RM0032
Reference manual

SPC56EL60 32-bit MCU family built on the embedded Power Architecture®

Introduction

The SPC56ELxx microcontroller is based on the Power Architecture® and targets the electric power steering, chassis, and safety applications that require a high safety integrity level.

All SPC56ELxx devices are built around a dual-core safety platform with an innovative safety concept targeting ISO26262 ASILD and IEC61508 SIL3 integrity levels[a]. To minimize additional software and module level features to reach this target, on-chip redundancy is provided for the critical components of the microcontroller (CPU core, DMA controller, interrupt controller, crossbar bus system, memory protection unit, Flash-memory controller and RAM controllers, peripheral bus bridge, system timers, and watchdog timer). Lock step Redundancy Checking Units are implemented at each output of this Sphere of Replication (SoR). ECC is available for on-chip RAM and Flash memories. A programmable fault collection and control unit monitors the integrity status of the device and provides flexible safe state control.

The host processor core of the SPC56ELxx is a CPU from the e200z4 family of compatible Power Architecture cores. The device's 5-stage pipeline dual issue core provides a high efficiency allowing high performance with minimum power dissipation.

The peripheral set is compatible with the SPC560P device family and provides high-end electrical motor control capability with very low CPU intervention due to the on-chip crosstriggering unit.

The SPC56ELxx is developed with high-performance 90-nm embedded Flash-memory technology that provides substantial cost reduction per feature and significant performance improvement.

a. All statements on functional safety in this chapter are under the condition that the requirements given in the Safety Application Guide are followed.

Contents

1 Preface 63

1.1 Overview 63
1.2 Audience 63
1.3 Chapter organization and device-specific information 63
1.4 Information about different device versions (“cuts”) 63
1.5 Acronyms and abbreviations 64
1.6 References 64

2 Introduction 65

2.1 The SPC56ELxx microcontroller 65
2.2 SPC56ELxx device summary 65
2.3 Device block diagram 67
2.4 Feature summary 69
2.5 Feature details 71
2.5.1 High-performance e200z4d core 71
2.5.2 Crossbar switch (XBAR) 72
2.5.3 Memory Protection Unit (MPU) 72
2.5.4 Enhanced Direct Memory Access (eDMA) 72
2.5.5 On-chip flash memory with ECC 73
2.5.6 On-chip SRAM with ECC 73
2.5.7 Platform flash memory controller 74
2.5.8 Platform Static RAM Controller (SRAMC) 74
2.5.9 Memory subsystem access time 75
2.5.10 Error Correction Status Module (ECSM) 75
2.5.11 Peripheral bridge (PBRIDGE) 76
2.5.12 Interrupt Controller (INTC) 76
2.5.13 System clocks and clock generation 77
2.5.14 Frequency-Modulated Phase-Locked Loop (FMPLL) 77
2.5.15 Main oscillator 78
2.5.16 Internal Reference Clock (RC) oscillator 78
2.5.17 Clock, reset, power, mode and test control modules (MC_CGM, MC_RGM, MC_PCU, and MC_ME) 79
2.5.18 Periodic Interrupt Timer Module (PIT) 79

2.5.19 System Timer Module (STM) . . . 79
2.5.20 Software Watchdog Timer (SWT) . . . 79
2.5.21 Fault Collection and Control Unit (FCCU) . . . 80
2.5.22 System Integration Unit Lite (SIUL) . . . 80
2.5.23 Non-Maskable Interrupt (NMI) . . . 80
2.5.24 Boot Assist Module (BAM) . . . 80
2.5.25 System Status and Configuration Module (SSCM) . . . 81
2.5.26 FlexCAN . . . 81
2.5.27 FlexRay . . . 82
2.5.28 Serial communication interface module (LINFlexD) . . . 82
2.5.29 Deserial Serial Peripheral Interface (DSPI) . . . 83
2.5.30 FlexPWM . . . 83
2.5.31 eTimer module . . . 85
2.5.32 Sine Wave Generator (SWG) . . . 85
2.5.33 Analog-to-Digital Converter module (ADC) . . . 86
2.5.34 Cross Triggering Unit (CTU) . . . 86
2.5.35 Cyclic Redundancy Checker (CRC) Unit . . . 87
2.5.36 Redundancy Control and Checker Unit (RCCU) . . . 87
2.5.37 Junction temperature sensor . . . 88
2.5.38 Nexus Port Controller (NPC) . . . 88
2.5.39 IEEE 1149.1 JTAG Controller (JTAGC) . . . 89
2.5.40 Voltage regulator / Power Management Unit (PMU) . . . 90
2.5.41 Built-In Self-Test (BIST) capability . . . 90

3 Memory Map . . . 91

4 Signal Description . . . 97

4.1 Package pinouts . . . 97
4.2 Supply pins . . . 135
4.3 System pins . . . 137
4.4 Pin muxing . . . 139
4.5 Mapping of ports to PGPDO/I registers . . . 163

5 Operating Modes . . . 164

5.1 Overview . . . 164
5.2 Lock Step Mode (LSM) . . . 164

5.3 Decoupled Parallel Mode (DPM) 165
5.4 Selecting LSM or DPM 165
5.4.1 Entering LSM 166
5.4.2 Entering DPM 166

6 Device Boot Modes 168
6.1 Boot mode functionality 168
6.2 Hardware configuration 168
6.2.1 Single Chip boot mode 168
6.3 Boot-sector search 169
6.3.1 Potential boot sectors 169
6.3.2 Reset Configuration Half-Word 170
6.3.3 Boot and alternate boot 171
6.4 Device behavior by boot mode 171
6.4.1 Single Chip Mode — Unsecured 171
6.4.2 Single Chip Mode — Secured 172
6.4.3 Serial Boot Loader Mode — Public Password Enabled 172
6.4.4 Serial Boot Loader Mode — Flash Memory Password Enabled 172
6.4.5 Standby Boot Mode 172
6.4.6 Static Mode 172

7 Device Security. 173
7.1 Security 173
7.1.1 Securing the microcontroller 173
7.1.2 Unsecuring the microcontroller 174
7.2 Serial access 175

8 Functional Safety 176
8.1 Overview 176
8.2 Redundancy 176
8.3 Built-In Self-Test (BIST) 176
8.3.1 BIST during boot 176
8.3.2 Software-triggered BIST during operation 177
8.3.3 Software-triggered self-tests after boot 177
8.4 Memory error detection and correction 177
8.5 Monitoring 177

8.6 Software measures 177
8.7 Fault reaction 178
8.8 External measures 178

9 Analog-to-Digital Converter (ADC) 179
9.1 Introduction 179
9.2 Features 179
9.3 Memory map and register descriptions 180
9.3.1 Memory map 180
9.3.2 Control logic registers 183
9.3.3 Interrupt registers 186
9.3.4 Watchdog Threshold Interrupt Status Register (WTISR) 189
9.3.5 Watchdog Threshold Interrupt Mask Register (WTIMR) 190
9.3.6 DMA Enable Register (DMAE) 191
9.3.7 DMA Channel Select Register 0 (DMAR0) 191
9.3.8 Threshold Registers (THRHLR*n*) 192
9.3.9 Presampling registers 192
9.3.10 Conversion timing registers 194
9.3.11 Mask registers 196
9.3.12 Power Down Exit Delay Register (PDEDR) 197
9.3.13 Channel Data Registers (CDR*n*) 197
9.3.14 Channel Watchdog Selection Registers (CWSEL*n*) 198
9.3.15 Channel Watchdog Enable Register 0 (CWENR0) 199
9.3.16 Analog Watchdog Out of Range Register 0 (AWORR0) 200
9.3.17 Self test registers 200
9.4 Functional description 215
9.4.1 Inter-module communication 215
9.4.2 Analog channel conversion 216
9.4.3 Analog clock generator and conversion timings 219
9.4.4 ADC sampling and conversion timing 219
9.4.5 Presampling 220
9.4.6 Programmable analog watchdog 221
9.4.7 DMA functionality 222
9.4.8 Interrupts 222
9.4.9 Power-down mode 223
9.4.10 Auto-clock-off mode 223

9.4.11 Self testing . . . 223

10 Boot Assist Module (BAM) . . . 232

10.1 Overview . . . 232
10.2 Features . . . 232
10.3 Memory map . . . 233
10.4 Functional description . . . 233
10.4.1 Entering boot modes . . . 233
10.4.2 Boot through BAM . . . 234
10.4.3 UART Boot — autobaud disabled . . . 239
10.4.4 CAN Boot — autobaud disabled . . . 240
10.4.5 Boot with Autobaud feature [cut2/3 only] . . . 242
10.4.6 Reading from Test Flash [cut2/3only] . . . 247
10.4.7 Inhibiting BAM operation . . . 247
10.4.8 Interrupt . . . 248

11 Clock Architecture . . . 249

11.1 Clock generation . . . 249
11.2 Clock distribution . . . 250
11.3 Detailed module descriptions . . . 252

12 Clock Generation Module (MC_CGM). . . . 253

12.1 Introduction . . . 253
12.1.1 Overview . . . 253
12.1.2 Features . . . 255
12.2 External signal description . . . 255
12.3 Memory map and register definition . . . 255
12.3.1 Register descriptions . . . 259
12.4 Functional description . . . 268
12.4.1 System clock generation . . . 268
12.4.2 Auxiliary clock generation . . . 269
12.4.3 Functional description of dividers . . . 272
12.4.4 Output clock multiplexing . . . 272
12.4.5 Output Clock Division Selection . . . 273

13 Clock Monitor Unit (CMU) . . . 274

13.1 Overview . . . 274
13.2 Main features . . . 274
13.3 Memory map and register description . . . 274
13.3.1 Control status register (CMU_CSR) . . . 275
13.3.2 Frequency display register (CMU_FDR) . . . 276
13.3.3 High-frequency reference register A (CMU_HFREFR_A) . . . 277
13.3.4 Low-frequency reference register A (CMU_LFREFR_A) . . . 277
13.3.5 Interrupt status register (CMU_ISR) . . . 278
13.3.6 Measurement duration register (CMU_MDR) . . . 279
13.4 Functional description . . . 279
13.4.1 XOSC clock monitor . . . 280

14 Cross-Triggering Unit (CTU) . . . 283
14.1 Introduction . . . 283
14.2 Block diagram . . . 283
14.3 CTU overview . . . 283
14.4 Functional description . . . 284
14.4.1 Interaction with other peripherals . . . 284
14.4.2 Trigger events features . . . 285
14.4.3 Trigger generator subunit (TGS) . . . 285
14.4.4 TGS in triggered mode . . . 286
14.4.5 TGS in sequential mode . . . 287
14.4.6 TGS counter . . . 288
14.5 Scheduler subunit (SU) . . . 289
14.5.1 ADC commands list . . . 291
14.5.2 ADC commands list format . . . 291
14.5.3 ADC results . . . 292
14.6 Reload mechanism . . . 292
14.7 Power safety mode . . . 293
14.7.1 STOP mode . . . 293
14.8 Interrupts and DMA requests . . . 294
14.8.1 DMA support . . . 294
14.8.2 CTU faults and errors . . . 294
14.8.3 CTU interrupt/DMA requests . . . 295
14.9 Conversion time evaluate . . . 297
14.10 Memory map . . . 297

14.10.1 Trigger generator subunit input selection register (TGSISR) 299
14.10.2 Trigger generator subunit control register (TGSCR) 301
14.10.3 TxCR - Trigger x compare register (x = 0,...,7) 302
14.10.4 TGS counter compare register (TGSCCR) 302
14.10.5 TGS counter reload register (TGSCRR) 302
14.10.6 Commands list control register 1 (CLCR1) 303
14.10.7 Commands list control register 2 (CLCR2) 304
14.10.8 Trigger handler control registers (THCR1 and THCR2) 305
14.10.9 Commands list register x (x = 1,...,24) (CLRx) 306
14.10.10 Cross triggering unit error flag register (CTUEFR) 308
14.10.11 Cross triggering unit interrupt flag register (CTUIFR) 309
14.10.12 Cross triggering unit interrupt/DMA register (CTUIR) 310
14.10.13 Control ON time register (COTR) 310
14.10.14 Cross triggering unit control register (CTUCR) 311
14.10.15 Cross triggering unit digital filter (CTUDF) 312
14.10.16 Cross triggering unit expected value A (CTU_EXP_A) 313
14.10.17 Cross triggering unit expected value B (CTU_EXP_B) 313
14.10.18 Cross triggering unit counter range (CTU_CNTRNG) 313
14.10.19 FIFO DMA control register (FDCR) 314
14.10.20 FIFO control register (FCR) 315
14.10.21 FIFO threshold (FTH) 316
14.10.22 FIFO status register (FST) 317
14.10.23 FIFO Right aligned data x (x = 0,...,3) (FRx) 319
14.10.24 FIFO signed Left aligned data x (x = 0,...,3) (FLx) 319

15 Crossbar Switch (XBAR) 321

15.1 Information specific to this device 321
15.1.1 Register availability 321
15.1.2 MPR reset value 321
15.1.3 max_halt signal unavailable 321
15.1.4 Logical master IDs 321
15.1.5 Master port allocation 322
15.1.6 Slave port allocation 322
15.2 Introduction 323
15.2.1 Overview 323
15.2.2 Features 323
15.2.3 Limitations 323

15.2.4 General Operation . . . 323
15.3 XBAR registers . . . 324
15.3.1 Register summary . . . 324
15.3.2 XBAR register descriptions . . . 326
15.3.3 Coherency . . . 333
15.4 Function . . . 333
15.4.1 Arbitration . . . 333
15.4.2 Priority Assignment . . . 335
15.4.3 Master Port Functionality . . . 335
15.4.4 Slave Port Functionality . . . 338
15.5 Initialization/Application Information . . . 344
15.6 Interface . . . 344
15.6.1 Overview . . . 344
15.6.2 Master Ports . . . 344
15.6.3 Slave Ports . . . 345

16 Cyclic Redundancy Checker (CRC) Unit . . . 347
16.1 Introduction . . . 347
16.1.1 Glossary . . . 347
16.2 Features . . . 347
16.3 Block diagram . . . 347
16.4 Signal description . . . 348
16.5 Register description . . . 348
16.5.1 CRC Configuration Register (CRC_CFG) . . . 349
16.5.2 CRC Input Register (CRC_INP) . . . 350
16.5.3 CRC Current Status Register (CRC_CSTAT) . . . 351
16.5.4 CRC Output Register (CRC_OUTP) . . . 352
16.6 Functional description . . . 352
16.7 Use cases and limitations . . . 354

17 Deserial Serial Peripheral Interface (DSPI) . . . 357
17.1 Introduction . . . 357
17.1.1 Overview . . . 357
17.1.2 Features . . . 358
17.1.3 DSPI configurations . . . 358
17.1.4 Modes of operation . . . 359

17.2 External signal description 360
17.2.1 PCS[0]/SS — Peripheral Chip Select/Slave Select 360
17.2.2 PCS[1] - PCS[3] — Peripheral Chip Selects 1–3 361
17.2.3 PCS[4] — Peripheral Chip Select 4 361
17.2.4 PCS[5]/PCSS — Peripheral Chip Select 5/Peripheral Chip Select Strobe 361
17.2.5 PCS[6] - PCS[7] — Peripheral Chip Selects 6 - 7 361
17.2.6 SIN — Serial Input 361
17.2.7 SOUT — Serial Output 361
17.2.8 SCK — Serial Clock 361
17.3 Memory map and register definition 361
17.3.1 Memory map 361
17.3.2 Register descriptions 362
17.4 Functional description 381
17.4.1 Start and Stop of DSPI Transfers 382
17.4.2 Serial Peripheral Interface (SPI) Configuration 383
17.4.3 DSPI baud rate and clock delay generation 386
17.4.4 Transfer formats 388
17.4.5 Continuous Serial Communications Clock 397
17.4.6 Interrupts/DMA Requests 398
17.4.7 Power Saving Features 399
17.5 Initialization/Application Information 400
17.5.1 How to Manage DSPI Queues 400
17.5.2 Switching Master and Slave Mode 401
17.5.3 Baud Rate Settings 401
17.5.4 Delay Settings 402
17.5.5 Calculation of FIFO Pointer Addresses 402

18 e200z4d Core Complex Overview 405
18.1 Overview 405
18.2 Features 407
18.2.1 Execution Unit Features 407
18.2.2 L1 Cache Features 408
18.2.3 Memory Management Unit Features 409
18.2.4 Exernal core complex interface features 409
18.2.5 Nexus 3+ Features 409

18.3 Programming model 410
18.3.1 Register set 410
18.3.2 Instruction set 413
18.3.3 Interrupts and Exception Handling 414
18.4 Microarchitecture summary 416
18.5 Availability of detailed documentation 417

19 eDMA Channel Mux (DMA_MUX) 418
19.1 Introduction 418
19.1.1 Overview 418
19.1.2 Features 418
19.1.3 Modes of operation 419
19.2 External signal description 419
19.2.1 Overview 419
19.3 Memory map and register definition 419
19.3.1 Register descriptions 420
19.4 DMA_MUX request source slot mapping 421
19.5 DMA_MUX trigger inputs 422
19.6 Functional description 422
19.6.1 DMA Channels with periodic triggering capability 422
19.6.2 DMA channels with no triggering capability 424
19.6.3 "Always Enabled" DMA sources 424
19.7 Initialization/application Information 425
19.7.1 Reset 425
19.7.2 Enabling and configuring sources 425

20 Enhanced Direct Memory Access (eDMA) 430
20.1 Introduction 430
20.1.1 Overview 431
20.1.2 Features 431
20.2 Memory map/register definition 432
20.2.1 Register descriptions 433
20.3 Functional description 462
20.3.1 eDMA microarchitecture 462
20.3.2 eDMA basic data flow 463
20.3.3 eDMA performance 466

20.4 Initialization/application information 470
20.4.1 eDMA initialization 470
20.4.2 eDMA programming errors 470
20.4.3 eDMA arbitration mode considerations 470
20.4.4 eDMA transfer 471
20.4.5 TCD status 473
20.4.6 Channel linking 474
20.4.7 Dynamic programming 475

21 Enhanced Motor Control Timer (eTimer) 477
21.1 Introduction 477
21.2 Features 479
21.3 External signal descriptions 480
21.3.1 TIO[5:0] - Timer Input/Outputs 480
21.3.2 TAI[2:0] - Timer Auxiliary Inputs 480
21.4 Memory map and register definition 481
21.4.1 Module memory map 481
21.4.2 Register descriptions 481
21.4.3 Timer Channel Registers 482
21.4.4 Watchdog Timer Registers 497
21.4.5 Configuration Registers 498
21.5 Functional description 500
21.5.1 General 500
21.5.2 Counting Modes 501
21.5.3 Other Features 507
21.6 Interrupts 508
21.7 DMA 509

22 Error Correction Status Module (ECSM) 510
22.1 Introduction 510
22.2 Overview 510
22.3 Features 510
22.4 Memory map and register description 510
22.4.1 Memory map 510
22.4.2 Register description 511

23 Fault Collection and Control Unit (FCCU) 536

23.1 Introduction 536

23.1.1 Glossary and acronyms 536

23.2 Main features 537

23.3 Block diagram 538

23.4 Signal description 539

23.5 Register interface 539

23.6 Memory map and register description 539

23.6.1 FCCU Control Register (FCCU_CTRL) 541

23.6.2 FCCU CTRL Key Register (FCCU_CTRLK) 544

23.6.3 FCCU Configuration Register (FCCU_CFG) 545

23.6.4 FCCU CF Configuration Register (FCCU_CF_CFG0..3) 546

23.6.5 FCCU NCF Configuration Register (FCCU_NCF_CFG0..3) 548

23.6.6 FCCU CFS Configuration Register (FCCU_CFS_CFG0..7) 549

23.6.7 FCCU NCFS Configuration Register (FCCU_NCFS_CFG0..7) 550

23.6.8 FCCU CF Status Register (FCCU_CF_S0..3) 550

23.6.9 FCCU CF Key Register (FCCU_CFK) 552

23.6.10 FCCU NCF Status Register (FCCU_NCF_S0..3) 553

23.6.11 FCCU NCF Key Register (FCCU_NCFK) 554

23.6.12 FCCU NCF Enable Register (FCCU_NCF_E0..3) 555

23.6.13 FCCU NCF Time-out Enable Register (FCCU_NCF_TOE0..3) 556

23.6.14 FCCU NCF Time-out Register (FCCU_NCF_TO) 557

23.6.15 FCCU CFG Timeout Register (FCCU_CFG_TO) 557

23.6.16 FCCU IO Control Register (FCCU_EINOUT) 558

23.6.17 FCCU Status Register (FCCU_STAT) 559

23.6.18 FCCU SC Freeze Status Register (FCCU_SCFS) 560

23.6.19 FCCU CF Fake Register (FCCU_CFF) 562

23.6.20 FCCU NCF Fake Register (FCCU_NCFF) 563

23.6.21 FCCU IRQ Status Register (FCCU_IRQ_STAT) 563

23.6.22 FCCU IRQ Enable Register (FCCU_IRQ_EN) 564

23.6.23 FCCU XTMR Register (FCCU_XTMR) 565

23.6.24 FCCU MCS Register (FCCU_MCS) 566

23.7 Functional description 567

23.7.1 Definitions 567

23.7.2 FSM description 567

23.7.3 Self-checking capabilities 569

23.7.4 Reset interface . . . 570
23.7.5 Fault priority scheme and nesting . . . 570
23.7.6 Fault recovery . . . 571
23.7.7 WKUP/NMI interface . . . 577
23.7.8 STCU interface . . . 578
23.7.9 NVM interface . . . 579
23.7.10 FCCU_F interface . . . 580
23.7.11 Fault mapping . . . 584

24 Flash Memory . . . 588
24.1 Flash memory block (C90FL) . . . 588
24.1.1 C90FL block overview . . . 588
24.1.2 C90FL block features . . . 590
24.1.3 C90FL modes of operation . . . 590
24.1.4 C90FL block diagram . . . 590
24.1.5 C90FL memory map and register definition . . . 591
24.1.6 C90FL flash memory functional description (user mode) . . . 613
24.1.7 User option bits . . . 620
24.1.8 Test flash memory . . . 622
24.2 Dual-ported platform flash memory controller (PFLASH2P) . . . 623
24.2.1 Introduction . . . 623
24.2.2 Registers . . . 627
24.2.3 Functional description . . . 633

25 FlexCAN Module . . . 638
25.1 Introduction . . . 638
25.1.1 Overview . . . 639
25.1.2 FlexCAN module features . . . 639
25.1.3 Modes of operation . . . 640
25.2 External signal description . . . 641
25.2.1 CAN Rx . . . 641
25.2.2 CAN Tx . . . 641
25.3 Memory map and register definition . . . 641
25.3.1 FlexCAN memory mapping . . . 641
25.3.2 Message Buffer Structure . . . 643
25.3.3 Rx FIFO Structure . . . 646
25.3.4 Register descriptions . . . 648

25.4 Functional Description 664
25.4.1 Overview 664
25.4.2 Transmit Process 665
25.4.3 Arbitration process 665
25.4.4 Receive Process 666
25.4.5 Matching Process 668
25.4.6 Data Coherence 669
25.4.7 Rx FIFO 671
25.4.8 CAN Protocol Related Features 672
25.4.9 Modes of Operation Details 677
25.4.10 Interrupts 678
25.4.11 Bus Interface 679
25.5 Initialization/Application Information 679
25.5.1 FlexCAN Initialization Sequence 679
25.5.2 FlexCAN addressing and RAM size configurations 680

26 Flexible Motor Control Pulse Width Modulator Module (FlexPWM) . 681
26.1 Introduction 681
26.1.1 Overview 681
26.1.2 Features 682
26.1.3 Modes of operation 682
26.1.4 Block diagrams 684
26.2 External signal descriptions 685
26.2.1 PWMA[n] and PWMB[n] — External PWM Pair 685
26.2.2 PWMX[n] — Auxiliary PWM signal 685
26.2.3 FAULT[n] — Fault inputs 685
26.2.4 EXT_SYNC — External Synchronization Signal 686
26.2.5 EXT_FORCE — External Output Force Signal 686
26.2.6 EXTA[n] and EXTB[n] — Alternate PWM Control Signals 686
26.2.7 OUT_TRIG0[n] and OUT_TRIG1[n] — Output Triggers 686
26.2.8 EXT_CLK — External Clock Signal 686
26.3 Functional Description 686
26.3.1 Block Diagram 686
26.3.2 PWM Capabilities 686
26.3.3 Functional details 696
26.3.4 PWM generator loading 713

26.4 Memory Map and Registers 716
26.4.1 Module Memory Map 716
26.4.2 Register Descriptions 717
26.4.3 Submodule Registers 717
26.4.4 Configuration registers 734
26.4.5 Fault Channel Registers 741
26.5 Interrupts 744
26.6 DMA 745

27 FlexRay Communication Controller 746
27.1 Introduction 746
27.1.1 Reference 746
27.1.2 Glossary 746
27.1.3 Color coding 747
27.1.4 Overview 747
27.1.5 Features 749
27.1.6 Modes of operation 750
27.2 External Signal Description 751
27.2.1 Detailed Signal Descriptions 751
27.3 Controller Host Interface Clocking 752
27.4 Protocol Engine Clocking 752
27.5 Memory Map and Register Description 753
27.5.1 Memory Map 753
27.5.2 Register Descriptions 757
27.6 Functional Description 840
27.6.1 Message Buffer Concept 841
27.6.2 Physical Message Buffer 841
27.6.3 Message Buffer Types 842
27.6.4 FlexRay Memory Area Layout 847
27.6.5 Physical Message Buffer Description 850
27.6.6 Individual Message Buffer Functional Description 860
27.6.7 Individual Message Buffer Search 885
27.6.8 Individual Message Buffer Reconfiguration 888
27.6.9 Receive FIFOs 889
27.6.10 Channel Device Modes 895
27.6.11 External Clock Synchronization 897

27.6.12 Sync Frame ID and Sync Frame Deviation Tables 898
27.6.13 MTS Generation 901
27.6.14 Key Slot Transmission 902
27.6.15 Sync Frame Filtering 903
27.6.16 Strobe Signal Support 904
27.6.17 Timer Support 905
27.6.18 Slot Status Monitoring 906
27.6.19 System Bus Access 909
27.6.20 Interrupt Support 910
27.6.21 Lower Bit Rate Support 914
27.6.22 PE Data Memory (PE DRAM) 915
27.6.23 CHI Lookup-Table Memory (CHI LRAM) 916
27.6.24 Memory Content Error Detection 917
27.6.25 Memory Error Injection 921
27.7 Application Information 922
27.7.1 Initialization Sequence 922
27.7.2 CHI LRAM Error Injection out of POC:default config 924
27.7.3 PE DRAM Error Injection out of POC:default config 924
27.7.4 Shut Down Sequence 924
27.7.5 Number of Usable Message Buffers 924
27.7.6 Protocol Control Command Execution 925
27.7.7 Message Buffer Search on Simple Message Buffer Configuration . . . 926

28 Frequency-Modulated Phase-Locked Loop (FMPLL) 929
28.1 Introduction 929
28.2 Overview 929
28.3 Features 930
28.4 Memory map 930
28.5 Register descriptions 930
28.5.1 Control Register (CR) 931
28.5.2 Modulation Register (MR) 933
28.6 Functional description 934
28.6.1 Normal mode 934
28.6.2 Progressive clock switching 935
28.6.3 Normal Mode with frequency modulation 936
28.6.4 Powerdown mode 937

28.7 Requirements 937
28.8 Recommendations 937

29 Interrupt Controller (INTC). 939
29.1 Introduction 939
29.1.1 Module overview 939
29.1.2 Block diagram 939
29.1.3 Features 940
29.2 Modes of Operation 941
29.2.1 Normal Mode 941
29.2.2 Debug Mode 942
29.2.3 Stop Mode 942
29.2.4 Factory Test Mode 942
29.3 External Signal Description 942
29.4 Memory map/register definition 942
29.4.1 Memory map 942
29.4.2 Register Information 943
29.4.3 INTC Block Configuration Register (INTC_BCR) 943
29.4.4 INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0) . 944
29.4.5 INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0) 945
29.4.6 INTC End of Interrupt Register for Processor 0 (INTC_EOIR_PRC0) . 946
29.4.7 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3 - INTC_SSCIR4_7) 947
29.4.8 INTC Priority Select Registers (INTC_PSR0_3 - INTC_PSR252_255) 948
29.5 Functional Description 949
29.5.1 Interrupt Request Sources 949
29.5.2 Priority Management 950
29.5.3 Handshaking with processor 951
29.6 Initialization/application information 953
29.6.1 Initialization flow 953
29.6.2 Interrupt exception handler 954
29.6.3 Code Compression’s Impact on Vector Table 956
29.6.4 ISR, RTOS, and Task Hierarchy 956
29.6.5 Order of Execution 956
29.6.6 Priority Ceiling Protocol 957
29.6.7 Selecting Priorities According to Request Rates and Deadlines 958

29.6.8 Software settable Interrupt Requests . . . 959
29.6.9 Lowering Priority Within an ISR . . . 960
29.6.10 Negating an Interrupt Request Outside of its ISR . . . 960
29.6.11 Examining LIFO contents . . . 961
29.7 Interrupt sources . . . 961

30 JTAG Controller (JTAGC) . . . 970
30.1 Introduction . . . 970
30.1.1 Overview . . . 970
30.1.2 Features . . . 971
30.1.3 Modes of operation . . . 971
30.2 External signal description . . . 972
30.2.1 Overview . . . 972
30.2.2 Detailed Signal Descriptions . . . 972
30.3 Register Definition . . . 973
30.3.1 Register descriptions . . . 973
30.4 Functional Description . . . 976
30.4.1 JTAGC Reset Configuration . . . 976
30.4.2 IEEE 1149.1-2001 (JTAG) Test Access Port . . . 976
30.4.3 TAP Controller State Machine . . . 977
30.4.4 JTAGC Block Instructions . . . 979
30.4.5 Boundary Scan . . . 981
30.5 Initialization/Application Information . . . 981

31 LIN Controller (LINFlexD) . . . 982
31.1 Introduction . . . 982
31.2 Main features . . . 982
31.2.1 LIN mode features . . . 983
31.2.2 UART mode features . . . 983
31.3 The LIN protocol . . . 984
31.3.1 Dominant and recessive logic levels . . . 984
31.3.2 LIN frames . . . 984
31.3.3 LIN header . . . 985
31.3.4 Response . . . 986
31.4 LINFlexD and software intervention . . . 987
31.5 Summary of operating modes . . . 987

31.6 Controller-level operating modes 988
31.6.1 Initialization mode 988
31.6.2 Normal mode 989
31.6.3 Sleep (low-power) mode 989
31.7 LIN modes 989
31.7.1 Master mode 989
31.7.2 Slave mode 990
31.7.3 Slave mode with identifier filtering 993
31.7.4 Slave mode with automatic resynchronization 996
31.8 Test modes 997
31.8.1 Loop Back mode 997
31.8.2 Self Test mode 998
31.9 UART mode 998
31.9.1 Data frame structure 998
31.9.2 Buffer 999
31.9.3 UART transmitter 1000
31.9.4 UART receiver 1001
31.10 Memory map and register description 1003
31.10.1 LIN control register 1 (LINCR1) 1004
31.10.2 LIN interrupt enable register (LINIER) 1007
31.10.3 LIN status register (LINSR) 1009
31.10.4 LIN error status register (LINESR) 1012
31.10.5 UART mode control register (UARTCR) 1013
31.10.6 UART mode status register (UARTSR) 1016
31.10.7 LIN timeout control status register (LINTCSR) 1018
31.10.8 LIN output compare register (LINOCR) 1019
31.10.9 LIN timeout control register (LINTOCR) 1020
31.10.10 LIN fractional baud rate register (LINFBRR) 1021
31.10.11 LIN integer baud rate register (LINIBRR) 1021
31.10.12 LIN checksum field register (LINCFR) 1022
31.10.13 LIN control register 2 (LINCR2) 1023
31.10.14 Buffer identifier register (BIDR) 1024
31.10.15 Buffer data register least significant (BDRL) 1025
31.10.16 Buffer data register most significant (BDRM) 1026
31.10.17 Identifier filter enable register (IFER) 1027
31.10.18 Identifier filter match index (IFMI) 1027

31.10.19 Identifier filter mode register (IFMR) 1028
31.10.20 Identifier filter control registers (IFCR0–IFCR15) 1029
31.10.21 Global control register (GCR) 1030
31.10.22 UART preset timeout register (UARTPTO) 1032
31.10.23 UART current timeout register (UARTCTO) 1032
31.10.24 DMA Tx enable register (DMATXE) 1034
31.10.25 DMA Rx enable register (DMARXE) 1034
31.11 DMA interface 1035
31.11.1 Master node, TX mode 1035
31.11.2 Master node, RX mode 1038
31.11.3 Slave node, TX mode 1040
31.11.4 Slave node, RX mode 1043
31.11.5 UART node, TX mode 1046
31.11.6 UART node, RX mode 1048
31.11.7 Use cases and limitations 1051
31.12 Functional description 1052
31.12.1 8-bit timeout counter 1052
31.12.2 Interrupts 1053
31.12.3 Fractional baud rate generation 1054
31.13 Programming considerations 1055
31.13.1 Master node 1056
31.13.2 Slave node 1057
31.13.3 Extended frames 1060
31.13.4 Timeout 1061
31.13.5 UART mode 1061

32 Memory Protection Unit (MPU) 1062
32.1 Introduction 1062
32.2 Block diagram 1062
32.3 Features 1064
32.4 Modes of operation 1064
32.5 External signal description 1065
32.6 Memory map and register definition 1065
32.6.1 MPU Control/Error Status Register (MPU_CESR) 1066
32.6.2 MPU Error Address Register, Slave Port n (MPU_EARn) 1067
32.6.3 MPU Error Detail Register, Slave Port n (MPU_EDRn) 1068

32.6.4 MPU Region Descriptor n (MPU_RGDn) 1069
32.6.5 MPU Region Descriptor Alternate Access Control n (MPU_RGDAACn) 1073
32.7 Functional description 1075
32.7.1 Access evaluation macro 1076
32.7.2 Putting it all together and AHB error terminations 1078
32.8 Initialization information 1079
32.9 Application information 1079

33 Mode Entry Module (MC_ME) 1082
33.1 Introduction 1082
33.1.1 Overview 1082
33.1.2 Features 1084
33.1.3 Modes of operation 1084
33.2 External signal description 1085
33.3 Memory map and register definition 1085
33.3.1 Memory map 1085
33.3.2 Register description 1092
33.4 Functional description 1113
33.4.1 Mode Transition Request 1113
33.4.2 Mode details 1114
33.4.3 Mode transition process 1117
33.4.4 Protection of mode configuration registers 1125
33.4.5 Mode Transition Interrupts 1125
33.4.6 Peripheral Clock Gating 1127
33.4.7 Application example 1127

34 Nexus Crossbar Slave Port Data Trace Module (NXSS) [cut2/3 only] 1129
34.1 Introduction 1129
34.2 Block diagram 1129
34.3 Features 1130
34.4 External signal description 1130
34.4.1 Rules for output messages 1130
34.4.2 Auxiliary port arbitration 1130
34.5 NXSS programmer model 1130

34.5.1 Development Control Registers (DC1 and DC2) 1131
34.5.2 Watchpoint Trigger Register (WT) 1132
34.5.3 Data Trace Control Register (DTC) 1133
34.5.4 Data Trace Start Address Registers 1 and 2 (DTSA1 and DTSA2) . . 1134
34.5.5 Data Trace End Address Registers 1 and 2 (DTEA1 and DTEA2) . . 1135
34.5.6 Breakpoint / Watchpoint Control Register 1 (BWC1) 1135
34.5.7 Breakpoint / Watchpoint Control Register 2 (BWC2) 1136
34.5.8 Breakpoint/Watchpoint Address Registers 1 and 2 (BWA1 and BWA2) 1137
34.5.9 Unimplemented registers 1137
34.6 Functional description 1137
34.6.1 TCODEs supported by NXSS_0 and NXSS_1 1137
34.6.2 Data Trace 1139
34.6.3 Data Trace Messaging (DTM) 1139
34.6.4 DTM Message Formats 1139
34.6.5 DTM operation 1142
34.7 Watchpoint support 1143
34.7.1 Watchpoint messaging 1143
34.7.2 Watchpoint error message 1143

35 Nexus Port Controller (NPC) 1145

35.1 Information specific to this device 1145
35.1.1 Parameter values 1145
35.1.2 Unavailable features 1145
35.2 Introduction 1145
35.2.1 Overview 1146
35.2.2 Features 1147
35.2.3 Modes of operation 1147
35.3 External signal description 1148
35.3.1 Overview 1148
35.3.2 Detailed signal descriptions 1149
35.4 Register definition 1150
35.4.1 Register descriptions 1151
35.5 Functional description 1155
35.5.1 NPC reset configuration 1155
35.5.2 Auxiliary Output Port 1155

35.5.3 IEEE 1149.1-2001 (JTAG) TAP 1158
35.5.4 Nexus JTAG Port Sharing 1163
35.5.5 MCKO 1163
35.5.6 EVTO Sharing 1163
35.5.7 Nexus Reset Control 1163
35.5.8 System Clock Locked Indication 1164
35.6 Initialization/Application Information 1164
35.6.1 Accessing NPC tool-mapped registers 1164

36 Oscillators 1165
36.1 IRCOSC 16 MHz internal RC oscillator 1165
36.2 XOSC external oscillator 1166
36.2.1 Functional description 1166
36.2.2 NVM interface 1167
36.2.3 Register description 1167

37 Periodic Interrupt Timer (PIT) 1169
37.1 Introduction 1169
37.1.1 Overview 1169
37.1.2 Features 1169
37.2 Signal description 1170
37.3 Memory map and register description 1170
37.3.1 Memory map 1170
37.3.2 Register descriptions 1170
37.4 Functional Description 1175
37.4.1 General 1175
37.4.2 Interrupts 1176
37.5 Initialization and Application Information 1177
37.5.1 Example Configuration 1177

38 Peripheral Bridge (PBRIDGE) 1178
38.1 Introduction 1178
38.2 Block interface 1178
38.3 Features 1179
38.4 Memory map and register description 1179
38.4.1 Register access 1179

38.4.2 Memory map 1179
38.4.3 Register descriptions 1180
38.5 Functional description 1183
38.5.1 Access support 1184
38.5.2 General operation 1184

39 Power Control Unit (MC_PCU) 1185
39.1 Introduction 1185
39.1.1 Overview 1185
39.1.2 Features 1185
39.2 External Signal Description 1186
39.3 Memory Map and Register Definition 1186
39.3.1 Memory Map 1186
39.3.2 Register Descriptions 1186

40 Power Management Unit (PMU) 1188
40.1 Overview 1188
40.2 Block diagram 1188
40.3 High-power regulators 1189
40.4 High- and low-voltage detectors (HVD, LVD) 1189
40.4.1 Current comparator 1190
40.5 Power-up and initialization 1190
40.6 Built In self-test (BIST) 1190
40.7 Memory map and register description 1192
40.7.1 PMUCTRL status register (PMUCTRL_STATUS) 1193
40.7.2 PMUCTRL control register (PMUCTRL_CTRL) 1194
40.7.3 PMUCTRL mask fault register (PMUCTRL_MASKF) 1195
40.7.4 PMUCTRL fault monitor register (PMUCTRL_FAULT) 1196
40.7.5 PMUCTRL interrupt request status register (PMUCTRL_IRQS) 1197
40.7.6 PMUCTRL interrupt request enable register (PMUCTRL_IRQE) . . . 1198

41 Register Protection (REG_PROT) 1200
41.1 Introduction 1200
41.1.1 Overview 1200
41.1.2 Features 1200
41.1.3 Modes of operation 1201

41.2 External signal description 1201
41.3 Memory map and register definition 1201
41.3.1 Memory map 1202
41.3.2 Register descriptions 1203
41.4 Functional description 1205
41.4.1 General 1205
41.4.2 Change lock settings 1206
41.4.3 Access errors 1208
41.5 Initialization/application information 1209
41.5.1 Reset 1209
41.5.2 Writing C code using the register protection scheme 1209
41.6 SPC56ELxx registers under protection 1210

42 Reset Generation Module (MC_RGM). 1251
42.1 Introduction 1251
42.1.1 Overview 1251
42.1.2 Features 1253
42.1.3 Reset sources 1253
42.2 External signal description 1254
42.3 Memory map and register definition 1254
42.3.1 Register descriptions 1256
42.4 Functional description 1269
42.4.1 Reset State Machine 1269
42.4.2 Destructive Resets 1272
42.4.3 External Reset 1272
42.4.4 Functional resets 1272
42.4.5 Alternate event generation 1273
42.4.6 Application Information 1274
42.4.7 Boot mode capturing 1297

43 Self-Test Control Unit (STCU) 1298
43.1 Introduction 1298
43.1.1 Acronyms, abbreviations, and terms 1298
43.1.2 Safety Integrity Subsystem 1299
43.1.3 Integrity SW operations 1301
43.2 STCU main features 1301

43.3 Block diagram and components 1302
43.4 Memory map and register definition 1302
43.4.1 Memory map 1303
43.4.2 Register conventions 1304
43.4.3 Detailed register descriptions 1304
43.5 LBIST partitioning 1315
43.6 MBIST partitioning 1319
43.7 Self-test bypass and MBIST-only mode 1320

44 Semaphore Unit (SEMA4) 1322
44.1 Introduction 1322
44.1.1 Block diagram 1322
44.1.2 Features 1324
44.1.3 Modes of operation 1324
44.2 Signal description 1324
44.3 Memory map and register description 1324
44.3.1 Semaphores gate *n* register (SEMA4_GATE*n*) 1325
44.3.2 Semaphores processor *n* IRQ notification enable (SEMA4_CP{0,1}INE) 1326
44.3.3 Semaphores processor *n* IRQ notification (SEMA4_CP{0,1}NTF) . . 1327
44.3.4 Semaphores (secure) reset gate *n* (SEMA4_RSTGT) 1327
44.3.5 Semaphores (secure) Reset IRQ notification (SEMA4_RSTNTF) . . 1329
44.4 Functional description 1331
44.4.1 Semaphore usage 1333
44.5 Initialization information 1333
44.6 Application information 1333
44.7 DMA requests 1335

45 Sine Wave Generator (SWG) 1336
45.1 Introduction 1336
45.2 Features 1336
45.3 Memory map and register description 1336
45.3.1 SWG control register (SWG_CTRL) 1336
45.3.2 SWG status register (SWG_STAT) 1338
45.4 Functional description 1339

45.4.1 SWG operation after a power-on reset 1339
45.4.2 Output sine wave frequency 1339
45.4.3 Output sine wave amplitude 1339
45.5 Initialization / application information 1339
45.5.1 Changing the output frequency 1339
45.5.2 Preserving the SWG_CTRL data 1340

46 Software Watchdog Timer (SWT) 1341
46.1 Introduction 1341
46.1.1 Overview 1341
46.1.2 Features 1341
46.1.3 Modes of Operation 1341
46.2 External Signal Description 1341
46.3 Memory Map and Register Definition 1341
46.3.1 Memory Map 1342
46.3.2 Register Descriptions 1342
46.4 Functional Description 1347

47 Static RAM (SRAM) 1349
47.1 Introduction 1349
47.2 SRAM operating mode 1349
47.3 Registers 1349
47.4 SRAM ECC mechanism 1349
47.4.1 Access timing 1350
47.4.2 Reset effects on SRAM accesses on cut1 1351
47.5 Functional description 1351
47.6 Initialization and Application Information 1351

48 System Integration Unit Lite (SIUL) 1352
48.1 Introduction 1352
48.2 Overview 1352
48.3 Features 1354
48.3.1 Register protection 1354
48.4 External signal description 1354
48.4.1 Detailed signal descriptions 1355

48.5 Memory map and register description 1355
48.5.1 SIUL memory map 1355
48.5.2 Register description 1356
48.6 Functional Description 1371
48.6.1 General 1371
48.6.2 Pad control 1371
48.6.3 General purpose input and output pads (GPIO) 1372
48.6.4 External interrupts 1373
48.7 Pin muxing 1374

49 System Status and Configuration Module (SSCM) 1375
49.1 Introduction 1375
49.1.1 Overview 1375
49.1.2 Features 1375
49.1.3 Modes of operation 1376
49.2 External signal description 1376
49.3 Memory map/register definition 1376
49.3.1 Register descriptions 1377
49.4 Functional description 1385
49.5 Initialization/application information 1385
49.5.1 Reset 1385

50 System Timer Module (STM) 1386
50.1 Introduction 1386
50.1.1 Overview 1386
50.1.2 Features 1386
50.1.3 Modes of operation 1386
50.2 External signal description 1386
50.3 Memory map and register definition 1386
50.3.1 Memory map 1386
50.3.2 Register descriptions 1387
50.4 Functional Description 1391

51 Temperature Sensor (TSENS) 1392
51.1 Introduction 1392
51.2 Features 1392

51.3 Signals 1392
51.4 Modes of operation 1393
51.5 Obtaining the device temperature using TSENS 1393
51.5.1 TSENS calibration constants 1393
51.5.2 Equations for converting TSENS voltage to device temperature 1394

52 Wakeup Unit (WKPU) 1395
52.1 Introduction 1395
52.1.1 Overview 1395
52.1.2 Features 1395
52.2 External signal description 1396
52.3 Memory map and register description 1396
52.3.1 Memory map 1396
52.3.2 Register descriptions 1396
52.4 Functional description 1399
52.4.1 General 1399
52.4.2 Non-maskable interrupts 1399

Appendix A Revision History. 1401

List of tables

Table 1. SPC56ELxx device summary . . . 66
Table 2. Platform memory access time summary . . . 75
Table 3. SPC56ELxx memory map, ordered by start address . . . 91
Table 4. SPC56ELxx memory map, ordered by module name . . . 93
Table 5. SRAM map for SPC56ELxx in LSM . . . 96
Table 6. SRAM map for SPC56ELxx in DPM . . . 96
Table 7. LQFP100 pin function summary . . . 100
Table 8. LQFP144 pin function summary . . . 108
Table 9. LFBGA257 pin function summary . . . 119
Table 10. Supply pins . . . 135
Table 11. System pins . . . 137
Table 12. Pin muxing . . . 139
Table 13. Mapping of ports to PGPDO registers . . . 163
Table 14. Mapping of ports to PGPDI registers . . . 163
Table 15. Hardware configuration . . . 168
Table 16. Bit access descriptions . . . 180
Table 17. ADC memory map . . . 180
Table 18. MCR field descriptions . . . 183
Table 19. MSR field descriptions . . . 185
Table 20. ISR field descriptions . . . 187
Table 21. CEOCFR0 field descriptions . . . 187
Table 22. IMR field descriptions . . . 188
Table 23. CIMR0 field descriptions . . . 189
Table 24. WTISR field descriptions . . . 190
Table 25. WTIMR field descriptions . . . 190
Table 26. DMAE field descriptions . . . 191
Table 27. DMAR0 field descriptions . . . 191
Table 28. THRHLR*n* field descriptions . . . 192
Table 29. PSCR field descriptions . . . 193
Table 30. Presampling voltages . . . 193
Table 31. PSR0 field descriptions . . . 193
Table 32. CTR0 field descriptions . . . 194
Table 33. Minimum AD_ck frequency . . . 195
Table 34. CTR1 field descriptions . . . 195
Table 35. NCMR0 field descriptions . . . 196
Table 36. JCMR0 field descriptions . . . 196
Table 37. PDEDR field descriptions . . . 197
Table 38. CDR*n* field descriptions . . . 197
Table 39. CWSEL*n* field descriptions . . . 199
Table 40. CWENR0 field descriptions . . . 199
Table 41. AWORR0 field descriptions . . . 200
Table 42. STCR1 field descriptions . . . 201
Table 43. STCR2 field descriptions . . . 202
Table 44. STCR3 field descriptions . . . 204
Table 45. STBRR field descriptions . . . 205
Table 46. STSR1 field descriptions . . . 206
Table 47. STSR2 field descriptions . . . 208
Table 48. STSR3 field descriptions . . . 208

Table 49. STSR4 field descriptions . . . 209
Table 50. STDR1 field descriptions . . . 209
Table 51. STDR2 field descriptions . . . 210
Table 52. STAW0R field descriptions . . . 211
Table 53. STAW1AR field descriptions . . . 211
Table 54. STAW1BR field descriptions . . . 212
Table 55. STAW2R field descriptions . . . 212
Table 56. STAW3R field descriptions . . . 213
Table 57. STAW4R field descriptions . . . 214
Table 58. STAW5R field descriptions . . . 214
Table 59. Values of WDGxH and WDGxL fields . . . 221
Table 60. Test channel activation . . . 224
Table 61. CTU-CLRx-ST1, ST0 meaning . . . 226
Table 62. CTU-CLRx-ALG1, ALG0 meaning . . . 227
Table 63. Algorithm S (STEP1) threshold comparison . . . 228
Table 64. BAM memory organization . . . 233
Table 65. Hardware configuration to select boot mode . . . 234
Table 66. Fields of SSCM STATUS register read by BAM to detect the chosen boot mode . . . 235
Table 67. LINFlex bit timing in UART mode . . . 240
Table 68. UART boot mode download protocol (autobaud disabled) . . . 240
Table 69. FlexCAN boot mode download protocol (autobaud disabled) . . . 241
Table 70. Clock distribution . . . 250
Table 71. MC_CGM register description . . . 255
Table 72. MC_CGM memory map . . . 256
Table 73. Output Clock Enable Register (CGM_OC_EN) Field Descriptions . . . 260
Table 74. Output Clock Division Select Register (CGM_OCDS_SC) Field Descriptions . . . 261
Table 75. System Clock Select Status Register (CGM_SC_SS) field descriptions . . . 262
Table 76. System Clock Divider Configuration Registers (CGM_SC_DC0) Field Descriptions . . . 262
Table 77. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC) Field Descriptions . . . 263
Table 78. Auxiliary Clock 0 Divider Configuration Registers (CGM_AC0_DC0...1) field descriptions . . . 264
Table 79. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC) Field Descriptions . . . 265
Table 80. Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0) field descriptions . . . 265
Table 81. Auxiliary Clock 2 Select Control Register (CGM_AC2_SC) Field Descriptions . . . 266
Table 82. Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0) field descriptions . . . 267
Table 83. Auxiliary Clock 3 Select Control Register (CGM_AC3_SC) Field Descriptions . . . 267
Table 84. Auxiliary Clock 4 Select Control Register (CGM_AC4_SC) Field Descriptions . . . 268
Table 85. CMU module summary . . . 274
Table 86. CMU base addresses . . . 274
Table 87. CMU memory map . . . 275
Table 88. CMU_CSR field descriptions . . . 275
Table 89. CMU_FDR field descriptions . . . 276
Table 90. CMU_HFREFR_A field descriptions . . . 277
Table 91. CMU_LFREFR_A fields descriptions . . . 277
Table 92. CMU_ISR field descriptions . . . 278
Table 93. CMU_MDR field descriptions . . . 279
Table 94. CTU interrupts . . . 295
Table 95. TGS registers . . . 297
Table 96. SU registers . . . 298
Table 97. FIFO registers . . . 298
Table 98. Other CTU registers . . . 298
Table 99. TGSISR Field descriptions . . . 299

Table 100. TGSCR field descriptions . . . 301
Table 101. TxCR field descriptions . . . 302
Table 102. TGSCCR field format . . . 302
Table 103. TGSCRR field descriptions . . . 303
Table 104. CLCR1 field description . . . 303
Table 105. CLCR2 field description . . . 304
Table 106. THCR1 and THCR2 field descriptions . . . 305
Table 107. CLRx (CMS = 0) field descriptions . . . 306
Table 108. CLRx (CMS = 1) field descriptions . . . 306
Table 109. CLRx (ST1 = 0) field descriptions . . . 307
Table 110. CLR*x* addresses . . . 307
Table 111. CTUEFR field description . . . 308
Table 112. CTUIFR field descriptions . . . 309
Table 113. CTUIR field description . . . 310
Table 114. COTR field description . . . 311
Table 115. CTUCR field descriptions . . . 311
Table 116. CTUDF field descriptions . . . 313
Table 117. CTU_EXP_A field descriptions . . . 313
Table 118. CTU_EXP_B field descriptions . . . 313
Table 119. CTU_CNTRNG field descriptions . . . 314
Table 120. FDCR field description . . . 314
Table 121. FCR field descriptions . . . 315
Table 122. FST field descriptions . . . 317
Table 123. FRx filed descriptions . . . 319
Table 124. FLx field description . . . 320
Table 125. Logical master IDs . . . 321
Table 126. XBAR master port allocation . . . 322
Table 127. XBAR slave port allocation . . . 322
Table 128. XBAR register configuration summary . . . 325
Table 129. XBAR register summary . . . 325
Table 130. Register terms . . . 326
Table 131. Master Priority Register descriptions . . . 327
Table 132. Slave General Purpose Control Register Descriptions . . . 330
Table 133. Master General Purpose Control Register Descriptions . . . 332
Table 134. CRC registers . . . 348
Table 135. CRC_CFG field descriptions . . . 350
Table 136. CRC_INP field descriptions . . . 351
Table 137. CRC_CSTAT field descriptions . . . 351
Table 138. CRC_OUTP field descriptions . . . 352
Table 139. Signal properties . . . 360
Table 140. DSPI memory map . . . 362
Table 141. DSPI_MCR field descriptions . . . 364
Table 142. DSPI_HCR field descriptions . . . 366
Table 143. DSPI_TCR Field Descriptions . . . 367
Table 144. DSPI_CTAR*n* field descriptions in master mode . . . 369
Table 145. DSPI SCK Duty Cycle . . . 371
Table 146. Delay Scaler Encoding . . . 372
Table 147. DSPI Baud Rate Scaler . . . 372
Table 148. DSPI_CTAR*0*, DSPI_CTAR*1* Field Descriptions in slave mode . . . 372
Table 149. DSPI_SR field descriptions . . . 374
Table 150. DSPI_RSER field descriptions . . . 376
Table 151. DSPI_PUSHR field descriptions in master mode . . . 378

Table 152. DSPI_PUSHR field descriptions in slave mode . . . 379
Table 153. DSPI_POPR field descriptions . . . 380
Table 154. DSPI_TXFR*n* Field Descriptions . . . 381
Table 155. DSPI_RXFR*n* Field Descriptions . . . 381
Table 156. Baud Rate Computation Example . . . 386
Table 157. PCS to SCK Delay Computation Example . . . 386
Table 158. After SCK Delay Computation Example . . . 387
Table 159. Delay after Transfer Computation Example . . . 387
Table 160. Peripheral Chip Select Strobe Assert Computation Example . . . 388
Table 161. Peripheral Chip Select Strobe Negate Computation Example . . . 388
Table 162. Interrupt and DMA Request Conditions . . . 398
Table 163. Baud Rate Values (bps) . . . 401
Table 164. Delay Values . . . 402
Table 165. PVR field descriptions . . . 412
Table 166. PIR field descriptions . . . 413
Table 167. SVR field descriptions . . . 413
Table 168. Interrupt Registers . . . 414
Table 169. Exceptions and conditions . . . 415
Table 170. DMA_MUX memory map . . . 419
Table 171. CHCONFIGxx Field Descriptions . . . 420
Table 172. Channel and Trigger Enabling . . . 420
Table 173. DMA_MUX source slot mapping . . . 421
Table 174. DMA_MUX trigger sources . . . 422
Table 175. eDMA 32-bit memory map . . . 432
Table 176. eDMA Control Register (DMACR) field descriptions . . . 435
Table 177. eDMA Error Status (DMAES) field descriptions . . . 437
Table 178. DMAERQL field descriptions . . . 439
Table 179. DMAEEIL field descriptions . . . 440
Table 180. DMASERQ field descriptions . . . 440
Table 181. DMASERQ[SERQ] field structure . . . 441
Table 182. DMACERQ field descriptions . . . 441
Table 183. DMACERQ[CERQ] field structure . . . 441
Table 184. DMASEEI field descriptions . . . 442
Table 185. DMASEEI[SEEI] field structure . . . 442
Table 186. DMACEEI field descriptions . . . 443
Table 187. DMACEEI[CEEI] field structure . . . 443
Table 188. DMACINT field descriptions . . . 444
Table 189. DMACINT[CINT] field structure . . . 444
Table 190. DMACERR field descriptions . . . 444
Table 191. DMACERR[CERR] field structure . . . 445
Table 192. DMASSRT field descriptions . . . 445
Table 193. DMASSRT[SSRT] field structure . . . 445
Table 194. DMACDNE field descriptions . . . 446
Table 195. DMACDNE[CDNE] field structure . . . 446
Table 196. DMAINTL field descriptions . . . 447
Table 197. DMAERRL field descriptions . . . 448
Table 198. DMAHRSL field descriptions . . . 449
Table 200. TCDn 32-bit memory structure . . . 450
Table 202. TCDn Word 1 (TCDn.{smod,ssize,dmod,dsize,soff}) field descriptions . . . 452
Table 204. TCDn Word 2 (TCDn.nbytes) field descriptions . . . 454
Table 207. TCDn Word 5 (TCDn.{doff,citer}) Field Descriptions . . . 456
Table 209. TCDn Word 7 (TCDn.{biter, control/status}) field descriptions . . . 459

Table 210. eDMA peak transfer rates [MB/s] . . . 467
Table 211. eDMA peak request rate [MReq/sec] . . . 468
Table 212. eTimer module-to-module input signals . . . 480
Table 213. eTimer memory map . . . 481
Table 214. Count Mode Values . . . 484
Table 215. Primary Count Source Values . . . 484
Table 216. Secondary Count Source Values . . . 486
Table 217. Values for Co-channel Initilization . . . 488
Table 218. OUTMODE Values . . . 489
Table 219. Values for Reload on Capture . . . 490
Table 220. Values for DBGEN . . . 490
Table 221. Values for Compare Load Control 2 . . . 494
Table 222. Values for Compare Load Control 1 . . . 495
Table 223. Values for Compare Mode . . . 495
Table 224. Values for Capture 2 Mode Control . . . 495
Table 225. Values for Capture 1 Mode Control . . . 496
Table 226. Values for DREQn . . . 499
Table 227. Interrupt summary . . . 508
Table 228. DMA Summary . . . 509
Table 229. ECSM memory map . . . 510
Table 230. Processor Core Type (PCT) Field Descriptions . . . 512
Table 231. REV field descriptions . . . 512
Table 232. PLAMC field descriptions . . . 513
Table 233. PLASC field descriptions . . . 514
Table 234. IOPMC field descriptions . . . 514
Table 235. MRSR field descriptions . . . 515
Table 236. MUDCR field descriptions . . . 516
Table 237. AHB response and ECC reporting for even and odd ECC . . . 516
Table 238. ECR field descriptions . . . 518
Table 239. ESR field descriptions . . . 520
Table 240. ECC Error Generation (EEGR) Field Descriptions . . . 522
Table 241. PFEAR field descriptions . . . 525
Table 242. PFEMR field descriptions . . . 526
Table 243. PFEAT field descriptions . . . 527
Table 244. PFEDRL and PFEDRH field descriptions . . . 528
Table 245. PREAR field descriptions . . . 529
Table 246. PRESR field descriptions . . . 530
Table 247. Platform RAM syndrome mapping for single-bit correctable errors . . . 530
Table 248. PREMR field descriptions . . . 533
Table 249. PREAT field descriptions . . . 534
Table 250. PREDRL and PREDRH field descriptions . . . 535
Table 251. Acronyms . . . 536
Table 252. FCCU memory map . . . 539
Table 253. FCCU_CTRL field descriptions . . . 542
Table 254. FCCU_CTRLK field descriptions . . . 544
Table 255. FCCU_CFG field descriptions . . . 545
Table 256. FCCU_CF_CFG0..3 field descriptions . . . 547
Table 257. FCCU_NCF_CFG0..3 field descriptions . . . 548
Table 258. FCCU_CFS_CFG0..7 field descriptions . . . 549
Table 259. FCCU_NCFS_CFG0..7 field descriptions . . . 550
Table 260. FCCU_CFS0..3 field descriptions . . . 552
Table 261. FCCU_CFK field descriptions . . . 552

Table 262. FCCU_NCFS0..3 field descriptions . . . 554
Table 263. FCCU_NCFK field descriptions . . . 555
Table 264. FCCU_NCFE0..3 field descriptions . . . 556
Table 265. FCCU_NCF_TOE0..3 field descriptions . . . 556
Table 266. FCCU_NCF_TO field descriptions . . . 557
Table 267. FCCU_CFG_TO field descriptions . . . 558
Table 268. Bi-stable encoding . . . 558
Table 269. FCCU_EINOUT field descriptions . . . 559
Table 270. FCCU_STAT field descriptions . . . 560
Table 271. FCCU_SC field descriptions . . . 561
Table 272. FCCU_CFF field descriptions . . . 562
Table 273. FCCU_NCFF field descriptions . . . 563
Table 274. FCCU_IRQ_STAT field descriptions . . . 564
Table 275. FCCU_IRQ_EN field descriptions . . . 565
Table 276. Timer state/value . . . 565
Table 277. FCCU_XTMR field descriptions . . . 566
Table 278. FCCU_MCS field descriptions . . . 567
Table 279. Reset sources . . . 570
Table 280. NVM configuration . . . 579
Table 281. Dual-rail encoding . . . 581
Table 282. Time switching encoding . . . 582
Table 283. Bi-stable encoding . . . 583
Table 284. FCCU mapping of critical faults . . . 584
Table 285. FCCU mapping of non-critical faults . . . 586
Table 286. C90FL flash memory map . . . 591
Table 287. Register memory map . . . 592
Table 288. MCR Field Descriptions . . . 593
Table 289. MCR bit set/clear priority levels . . . 597
Table 290. LML Field Descriptions . . . 598
Table 291. HBL Field Descriptions . . . 600
Table 292. SLL Field Descriptions . . . 601
Table 293. LMS Field Descriptions . . . 602
Table 294. HBS Field Descriptions . . . 603
Table 295. ADR Field Descriptions . . . 604
Table 296. UT0 Field Descriptions . . . 605
Table 297. UT1 Field Descriptions . . . 607
Table 298. UT2 Field Descriptions . . . 608
Table 299. UM0 Field Descriptions . . . 609
Table 300. UM1 Field Descriptions . . . 609
Table 301. UM2 Field Descriptions . . . 610
Table 302. UM3 Field Descriptions . . . 610
Table 303. UM4 field descriptions . . . 611
Table 304. NVPWD0 field descriptions . . . 611
Table 305. NVPWD1 field descriptions . . . 612
Table 306. NVSCI field descriptions . . . 613
Table 307. User option bits . . . 621
Table 308. SWT default states . . . 621
Table 309. Test flash information . . . 622
Table 310. PFCR0 field descriptions . . . 629
Table 311. PFLASH Configuration Register 0 (PFCR0) settings for different frequencies . . . 632
Table 312. PFAPR field descriptions . . . 633
Table 313. Additional wait-state encoding . . . 636

Table 314. FlexCAN signals . . . 641
Table 315. Module Memory Map . . . 642
Table 316. Message Buffer (MB0) Memory Mapping . . . 643
Table 317. Message Buffer Code for Rx buffers . . . 643
Table 318. Message Buffer Code for Tx buffers . . . 644
Table 319. IDAM Coding . . . 652
Table 320. Fault Confinement State . . . 661
Table 321. Recommended FEN and BCC settings . . . 672
Table 322. Time Segment Syntax . . . 675
Table 323. CAN Standard Compliant Bit Time Segment Settings . . . 676
Table 324. Minimum Ratio Between Peripheral Clock Frequency and CAN Bit Rate . . . 677
Table 325. Modes when PWM operation is restricted . . . 683
Table 326. Fault Mapping . . . 711
Table 327. PWM Registers . . . 716
Table 328. PWM Reload Frequency . . . 721
Table 329. PWM Prescaler . . . 722
Table 330. Interrupt summary . . . 744
Table 331. DMA Summary . . . 745
Table 332. List of Terms . . . 746
Table 333. External Signal Properties . . . 751
Table 334. FlexRay Memory Map . . . 753
Table 335. Register Access Conventions . . . 757
Table 336. Additional Register Reset Conditions . . . 757
Table 337. Register write access restrictions . . . 758
Table 338. FR_MVR Field Descriptions . . . 759
Table 339. FR_MCR Field Descriptions . . . 760
Table 340. FlexRay Channel Selection . . . 761
Table 341. FR_SYMBADR Field Descriptions . . . 762
Table 342. FR_STBSCR Field Descriptions . . . 763
Table 343. Strobe signal mapping . . . 763
Table 344. FR_MBDSR Field Descriptions . . . 764
Table 345. FR_MBSSUTR Field Descriptions . . . 765
Table 346. FR_PEDRAR Field Descriptions . . . 766
Table 347. FR_POCR Field Descriptions . . . 767
Table 348. FR_GIFER field descriptions . . . 769
Table 349. FR_PIFR0 field descriptions . . . 771
Table 350. FR_PIFR1 field descriptions . . . 774
Table 351. FR_PIER0 field descriptions . . . 775
Table 352. FR_PIER1 Field Descriptions . . . 777
Table 353. FR_CHIERFR field descriptions . . . 778
Table 354. FR_MBIVEC field descriptions . . . 780
Table 355. FR_CASERCR field descriptions . . . 781
Table 356. FR_CBSERCR field descriptions . . . 781
Table 357. FR_PSR0 field descriptions . . . 782
Table 358. FR_PSR1 Field Descriptions . . . 784
Table 359. FR_PSR2 field descriptions . . . 785
Table 360. FR_PSR3 field descriptions . . . 787
Table 361. FR_MTCTR Field Descriptions . . . 789
Table 362. FR_CYCTR Field Descriptions . . . 789
Table 363. FR_SLTCTAR Field Descriptions . . . 790
Table 364. FR_SLTCTBR Field Descriptions . . . 790
Table 365. FR_RTCORVR Field Descriptions . . . 791

Table 366. FR_OFCORVR Field Descriptions . . . 792
Table 367. FR_CIFR Field Descriptions . . . 792
Table 368. FR_SYMATOR Field Descriptions . . . 793
Table 369. FR_SFCNTR Field Descriptions . . . 794
Table 370. FR_SFTOR Field Description . . . 795
Table 371. FR_SFTCCSR Field Descriptions . . . 795
Table 372. FR_SFIDRFR Field Descriptions . . . 797
Table 373. FR_SFIDAFVR Field Descriptions . . . 797
Table 374. FR_SFIDAFMR Field Descriptions . . . 797
Table 375. NMVR[0:5] Field Descriptions . . . 798
Table 376. Mapping of NMVRn to the Received Payload Bytes NMVn . . . 798
Table 377. FR_NMVLR Field Descriptions . . . 799
Table 378. FR_TICCR Field Descriptions . . . 799
Table 379. FR_TI1CYSR Field Descriptions . . . 800
Table 380. FR_TI1MTOR Field Descriptions . . . 801
Table 381. FR_TI2CR0 Field Descriptions . . . 801
Table 382. FR_TI2CR1 Field Descriptions . . . 802
Table 383. FR_SSSR Field Descriptions . . . 803
Table 384. Mapping Between FR_SSSRn and FR_SSRn . . . 803
Table 385. FR_SSCCR Field Descriptions . . . 804
Table 386. Mapping between internal FR_SSCCRn and FR_SSCRn . . . 805
Table 387. FR_SSR0–FR_SSR7 Field Descriptions . . . 806
Table 388. FR_SSCR0–FR_SSCR3 Field Descriptions . . . 808
Table 389. FR_MTSACFR Field Descriptions . . . 808
Table 390. MTSBCFR Field Descriptions . . . 809
Table 391. FR_RSBIR Field Descriptions . . . 809
Table 392. FR_RFSYMBADR Field Descriptions . . . 810
Table 393. FR_RFPTR Field Descriptions . . . 811
Table 394. SEL Controlled Receiver FIFO Registers . . . 811
Table 395. FR_RFWMSR Field Descriptions . . . 811
Table 396. FR_RFSIR Field Descriptions . . . 812
Table 397. RFDSR Field Descriptions . . . 812
Table 398. FR_RFARIR Field Descriptions . . . 813
Table 399. FR_RFBRIR Field Descriptions . . . 813
Table 400. FR_RFFLPCR Field Descriptions . . . 814
Table 401. FR_RFMIDAFVR Field Descriptions . . . 814
Table 402. FR_RFMIDAFMR Field Descriptions . . . 815
Table 403. FR_RFFIDRFVR Field Descriptions . . . 815
Table 404. FR_RFFIDRFMR Field Descriptions . . . 816
Table 405. FR_RFRFCFR Field Descriptions . . . 816
Table 406. FR_RFRFCTR Field Descriptions . . . 817
Table 407. FR_LDTXSLAR Field Descriptions . . . 818
Table 408. FR_LDTXSLBR Field Descriptions . . . 819
Table 409. Protocol Configuration Register Fields . . . 819
Table 410. Wakeup Channel Selection . . . 821
Table 411. FR_EEIFER Field Descriptions . . . 830
Table 412. FR_EERICR Field Descriptions . . . 832
Table 413. FR_EERAR Field Descriptions . . . 833
Table 414. FR_EERDR Field Descriptions . . . 834
Table 415. Valid Bits in FR_EERDR[DATA] / FR_EEIDR[DATA] field . . . 834
Table 416. FR_EERSR Field Descriptions . . . 834
Table 417. FR_EEIAR Field Descriptions . . . 835

Table 418. FR_EEIDR Field Descriptions . . . 836
Table 419. FR_EEICR Field Descriptions . . . 836
Table 420. FR_MBCCSRn Field Descriptions . . . 837
Table 421. FR_MBCCFRn Field Descriptions . . . 839
Table 422. Channel Assignment Description . . . 839
Table 423. FR_MBFIDRn Field Descriptions . . . 840
Table 424. FR_MBIDXRn Field Descriptions . . . 840
Table 425. Frame Header Write Access Constraints (Transmit Message Buffer) . . . 852
Table 426. Frame Header Field Descriptions (Receive Message Buffer and Receive FFO) . . . 853
Table 427. Frame Header Field Descriptions (Transmit Message Buffer) . . . 853
Table 428. Receive Message Buffer Slot Status Content . . . 854
Table 429. Receive Message Buffer Slot Status Field Descriptions (Continued) . . . 855
Table 430. Transmit Message Buffer Slot Status Content . . . 856
Table 431. Transmit Message Buffer Slot Status Structure Field Descriptions . . . 857
Table 432. Message Buffer Data Field Minimum Length . . . 858
Table 433. Frame Data Write Access Constraints . . . 859
Table 434. Frame Data Field Descriptions . . . 859
Table 435. Individual Message Buffer Types . . . 861
Table 436. Single Transmit Message Buffer Access Regions Description . . . 862
Table 437. Single Transmit Message Buffer State Description . . . 863
Table 438. Single Transmit Message Buffer Application Transitions . . . 865
Table 439. Single Transmit Message Buffer Module Transitions . . . 865
Table 440. Single Transmit Message Buffer Transition Priorities . . . 866
Table 441. Receive Message Buffer Access Region Description . . . 871
Table 442. Receive Message Buffer States and Access . . . 872
Table 443. Receive Message Buffer Application Transitions . . . 873
Table 444. Receive Message Buffer Module Transitions . . . 873
Table 445. Receive Message Buffer Transition Priorities . . . 874
Table 446. Receive Message Buffer Update (Continued) . . . 875
Table 447. Double Transmit Message Buffer Access Regions Description . . . 878
Table 448. Double Transmit Message Buffer State Description (Commit Side) . . . 879
Table 449. Double Transmit Message Buffer State Description (Transmit Side) . . . 880
Table 450. Double Transmit Message Buffer Host Transitions . . . 882
Table 451. Double Transmit Message Buffer Module Transitions . . . 882
Table 452. Double Transmit Message Buffer Transition Priorities . . . 883
Table 453. Message Buffer Search Priority (static segment) . . . 886
Table 454. Message Buffer Search Priority (dynamic segment) . . . 886
Table 455. Sync Frame Table Generation Modes . . . 900
Table 456. Key Slot Frame Type . . . 903
Table 457. Slot Status Content . . . 907
Table 458. FlexRay Channel Bit Rate Control . . . 915
Table 459. PE DRAM Layout . . . 915
Table 460. CHI LRAM Layout . . . 916
Table 461. Detected Memory Error Types . . . 917
Table 462. PE DRAM checkbits coding . . . 918
Table 463. FR_EERCR[CODE] PE DRAM Syndrome Coding . . . 919
Table 464. CHI LRAM checkbits coding . . . 919
Table 465. FR_EERCR[CODE] CHI LRAM Syndrome Coding . . . 920
Table 466. Minimum f_{chi} [MHz] examples (64 message buffers) . . . 925
Table 467. Protocol Control Command Priorities . . . 926
Table 468. Transmit Buffer Configuration . . . 926
Table 469. Receive Buffer Configuration . . . 927

Table 470. FMPLL Memory Map . . . 930
Table 471. CR Field Descriptions . . . 931
Table 472. Input divide ratios . . . 932
Table 473. Output Divide Ratios . . . 932
Table 474. Loop Divide Ratios . . . 933
Table 475. MR Field Descriptions . . . 934
Table 476. Progressive clock switching for 120 MHz FMPLL operation . . . 935
Table 477. INTC memory map . . . 942
Table 478. PRI Values . . . 945
Table 479. Order of ISR execution example . . . 957
Table 480. Interrupt sources for INTC_0 and INTC_1 . . . 961
Table 481. JTAG Signal Properties . . . 972
Table 482. TEST_CTRL field descriptions . . . 975
Table 483. JTAG Instructions . . . 979
Table 484. Errors in Master mode . . . 990
Table 485. Errors in Slave mode . . . 992
Table 486. Filter submodes . . . 994
Table 487. Filter to interrupt vector correlation . . . 995
Table 488. UART buffer structure . . . 1000
Table 489. BDRL access in UART mode . . . 1000
Table 490. BDRM access in UART mode . . . 1001
Table 491. UART receiver scenarios . . . 1002
Table 492. LINFlexD memory map . . . 1003
Table 493. LINCR1 field descriptions . . . 1004
Table 494. Checksum bits configuration . . . 1006
Table 495. LIN master break length selection . . . 1006
Table 496. Operating mode selection . . . 1007
Table 497. LINIER field descriptions . . . 1007
Table 498. LINSR field descriptions . . . 1010
Table 499. LINESR field descriptions . . . 1012
Table 500. UARTCR field descriptions . . . 1014
Table 501. UARTSR field descriptions . . . 1016
Table 502. LINTCSR field descriptions . . . 1018
Table 503. LINOCR field descriptions . . . 1019
Table 504. LINTOCR field descriptions . . . 1020
Table 505. LINFBRR field descriptions . . . 1021
Table 506. LINIBRR field descriptions . . . 1022
Table 507. Integer baud rate selection . . . 1022
Table 508. LINCFR field descriptions . . . 1022
Table 509. LINCR2 field descriptions . . . 1023
Table 510. BIDR field descriptions . . . 1025
Table 511. BDRL field descriptions . . . 1025
Table 512. BDRM field descriptions . . . 1026
Table 513. IFER field descriptions . . . 1027
Table 514. IFMI field descriptions . . . 1028
Table 515. IFMR field descriptions . . . 1028
Table 516. IFMR[IFM] configuration . . . 1028
Table 517. IFCR functionality based on mode . . . 1029
Table 518. IFCR field descriptions . . . 1030
Table 519. GCR field descriptions . . . 1031
Table 520. UARTPTO field descriptions . . . 1032
Table 521. UARTCTO field descriptions . . . 1033

Table 522. DMATXE field descriptions . . . 1034
Table 523. DMARXE field descriptions . . . 1035
Table 524. Register settings (master node, TX mode) . . . 1036
Table 525. TCD settings (master node, TX mode) . . . 1038
Table 526. TCD settings (master node, RX mode) . . . 1040
Table 527. Register settings (slave node, TX mode) . . . 1041
Table 528. TCD settings (slave node, TX mode) . . . 1043
Table 529. Register settings (slave node, RX mode) . . . 1044
Table 530. TCD settings (slave node, RX mode) . . . 1045
Table 531. TCD settings (UART node, TX mode) . . . 1048
Table 532. TCD settings (UART node, RX mode) . . . 1051
Table 533. LINFlexD interrupt control . . . 1053
Table 534. Error calculation for programmed baud rates . . . 1055
Table 535. MPU port allocation . . . 1064
Table 536. MPU memory map . . . 1065
Table 537. MPU_CESR field descriptions . . . 1067
Table 538. MPU_EARn field descriptions . . . 1068
Table 539. MPU_EDRn field descriptions . . . 1069
Table 540. MPU_RGDn.Word0 field descriptions . . . 1070
Table 541. MPU_RGDn.Word1 field descriptions . . . 1070
Table 542. MPU_RGDn.Word2 field descriptions . . . 1071
Table 543. MPU_RGDn.Word3 field descriptions . . . 1073
Table 544. MPU_RGDAACn field descriptions . . . 1074
Table 545. Protection violation definition . . . 1077
Table 546. Overlapping region descriptor example . . . 1081
Table 547. MC_ME mode descriptions . . . 1084
Table 548. MC_ME register description . . . 1085
Table 549. MC_ME memory map . . . 1088
Table 550. Global Status Register (ME_GS) field descriptions . . . 1093
Table 551. Mode Control Register (ME_MCTL) Field Descriptions . . . 1095
Table 552. Mode Enable Register (ME_ME) Field Descriptions . . . 1097
Table 553. Interrupt Status Register (ME_IS) field descriptions . . . 1098
Table 554. Interrupt Mask Register (ME_IM) field descriptions . . . 1100
Table 555. Invalid Mode Transition Status Register (ME_IMTS) Field Descriptions . . . 1101
Table 556. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions . . . 1102
Table 557. Mode Configuration Registers (ME_<mode>_MC) Field Descriptions . . . 1108
Table 558. Peripheral Status Registers 0...4 (ME_PS0...4) Field Descriptions . . . 1110
Table 559. Run Peripheral Configuration Registers (ME_RUN_PC0...7) Field Descriptions . . . 1111
Table 560. Low-Power Peripheral Configuration Registers (ME_LP_PC0...7) Field Descriptions . . . 1112
Table 561. Peripheral Control Registers (ME_PCTL0...143) Field Descriptions . . . 1112
Table 562. MC_ME Resource Control Overview . . . 1118
Table 563. MC_ME system clock selection overview . . . 1122
Table 564. Registers available in the NXSS programmer model . . . 1130
Table 565. DC1 field descriptions . . . 1131
Table 566. DC2 field description . . . 1132
Table 567. WT field descriptions . . . 1133
Table 568. DTC field descriptions . . . 1133
Table 569. Data trace address range options . . . 1135
Table 570. BWC1 field descriptions . . . 1136
Table 571. BWC2 field descriptions . . . 1136
Table 572. Public TCODEs supported . . . 1138
Table 573. Error code (ECODE) encoding (TCODE = 8) . . . 1138

Table 574. Data Trace Size (DSZ) encodings (TCODE = 5, 6, 13, 14) . . . 1139
Table 575. Data trace exception summary . . . 1141
Table 576. Watchpoint source description (WPHIT field descriptor) . . . 1143
Table 577. Device parameter values . . . 1145
Table 578. NPC Signal Properties . . . 1148
Table 579. NPC Registers . . . 1151
Table 580. DID Field Descriptions . . . 1152
Table 581. PCR field descriptions . . . 1153
Table 582. MCKO_DIV Values . . . 1154
Table 583. NPC Reset Configuration Options . . . 1155
Table 584. NPC Output Messages . . . 1157
Table 585. Implemented Instructions . . . 1158
Table 586. Loading NEXUS-ENABLE instruction . . . 1161
Table 587. Write to a 32-Bit Nexus Client Register . . . 1162
Table 588. RC Digital Interface Registers - base address 0xC3FE_0060 . . . 1165
Table 589. RC_CTL field descriptions . . . 1166
Table 590. Truth table of the external crystal oscillator . . . 1166
Table 591. OSC_CTL field descriptions . . . 1168
Table 592. PIT memory map . . . 1170
Table 593. Timer Channel n . . . 1170
Table 594. PITMCR Field Descriptions . . . 1171
Table 595. LDVAL Field Descriptions . . . 1172
Table 596. CVAL Field Descriptions . . . 1173
Table 597. TCTRL Field Descriptions . . . 1174
Table 598. TFLG Field Descriptions . . . 1175
Table 599. PBRIDGE registers . . . 1179
Table 600. PBRIDGE memory map . . . 1180
Table 601. MPROT*n* field reset values . . . 1181
Table 602. MPROT*n* field structure descriptions . . . 1181
Table 603. PACR*n* field structure descriptions . . . 1182
Table 604. On-platform peripherals and PACR numbers . . . 1182
Table 605. Off-platform peripherals and OPACR numbers . . . 1182
Table 606. MC_PCU Register Description . . . 1186
Table 607. MC_PCU Memory Map . . . 1186
Table 608. Power Domain Status Register (PCU_PSTAT) Field Descriptions . . . 1187
Table 609. PMU blocks . . . 1189
Table 610. Fault assertion conditions . . . 1191
Table 611. PMU memory map . . . 1192
Table 612. PMUCTRL_STATUS field descriptions . . . 1193
Table 613. PMUCTRL_CTRL field descriptions . . . 1194
Table 614. PMUCTRL_MASKF field descriptions . . . 1195
Table 615. PMUCTRL_FAULT field descriptions . . . 1196
Table 616. PMUCTRL_IRQS field descriptions . . . 1197
Table 617. PMUCTRL_IRQE field descriptions . . . 1198
Table 618. REG_PROT memory map . . . 1202
Table 619. SLBR*n* field descriptions . . . 1204
Table 620. Soft lock bits vs. protected address . . . 1204
Table 621. GCR field descriptions . . . 1205
Table 622. SPC56ELxx register protection . . . 1210
Table 687. Acronyms and abbreviated terms . . . 1298
Table 688. STCU register map . . . 1303
Table 689. STCU_SKC field descriptions . . . 1305

Table 690. STCU_CFG field descriptions . . . 1305
Table 691. STCU_ERR field descriptions . . . 1307
Table 692. STCU_ERRK field descriptions . . . 1308
Table 693. STCU_LBS field descriptions . . . 1308
Table 694. STCU_LBE field descriptions . . . 1309
Table 695. STCU_MBSL field descriptions . . . 1310
Table 696. STCU_MBSH field descriptions . . . 1311
Table 697. STCU_MBEL field descriptions . . . 1311
Table 698. STCU_MBEH field descriptions . . . 1312
Table 699. STCU_LB_MISREL field descriptions . . . 1313
Table 700. STCU_LB_MISREH field descriptions . . . 1313
Table 701. STCU_LB_MISRRL field descriptions . . . 1314
Table 702. STCU_LB_MISRRH field descriptions . . . 1314
Table 703. LBIST partitioning for cut1 . . . 1315
Table 704. LBIST partitioning for cut2/3 . . . 1317
Table 705. MBIST partitioning . . . 1319
Table 706. SEMA4 memory map . . . 1324
Table 707. SEMA4_GATE*n* field descriptions . . . 1326
Table 708. SEMA4_CP{0,1}NTF field descriptions . . . 1327
Table 709. SEMA4_CP{0,1}NTF field descriptions . . . 1327
Table 710. SEMA4_RSTGT field descriptions . . . 1329
Table 711. SEMA4_RSTNTF field descriptions . . . 1331
Table 712. SWG memory map . . . 1336
Table 713. SWG_CTRL field descriptions . . . 1337
Table 714. Sine wave amplitude (approximate) as a function of SWG_CTRL[IOAMPL] . . . 1337
Table 715. SWG_STAT field descriptions . . . 1339
Table 716. SWT Memory Map . . . 1342
Table 717. SWT_CR Field Descriptions . . . 1343
Table 718. SWT_IR Field Descriptions . . . 1344
Table 719. SWT_TO Register Field Descriptions . . . 1345
Table 720. SWT_WN Register Field Descriptions . . . 1345
Table 721. SWT_SR Field Descriptions . . . 1346
Table 722. SWT_CO Register Field Descriptions . . . 1346
Table 723. SWT_SK field descriptions . . . 1347
Table 724. SRAM operating modes . . . 1349
Table 725. Number of Wait States Required for SRAM Operations . . . 1350
Table 726. SIUL signal properties . . . 1354
Table 727. SIUL memory map . . . 1355
Table 728. MIDR1 Field Descriptions . . . 1357
Table 729. MIDR2 field descriptions . . . 1358
Table 730. ISR Field Descriptions . . . 1359
Table 731. IRER Field Descriptions . . . 1359
Table 732. IREER Field Descriptions . . . 1360
Table 733. IFEER Field Descriptions . . . 1360
Table 734. IFER Field Descriptions . . . 1361
Table 735. PCR0 Field Descriptions . . . 1362
Table 736. PSMI0_3 field descriptions . . . 1364
Table 737. GPDO0_3 field descriptions . . . 1365
Table 738. GPDO0_3–GPDI104_107 Field Descriptions . . . 1366
Table 739. PGPDO0–PGPDO3 Field Descriptions . . . 1367
Table 740. PGPDI0_3 Field Descriptions . . . 1368
Table 741. MPGPDO0_6 Field Descriptions . . . 1369

Table 742. IFMC0_31 field descriptions . . . 1370
Table 743. IFCPR Field Descriptions . . . 1371
Table 744. Module memory map . . . 1376
Table 745. STATUS Allowed Register Accesses . . . 1377
Table 746. STATUS field descriptions . . . 1377
Table 747. MEMCONFIG Field Descriptions . . . 1379
Table 748. MEMCONFIG Allowed Register Accesses . . . 1379
Table 749. ERROR Field Descriptions . . . 1380
Table 750. ERROR Allowed Register Accesses . . . 1380
Table 751. DEBUGPORT Field Descriptions . . . 1381
Table 752. Debug Status Port Modes . . . 1381
Table 753. DEBUGPORT Allowed Register Accesses . . . 1381
Table 754. DPMBOOT Field Descriptions . . . 1382
Table 755. DPMKEY Field Descriptions . . . 1383
Table 756. UOPS field descriptions . . . 1384
Table 757. SCTR Field Descriptions . . . 1385
Table 758. STM memory map . . . 1386
Table 759. STM_CR Field Descriptions . . . 1388
Table 760. STM_CNT Field Descriptions . . . 1389
Table 761. STM_CCRn Field Descriptions . . . 1389
Table 762. STM_CIRn Field Descriptions . . . 1390
Table 763. STM_CMPn Register Field Descriptions . . . 1390
Table 764. TSENS calibration constants . . . 1393
Table 765. WKPU memory map . . . 1396
Table 766. NSR Field Descriptions . . . 1397
Table 767. NCR Field Descriptions . . . 1398
Table 768. Document revision history . . . 1401

List of figures

Figure 1. SPC56ELxx block diagram . . . 68
Figure 2. LQFP100 pinout . . . 97
Figure 3. LQFP144 pinout . . . 98
Figure 4. LFBGA257 ballmap . . . 99
Figure 5. Simplified boot process in DPM . . . 166
Figure 6. Flash partitioning and RCHW search . . . 170
Figure 7. Reset Configuration Half Word (RCHW) . . . 171
Figure 8. Nonvolatile System Censorship Information Register (NVSCI) for cut1 . . . 173
Figure 9. Nonvolatile System Censorship Information Register (NVSCI) for cut2/3 . . . 174
Figure 10. Main Configuration Register (MCR) . . . 183
Figure 11. Main Status Register (MSR) . . . 185
Figure 12. Interrupt Status Register (ISR) . . . 186
Figure 13. Channel Pending Register 0 (CEOCFR0) . . . 187
Figure 14. Interrupt Mask Register (IMR) . . . 188
Figure 15. Channel Interrupt Mask Register 0 (CIMR0) . . . 189
Figure 16. Watchdog Threshold Interrupt Status Register (WTISR) . . . 189
Figure 17. Watchdog Threshold Interrupt Mask Register (WTIMR) . . . 190
Figure 18. DMA Enable Register (DMAE) . . . 191
Figure 19. DMA Channel Select Register 0 (DMAR0) . . . 191
Figure 20. Threshold Registers (THRHLR*n*) . . . 192
Figure 21. Presampling Control Register (PSCR) . . . 192
Figure 22. Presampling Register 0 (PSR0) . . . 193
Figure 23. Conversion Timing Register 0 (CTR0) . . . 194
Figure 24. Conversion timing . . . 194
Figure 25. Conversion Timing Register 1 (CTR1) . . . 195
Figure 26. Normal Conversion Mask Register 0 (NCMR0) . . . 196
Figure 27. Injected Conversion Mask Register 0 (JCMR0) . . . 196
Figure 28. Power-down Exit Delay Register (PDEDR) . . . 197
Figure 29. Channel Data Register *n* (CDR*n*) . . . 197
Figure 30. Channel Watchdog Selection Register 0 (CWSEL0) . . . 198
Figure 31. Channel Watchdog Selection Register 1 (CWSEL1) . . . 198
Figure 32. Channel Watchdog Enable Register 0 (CWENR0) . . . 199
Figure 33. Analog Watchdog Out of Range Register 0 (AWORR0) . . . 200
Figure 34. Self Test Configuration Register 1 (STCR1) . . . 200
Figure 35. Self Test Configuration Register 2 (STCR2) . . . 201
Figure 36. Self Test Configuration Register 3 (STCR3) . . . 203
Figure 37. Self Test Baud Rate Register (STBRR) . . . 204
Figure 38. Self Test Status Register 1 (STSR1) . . . 205
Figure 39. Self Test Status Register 2 (STSR2) . . . 207
Figure 40. Self Test Status Register 3 (STSR3) . . . 208
Figure 41. Self Test Status Register 4 (STSR4) . . . 208
Figure 42. Self Test Data Register 1 (STDR1) . . . 209
Figure 43. Self Test Data Register 2 (STDR2) . . . 210
Figure 44. Self Test Analog Watchdog Register 0 (STAW0R) . . . 210
Figure 45. Self Test Analog Watchdog Register 1A (STAW1AR) . . . 211
Figure 46. Self Test Analog Watchdog Register 1B (STAW1BR) . . . 212
Figure 47. Self Test Analog Watchdog Register 2 (STAW2R) . . . 212
Figure 48. Self Test Analog Watchdog Register 3 (STAW3R) . . . 213

Figure 49. Self Test Analog Watchdog Register 4 (STAW4R). . . . 213
Figure 50. Self Test Analog Watchdog Register 5 (STAW5R). . . . 214
Figure 51. ADC interaction with other modules . . . 215
Figure 52. CTU/ADC interface . . . 216
Figure 53. Normal conversion flow . . . 217
Figure 54. Injected sample/conversion sequence . . . 218
Figure 55. Presampling sequence . . . 220
Figure 56. Presampling sequence with PRECONV = 1 . . . 220
Figure 57. Guarded area . . . 221
Figure 58. Test channel conversion example . . . 225
Figure 59. Interface between ADC and CTU to manage self test . . . 226
Figure 60. Boot mode selection . . . 234
Figure 61. BAM logic flow . . . 235
Figure 62. Password check flow . . . 238
Figure 63. Start address, VLE bit, and download size in bytes . . . 239
Figure 64. FlexCAN bit timing . . . 241
Figure 65. Autobaud configuration and detection/measurement flow . . . 243
Figure 66. Autobaud measurement / UART protocol . . . 244
Figure 67. Baud rate Deviation Between Host and SPC56ELxx . . . 244
Figure 68. Autobaud measurement / CAN protocol . . . 245
Figure 69. System clock generation . . . 249
Figure 70. MC_CGM block diagram . . . 254
Figure 71. Output Clock Enable Register (CGM_OC_EN) . . . 260
Figure 72. Output Clock Division Select Register (CGM_OCDS_SC) . . . 260
Figure 73. System Clock Select Status Register (CGM_SC_SS) . . . 261
Figure 74. System Clock Divider Configuration Registers (CGM_SC_DC0) . . . 262
Figure 75. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC) . . . 263
Figure 76. Auxiliary Clock 0 Divider Configuration Register 0 (CGM_AC0_DC0) . . . 263
Figure 77. Auxiliary Clock 0 Divider Configuration Register 1 (CGM_AC0_DC1) . . . 264
Figure 78. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC) . . . 264
Figure 79. Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0) . . . 265
Figure 80. Auxiliary Clock 2 Select Control Register (CGM_AC2_SC) . . . 266
Figure 81. Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0) . . . 266
Figure 82. Auxiliary Clock 3 Select Control Register (CGM_AC3_SC) . . . 267
Figure 83. Auxiliary Clock 4 Select Control Register (CGM_AC4_SC) . . . 268
Figure 84. MC_CGM system clock generation overview . . . 269
Figure 85. MC_CGM auxiliary clock 0 generation overview . . . 270
Figure 86. MC_CGM auxiliary clock 1 generation overview . . . 270
Figure 87. MC_CGM auxiliary clock 2 generation overview . . . 271
Figure 88. MC_CGM auxiliary clock 3 generation overview . . . 271
Figure 89. MC_CGM auxiliary clock 4 generation overview . . . 272
Figure 90. MC_CGM Output Clock Multiplexer and Port Pin B[6] Generation . . . 273
Figure 91. Control status register (CMU_CSR) . . . 275
Figure 92. Frequency display Register (CMU_FDR) . . . 276
Figure 93. High-frequency reference register A (CMU_HFREFR_A) . . . 277
Figure 94. Low-frequency reference register A (CMU_LFREFR_A) . . . 277
Figure 95. Interrupt status register (CMU_ISR) . . . 278
Figure 96. Measurement duration register (CMU_MDR) . . . 279
Figure 97. Cross triggering unit block diagram . . . 283
Figure 98. CTU interaction with other peripherals . . . 285
Figure 99. TGS in triggered mode . . . 286
Figure 100. Example timing for TGS in triggered mode . . . 287

Figure 101. TGS in sequential mode . . . 288
Figure 102. Example timing for TGS in sequential mode . . . 288
Figure 103. TGS counter cases . . . 289
Figure 104. Scheduler subunit . . . 290
Figure 105. Reload error scenario . . . 293
Figure 106. Trigger generator subunit input sel. register(TGSISR) . . . 299
Figure 107. Trigger generator subunit control register (TGSCR) . . . 301
Figure 108. TxCR - Trigger x compare register (x = 0,...,7) . . . 302
Figure 109. TGS counter compare register (TGSCCR) . . . 302
Figure 110. TGS counter reload register (TGSCRR) . . . 302
Figure 111. Commands list control register 1 (CLCR1) . . . 303
Figure 112. Commands list control register 2 (CLCR2) . . . 304
Figure 113. Trigger handler control register 1 (THCR1) . . . 305
Figure 114. Trigger handler control register 2 (THCR2) . . . 305
Figure 115. Commands list register x (x = 1,...,24) (CMS=0) . . . 306
Figure 116. Commands list register x (x = 1,...,24) (CMS=1) . . . 306
Figure 117. Commands list register x (x = 1,...,24) (ST1 = 0) . . . 307
Figure 118. Cross triggering unit error flag register (CTUEFR) . . . 308
Figure 119. Cross triggering unit interrupt flag register (CTUIFR) . . . 309
Figure 120. Cross triggering unit interrupt/DMA register (CTUIR) . . . 310
Figure 121. Control On time register (COTR) . . . 310
Figure 122. Cross triggering unit control Register (CTUCR) . . . 311
Figure 123. Cross triggering unit digital filter (CTUDF) . . . 312
Figure 124. Cross triggering unit expected value A (CTU_EXPECTED_A) . . . 313
Figure 125. Cross triggering unit expected value B (CTU_EXPECTED_B) . . . 313
Figure 126. Cross triggering unit count range (CTU_CNTRNG) . . . 314
Figure 127. FIFO DMA control register (FDCR) . . . 314
Figure 128. FIFO control register (FCR) . . . 315
Figure 129. FIFO threshold (FTH) . . . 316
Figure 130. FIFO status (FST) . . . 317
Figure 131. FIFO Right aligned data x . . . 319
Figure 132. FIFO signed left aligned data x . . . 319
Figure 133. Key to Register Fields . . . 325
Figure 134. Master Priority Register n . . . 327
Figure 135. Slave General Purpose Control Register n . . . 330
Figure 136. Master General Purpose Control Register n . . . 332
Figure 137. Low to high priority mastership change . . . 340
Figure 138. High to low priority mastership change . . . 341
Figure 139. Round-robin mastership change . . . 341
Figure 140. Parking on a specific master . . . 343
Figure 141. Parking on last master . . . 343
Figure 142. CRC top level diagram . . . 348
Figure 143. CRC Configuration Register (CRC_CFG) for cut1 . . . 349
Figure 144. CRC Configuration Register (CRC_CFG) for cut2/3 . . . 349
Figure 145. CRC Input Register (CRC_INP) . . . 350
Figure 146. CRC Current Status Register (CRC_CSTAT) . . . 351
Figure 147. CRC Output Register (CRC_OUTP) . . . 352
Figure 148. CRC-CCITT engine concept scheme . . . 353
Figure 149. CRC computation flow . . . 354
Figure 150. DMA-CRC (Tx flow, Rx flow) . . . 356
Figure 151. DSPI block diagram . . . 357
Figure 152. DSPI with queues and DMA . . . 359

Figure 153. DSPI Module Configuration Register (DSPI_MCR) for cut1 . . . 363
Figure 154. DSPI Module Configuration Register (DSPI_MCR) for cut2/3 . . . 363
Figure 155. DSPI Hardware Configuration Register (DSPI_HCR) [cut2/3 only] . . . 366
Figure 156. DSPI Transfer Count Register (DSPI_TCR) . . . 367
Figure 157. DSPI Clock and Transfer Attributes Register 0–3 (DSPI_CTAR0–DSPI_CTAR3) in master mode . . . 368
Figure 158. DSPI Clock and Transfer Attributes Register 0, 1 (DSPI_CTAR0 for cut1, DSPI_CTAR1) in slave mode . . . 368
Figure 159. DSPI Clock and Transfer Attributes Register 0 (DSPI_CTAR0) in slave mode for cut2/3 369
Figure 160. DSPI Status Register (DSPI_SR) for cut1 . . . 373
Figure 161. DSPI Status Register (DSPI_SR) for cut2/3 . . . 374
Figure 162. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER) for cut1 . . . 376
Figure 163. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER) for cut2/3 . . . 376
Figure 164. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode for cut1 . . . 378
Figure 165. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode for cut2/3 . . . 378
Figure 166. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in slave mode . . . 379
Figure 167. DSPI POP RX FIFO Register (DSPI_POPR) . . . 380
Figure 168. DSPI Transmit FIFO Register 0–4 (DSPI_TXFR0–DSPI_TXFR4) . . . 380
Figure 169. DSPI Receive FIFO Registers 0–4 (DSPI_RXFR0–DSPI_RXFR4) . . . 381
Figure 170. SPI Serial Protocol Overview . . . 382
Figure 171. Communications clock prescalers and scalers . . . 386
Figure 172. Peripheral Chip Select Strobe Timing . . . 387
Figure 173. DSPI Transfer Timing Diagram (MTFE=0, CPHA=0, FMSZ=8) . . . 389
Figure 174. DSPI Transfer Timing Diagram (MTFE=0, CPHA=1, FMSZ=8) . . . 390
Figure 175. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/4$) . . . 392
Figure 176. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/2$) . . . 392
Figure 177. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/3$) . . . 393
Figure 178. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/2$) . . . 394
Figure 179. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/3$) . . . 394
Figure 180. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/4$) . . . 395
Figure 181. Example of Non-Continuous Format (CPHA=1, CONT=0) . . . 395
Figure 182. Example of Continuous Transfer (CPHA=1, CONT=1) . . . 396
Figure 183. Continuous SCK Timing Diagram (CONT=0) . . . 397
Figure 184. Continuous SCK Timing Diagram (CONT=1) . . . 398
Figure 185. TX FIFO Pointers and Counter . . . 403
Figure 186. e200z4d block diagram . . . 406
Figure 187. e200z4d Supervisor Mode Programmer's Model . . . 411
Figure 188. e200z4d User Mode Programmer's Model SPRs . . . 412
Figure 189. Processor Version Register (PVR) . . . 412
Figure 190. Processor ID Register (PIR) . . . 413
Figure 191. System Version Register (SVR) . . . 413
Figure 192. DMA_MUX block diagram . . . 418
Figure 193. Channel Configuration Registers (CHCONFIG#n) . . . 420
Figure 194. DMA_MUX triggered channels . . . 423
Figure 195. DMA_MUX Channel Triggering: Normal Operation . . . 423
Figure 196. DMA_MUX Channel Triggering: Ignored Trigger . . . 424
Figure 197. eDMA block diagram . . . 430
Figure 198. eDMA Control Register (DMACR) . . . 434
Figure 199. eDMA Error Status (DMAES) Register . . . 437
Figure 200. eDMA Enable Request Low (DMAERQL) Register . . . 439
Figure 201. eDMA Enable Error Interrupt Low (DMAEEIL) Register . . . 440
Figure 202. eDMA Set Enable Request (DMASERQ) Register . . . 440

Figure 203. eDMA Clear Enable Request (DMACERQ) Register 441
Figure 204. eDMA Set Enable Error Interrupt (DMASEEI) Register 442
Figure 205. eDMA Clear Enable Error Interrupt (DMACEEI) Register. 443
Figure 206. eDMA Clear Interrupt Request (DMACINT) Register 443
Figure 207. eDMA Clear Error (DMACERR) Register 444
Figure 208. eDMA Set START Bit (DMASSRT) Register 445
Figure 209. eDMA Clear DONE Status (DMACDNE) Register 446
Figure 210. eDMA Interrupt Request Low (DMAINTL) Register 447
Figure 211. eDMA Error Low (DMAERRL) Register 448
Figure 212. eDMA Hardware Request Status Low (DMAHRSL) Register. 448
Figure 213. eDMA Channel n Priority (DCHPRIn) Register. 449
Figure 214. TCDn Word 0 (TCDn.saddr) Field 451
Figure 215. TCDn Word 1 (TCDn.{soff,smod,ssize,dmod,dsize}) Fields. 451
Figure 216. TCDn Word 2 (TCDn.nbytes) Field (DMACR[EMLM] = 0) 453
Figure 217. TCDn Word 2 (TCDn.nbytes) Field (DMACR[EMLM] = 1) 454
Figure 218. TCDn Word 3 (TCDn.slast) Field 455
Figure 219. TCDn Word 4 (TCDn.daddr) Field 455
Figure 220. TCDn Word 5 (TCDn.{citer,doff}) Fields 456
Figure 221. TCDn Word 6 (TCDn.dlast_sga) Field 457
Figure 222. TCDn Word 7 (TCDn.{biter,control/status}) Fields 458
Figure 223. eDMA operation, part 1. 464
Figure 224. eDMA operation, part 2. 465
Figure 225. eDMA operation, part 3. 466
Figure 226. eTimer block diagram 478
Figure 227. eTimer channel block diagram 479
Figure 228. Compare Register 1 (COMP1) 482
Figure 229. Compare Register 2 (COMP2) 482
Figure 230. Capture Register 1 (CAPT1). 482
Figure 231. Capture Register 2 (CAPT2). 483
Figure 232. Load Register (LOAD) 483
Figure 233. Hold Register (HOLD). 483
Figure 234. Counter (CNTR) 483
Figure 235. Control Register 1 (CTRL1) 484
Figure 236. Control Register 2 (CTRL2) 487
Figure 237. Control Register 3 (CTRL3) 490
Figure 238. Status Register (STS). 491
Figure 239. Interrupt and DMA Enable Register (INTDMA). 492
Figure 240. Comparator Load 1 (CMPLD1). 493
Figure 241. Comparator Load 2 (CMPLD2). 494
Figure 242. Compare and Capture Control Register (CCCTRL) 494
Figure 243. Input Filter Register (FILT) 497
Figure 244. Watchdog Time-out Low Word Register (WDTOL). 498
Figure 245. Watchdog Time-out High Word Register (WDTOH) 498
Figure 246. Channel Enable Register (ENBL) 498
Figure 247. DMA Request 0 Select Register (DREQ0) 499
Figure 248. DMA Request 1 Select Register (DREQ1) 499
Figure 249. Quadrature Incremental Position Encoder 503
Figure 250. Triggered Count Mode (Length=1) 503
Figure 251. One-Shot Mode (Length=1) 503
Figure 252. Pulse Output Mode 504
Figure 253. Variable PWM Waveform 505
Figure 254. Variable Frequency PWM Mode Timing 505

Figure 255. Compare Register and OFLAG Timing . . . 506
Figure 256. Processor Core Type (PCT) Register . . . 512
Figure 257. Chip-Defined Platform Revision (REV) Register . . . 512
Figure 258. Platform Crossbar Master Configuration (PLAMC) . . . 513
Figure 259. Platform Crossbar Slave Configuration (PLASC) . . . 513
Figure 260. IPS On-Platform Module Configuration (IOPMC) Register . . . 514
Figure 261. Miscellaneous Reset Status (MRSR) Register . . . 515
Figure 262. Miscellaneous User-Defined Control (MUDCR) Register . . . 515
Figure 263. ECC Configuration (ECR) Register . . . 518
Figure 264. ECC Status (ESR) Register . . . 520
Figure 265. ECC Error Generation (EEGR) Register . . . 522
Figure 266. Platform Flash Memory ECC Address (PFEAR) Register . . . 525
Figure 267. Platform Flash Memory ECC Master Number (PFEMR) Register . . . 526
Figure 268. Platform Flash Memory ECC Attributes (PFEAT) Register . . . 526
Figure 269. Platform Flash Memory ECC Data High Register (PFEDRH) . . . 527
Figure 270. Platform Flash Memory ECC Data Low Register (PFEDRL) . . . 528
Figure 271. Platform RAM ECC Address (PREAR) Register . . . 529
Figure 272. Platform RAM ECC Syndrome (PRESR) Register . . . 530
Figure 273. Platform RAM ECC Master Number (PREMR) Register . . . 533
Figure 274. Platform RAM ECC Attributes (PREAT) Register . . . 533
Figure 275. Platform RAM ECC Data High Register (PREDRH) . . . 534
Figure 276. Platform RAM ECC Data Low Register (PREDRL) . . . 535
Figure 277. FCCU top-level diagram . . . 538
Figure 278. FCCU Control Register (FCCU_CTRL) . . . 542
Figure 279. FCCU CTRL Key Register (FCCU_CTRLK) . . . 544
Figure 280. FCCU Configuration Register (FCCU_CFG) . . . 545
Figure 281. FCCU CF Configuration Register (FCCU_CF_CFG0..3) . . . 547
Figure 282. FCCU NCF Configuration Register (FCCU_NCF_CFG0..3) . . . 548
Figure 283. FCCU CFS Configuration Register 0 (FCCU_CFS_CFG0) . . . 549
Figure 284. FCCU CFS Configuration Register 1 (FCCU_CFS_CFG1) . . . 549
Figure 285. FCCU NCFS Configuration Register (FCCU_NCFS_CFG0..7) . . . 550
Figure 286. FCCU CF Status Register (FCCU_CFS0..3) . . . 551
Figure 287. FCCU CF Key Register (FCCU_CFK) . . . 552
Figure 288. FCCU NCF Status Register (FCCU_NCFS0..3) . . . 554
Figure 289. FCCU NCF Key Register (FCCU_NCFK) . . . 555
Figure 290. FCCU NCF Enable Register (FCCU_NCFE0..3) . . . 555
Figure 291. FCCU NCF Time-out Enable Register (FCCU_NCF_TOE0..3) . . . 556
Figure 292. FCCU NCF Time-out Register (FCCU_NCF_TO) . . . 557
Figure 293. FCCU CFG Timeout Register (FCCU_CFG_TO) . . . 558
Figure 294. FCCU IO Control Register (FCCU_EINOUT) . . . 559
Figure 295. FCCU Status Register (FCCU_STAT) . . . 560
Figure 296. FCCU SC Freeze Status Register (FCCU_SCFS) . . . 561
Figure 297. FCCU CF Fake Register (FCCU_CFF) . . . 562
Figure 298. FCCU NCF Fake Register (FCCU_NCFF) . . . 563
Figure 299. FCCU IRQ Status Register (FCCU_IRQ_STAT) . . . 564
Figure 300. FCCU IRQ Enable Register (FCCU_IRQ_EN) . . . 565
Figure 301. FCCU XTMR Register (FCCU_XTMR) . . . 566
Figure 302. FCCU MCS Register (FCCU_MCS) . . . 566
Figure 303. FCCU state diagram . . . 569
Figure 304. Self-checking reaction . . . 570
Figure 305. Critical FAULT recovery (a) . . . 572
Figure 306. Critical FAULT recovery (b) . . . 572

Figure 307. Non-critical FAULT (ALARM state) recovery (a) . . . 573
Figure 308. Non-critical FAULT (ALARM state) recovery (b) . . . 573
Figure 309. Non-critical FAULT (ALARM -> FAULT state) recovery . . . 574
Figure 310. Critical FAULT (nesting) recovery . . . 575
Figure 311. Critical FAULT (and non-critical FAULT nesting) recovery . . . 577
Figure 312. NMI/WKUP scheme . . . 578
Figure 313. STCU-FCCU (case a) . . . 578
Figure 314. STCU-FCCU (case b) . . . 579
Figure 315. STCU-FCCU (case c) . . . 579
Figure 316. NVM interface . . . 580
Figure 317. Dual-rail protocol (slow switching mode) . . . 582
Figure 318. Dual-rail protocol (fast switching mode) . . . 582
Figure 319. Time-switching protocol . . . 583
Figure 320. Bi-stable protocol . . . 584
Figure 321. C90FL flash memory array diagram . . . 589
Figure 322. C90FL flash memory system block diagram . . . 591
Figure 323. Module Configuration Register (MCR) . . . 593
Figure 324. LML Register . . . 597
Figure 325. HBL Register . . . 599
Figure 326. SLL Register . . . 600
Figure 327. LMS Register . . . 601
Figure 328. HBS Register . . . 603
Figure 329. ADR Register . . . 604
Figure 330. UT0 Register . . . 605
Figure 331. UT1 Register . . . 607
Figure 332. UT2 Register . . . 607
Figure 333. UM0 Register . . . 608
Figure 334. UM1 Register . . . 609
Figure 335. UM2 Register . . . 610
Figure 336. UM3 Register . . . 610
Figure 337. UM4 register . . . 611
Figure 338. Nonvolatile Private Censorship Password 0 register (NVPWD0) . . . 611
Figure 339. Nonvolatile Private Censorship Password 1 register (NVPWD1) . . . 612
Figure 340. Nonvolatile System Censoring Information register (NVSCI) for cut1 . . . 612
Figure 341. Nonvolatile System Censoring Information register (NVSCI) for cut2/3 . . . 613
Figure 342. System Block Diagram with PFLASH Controller . . . 625
Figure 343. PFLASH2P High Level Block Diagram . . . 626
Figure 344. PFLASH2P Memory Map . . . 627
Figure 345. PFLASH Configuration Register 0 (PFCR0) for port 0 . . . 628
Figure 346. PFLASH Access Protection Register (PFAPR) . . . 632
Figure 347. FlexCAN block diagram . . . 638
Figure 348. Message Buffer Structure . . . 643
Figure 349. Rx FIFO Structure . . . 646
Figure 350. ID Table 0 - 7 . . . 647
Figure 351. Module Configuration Register (MCR) . . . 648
Figure 352. Control Register (CTRL) . . . 652
Figure 353. Free Running Timer (TIMER) . . . 656
Figure 354. Rx Global Mask Register (RXGMASK) . . . 657
Figure 355. Error Counter Register (ECR) . . . 659
Figure 356. Error and Status Register (ESR) . . . 659
Figure 357. Interrupt Masks 1 Register (IMASK1) . . . 662
Figure 358. Interrupt Flags 1 Register (IFLAG1) . . . 663

Figure 359. Rx Individual Mask Registers (RXIMR0 - RXIMR31) 664
Figure 360. CAN Engine Clocking Scheme 674
Figure 361. Segments within the Bit Time 675
Figure 362. Arbitration, Match and Move Time Windows 676
Figure 363. PWM block diagram 684
Figure 364. PWM Submodule Block Diagram 685
Figure 365. Center Aligned Example 687
Figure 366. Edge Aligned Example (INIT=VAL2=VAL4) 688
Figure 367. Phase Shifted Outputs Example 689
Figure 368. Phase Shifted PWMs Applied to a Transformer Primary 690
Figure 369. Double Switching Output Example 691
Figure 370. Multiple Output Trigger Generation in Hardware 692
Figure 371. Multiple Output Triggers Over Several PWM Cycles 693
Figure 372. Capture Capabilities of the E-Capture Circuit 694
Figure 373. Output Pulse Width Measurement Possible with the E-Capture Circuit 695
Figure 374. Sensorless BLDC Commutation Using the Force Out Function 696
Figure 375. Clocking block diagram for each PWM submodule 697
Figure 376. Register Reload Logic 698
Figure 377. Submodule timer synchronization 698
Figure 378. PWM Generation Hardware 700
Figure 379. Force Out Logic 702
Figure 380. Complementary Channel Pair 703
Figure 381. Typical 3 Phase AC Motor Drive 703
Figure 382. Deadtime Insertion and Fine Control Logic 704
Figure 383. Deadtime Insertion 705
Figure 384. Deadtime Distortion 706
Figure 385. Current-status Sense Scheme for Deadtime Correction 707
Figure 386. Output Voltage Waveforms 708
Figure 387. Output Logic Section 709
Figure 388. Enhanced Capture (E-Capture) Logic 710
Figure 389. Fault Decoder for PWMA 711
Figure 390. Automatic Fault Clearing 712
Figure 391. Manual Fault Clearing (FSAFE=0) 713
Figure 392. Manual Fault Clearing (FSAFE=1) 713
Figure 393. Full Cycle Reload Frequency Change 714
Figure 394. Half Cycle Reload Frequency Change 714
Figure 395. Full and Half Cycle Reload Frequency Change 715
Figure 396. PWMF Reload Interrupt Request 715
Figure 397. Counter Register (CNT) 718
Figure 398. Initial Count Register (INIT) 718
Figure 399. Control 2 Register (CTRL2) 718
Figure 400. Control 1 Register (CTRL1) 721
Figure 401. Value Register 0 (VAL0) 723
Figure 402. Value Register 1 (VAL1) 723
Figure 403. Value Register 2 (VAL2) 724
Figure 404. Value Register 3 (VAL3) 724
Figure 405. Value Register 4 (VAL4) 724
Figure 406. Value Register 5 (VAL5) 725
Figure 407. Output Control Register (OCTRL) 725
Figure 408. Status Register (STS) 727
Figure 409. Interrupt Enable Register (INTEN) 728
Figure 410. DMA Enable Register (DMAEN) 729

Figure 411. Output Trigger Control Register (TCTRL). 730
Figure 412. Fault Disable Mapping Register (DISMAP). 730
Figure 413. Deadtime Count Register 0 (DTCNT0). 731
Figure 414. Deadtime Count Register 1 (DTCNT1). 731
Figure 415. Capture Control X Register (CAPTCTRLX) 731
Figure 416. Capture Compare X Register (CAPTCMPX . . . . .)733
Figure 417. Capture Value 0 Register (CVAL0). 733
Figure 418. Capture Value 0 Cycle Register (CVAL0CYC). 734
Figure 419. Capture Value 1 Register (CVAL1). 734
Figure 420. Capture Value 1 Cycle Register (CVAL1CYC). 734
Figure 421. Output Enable Register (OUTEN) 735
Figure 422. Mask Register (MASK) 735
Figure 423. Software Controlled Output Register (SWCOUT). 736
Figure 424. Deadtime Source Select Register (DTSRCSEL) 737
Figure 425. Master Control Register (MCTRL) 739
Figure 426. Fault Control Register (FCTRL) 741
Figure 427. Fault Status Register (FSTS) 742
Figure 428. Fault Filter Register (FFILT) 743
Figure 429. FLEXRAY block diagram 748
Figure 430. Module Version Register (FR_MVR) 759
Figure 431. Module Configuration Register (FR_MCR). 759
Figure 432. System Memory Base Address High Register (FR_SYMBADHR) 762
Figure 433. System Memory Base Address Low Register (FR_SYMBADLR). 762
Figure 434. Strobe Signal Control Register (FR_STBSCR). 762
Figure 435. Message Buffer Data Size Register (FR_MBDSR). 764
Figure 436. Message Buffer Segment Size and Utilization Register (FR_MBSSUTR) 764
Figure 437. PE DRAM Access Register (FR_PEDRAR) 765
Figure 438. PE DRAM Data Register (FR_PEDRDR) 766
Figure 439. Protocol Operation Control Register (FR_POCR) 766
Figure 440. Global Interrupt Flag and Enable Register (FR_GIFER) 768
Figure 441. Protocol Interrupt Flag Register 0 (FR_PIFR0). 771
Figure 442. Protocol Interrupt Flag Register 1 (FR_PIFR1). 773
Figure 443. Protocol Interrupt Enable Register 0 (FR_PIER0) 774
Figure 444. Protocol Interrupt Enable Register 1 (FR_PIER1) 776
Figure 445. CHI Error Flag Register (FR_CHIERFR) 777
Figure 446. Message Buffer Interrupt Vector Register (FR_MBIVEC). 780
Figure 447. Channel A Status Error Counter Register (FR_CASERCR). 781
Figure 448. Channel B Status Error Counter Register (FR_CBSERCR). 781
Figure 449. Protocol Status Register 0 (FR_PSR0) 782
Figure 450. Protocol Status Register 1 (FR_PSR1) 783
Figure 451. Protocol Status Register 2 (FR_PSR2) 785
Figure 452. Protocol Status Register 3 (FR_PSR3) 787
Figure 453. Macrotick Counter Register (FR_MTCTR) 789
Figure 454. Cycle Counter Register (FR_CYCTR) 789
Figure 455. Slot Counter Channel A Register (FR_SLTCTAR) 790
Figure 456. Slot Counter Channel B Register (FR_SLTCTBR) 790
Figure 457. Rate Correction Value Register (FR_RTCORVR) 791
Figure 458. Offset Correction Value Register (FR_OFCORVR) 791
Figure 459. Combined Interrupt Flag Register (FR_CIFR) 792
Figure 460. System Memory Access Time-Out Register (FR_SYMATOR). 793
Figure 461. Sync Frame Counter Register (FR_SFCNTR) 794
Figure 462. Sync Frame Table Offset Register (FR_SFTOR) 794

Figure 463. Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR) 795
Figure 464. Sync Frame ID Rejection Filter Register (FR_SFIDRFR) 796
Figure 465. Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR) 797
Figure 466. Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR) 797
Figure 467. Network Management Vector Registers (FR_NMVR0–FR_NMVR5) 798
Figure 468. Network Management Vector Length Register (FR_NMVLR) 799
Figure 469. Timer Configuration and Control Register (FR_TICCR) 799
Figure 470. Timer 1 Cycle Set Register (FR_TI1CYSR) 800
Figure 471. Timer 1 Macrotick Offset Register (FR_TI1MTOR) 801
Figure 472. Timer 2 Configuration Register 0 (FR_TI2CR0) 801
Figure 473. Timer 2 Configuration Register 1 (FR_TI2CR1) 802
Figure 474. Slot Status Selection Register (FR_SSSR) 803
Figure 475. Slot Status Counter Condition Register (FR_SSCCR) 804
Figure 476. Slot Status Registers (FR_SSR0–FR_SSR7) 805
Figure 477. Slot Status Counter Registers (FR_SSCR0–FR_SSCR3) 807
Figure 478. MTS A Configuration Register (FR_MTSACFR) 808
Figure 479. MTS B Configuration Register (MTSBCFR) 808
Figure 480. Receive Shadow Buffer Index Register (FR_RSBIR) 809
Figure 481. Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR) 810
Figure 482. Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR) 810
Figure 483. Receive FIFO Periodic Timer Register (FR_RFPTR) 810
Figure 484. Receive FIFO Watermark and Selection Register (FR_RFWMSR) 811
Figure 485. Receive FIFO Start Index Register (FR_RFSIR) 812
Figure 486. Receive FIFO Depth and Size Register (RFDSR) 812
Figure 487. Receive FIFO A Read Index Register (FR_RFARIR) 813
Figure 488. Receive FIFO B Read Index Register (FR_RFBRIR) 813
Figure 489. Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR) 814
Figure 490. Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR) 814
Figure 491. Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR) 815
Figure 492. Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR) 815
Figure 493. Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR) 816
Figure 494. Receive FIFO Range Filter Configuration Register (FR_RFRFCFR) 816
Figure 495. Receive FIFO Range Filter Control Register (FR_RFRFCTR) 817
Figure 496. Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR) 818
Figure 497. Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR) 818
Figure 498. Protocol Configuration Register 0 (FR_PCR0) 821
Figure 499. Protocol Configuration Register 1 (FR_PCR1) 821
Figure 500. Protocol Configuration Register 2 (FR_PCR2) 822
Figure 501. Protocol Configuration Register 3 (FR_PCR3) 822
Figure 502. Protocol Configuration Register 4 (FR_PCR4) 822
Figure 503. Protocol Configuration Register 5 (FR_PCR5) 822
Figure 504. Protocol Configuration Register 6 (FR_PCR6) 823
Figure 505. Protocol Configuration Register 7 (FR_PCR7) 823
Figure 506. Protocol Configuration Register 8 (FR_PCR8) 823
Figure 507. Protocol Configuration Register 9 (FR_PCR9) 823
Figure 508. Protocol Configuration Register 10 (FR_PCR10) 824
Figure 509. Protocol Configuration Register 11 (FR_PCR11) 824
Figure 510. Protocol Configuration Register 12 (FR_PCR12) 824
Figure 511. Protocol Configuration Register 13 (FR_PCR13) 824
Figure 512. Protocol Configuration Register 14 (FR_PCR14) 825
Figure 513. Protocol Configuration Register 15 (FR_PCR15) 825
Figure 514. Protocol Configuration Register 16 (FR_PCR16) 825

Figure 515. Protocol Configuration Register 17 (FR_PCR17) . . . 825
Figure 516. Protocol Configuration Register 18 (FR_PCR18) . . . 826
Figure 517. Protocol Configuration Register 19 (FR_PCR19) . . . 826
Figure 518. Protocol Configuration Register 20 (FR_PCR20) . . . 826
Figure 519. Protocol Configuration Register 21 (FR_PCR21) . . . 826
Figure 520. Protocol Configuration Register 22 (FR_PCR22) . . . 827
Figure 521. Protocol Configuration Register 23 (FR_PCR23) . . . 827
Figure 522. Protocol Configuration Register 24 (FR_PCR24) . . . 827
Figure 523. Protocol Configuration Register 25 (FR_PCR25) . . . 827
Figure 524. Protocol Configuration Register 26 (FR_PCR26) . . . 828
Figure 525. Protocol Configuration Register 27 (FR_PCR27) . . . 828
Figure 526. Protocol Configuration Register 28 (FR_PCR28) . . . 828
Figure 527. Protocol Configuration Register 29 (FR_PCR29) . . . 828
Figure 528. Protocol Configuration Register 30 (FR_PCR30) . . . 829
Figure 529. ECC Error Interrupt Flag and Enable Register (FR_EEIFER) . . . 829
Figure 530. ECC Error Report and Injection Control Register (FR_EERICR) . . . 832
Figure 531. ECC Error Report Address Register (FR_EERAR) . . . 833
Figure 532. ECC Error Report Data Register (FR_EERDR) . . . 833
Figure 533. ECC Error Report Code Register (FR_EERCR) . . . 834
Figure 534. ECC Error Injection Address Register (FR_EEIAR) . . . 835
Figure 535. ECC Error Injection Data Register (FR_EEIDR) . . . 835
Figure 536. ECC Error Injection Code Register (FR_EEICR) . . . 836
Figure 537. Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn) . . . 836
Figure 538. Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn) . . . 838
Figure 539. Message Buffer Frame ID Registers (FR_MBFIDRn) . . . 840
Figure 540. Message Buffer Index Registers (FR_MBIDXRn) . . . 840
Figure 541. Physical Message Buffer Structure . . . 841
Figure 542. Individual Message Buffer Structure . . . 843
Figure 543. Receive Shadow Buffer Structure . . . 844
Figure 544. Receive FIFO Structure . . . 845
Figure 545. Example of FlexRay Memory Area Layout (FR_MCR[FAM] = 0) . . . 848
Figure 546. Example of FlexRay Memory Area Layout (FR_MCR[FAM] = 1) . . . 849
Figure 547. Frame Header Structure (Receive Message Buffer and Receive FIFO) . . . 851
Figure 548. Frame Header Structure (Transmit Message Buffer) . . . 851
Figure 549. Frame Header Structure (Transmit Message Buffer for Key Slot) . . . 851
Figure 550. Receive Message Buffer Slot Status Structure (ChAB) . . . 854
Figure 551. Receive Message Buffer Slot Status Structure (ChA) . . . 854
Figure 552. Receive Message Buffer Slot Status Structure (ChB) . . . 855
Figure 553. Transmit Message Buffer Slot Status Structure (ChAB) . . . 856
Figure 554. Transmit Message Buffer Slot Status Structure (ChA) . . . 856
Figure 555. Transmit Message Buffer Slot Status Structure (ChB) . . . 857
Figure 556. Message Buffer Data Field Structure . . . 858
Figure 557. Single Transmit Message Buffer Access Regions . . . 862
Figure 558. Single Transmit Message Buffer States . . . 863
Figure 559. Message Transmission Timing . . . 867
Figure 560. Message Transmission from HLck state with unlock . . . 867
Figure 561. Null Frame Transmission from Idle state . . . 868
Figure 562. Null Frame Transmission from HLck state . . . 868
Figure 563. Null Frame Transmission from HLck state with unlock . . . 868
Figure 564. Null Frame Transmission from Idle state with locking . . . 869
Figure 565. Receive Message Buffer Access Regions . . . 870
Figure 566. Receive Message Buffer States . . . 871

Figure 567. Message Reception Timing . . . 875
Figure 568. Double Transmit Buffer Structure and Data Flow . . . 877
Figure 569. Double Transmit Message Buffer Access Regions Layout . . . 878
Figure 570. Double Transmit Message Buffer State Diagram (Commit Side) . . . 879
Figure 571. Double Transmit Message Buffer State Diagram (Transmit Side) . . . 880
Figure 572. Internal Message Transfer in Streaming Commit Mode . . . 884
Figure 573. Internal Message Transfer in Immediate Commit Mode . . . 885
Figure 574. Inconsistent Channel Assignment . . . 888
Figure 575. Message Buffer Reconfiguration Scheme . . . 889
Figure 576. Received Frame FIFO Filter Path . . . 893
Figure 577. Dual Channel Device Mode . . . 896
Figure 578. Single Channel Device Mode (Channel A) . . . 897
Figure 579. Single Channel Device Mode (Channel B) . . . 897
Figure 580. External Offset Correction Write and Application Timing . . . 898
Figure 581. External Rate Correction Write and Application Timing . . . 898
Figure 582. Sync Table Memory Layout . . . 899
Figure 583. Sync Frame Table Trigger and Generation Timing . . . 901
Figure 584. Strobe Signal Timing (type = pulse, clk_offset = -2) . . . 904
Figure 585. Strobe Signal Timing (type = pulse, clk_offset = +4) . . . 904
Figure 586. Slot Status Vector Update . . . 906
Figure 587. Slot Status Counting and FR_SSCRn Update . . . 908
Figure 588. Scheme of FR_GIFER interrupt signal generation . . . 912
Figure 589. Scheme of FR_EEIFER interrupt signal generation . . . 913
Figure 590. Scheme of FR_CIFR flags generation . . . 914
Figure 591. Transmit Data Not Available . . . 928
Figure 592. Transmit Data Not Available . . . 928
Figure 593. FMPLL block diagram . . . 929
Figure 594. Control Register (CR) . . . 931
Figure 595. Modulation Register (MR) . . . 933
Figure 596. Illustration of progressive clock switching . . . 935
Figure 597. Frequency modulation . . . 937
Figure 598. INTC block diagram . . . 940
Figure 599. INTC Block Configuration Register (INTC_BCR) . . . 943
Figure 600. INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0) . . . 944
Figure 601. INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0) . . . 945
Figure 602. INTC End Of Interrupt Register for Processor 0 (INTC_EOIR_PRC0) . . . 946
Figure 603. INTC Software Set/Clear Interrupt Register 0 - 3 (INTC_SSCIR0_3) . . . 947
Figure 604. INTC Software Set/Clear Interrupt Register 4 - 7 (INTC_SSCIR4_7) . . . 947
Figure 605. INTC Priority Select Register 0 - 3 (INTC_PSR0_3) . . . 948
Figure 606. INTC Priority Select Register 252 - 255 (INTC_PSR252_255) . . . 948
Figure 607. Software Vector Mode Handshaking Timing Diagram . . . 952
Figure 608. Hardware Vector Mode Handshaking Timing Diagram . . . 953
Figure 609. JTAG STL (IEEE 1149.1) block diagram . . . 970
Figure 610. 5-Bit Instruction Register . . . 973
Figure 611. Device Identification Register . . . 974
Figure 612. TEST_CTRL register (cut2/3 only) . . . 975
Figure 613. CENSOR_CTRL Register . . . 976
Figure 614. Shifting Data Through a Register . . . 977
Figure 615. IEEE 1149.1-2001 TAP Controller Finite State Machine . . . 978
Figure 616. LINFlexD block diagram . . . 982
Figure 617. LIN network topology . . . 984
Figure 618. LIN frame structure . . . 985

Figure 619. Break field . . . 985
Figure 620. Sync pattern . . . 985
Figure 621. Structure of the data field . . . 986
Figure 622. Identifier . . . 986
Figure 623. LINFlexD controller operating modes . . . 988
Figure 624. Filter configuration - register organization . . . 994
Figure 625. Identifier match index . . . 995
Figure 626. LIN sync field measurement . . . 996
Figure 627. LINFlexD in Loop Back mode . . . 997
Figure 628. LINFlexD in Self Test mode . . . 998
Figure 629. UART mode 8-bit data frame . . . 998
Figure 630. UART mode 9-bit data frame . . . 999
Figure 631. UART mode 16-bit data frame . . . 999
Figure 632. UART mode 17-bit data frame . . . 999
Figure 633. LIN control register 1 (LINCR1) . . . 1004
Figure 634. LIN interrupt enable register (LINIER) . . . 1007
Figure 635. LIN status register (LINSR) . . . 1009
Figure 636. LIN error status register (LINESR) . . . 1012
Figure 637. UART mode control register (UARTCR) . . . 1013
Figure 638. UART mode status register (UARTSR) . . . 1016
Figure 639. LIN timeout control status register (LINTCSR) . . . 1018
Figure 640. LIN output compare register (LINOCR) . . . 1019
Figure 641. LIN timeout control register (LINTOCR) . . . 1020
Figure 642. LIN timeout control register (LINTOCR) . . . 1021
Figure 643. LIN integer baud rate register (LINIBRR) . . . 1021
Figure 644. LIN checksum field register (LINCFR) . . . 1022
Figure 645. LIN control register 2 (LINCR2) . . . 1023
Figure 646. Buffer identifier register (BIDR) . . . 1024
Figure 647. Buffer data register least significant (BDRL) . . . 1025
Figure 648. Buffer data register most significant (BDRM) . . . 1026
Figure 649. Identifier filter enable register (IFER) . . . 1027
Figure 650. Identifier filter match index (IFMI) . . . 1027
Figure 651. Identifier filter mode register (IFMR) . . . 1028
Figure 652. Identifier filter control registers (IFCR0–IFCR15) . . . 1029
Figure 653. Global control register (GCR) . . . 1030
Figure 654. UART preset timeout register (UARTPTO) . . . 1032
Figure 655. UART current timeout register (UARTCTO) . . . 1033
Figure 656. DMA Tx enable register (DMATXE) . . . 1034
Figure 657. DMA Rx enable register (DMARXE) . . . 1034
Figure 658. TCD chain memory map (master node, TX mode) . . . 1036
Figure 659. FSM to control the DMA TX interface (master node) . . . 1037
Figure 660. TCD chain memory map (master node, RX mode) . . . 1038
Figure 661. FSM to control the DMA RX interface (master node) . . . 1039
Figure 662. TCD chain memory map (slave node, TX mode) . . . 1040
Figure 663. FSM to control the DMA TX interface (slave node) . . . 1042
Figure 664. TCD chain memory map (slave node, RX mode) . . . 1043
Figure 665. FSM to control the DMA RX interface (slave node) . . . 1045
Figure 666. TCD chain memory map (UART node, TX mode) . . . 1046
Figure 667. FSM to control the DMA TX interface (UART node) . . . 1047
Figure 668. TCD chain memory map (UART node, RX mode) . . . 1048
Figure 669. FSM to control the DMA RX interface (UART node) . . . 1050
Figure 670. Header and response timeout . . . 1053

Figure 671. Interrupt diagram . . . 1054
Figure 672. Programming consideration: master node, transmitter . . . 1056
Figure 673. Programming consideration: master node, receiver . . . 1056
Figure 674. Programming consideration: master node, transmitter, bit error . . . 1056
Figure 675. Programming consideration: master node, receiver, checksum error . . . 1056
Figure 676. Programming consideration: slave node, transmitter, no filters . . . 1057
Figure 677. Programming consideration: slave node, receiver, no filters . . . 1057
Figure 678. Programming consideration: slave node, transmitter, no filters, bit error . . . 1057
Figure 679. Programming consideration: slave node, receiver, no filters, checksum error . . . 1057
Figure 680. Programming consideration: slave node, at least one TX filter, BF is reset, ID matches filter . . . 1058
Figure 681. Programming consideration: slave node, at least one RX filter, BF is reset, ID matches filter . . . 1058
Figure 682. Programming consideration: slave node, RX only, TX only, RX and TX filters, ID not matching filter, BF is reset . . . 1058
Figure 683. Programming consideration: slave node, TX filter, BF is set . . . 1059
Figure 684. Programming consideration: slave node, RX filter, BF is set . . . 1059
Figure 685. Programming consideration: slave node, TX filter, RX filter, BF is set . . . 1060
Figure 686. Programming consideration: extended frames . . . 1060
Figure 687. Programming consideration: response timeout . . . 1061
Figure 688. Programming consideration: frame timeout . . . 1061
Figure 689. Programming consideration: header timeout . . . 1061
Figure 690. Programming consideration: UART mode . . . 1061
Figure 691. MPU block diagram . . . 1063
Figure 692. MPU Control/Error Status Register (MPU_CESR) . . . 1067
Figure 693. MPU Error Address Register, Slave Port n (MPU_EARn) . . . 1068
Figure 694. MPU Error Detail Register, Slave Port n (MPU_EDRn) . . . 1068
Figure 695. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0) . . . 1070
Figure 696. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1) . . . 1070
Figure 697. MPU Region Descriptor, Word 2 Register (MPU_RGDn.Word2) . . . 1071
Figure 698. MPU Region Descriptor, Word 3 Register (MPU_RGDn.Word3) . . . 1073
Figure 699. MPU RGD Alternate Access Control n (MPU_RGDAACn) . . . 1074
Figure 700. MPU access evaluation macro . . . 1076
Figure 701. Access evaluation macro critical timing path . . . 1078
Figure 702. MC_ME block diagram . . . 1083
Figure 703. Global Status Register (ME_GS) . . . 1093
Figure 704. Mode Control Register (ME_MCTL) . . . 1095
Figure 705. Mode Enable Register (ME_ME) for cut1 . . . 1096
Figure 706. Mode Enable Register (ME_ME) for cut2/3 . . . 1096
Figure 707. Interrupt Status Register (ME_IS) for cut1 . . . 1098
Figure 708. Interrupt Status Register (ME_IS) for cut2/3 . . . 1098
Figure 709. Interrupt Mask Register (ME_IM) for cut1 . . . 1099
Figure 710. Interrupt Mask Register (ME_IM) for cut2/3 . . . 1099
Figure 711. Invalid Mode Transition Status Register (ME_IMTS) . . . 1100
Figure 712. Debug Mode Transition Status Register (ME_DMTS) . . . 1101
Figure 713. RESET Mode Configuration Register (ME_RESET_MC) . . . 1104
Figure 714. TEST Mode Configuration Register (ME_TEST_MC) . . . 1105
Figure 715. SAFE Mode Configuration Register (ME_SAFE_MC) . . . 1105
Figure 716. DRUN Mode Configuration Register (ME_DRUN_MC) . . . 1106
Figure 717. RUN0…3 Mode Configuration Registers (ME_RUN0...3_MC) . . . 1106
Figure 718. HALT0 Mode Configuration Register (ME_HALT0_MC) . . . 1107
Figure 719. STOP0 Mode Configuration Register (ME_STOP0_MC) . . . 1107

Figure 720. Peripheral Status Register 0 (ME_PS0) . . . 1109
Figure 721. Peripheral Status Register 1 (ME_PS1) . . . 1109
Figure 722. Peripheral Status Register 2 (ME_PS2) . . . 1110
Figure 723. Run Peripheral Configuration Registers (ME_RUN_PC0...7) . . . 1110
Figure 724. Low-Power Peripheral Configuration Registers (ME_LP_PC0...7) . . . 1111
Figure 725. Peripheral Control Registers (ME_PCTL0...143) . . . 1112
Figure 726. MC_ME Mode Diagram . . . 1114
Figure 727. MC_ME Transition Diagram . . . 1124
Figure 728. MC_ME Application Example Flow Diagram . . . 1128
Figure 729. NXSS block diagram . . . 1129
Figure 730. Development Control Register 1 (DC1) . . . 1131
Figure 731. Development Control Register 2 (DC2) . . . 1132
Figure 732. Watchpoint Trigger Register (WT) . . . 1132
Figure 733. Data Trace Control Register (DTC) . . . 1133
Figure 734. Data Trace Start Address Registers (DTSA1, DTSA2) . . . 1135
Figure 735. Data Trace Start Address Registers (DTEA1, DTEA2) . . . 1135
Figure 736. Break / Watchpoint Control Register 1 (BWC1) . . . 1135
Figure 737. Break / Watchpoint Control Register 2 (BWC2) . . . 1136
Figure 738. Breakpoint / Watchpoint Address Registers (BWA1, BWA2) . . . 1137
Figure 739. Data Write/Read Message Format . . . 1140
Figure 740. Error Message Format . . . 1140
Figure 741. Data Write/Read w/ Sync Message Format . . . 1141
Figure 742. Watchpoint message format . . . 1143
Figure 743. Error message format . . . 1144
Figure 744. Nexus Port Controller Block Diagram . . . 1146
Figure 745. 4-Bit Instruction Register . . . 1151
Figure 746. Nexus Device ID Register . . . 1152
Figure 747. Port Configuration Register (PCR) . . . 1153
Figure 748. MSEO Transfers (for 2-bit MSEO) . . . 1156
Figure 749. Message Field Sizes . . . 1157
Figure 750. Transmission Sequence of Messages . . . 1158
Figure 751. Shifting Data Into Register . . . 1158
Figure 752. IEEE 1149.1-2001 TAP Controller State Machine . . . 1160
Figure 753. NEXUS Controller State Machine . . . 1161
Figure 754. IEEE 1149.1 Controller Command Input . . . 1162
Figure 755. RC Control Register (RC_CTL) . . . 1165
Figure 756. External Crystal Oscillator Control Register (OSC_CTL) . . . 1167
Figure 757. PIT block diagram . . . 1169
Figure 758. PIT Module Control Registers (PITMCR) . . . 1171
Figure 759. Timer Load Value Register (LDVAL) . . . 1172
Figure 760. Current Timer Value Register (CVAL) . . . 1173
Figure 761. Timer Control Register (TCTRL) . . . 1174
Figure 762. Timer Flag Register (TFLG) . . . 1175
Figure 763. Stopping and Starting a Timer . . . 1176
Figure 764. Modifying Running Timer Period . . . 1176
Figure 765. Dynamically Setting a New Load Value . . . 1176
Figure 766. PBRIDGE interface . . . 1178
Figure 767. MPROT*n* field structure . . . 1181
Figure 768. PACR*n* field structure . . . 1181
Figure 769. MC_PCU Block Diagram . . . 1185
Figure 770. Power Domain Status Register (PCU_PSTAT) . . . 1187
Figure 771. PMU block diagram . . . 1188

Figure 772. Built-In self test flow . . . 1191
Figure 773. PMUCTRL status register (PMUCTRL_STATUS) . . . 1193
Figure 774. PMUCTRL control register (PMUCTRL_CTRL) . . . 1194
Figure 775. PMUCTRL mask fault register (PMUCTRL_MASKF) . . . 1195
Figure 776. PMUCTRL fault monitor register (PMUCTRL_FAULT) . . . 1196
Figure 777. PMUCTRL interrupt request status register (PMUCTRL_IRQS) . . . 1197
Figure 778. PMUCTRL interrupt request enable register (PMUCTRL_IRQE) . . . 1198
Figure 779. REG_PROT block diagram . . . 1200
Figure 780. REG_PROT Memory Diagram . . . 1201
Figure 781. Key to register fields . . . 1203
Figure 782. Soft lock bit register (SLBR*n*) . . . 1203
Figure 783. Global configuration register (GCR) . . . 1205
Figure 784. Change lock settings directly via area #4 . . . 1206
Figure 785. Change lock settings for 16-bit protected addresses . . . 1206
Figure 786. Change lock settings for 32-bit protected addresses . . . 1207
Figure 787. Change lock settings for mixed protection . . . 1207
Figure 788. Enable locking via mirror module space (area #3) . . . 1208
Figure 789. Enable locking for protected and unprotected addresses . . . 1208
Figure 790. MC_RGM block diagram . . . 1252
Figure 791. Functional Event Status Register (RGM_FES) . . . 1257
Figure 792. Destructive Event Status Register (RGM_DES) for cut1 . . . 1259
Figure 793. Destructive Event Status Register (RGM_DES) for cut2/3 . . . 1260
Figure 794. Functional Event Reset Disable Register (RGM_FERD) . . . 1261
Figure 795. Destructive Event Reset Disable Register (RGM_DERD) for cut1 . . . 1263
Figure 796. Destructive Event Reset Disable Register (RGM_DERD) for cut2/3 . . . 1263
Figure 797. Functional Event Alternate Request Register (RGM_FEAR) . . . 1264
Figure 798. Functional Event Short Sequence Register (RGM_FESS) . . . 1265
Figure 799. Functional Bidirectional Reset Enable Register (RGM_FBRE) . . . 1267
Figure 800. MC_RGM State Machine . . . 1270
Figure 801. Monotonic Power-Up . . . 1276
Figure 802. Non-Monotonic Power-Up (dip on 1.2V supply) . . . 1277
Figure 803. Non-Monotonic Power-Up (overshoot on 1.2V supply) . . . 1278
Figure 804. Non-Monotonic Power-Up (dip on 2.7V supply) . . . 1279
Figure 805. Post-Power-Up 1.2V LVD . . . 1280
Figure 806. Post-Power-Up 1.2V HVD . . . 1281
Figure 807. Post-Power-Up 2.7V LVD . . . 1282
Figure 808. The STCU within the Safety Integrity Subsystem . . . 1299
Figure 809. Boot sequence phase 1: Self test . . . 1300
Figure 810. Boot sequence phase 2: Functional reset . . . 1300
Figure 811. STCU block diagram . . . 1302
Figure 812. STCU SK Code Register (STCU_SKC) [cut2/3 only] . . . 1305
Figure 813. STCU Configuration Register (STCU_CFG) [cut1 only] . . . 1305
Figure 814. STCU Error Register (STCU_ERR) for cut1 . . . 1306
Figure 815. STCU Error Register (STCU_ERR) for cut2/3 . . . 1306
Figure 816. STCU Error Key Register (STCU_ERRK) [cut2/3 only] . . . 1308
Figure 817. STCU LBIST Status Register (STCU_LBS) . . . 1308
Figure 818. STCU LBIST End Flag Register (STCU_LBE) . . . 1309
Figure 819. STCU MBIST Status Low Register (STCU_MBSL) . . . 1310
Figure 820. STCU MBIST Status High Register (STCU_MBSH) . . . 1310
Figure 821. STCU MBIST End Flag Low Register (STCU_MBEL) . . . 1311
Figure 822. STCU MBIST End Flag High Register (STCU_MBEH) . . . 1312
Figure 823. STCU LBIST MISR Expected Low Register (STCU_LB_MISREL) [cut2/3 only] . . . 1312

Figure 824. STCU LBIST MISR Expected High Register (STCU_LB_MISREH) [cut2/3 only] 1313
Figure 825. STCU LBIST MISR Read Low Register (STCU_LB_MISRRL) 1313
Figure 826. STCU LBIST MISR Read High Register (STCU_LB_MISRRH) 1314
Figure 827. SEMA4 block diagram 1323
Figure 828. SEMA4 gate *n* register (SEMA4_GATE*n*) 1326
Figure 829. Semaphores processor *n* IRQ notification enable (SEMA4_CP{0,1}INE) 1326
Figure 830. Semaphores processor *n* IRQ notification (SEMA4_CP{0,1}NTF) 1327
Figure 831. Semaphores (secure) reset gate *n* (SEMA4_RSTGT) 1328
Figure 832. Semaphores (secure) Reset IRQ notification (SEMA4_RSTNTF) 1330
Figure 833. SWG block diagram 1336
Figure 834. SWG control register (SWG_CTRL) 1337
Figure 835. SWG status register (SWG_STAT) 1338
Figure 836. SWT Control Register (SWT_CR) 1343
Figure 837. SWT Interrupt Register (SWT_IR) 1344
Figure 838. SWT Time-Out Register (SWT_TO) 1345
Figure 839. SWT Window Register (SWT_WN) 1345
Figure 840. SWT Service Register (SWT_SR) 1346
Figure 841. SWT Counter Output Register (SWT_CO) 1346
Figure 842. SWT Service Key Register (SWT_SK) 1347
Figure 843. Pseudorandom Key Generator 1348
Figure 844. SIUL block diagram 1353
Figure 845. Key to Register Fields 1356
Figure 846. MCU ID Register 1 (MIDR1) 1357
Figure 847. MCU ID Register 2 (MIDR2) 1358
Figure 848. Interrupt Status Flag Register (ISR) 1359
Figure 849. Interrupt Request Enable Register (IRER) 1359
Figure 850. Interrupt Rising-Edge Event Enable Register (IREER) 1360
Figure 851. Interrupt Falling-Edge Event Enable Register (IFEER) 1360
Figure 852. Interrupt Filter Enable Register (IFER) 1361
Figure 853. Pad Configuration Registers (PCR0) 1361
Figure 854. Pad Selection for Multiplexed Inputs Register (PSMI0_3) 1364
Figure 855. Port GPIO Pad Data Output register 0–3 (GPDO0_3) 1365
Figure 856. Port GPIO Pad Data Input register 0–3 (GPDI0_3–GPDI104_107)) 1366
Figure 857. Parallel GPIO Pad Data Out Register (PGPDO0–PGPDO3) 1367
Figure 858. Parallel GPIO Pad Data In Register (PGPDI0–PGPDI3) 1368
Figure 859. Masked Parallel GPIO Pad Data Out Register (MPGPDO0) 1369
Figure 860. Interrupt Filter Maximum Counter Register (IFMC0–IFMC31) 1370
Figure 861. Interrupt Filter Clock Prescaler Register (IFCPR) 1371
Figure 862. Data port width configurations for different port width accesses 1372
Figure 863. External Interrupt pad diagram 1373
Figure 864. SSCM block diagram 1375
Figure 865. Status (STATUS) Register 1377
Figure 866. System Memory and ID (MEMCONFIG) Register 1378
Figure 867. Error Configuration (ERROR) Register 1379
Figure 868. Debug Status Port (DEBUGPORT) Register 1380
Figure 869. DPM Boot (DPMBOOT) Register 1382
Figure 870. Boot Key Register (DPMKEY) Register 1383
Figure 871. User Option Status (UOPS) Register 1384
Figure 872. SSCM Control (SCTR) Register 1384
Figure 873. Key to register fields 1387
Figure 874. STM Control Register (STM_CR) 1388
Figure 875. STM Count Register (STM_CNT) 1388

Figure 876. STM Channel Control Register (STM_CCRn) 1389
Figure 877. STM Channel Interrupt Register (STM_CIRn) 1390
Figure 878. STM Channel Compare Register (STM_CMPn) 1390
Figure 879. TSENS block diagram 1392
Figure 880. WKPU block diagram 1395
Figure 881. Key to Register Fields 1396
Figure 882. NMI Status Flag Register (NSR) 1397
Figure 883. NMI Configuration Register (NCR) 1398
Figure 884. NMI pad diagram 1400

1 Preface

1.1 Overview

The primary objective of this document is to define the functionality of the SPC56ELxx microcontroller for use by software and hardware developers. The SPC56ELxx is built on Power Architecture® technology and integrates technologies that are important for today's electric power steering, chassis, and safety applications that require a high safety integrity level.

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 SPC56ELxx 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 Chapter organization and device-specific information

This document includes chapters that describe:

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

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

1.4 Information about different device versions ("cuts")

The SPC56ELxx device is available in three silicon versions, or "cuts". These are referred to as "cut1" ,"cut2" and "cut3" throughout this document. Functional differences between the cuts are clearly identified with the labels "cut1" ,"cut2" and "cut3". The mask identification corresponding to each cut is listed here for reference:

- Mask 0M78X - Cut 1.0
- Mask 0N28C - Cut 2.0
- Mask 0N89D - Cut 3.0
- Mask 2N89D - Cut 3.1

All cut2 functionalities are applicable for cut3 except for the following :

- BAM code (cut2 uses ROM and Shadow Flash, cut3 ROM only)
- Chip ID's (MINOR_MASK_ID, JTAG_ID)
- POR and LVD flag behavioral

1.5 Acronyms and abbreviations

The following table lists some acronyms and abbreviations used in this document.

Term	Meaning
AUTOSAR	Automotive Open System Architecture
GPIO	General-purpose I/O
IEEE	Institute for Electrical and Electronics Engineers
JEDEC	Joint Electron Device Engineering Council
JTAG	Joint Test Action Group
Mux	Multiplex
Rx	Receive
SAG	Safety Application Guide
TBD	To be determined
Tx	Transmit
UART	Universal asynchronous receiver transmitter

1.6 References

In addition to this reference manual, the following documents provide additional information on the operation of the SPC56ELxx:

- 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

2 Introduction

2.1 The SPC56ELxx microcontroller

The SPC56ELxx microcontroller is based on the Power Architecture® and targets the electric power steering, chassis, and safety applications that require a high safety integrity level.

All SPC56ELxx devices are built around a dual-core safety platform with an innovative safety concept targeting ISO26262 ASILD and IEC61508 SIL3 integrity levels(b). To minimize additional software and module level features to reach this target, on-chip redundancy is provided for the critical components of the microcontroller (CPU core, eDMA controller, interrupt controller, crossbar bus system, memory protection unit, flash-memory controller and SRAM controllers, peripheral bus bridge, system timers, and watchdog timer). Lock step Redundancy Checking Units are implemented at each output of this Sphere of Replication (SoR). ECC is available for on-chip SRAM and flash memories. A programmable fault collection and control unit monitors the integrity status of the device and provides flexible safe state control.

The device's host processor core is a member of the e200z4 Power Architecture® compatible core family. The device's 5-stage pipeline dual issue core provides a high efficiency allowing high performance with minimum power dissipation.

The peripheral set is compatible with the SPC560P device family and provides high-end electrical motor control capability with very low CPU intervention due to the on-chip cross-triggering unit.

The SPC56ELxx is developed with high-performance 90-nm embedded flash-memory technology that provides substantial cost reduction per feature and significant performance improvement.

2.2 SPC56ELxx device summary

Table 1 summarizes the SPC56ELxx microcontroller.

b. All statements on functional safety in this chapter are under the condition that the requirements given in the Safety Application Guide are followed.

Table 1. SPC56ELxx device summary

Feature		SPC56EL60
CPU	Type	2 × e200z4 (in lock-step or decoupled operation)
	Architecture	Harvard
	Execution speed	0–120 MHz (+2% FM)
	DMIPS intrinsic performance	>240 MIPS
	SIMD (DSP + FPU)	Yes
	MMU	16 entry
	Instruction set PPC	Yes
	Instruction set VLE	Yes
	Instruction cache	4 KB, EDC
	MPU-16 regions	Yes, replicated module
	Semaphore unit (SEMA4)	Yes
Buses	Core bus	AHB, 32-bit address, 64-bit data
	Internal periphery bus	32-bit address, 32-bit data
Crossbar	Master × slave ports	Lock Step Mode: 4 × 3 Decoupled Parallel Mode: 6 × 3
Memory	Flash	1 MB, ECC, RWW
	Static RAM (SRAM)	128 KB, ECC
Modules	Interrupt Controller (INTC)	16 interrupt levels, replicated module
	Periodic Interrupt Timer (PIT)	1 × 4 channels
	System Timer Module (STM)	1 × 4 channels, replicated module
	Software Watchdog Timer (SWT)	Yes, replicated module
	eDMA	16 channels, replicated module
	FlexRay	1 × 64 message buffers, dual channel
	FlexCAN	2 × 32 message buffers
	LINFlexD (UART and LIN with DMA support)	2
	Clock out	Yes
	Fault Collection and Control Unit (FCCU)	Yes
	Cross Triggering Unit (CTU)	Yes
	eTimer	3 × 6 channels(1)
	FlexPWM	2 Module 4 × (2 + 1) channels(2)
	Analog-to-Digital Converter (ADC)	2 × 12-bit ADC, 16 channels per ADC (3 internal, 4 shared and 9 external)
	Sine Wave Generator (SWG)	32 point

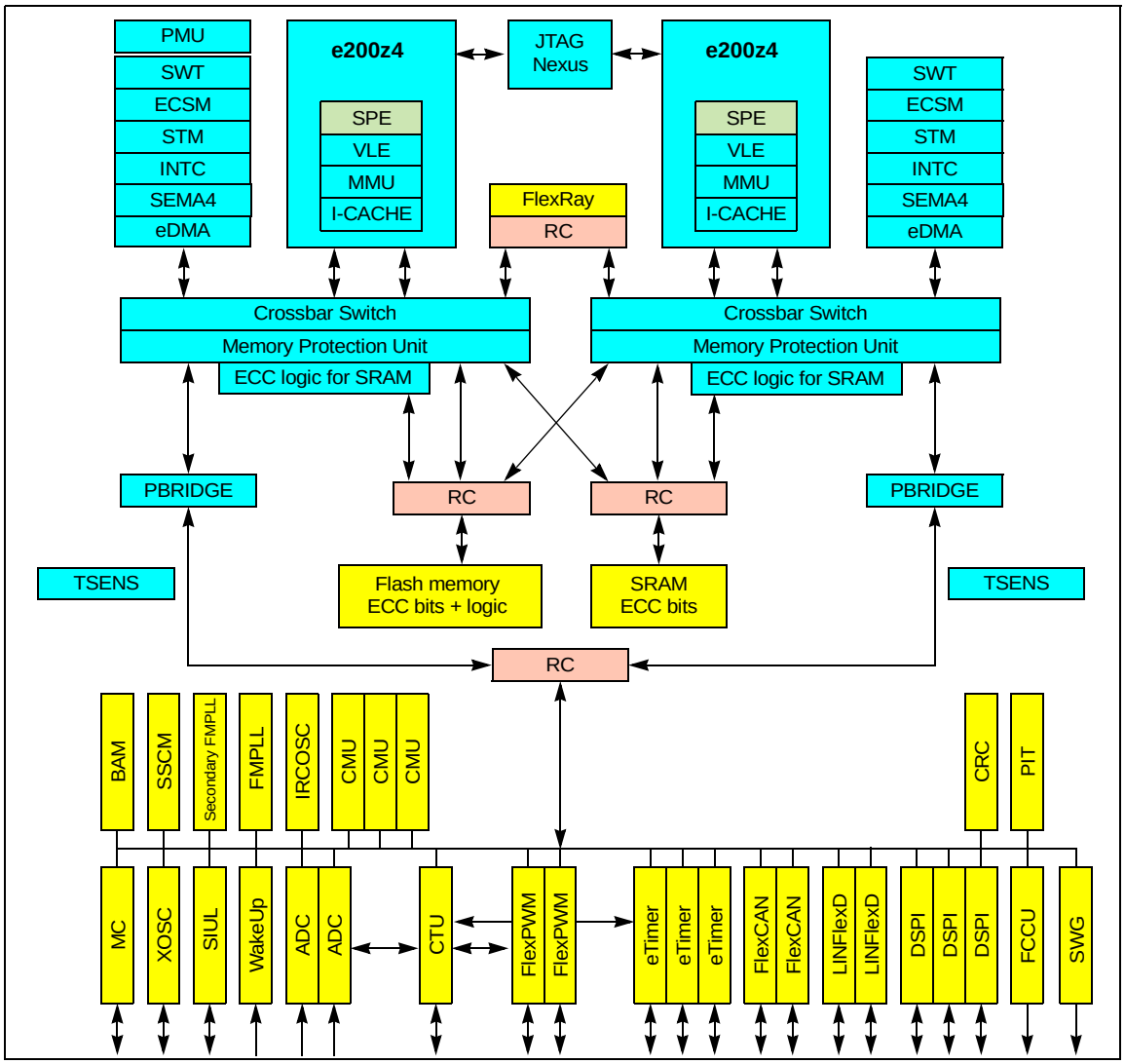
Table 1. SPC56ELxx device summary (continued)

Feature		SPC56EL60
Modules (cont.)	Deserial Serial Peripheral Interface (DSPI)	3 × DSPI as many as 8 chip selects
	Cyclic Redundancy Checker (CRC) unit	Yes
	Junction temperature sensor (TSENS)	Yes, replicated module
	Digital I/Os	≥ 16
Supply	Device power supply	3.3 V with integrated bypassable ballast transistor External ballast transistor not needed for bare die
	Analog reference voltage	3.0 V – 3.6 V and 4.5 V – 5.5 V
Clocking	Frequency-modulated phase-locked loop (FMPLL)	2
	Internal RC oscillator	16 MHz
	External crystal oscillator	4 – 40 MHz
Debug	Nexus	Level 3+
Packages	Known Good Die (KGD)	Yes
	LQFP	100 pins 144 pins
	LBGA(3)	LBGA257
Temperature	Temperature range (junction)	–40 to 150 °C
	Ambient temperature range using external ballast transistor (LQFP)	–40 to 125 °C

1. The third eTimer (eTimer_2) is available with external I/O access only in the BGA package, on the LQFP package eTimer_2 is available internally only without any external I/O access.
2. The second FlexPWM module is available only in the BGA package.
3. LBGA257 available only as development package.

2.3 Device block diagram

Figure 1 shows a top-level block diagram of the SPC56ELxx.



ADC – Analog-to-Digital Converter
BAM – Boot Assist Module
CMU – Clock Monitoring Unit
CRC – Cyclic Redundancy Check unit
CTU – Cross Triggering Unit
DSPI – Serial Peripherals Interface
ECC – Error Correction Code
ECSM – Error Correction Status Module
eDMA – Enhanced Direct Memory Access controller
FCCU – Fault Collection and Control Unit
FlexCAN – Controller Area Network controller
FMPLL – Frequency Modulated Phase Locked Loop
INTC – Interrupt Controller
IRCOSC – Internal RC Oscillator
JTAG – Joint Test Action Group interface
LINFlexD – LIN controller with DMA support
MC – Mode Entry, Clock, Reset, & Power
PBRIDGE – Peripheral bridge
PIT – Periodic Interrupt Timer
PMU – Power Management Unit
RC – Redundancy Checker
RTC – Real Time Clock
SEMA4 – Semaphore Unit
SIUL – System Integration Unit Lite
SSCM – System Status and Configuration Module
STM – System Timer Module
SWG – Sine Wave Generator
SWT – Software Watchdog Timer
TSENS – Temperature Sensor
XOSC – Crystal Oscillator

Figure 1. SPC56ELxx block diagram

2.4 Feature summary

- High-performance e200z4d dual core
  - 32-bit Power Architecture® technology CPU
  - Core frequency as high as 120 MHz
  - Dual issue five-stage pipeline core
  - Variable Length Encoding (VLE)
  - Memory Management Unit (MMU)
  - 4 KB instruction cache with error detection code
  - Signal processing engine (SPE)
- Memory available
  - 1 MB flash memory with ECC
  - 128 KB on-chip SRAM with ECC
  - Built-in RWW capabilities for EEPROM emulation
- SIL3/ASILD innovative safety concept: LockStep mode and Fail-safe protection
  - Sphere of replication (SoR) for key components (such as CPU core, eDMA, crossbar switch)
  - Fault collection and control unit (FCCU)
  - Redundancy control and checker unit (RCCU) on outputs of the SoR connected to FCCU
  - Boot-time Built-In Self-Test for Memory (MBIST) and Logic (LBIST) triggered by hardware
  - Boot-time Built-In Self-Test for ADC and flash memory triggered by software
  - Replicated safety enhanced watchdog
  - Replicated junction temperature sensor
  - Non-maskable interrupt (NMI)
  - 16-region memory protection unit (MPU)
  - Clock monitoring units (CMU)
  - Power management unit (PMU)
  - Cyclic redundancy check (CRC) unit
- Decoupled Parallel mode for high-performance use of replicated cores
- Nexus Class 3+ interface
- Interrupts
  - Replicated 16-priority controller
  - Replicated 16-channel eDMA controller
- GPIOs individually programmable as input, output or special function
- Three 6-channel general-purpose eTimer units
- 2 FlexPWM units
  - Four 16-bit channels per module
- Communications interfaces
  - 2 LINFlexD channels
  - 3 DSPI channels with automatic chip select generation
  - 2 FlexCAN interfaces (2.0B Active) with 32 message objects

  - FlexRay module (V2.1 Rev. A) with 2 channels, 64 message buffers and data rates up to 10 Mbit/s
- Two 12-bit analog-to-digital converters (ADCs)
  - 16 input channels
  - Programmable cross triggering unit (CTU) to synchronize ADCs conversion with timer and PWM
- Sine wave generator (D/A with low pass filter)
- On-chip CAN/UART bootstrap loader
- Single 3.0 V to 3.6 V voltage supply
- Ambient temperature range –40 °C to 125 °C
- Junction temperature range –40 °C to 150 °C

2.5 Feature details

2.5.1 High-performance e200z4d core

The e200z4d Power Architecture® core provides the following features:

- 2 independent execution units, both supporting fixed-point and floating-point operations
- Dual issue 32-bit Power Architecture technology compliant
  - 5-stage pipeline (IF, DEC, EX1, EX2, WB)
  - In-order execution and instruction retirement
- Full support for Power Architecture instruction set and Variable Length Encoding (VLE)
  - Mix of classic 32-bit and 16-bit instruction allowed
  - Optimization of code size possible
- Thirty-two 64-bit general purpose registers (GPRs)
- Harvard bus (32-bit address, 64-bit data)
  - I-Bus interface capable of one outstanding transaction plus one piped with no wait-on-data return
  - D-Bus interface capable of two transactions outstanding to fill AHB pipe
- I-cache and I-cache controller
  - 4 KB, 256-bit cache line (programmable for 2- or 4-way)
- No data cache
- 16-entry MMU
- 8-entry branch table buffer
- Branch look-ahead instruction buffer to accelerate branching
- Dedicated branch address calculator
- 3 cycles worst case for missed branch
- Load/store unit
  - Fully pipelined
  - Single-cycle load latency
  - Big- and little-endian modes supported
  - Misaligned access support
  - Single stall cycle on load to use
- Single-cycle throughput (2-cycle latency) integer 32 × 32 multiplication
- 4 – 14 cycles integer 32 × 32 division (average division on various benchmark of nine cycles)
- Single precision floating-point unit
  - 1 cycle throughput (2-cycle latency) floating-point 32 × 32 multiplication
  - Target 9 cycles (worst case acceptable is 12 cycles) throughput floating-point 32 × 32 division
  - Special square root and min/max function implemented

- Signal processing support: APU-SPE 1.1
  - Support for vectorized mode: as many as two floating-point instructions per clock
- Vectored interrupt support
- Reservation instruction to support read-modify-write constructs
- Extensive system development and tracing support via Nexus debug port

2.5.2 Crossbar switch (XBAR)

The XBAR multi-port crossbar switch supports simultaneous connections between four master ports and three slave ports. The crossbar supports a 32-bit address bus width and a 64-bit data bus width.

The crossbar allows four concurrent transactions to occur from any master port to any slave port, although one of those transfers must be an instruction fetch from internal flash memory. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions.

The crossbar provides the following features:

- 4 masters and 3 slaves supported per each replicated crossbar
  - Masters allocation for each crossbar: e200z4d core with two independent bus interface units (BIU) for I and D access (2 masters), one eDMA, one FlexRay
  - Slaves allocation for each crossbar: a redundant flash-memory controller with 2 slave ports to guarantee maximum flexibility to handle Instruction and Data array, one redundant SRAM controller with 1 slave port each and 1 redundant peripheral bus bridge
- 32-bit address bus and 64-bit data bus
- Programmable arbitration priority
  - Requesting masters can be treated with equal priority and are granted access to a slave port in round-robin method, based upon the ID of the last master to be granted access or a priority order can be assigned by software at application run time
- Temporary dynamic priority elevation of masters

The XBAR is replicated for each processing channel.

2.5.3 Memory Protection Unit (MPU)

The Memory Protection Unit splits the physical memory into 16 different regions. Each master (eDMA, FlexRay, CPU) can be assigned different access rights to each region.

- 16-region MPU with concurrent checks against each master access
- 32-byte granularity for protected address region

The memory protection unit is replicated for each processing channel.

2.5.4 Enhanced Direct Memory Access (eDMA)

The enhanced direct memory access (eDMA) controller is a second-generation module capable of performing complex data movements via 16 programmable channels, with minimal intervention from the host processor. The hardware microarchitecture includes a

DMA engine which performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels. This implementation is used to minimize the overall block size.

The eDMA module provides the following features:

- 16 channels supporting 8-, 16-, and 32-bit value single or block transfers
- Support variable sized queues and circular buffered queue
- Source and destination address registers independently configured to post-increment or stay constant
- Support major and minor loop offset
- Support minor and major loop done signals
- DMA task initiated either by hardware requestor or by software
- Each DMA task can optionally generate an interrupt at completion and retirement of the task
- Signal to indicate closure of last minor loop
- Transfer control descriptors mapped inside the SRAM

The eDMA controller is replicated for each processing channel.

2.5.5 On-chip flash memory with ECC

This device includes programmable, non-volatile flash memory. The non-volatile memory (NVM) can be used for instruction storage or data storage, or both. The flash memory module interfaces with the system bus through a dedicated flash memory array controller. It supports a 64-bit data bus width at the system bus port, and a 128-bit read data interface to flash memory. The module contains four 128-bit prefetch buffers. Prefetch buffer hits allow no-wait responses. Buffer misses incur a 3 wait state response at 120 MHz.

The flash memory module provides the following features

- 1 MB of flash memory in unique multi-partitioned hard macro
- Sectorization: 16 KB + 2 × 48 KB + 16 KB + 2 × 64 KB + 2 × 128 KB + 2 × 256 KB
- EEPROM emulation (in software) within same module but on different partition
- 16 KB test sector and 16 KB shadow block for test, censorship device and user option bits
- Wait states:
  - 3 wait states for frequencies =< 120 MHz
  - 2 wait states for frequencies =< 80 MHz
  - 1 wait state for frequencies =< 60 MHz
- Flash memory line 128-bit wide with 8-bit ECC on 64-bit word (total 144 bits)
- Accessed via a 64-bit wide bus for write and a 128-bit wide array for read operations
- 1-bit error correction, 2-bit error detection

2.5.6 On-chip SRAM with ECC

The SPC56ELxx SRAM provides a general-purpose single port memory.

ECC handling is done on a 32-bit boundary for data and it is extended to the address to have the highest possible diagnostic coverage including the array internal address decoder.

The SRAM module provides the following features:

- System SRAM: 128 KB
- ECC on 32-bit word (syndrome of 7 bits)
  - ECC covers SRAM bus address
- 1-bit error correction, 2-bit error detection
- Wait states:
  - 1 wait state for frequencies > 80 MHz and =< 120 MHz
  - 0 wait states for frequencies =< 80 MHz

2.5.7 Platform flash memory controller

The following list summarizes the key features of the flash memory controller:

- Single AHB port interface supports a 64-bit data bus. All AHB aligned and unaligned reads within the 32-bit container are supported. Only aligned word writes are supported.
- Array interfaces support a 128-bit read data bus and a 64-bit write data bus for each bank.
- Code flash (bank0) interface provides configurable read buffering and page prefetch support.
  - Four page-read buffers (each 128 bits wide) and a prefetch controller support speculative reading and optimized flash access.
- Single-cycle read responses (0 AHB data-phase wait states) for hits in the buffers. The buffers implement a least-recently-used replacement algorithm to maximize performance.
- 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 (on a per bank basis) to support use across a wide range of platforms and frequencies.
- Support of address-based read access timing for emulation of other memory types.
- Support for reporting of single- and multi-bit error events.
- Typical operating configuration loaded into programming model by system reset.

The platform flash controller is replicated for each processor.

2.5.8 Platform Static RAM Controller (SRAMC)

The SRAMC module is the platform SRAM array controller, with integrated error detection and correction.

The main features of the SRAMC provide connectivity for the following interfaces:

- XBAR Slave Port (64-bit data path)
- ECSM (ECC Error Reporting, error injection and configuration)
- SRAM array

The following functions are implemented:

- ECC encoding (32-bit boundary for data and complete address bus)
- ECC decoding (32-bit boundary and entire address)
- Address translation from the AHB protocol on the XBAR to the SRAM array

The platform SRAM controller is replicated for each processor.

2.5.9 Memory subsystem access time

Every memory access the CPU performs requires at least one system clock cycle for the data phase of the access. Slower memories or peripherals may require additional data phase wait states. Additional data phase wait states may also occur if the slave being accessed is not parked on the requesting master in the crossbar.

Table 2 shows the number of additional data phase wait states required for a range of memory accesses.

Table 2. Platform memory access time summary

AHB transfer	Data phase wait states	Description
e200z4d instruction fetch	0	Flash memory prefetch buffer hit (page hit)
e200z4d instruction fetch	3	Flash memory prefetch buffer miss (based on 4-cycle random flash array access time)
e200z4d data read	0–1	SRAM read
e200z4d data write	0	SRAM 32-bit write
e200z4d data write	0	SRAM 64-bit write (executed as 2 x 32-bit writes)
e200z4d data write	0–2	SRAM 8-,16-bit write (Read-modify-Write for ECC)
e200z4d flash memory read	0	Flash memory prefetch buffer hit (page hit)
e200z4d flash memory read	3	Flash memory prefetch buffer miss (at 120 MHz; includes 1 cycle of program flash memory controller arbitration)

2.5.10 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. A detected error (double error for flash memory or SRAM) is also reported to the FCCU. 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

2.5.11 Peripheral bridge (PBRIDGE)

The PBRIDGE implements the following features:

- Duplicated periphery
- Master access privilege level per peripheral (per master: read access enable; write access enable)
- Checker applied on PBRIDGE output toward periphery
- Byte endianess swap capability

2.5.12 Interrupt Controller (INTC)

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

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

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

The INTC provides the following features:

- Duplicated periphery
- Unique 9-bit vector per interrupt source
- 16 priority levels with fixed hardware arbitration within priority levels for each interrupt source
- Priority elevation for shared resource

The INTC is replicated for each processor.

2.5.13 System clocks and clock generation

The following list summarizes the system clock and clock generation on this device:

- Lock status continuously monitored by lock detect circuitry
- Loss-of-clock (LOC) detection for reference and feedback clocks
- On-chip loop filter (for improved electromagnetic interference performance and fewer external components required)
- Programmable output clock divider of system clock (÷1, ÷2, ÷4, ÷8)
- FlexPWM module and as many as three eTimer modules running on an auxiliary clock independent from system clock (with max frequency 120 MHz)
- On-chip crystal oscillator with automatic level control
- Dedicated internal 16 MHz internal RC oscillator for rapid start-up
  - Supports automated frequency trimming by hardware during device startup and by user application
- Auxiliary clock domain for motor control periphery (FlexPWM, eTimer, CTU, ADC, and SWG)

2.5.14 Frequency-Modulated Phase-Locked Loop (FMPLL)

Each device has two FMPLLs.

Each FMPLL allows the user to generate high speed system clocks starting from a minimum reference of 4 MHz input clock. Further, the FMPLL supports programmable frequency

modulation of the system clock. The FMPLL multiplication factor, output clock divider ratio are all software configurable. The FMPLLs have the following major features:

- Input frequency: 4–40 MHz continuous range (limited by the crystal oscillator)
- Voltage controlled oscillator (VCO) range: 256–512 MHz
- Frequency modulation via software control to reduce and control emission peaks
  - Modulation depth ±2% if centered or 0% to –4% if downshifted via software control register
  - Modulation frequency: triangular modulation with 25 kHz nominal rate
- Option to switch modulation on and off via software interface
- Output divider (ODF) for reduced frequency operation without re-lock
- 3 modes of operation
  - Bypass mode
  - Normal FMPLL mode with crystal reference (default)
  - Normal FMPLL mode with external reference
- Lock monitor circuitry with lock status
- Loss-of-lock detection for reference and feedback clocks
- Self-clocked mode (SCM) operation
- On-chip loop filter
- Auxiliary FMPLL
  - Used for FlexRay due to precise symbol rate requirement by the protocol
  - Used for motor control periphery and connected IP (A/D digital interface CTU) to allow independent frequencies of operation for PWM and timers and jitter-free control
  - Option to enable/disable modulation to avoid protocol violation on jitter and/or potential unadjusted error in electric motor control loop
  - Allows to run motor control periphery at different (precisely lower, equal or higher as required) frequency than the system to ensure higher resolution

2.5.15 Main oscillator

The main oscillator provides these features:

- Input frequency range 4–40 MHz
- Crystal input mode
- External reference clock (3.3 V) input mode
- FMPLL reference

2.5.16 Internal Reference Clock (RC) oscillator

The architecture uses constant current charging of a capacitor. The voltage at the capacitor is compared to the stable bandgap reference voltage. The RC oscillator is the device safe clock.

The RC oscillator provides these features:

- Nominal frequency 16 MHz
- ±5% variation over voltage and temperature after process trim
- Clock output of the RC oscillator serves as system clock source in case loss of lock or loss of clock is detected by the FMPLL
- RC oscillator is used as the default system clock during startup and can be used as back-up input source of FMPLL(s) in case XOSC fails

2.5.17 Clock, reset, power, mode and test control modules (MC_CGM, MC_RGM, MC_PCU, and MC_ME)

These modules provide the following:

- Clock gating and clock distribution control
- Halt, stop mode control
- Flexible configurable system and auxiliary clock dividers
- Various execution modes
  - HALT and STOP mode as reduced activity low power mode
  - Reset, Idle, Test, Safe
  - Various RUN modes with software selectable powered modules
  - No stand-by mode implemented (no internal switchable power domains)

2.5.18 Periodic Interrupt Timer Module (PIT)

The PIT module implements the following features:

- 4 general purpose interrupt timers
- 32-bit counter resolution
- Can be used for software tick or DMA trigger operation

2.5.19 System Timer Module (STM)

The STM implements the following features:

- Up-counter with 4 output compare registers
- OS task protection and hardware tick implementation per AUTOSAR[(c)] requirement

The STM is replicated for each processor.

2.5.20 Software Watchdog Timer (SWT)

This module implements the following features:

- Fault tolerant output
- Safe internal RC oscillator as reference clock
- Windowed watchdog
- Program flow control monitor with 16-bit pseudorandom key generation
- Allows a high level of safety (SIL3 monitor)

c. Automotive Open System Architecture

The SWT module is replicated for each processor.

2.5.21 Fault Collection and Control Unit (FCCU)

The FCCU module has the following features:

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

2.5.22 System Integration Unit Lite (SIUL)

The SIUL controls MCU reset configuration, pad configuration, external interrupt, general purpose I/O (GPIO), internal peripheral multiplexing, and system reset operation. The reset configuration block contains the external pin boot configuration logic. The pad configuration block controls the static electrical characteristics of I/O pins. The GPIO block provides uniform and discrete input/output control of the I/O pins of the MCU.

The SIU provides the following features:

- Centralized pad control on a per-pin basis
  - Pin function selection
  - Configurable weak pull-up/down
  - Configurable slew rate control (slow/medium/fast)
  - Hysteresis on GPIO pins
  - Configurable automatic safe mode pad control
- Input filtering for external interrupts

2.5.23 Non-Maskable Interrupt (NMI)

The non-maskable interrupt with de-glitching filter supports high-priority core exceptions.

2.5.24 Boot Assist Module (BAM)

The BAM is a block of read-only memory with hard-coded content. The BAM program is executed only if serial booting mode is selected via boot configuration pins.

The BAM provides the following features:

- Enables booting via serial mode (FlexCAN or LINFlex-UART)
- Supports programmable 64-bit password protection for serial boot mode
- Supports serial bootloading of either Power Architecture code (default) or VLE code
- Automatic switch to serial boot mode if internal flash memory is blank or invalid

2.5.25 System Status and Configuration Module (SSCM)

The SSCM on this device features the following:

- System configuration and status
- Debug port status and debug port enable
- Multiple boot code starting locations out of reset through implementation of search for valid Reset Configuration Half Word
- Sets up the MMU to allow user boot code to execute as either Power Architecture code (default) or as VLE code out of flash memory
- Triggering of device self-tests during reset phase of device boot

2.5.26 FlexCAN

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

The FlexCAN module provides the following features:

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

- Receive features
  - Individual programmable filters for each mailbox
  - 8 mailboxes configurable as a 6-entry receive FIFO
  - 8 programmable acceptance filters for receive FIFO
- Programmable clock source
  - System clock
  - Direct oscillator clock to avoid FMPLL jitter

2.5.27 FlexRay

The FlexRay module provides the following features:

- Full implementation of FlexRay Protocol Specification 2.1 Rev. A
- 64 configurable message buffers can be handled
- Dual channel or single channel mode of operation, each as fast as 10 Mbit/s data rate
- Message buffers configurable as transmit or receive
- Message buffer size configurable
- Message filtering for all message buffers based on Frame ID, cycle count, and message ID
- Programmable acceptance filters for receive FIFO
- Message buffer header, status, and payload data stored in system memory (SRAM)
- Internal FlexRay memories have error detection and correction

2.5.28 Serial communication interface module (LINFlexD)

The LINFlexD module (LINFlex with DMA support) on this device features the following:

- Supports LIN Master mode, LIN Slave mode and UART mode
- LIN state machine compliant to LIN1.3, 2.0, and 2.1 specifications
- Manages LIN frame transmission and reception without CPU intervention
- LIN features
  - Autonomous LIN frame handling
  - Message buffer to store as many as 8 data bytes
  - Supports messages as long as 64 bytes
  - Detection and flagging of LIN errors (Sync field, delimiter, ID parity, bit framing, checksum and Time-out errors)
  - Classic or extended checksum calculation
  - Configurable break duration of up to 50-bit times
  - Programmable baud rate prescalers (13-bit mantissa, 4-bit fractional)
  - Diagnostic features (Loop back, LIN bus stuck dominant detection)
  - Interrupt driven operation with 16 interrupt sources
- LIN slave mode features
  - Autonomous LIN header handling
  - Autonomous LIN response handling

- UART mode
  - Full-duplex operation
  - Standard non return-to-zero (NRZ) mark/space format
  - Data buffers with 4-byte receive, 4-byte transmit
  - Configurable word length (8-bit, 9-bit, 16-bit, or 17-bit words)
  - Configurable parity scheme: none, odd, even, always 0
  - Speed as fast as 2 Mbit/s
  - Error detection and flagging (Parity, Noise and Framing errors)
  - Interrupt driven operation with four interrupt sources
  - Separate transmitter and receiver CPU interrupt sources
  - 16-bit programmable baud-rate modulus counter and 16-bit fractional
  - Two receiver wake-up methods
- Support for DMA enabled transfers

2.5.29 Deserial Serial Peripheral Interface (DSPI)

The DSPI modules provide a synchronous serial interface for communication between the SPC56ELxx and external devices.

A DSPI module provides these features:

- Full duplex, synchronous transfers
- Master or slave operation
- Programmable master bit rates
- Programmable clock polarity and phase
- End-of-transmission interrupt flag
- Programmable transfer baud rate
- Programmable data frames from 4 to 16 bits
- As many as 8 chip select lines available, depending on package and pin multiplexing
- 4 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 as many as 5 transfers on the transmit and receive side
- Queueing operation possible through use of the eDMA
- General purpose I/O functionality on pins when not used for SPI

2.5.30 FlexPWM

The pulse width modulator module (FlexPWM) contains four PWM channels, each of which is configured to control a single half-bridge power stage. Two modules are present for external input/output on LFBGA257 devices; on the LQFP144 package, only one module is present for external input/output. Additionally, four fault input channels are provided per FlexPWM module.

This PWM is capable of controlling most motor types, including:

- AC induction motors (ACIM)
- Permanent Magnet AC motors (PMAC)
- Brushless (BLDC) and brush DC motors (BDC)
- Switched (SRM) and variable reluctance motors (VRM)
- Stepper motors

A FlexPWM module implements the following features:

- 16 bits of resolution for center, edge aligned, and asymmetrical PWMs
- Maximum operating frequency as high as 120 MHz
  - Clock source not modulated and independent from system clock (generated via secondary FMPLL)
- Fine granularity control for enhanced resolution of the PWM period
- PWM outputs can operate as complementary pairs or independent channels
- Ability to accept signed numbers for PWM generation
- Independent control of both edges of each PWM output
- Synchronization to external hardware or other PWM supported
- Double buffered PWM registers
  - Integral reload rates from 1 to 16
  - Half cycle reload capability
- Multiple ADC trigger events can be generated per PWM cycle via hardware
- Fault inputs can be assigned to control multiple PWM outputs
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom deadtime insertion
- Each complementary pair can operate with its own PWM frequency and deadtime values
- Individual software control for each PWM output
- All outputs can be forced to a value simultaneously
- PWMX pin can optionally output a third signal from each channel
- Channels not used for PWM generation can be used for buffered output compare functions
- Channels not used for PWM generation can be used for input capture functions
- Enhanced dual edge capture functionality
- Option to supply the source for each complementary PWM signal pair from any of the following:
  - External digital pin
  - Internal timer channel
  - External ADC input, taking into account values set in ADC high- and low-limit registers
- DMA support

2.5.31 eTimer module

The MPC5643L provides three eTimer modules (on the LQFP package eTimer_2 is available internally only without any external I/O access). Six 16-bit general purpose up/down timer/counters per module are implemented with the following features:

- Maximum clock frequency of 120 MHz
- Individual channel capability
  - Input capture trigger
  - Output compare
  - Double buffer (to capture rising edge and falling edge)
  - Separate prescaler for each counter
  - Selectable clock source
  - 0–100% pulse measurement
  - Rotation direction flag (Quad decoder mode)
- Maximum count rate
  - Equals peripheral clock divided by 2 for external event counting
  - Equals peripheral clock for internal clock counting
- Cascadeable counters
- Programmable count modulo
- Quadrature decode capabilities
- Counters can share available input pins
- Count once or repeatedly
- Preloadable counters
- Pins available as GPIO when timer functionality not in use
- DMA support

2.5.32 Sine Wave Generator (SWG)

A digital-to-analog converter is available to generate a sine wave based on 32 stored values for external devices (ex: resolver).

2.5.33 Analog-to-Digital Converter module (ADC)

The ADC module features include:

Analog part:

- 2 on-chip ADCs
  - 12-bit resolution SAR architecture
  - Same digital interface as in the SPC560P family
  - A/D Channels: 9 external, 3 internal and 4 shared with other A/D (total 16 channels)
  - One channel dedicated to each T-sensor to enable temperature reading during application
  - Separated reference for each ADC
  - Shared analog supply voltage for both ADCs
  - One sample and hold unit per ADC
  - Adjustable sampling and conversion time

Digital part:

- 4 analog watchdogs comparing ADC results against predefined levels (low, high, range) before results are stored in the appropriate ADC result location
- 2 modes of operation: CPU Mode or CTU Mode
- CPU mode features
  - Register based interface with the CPU: one result register per channel
  - ADC state machine managing three request flows: regular command, hardware injected command, software injected command
  - Selectable priority between software and hardware injected commands
  - 4 analog watchdogs comparing ADC results against predefined levels (low, high, range)
  - DMA compatible interface
- CTU mode features
  - Triggered mode only
  - 4 independent result queues (1×16 entries, 2×8 entries, 1×4 entries)
  - Result alignment circuitry (left justified; right justified)
  - 32-bit read mode allows to have channel ID on one of the 16-bit parts
  - DMA compatible interfaces
- Built-in self-test features triggered by software

2.5.34 Cross Triggering Unit (CTU)

The ADC cross triggering unit allows automatic generation of ADC conversion requests on user selected conditions without CPU load during the PWM period and with minimized CPU load for dynamic configuration.

The CTU implements the following features:

- Cross triggering between ADC, FlexPWM, eTimer, and external pins
- Double buffered trigger generation unit with as many as 8 independent triggers generated from external triggers
- Maximum operating frequency less than or equal to 120 MHz
- Trigger generation unit configurable in sequential mode or in triggered mode
- Trigger delay unit to compensate the delay of external low pass filter
- Double buffered global trigger unit allowing eTimer synchronization and/or ADC command generation
- Double buffered ADC command list pointers to minimize ADC-trigger unit update
- Double buffered ADC conversion command list with as many as 24 ADC commands
- Each trigger capable of generating consecutive commands
- ADC conversion command allows control of ADC channel from each ADC, single or synchronous sampling, independent result queue selection
- DMA support with safety features

2.5.35 Cyclic Redundancy Checker (CRC) Unit

The CRC module is a configurable multiple data flow unit to compute CRC signatures on data written to its input register.

The CRC unit has the following features:

- 3 sets of registers to allow 3 concurrent contexts with possibly different CRC computations, each with a selectable polynomial and seed
- Computes 16- or 32-bit wide CRC on the fly (single-cycle computation) and stores result in internal register.
  The following standard CRC polynomials are implemented:
  - $x^8 + x^4 + x^3 + x^2 + 1$ [8-bit CRC]
  - $x^{16} + x^{12} + x^5 + 1$ *[16-bit CRC-CCITT]*
  - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ [32-bit CRC-ethernet(32)]
- Key engine to be coupled with communication periphery where CRC application is added to allow implementation of safe communication protocol
- Offloads core from cycle-consuming CRC and helps checking configuration signature for safe start-up or periodic procedures
- CRC unit connected as peripheral bus on internal peripheral bus
- DMA support

2.5.36 Redundancy Control and Checker Unit (RCCU)

The RCCU checks all outputs of the sphere of replication (addresses, data, control signals). It has the following features:

- Duplicated module to guarantee highest possible diagnostic coverage (check of checker)
- Multiple times replicated IPs are used as checkers on the SoR outputs

2.5.37 Junction temperature sensor

The junction temperature sensor provides a value via an ADC channel that can be used by software to calculate the device junction temperature.

The key parameters of the junction temperature sensor include:

- Nominal temperature range from –40 to 150 °C
- Software temperature alarm via analog ADC comparator possible

2.5.38 Nexus Port Controller (NPC)

The NPC module provides real-time development support capabilities for this device in compliance with the IEEE-ISTO 5001-2003. This development support is supplied for MCUs without requiring external address and data pins for internal visibility.

The NPC block interfaces to the host processor and internal buses to provide development support as per the IEEE-ISTO 5001-2003 Class 3+, including selected features from Class 4 standard.

The development support provided includes program trace, data trace, watchpoint trace, ownership trace, run-time access to the MCUs internal memory map and access to the Power Architecture internal registers during halt. The Nexus interface also supports a JTAG only mode using only the JTAG pins. The following features are implemented:

- Full and reduced port modes
- MCKO (message clock out) pin
- 4 or 12 MDO (message data out) pins[(d)]
- 2 $\overline{\text{MSEO}}$ (message start/end out) pins
- $\overline{\text{EVTO}}$ (event out) pin
  - Auxiliary input port
- $\overline{\text{EVTI}}$ (event in) pin
- 5-pin JTAG port (JCOMP, TDI, TDO, TMS, and TCK)
  - Supports JTAG mode
- Host processor (e200) development support features
  - Data trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tool to trace reads or writes, or both, 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.
  - Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct branches, indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires

d. 4 MDO pins on LQFP144 package, 12 MDO pins on LFBGA257 package.

between the discontinuities. Thus, static code may be traced.

- Watchpoint messaging (WPM) via the auxiliary port
- Watchpoint trigger enable of program and/or data trace messaging
- Data tracing of instruction fetches via private opcodes

2.5.39 IEEE 1149.1 JTAG Controller (JTAGC)

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

The JTAG controller provides the following features:

- IEEE Test Access Port (TAP) interface with 5 pins:
  - TDI
  - TMS
  - TCK
  - TDO
  - JCOMP
- Selectable modes of operation include JTAGC/debug or normal system operation
- 5-bit instruction register that supports the following IEEE 1149.1-2001 defined instructions:
  - BYPASS
  - IDCODE
  - EXTEST
  - SAMPLE
  - SAMPLE/PRELOAD
- 3 test data registers: a bypass register, a boundary scan register, and a device identification register. The size of the boundary scan register is parameterized to support a variety of boundary scan chain lengths.
- TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry

2.5.40 Voltage regulator / Power Management Unit (PMU)

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

- Single external rail required
- Single high supply required: nominal 3.3 V both for packaged and Known Good Die option
  - Packaged option requires external ballast transistor due to reduced dissipation capacity at high temperature but can use embedded transistor if power dissipation is maintained within package dissipation capacity (lower frequency of operation)
  - Known Good Die option uses embedded ballast transistor as dissipation capacity is increased to reduce system cost
- All I/Os are at same voltage as external supply (3.3 V nominal)
- Duplicated Low-Voltage Detectors (LVD) to guarantee proper operation at all stages (reset, configuration, normal operation) and, to maximize safety coverage, one LVD can be tested while the other operates (on-line self-testing feature)

2.5.41 Built-In Self-Test (BIST) capability

This device includes the following protection against latent faults:

- Boot-time Memory Built-In Self-Test (MBIST)
- Boot-time scan-based Logic Built-In Self-Test (LBIST)
- Run-time ADC Built-In Self-Test (BIST)
- Run-time Built-In Self Test of LVDs

3 Memory Map

Table 3 shows the memory map for the SPC56ELxx.

All addresses on the SPC56ELxx, including those that are reserved, are identified in the table. The addresses represent the physical addresses assigned to each region or module name. The table also identifies the associated peripheral control register (PCTL) number and mode (LS, DP, or both).

All memory not listed in this table is reserved, and access to those locations may produce undesirable results.

Table 3. SPC56ELxx memory map, ordered by start address

Start Address	Size (KB)	PCTL number	Mode	Region / Module Name
Flash memory				
0x0000_0000	16	—	LS/DP	Flash-memory array partition 1 (low address) or test flash memory(1)
0x0000_4000	48	—	LS/DP	Flash-memory array partition 1 (low address)
0x0001_0000	48	—	LS/DP	Flash-memory array partition 1 (low address)
0x0001_C000	16	—	LS/DP	Flash-memory array partition 1 (low address)
0x0002_0000	64	—	LS/DP	Flash-memory array partition 2 (low address)
0x0003_0000	64	—	LS/DP	Flash-memory array partition 2 (low address)
0x0004_0000	128	—	LS/DP	Flash-memory array partition 3 (mid address)
0x0006_0000	128	—	LS/DP	Flash-memory array partition 3 (mid address)
0x0008_0000	256	—	LS/DP	Flash-memory array partition 4 (high address)
0x000C_0000	256	—	LS/DP	Flash-memory array partition 4 (high address)
0x00F0_0000	16	—	LS/DP	Shadow block
0x0100_0000	507904	—	LS/DP	Flash-memory emulation mapping
Static RAM				
0x4000_0000	64	—	DP	SRAM
	128	—	LS	SRAM
0x5000_0000	64	—	DP	SRAM(2)
On-platform 1 peripherals(3)				
0x8FF0_0000	16	—	DP	PBRIDGE_1
0x8FF0_4000	16	—	DP	XBAR_1
0x8FF1_0000	16	—	DP	MPU_1
0x8FF2_4000	16	—	DP	Semaphores (SEMA4_1)
0x8FF3_8000	16	—	DP	SWT_1
0x8FF3_C000	16	—	DP	STM_1
0x8FF4_0000	16	—	DP	ECSM_1

Table 3. SPC56ELxx memory map, ordered by start address (continued)

Start Address	Size (KB)	PCTL number	Mode	Region / Module Name
0x8FF4_8000	16	—	DP	INTC_1
Off-platform peripherals (mirrored to memory range 0xFFE8_0000–0xFFEF_FFFF)				
0xC3F8_8000	16	66	LS/DP	Flash 0 configuration (FLASH0)
0xC3F9_0000	16	68	LS/DP	System integration unit lite (SIUL)
0xC3F9_4000	16	69	LS/DP	Wakeup unit (WKPU)
0xC3FD_8000	16	86	LS/DP	System status and configuration module (SSCM)
0xC3FD_C000	16	87	LS/DP	Mode entry module (MC_ME)
0xC3FE_0000	16	88	LS/DP	Clock generation module (MC_CGM)
0xC3FE_4000	16	89	LS/DP	Reset generation module (MC_RGM)
0xC3FE_8000	16	90	LS/DP	Power control unit (MC_PCU)
0xC3FF_0000	16	92	LS/DP	Periodic interrupt timer (PIT)
0xC3FF_4000	16	93	LS/DP	Self-test control unit (STCU)
Off-platform peripherals				
0xFFE0_0000	16	32	LS/DP	Analog-to-digital converter 0 (ADC_0)
0xFFE0_4000	16	33	LS/DP	Analog-to-digital converter 1 (ADC_1)
0xFFE0_C000	16	35	LS/DP	Cross triggering unit (CTU)
0xFFE1_8000	16	38	LS/DP	eTimer_0
0xFFE1_C000	16	39	LS/DP	eTimer_1
0xFFE2_0000	16	40	LS/DP	eTimer_2
0xFFE2_4000	16	41	LS/DP	FlexPWM_0
0xFFE2_8000	16	42	LS/DP	FlexPWM_1
0xFFE4_0000	16	48	LS/DP	LINFlexD_0
0xFFE4_4000	16	49	LS/DP	LINFlexD_1
0xFFE6_8000	16	58	LS/DP	Cyclic redundancy check unit (CRC)
0xFFE6_C000	16	59	LS/DP	Fault collection and control unit (FCCU)
0xFFE7_8000	16	62	LS/DP	Sine wave generator (SWG)
On Platform 0 Peripherals				
0xFFF0_0000	16	—	LS	PBRIDGE_0, PBRIDGE_1
			DP	PBRIDGE_0
0xFFF0_4000	16	—	LS	XBAR_0, XBAR_1
			DP	XBAR_0
0xFFF1_0000	16	—	LS	MPU_0, MPU_1
			DP	MPU_0
0xFFF2_4000	16	—	DP	Semaphores (SEMA4_0)

Table 3. SPC56ELxx memory map, ordered by start address (continued)

Start Address	Size (KB)	PCTL number	Mode	Region / Module Name
0xFFF3_8000	16	—	LS	SWT_0, SWT_1
			DP	SWT_0
0xFFF3_C000	16	—	LS	STM_0, STM_1
			DP	STM_0
0xFFF4_0000	16	—	LS	ECSM_0, ECSM_1
			DP	ECSM_0
0xFFF4_4000	16	—	LS	eDMA_0, eDMA_1
			DP	eDMA_0
0xFFF4_8000	16	—	LS	INTC_0, INTC_1
			DP	INTC_0
Off Platform Peripherals				
0xFFF9_0000	16	4	LS/DP	DSPI_0
0xFFF9_4000	16	5	LS/DP	DSPI_1
0xFFF9_8000	16	6	LS/DP	DSPI_2
0xFFFC_0000	16	16	LS/DP	FlexCAN_0
0xFFFC_4000	16	17	LS/DP	FlexCAN_1
0xFFFD_C000	16	23	LS	eDMA channel multiplexer (DMA_MUX)
			DP	eDMA channel multiplexer (DMA_MUX)(4)
0xFFFE_0000	16	24	LS/DP	FlexRay controller (FlexRay)
0xFFFF_C000	16	31	LS/DP	Boot Assist Module (BAM)

1. Test flash memory is mapped to this address if the SCTR[TFE] bit in the SSCM is set (see *Section 49.3.1.8, SSCM Control Register (SCTR)*).
2. This range cannot be accessed by DMA_0.
3. These peripherals are not accessible to DMA_0 in DP mode.
4. DMA_MUX_1 is disabled in DP mode.

Table 4. SPC56ELxx memory map, ordered by module name

Start Address	Size (KB)	PCTL number	Mode	Region / Module Name
0xFFE0_0000	16	32	LS/DP	Analog-to-digital converter 0 (ADC_0)
0xFFE0_4000	16	33	LS/DP	Analog-to-digital converter 1 (ADC_1)
0xFFFF_C000	16	31	LS/DP	Boot assist module (BAM)
0xC3FE_0000	16	88	LS/DP	Clock generation module (MC_CGM)
0xFFE0_C000	16	35	LS/DP	Cross triggering unit (CTU)
0xFFE6_8000	16	58	LS/DP	Cyclic redundancy check unit (CRC)
0xFFF9_0000	16	4	LS/DP	DSPI_0

Table 4. SPC56ELxx memory map, ordered by module name (continued)

Start Address	Size (KB)	PCTL number	Mode	Region / Module Name
0xFFF9_4000	16	5	LS/DP	DSPI_1
0xFFF9_8000	16	6	LS/DP	DSPI_2
0x8FF4_0000	16	—	DP	ECSM_1
0xFFF4_0000	16	—	LS	ECSM_0, ECSM_1
			DP	ECSM_0
0xFFF4_4000	16	—	LS	eDMA_0, eDMA_1
			DP	eDMA_0
0xFFFD_C000	16	23	LS	eDMA channel multiplexer (DMA_MUX)
			DP	eDMA channel multiplexer (DMA_MUX)(1)
0xFFE1_8000	16	38	LS/DP	eTimer_0
0xFFE1_C000	16	39	LS/DP	eTimer_1
0xFFE2_0000	16	40	LS/DP	eTimer_2
0xFFE6_C000	16	59	LS/DP	Fault collection and control unit (FCCU)
0xC3F8_8000	16	66	LS/DP	Flash 0 configuration (FLASH0)
0x0000_0000	16	—	LS/DP	Flash-memory array partition 1 (low address) or test flash memory(2)
0x0000_4000	48	—	LS/DP	Flash-memory array partition 1 (low address)
0x0001_0000	48	—	LS/DP	Flash-memory array partition 1 (low address)
0x0001_C000	16	—	LS/DP	Flash-memory array partition 1 (low address)
0x0002_0000	64	—	LS/DP	Flash-memory array partition 2 (low address)
0x0003_0000	64	—	LS/DP	Flash-memory array partition 2 (low address)
0x0004_0000	128	—	LS/DP	Flash-memory array partition 3 (mid address)
0x0006_0000	128	—	LS/DP	Flash-memory array partition 3 (mid address)
0x0008_0000	256	—	LS/DP	Flash-memory array partition 4 (high address)
0x000C_0000	256	—	LS/DP	Flash-memory array partition 4 (high address)
0x0100_0000	507904	—	LS/DP	Flash-memory emulation mapping
0xFFFC_0000	16	16	LS/DP	FlexCAN_0
0xFFFC_4000	16	17	LS/DP	FlexCAN_1
0xFFE2_4000	16	41	LS/DP	FlexPWM_0
0xFFE2_8000	16	42	LS/DP	FlexPWM_1
0xFFFE_0000	16	24	LS/DP	FlexRay controller (FlexRay)
0x8FF4_8000	16	—	DP	INTC_1
0xFFF4_8000	16	—	LS	INTC_0, INTC_1
			DP	INTC_0

Table 4. SPC56ELxx memory map, ordered by module name (continued)

Start Address	Size (KB)	PCTL number	Mode	Region / Module Name
0xFFE4_0000	16	48	LS/DP	LINFlexD_0
0xFFE4_4000	16	49	LS/DP	LINFlexD_1
0x8FF1_0000	16	—	DP	MPU_1
0xFFF1_0000	16	—	LS	MPU_0, MPU_1
			DP	MPU_0
0xC3FD_C000	16	87	LS/DP	Mode entry module (MC_ME)
0xC3FF_0000	16	92	LS/DP	Periodic interrupt timer (PIT)
0xC3FE_8000	16	90	LS/DP	Power control unit (MC_PCU)
0x8FF0_0000	16	—	DP	PBRIDGE_1
0xFFF0_0000	16	—	LS	PBRIDGE_0, PBRIDGE_1
			DP	PBRIDGE_0
0xC3FE_4000	16	89	LS/DP	Reset generation module (MC_RGM)
0xC3FF_4000	16	93	LS/DP	Self-test control unit (STCU)
0x8FF2_4000	16	—	DP	Semaphores (SEMA4_1)
0xFFF2_4000	16	—	DP	Semaphores (SEMA4_0)
0x00F0_0000	16	—	LS/DP	Shadow block
0xFFE7_8000	16	62	LS/DP	Sine wave generator (SWG)
0x8FF3_8000	16	—	DP	SWT_1
0xFFF3_8000	16	—	LS	SWT_0, SWT_1
			DP	SWT_0
0x4000_0000	64	—	DP	SRAM
	128	—	LS	SRAM
0x5000_0000	64	—	DP	SRAM(3)
0x8FF3_C000	16	—	DP	STM_1
0xFFF3_C000	16	—	LS	STM_0, STM_1
			DP	STM_0
0xC3F9_0000	16	68	LS/DP	System integration unit lite (SIUL)
0xC3FD_8000	16	86	LS/DP	System status and configuration module (SSCM)
0xC3F9_4000	16	69	LS/DP	Wakeup unit (WKPU)
0x8FF0_4000	16	—	DP	XBAR_1
0xFFF0_4000	16	—	LS	XBAR_0, XBAR_1
			DP	XBAR_0

1. DMA_MUX_1 is disabled in DP mode.
2. Test flash memory is mapped to this address if the SCTR[TFE] bit in the SSCM is set (see *Section 49.3.1.8, SSCM Control Register (SCTR)*).

3. This range cannot be accessed by eDMA_0.

Table 5. SRAM map for SPC56ELxx in LSM

Start address	End address	Size (KB)	Region(1)
0x4000_0000	0x4001_FFFF	128	SRAM
0x4002_0000	0x5FFF_FFFF	524160	Reserved

1. Access to reserved memory space can cause unexpected behavior. The MPU must be configured to force a deterministic action for accesses to reserved memory space.

Table 6. SRAM map for SPC56ELxx in DPM

Start address	End address	Size (KB)	Region(1)
0x4000_0000	0x4000_FFFF	64	SRAM
0x4001_0000	0x4FFF_FFFF	262080	Reserved
0x5000_0000	0x5000_FFFF	64	SRAM(2)
0x5001_0000	0x5FFF_FFFF	262080	Reserved

1. Access to reserved memory space can cause unexpected behavior. The MPU must be configured to force a deterministic action for accesses to reserved memory space.
2. This range cannot be accessed by eDMA_0.

4 Signal Description

This chapter describes the signals of the SPC56ELxx.

4.1 Package pinouts

Figure 2 shows the LQFP100 pinout.

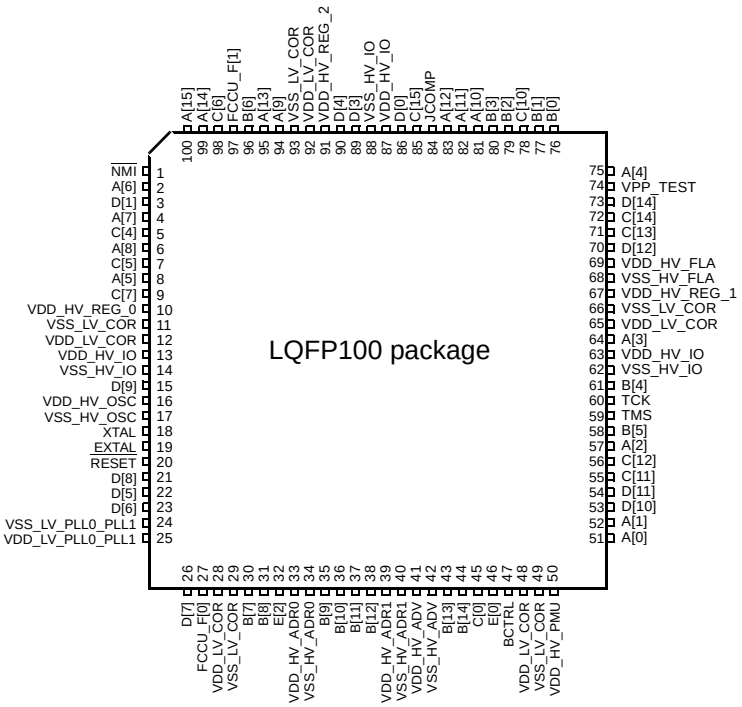


Figure 2. LQFP100 pinout

Figure 3 shows the LQFP144 pinout.

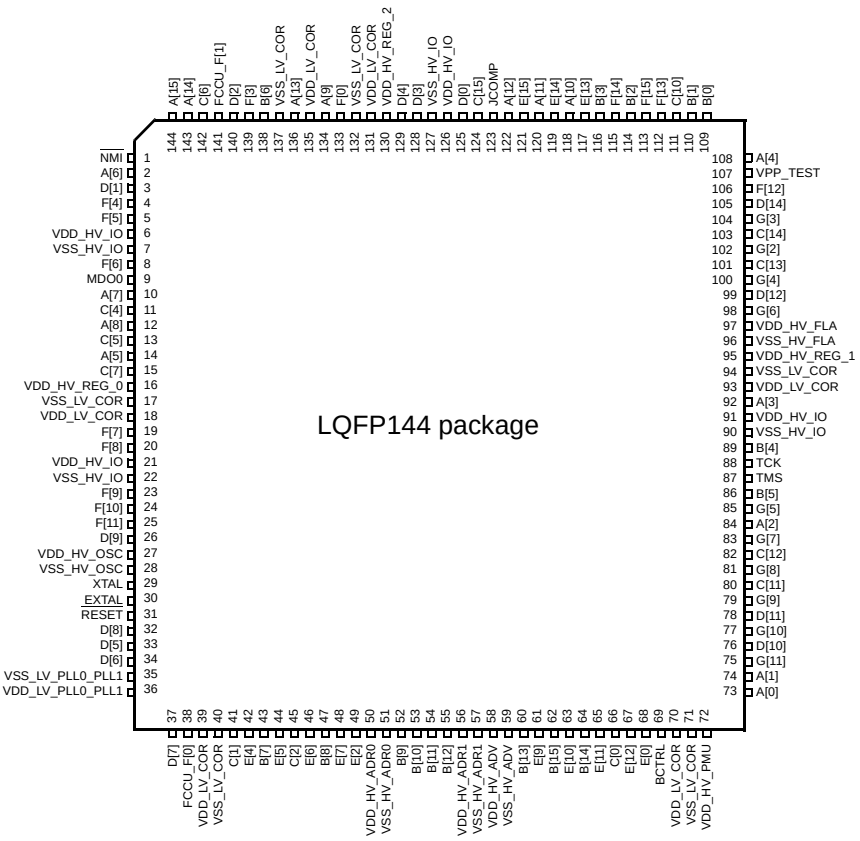


Figure 3. LQFP144 pinout

Figure 4 shows the LFBGA257 ballmap.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A	$V_{SS_HV_IO}$	$V_{SS_HV_IO}$	$V_{DD_HV_IO}$	H[2]	H[0]	G[14]	D[3]	C[15]	$V_{DD_HV_IO}$	A[12]	H[10]	H[14]	A[10]	B[2]	C[10]	$V_{SS_HV_IO}$	$V_{SS_HV_IO}$
B	$V_{SS_HV_IO}$	$V_{SS_HV_IO}$	B[6]	A[14]	F[3]	A[9]	D[4]	D[0]	$V_{SS_HV_IO}$	H[12]	E[15]	E[14]	B[3]	F[13]	B[0]	$V_{DD_HV_IO}$	$V_{SS_HV_IO}$
C	$V_{DD_HV_IO}$	NC(1)	$V_{SS_HV_IO}$	FCCU_F[1]	D[2]	A[13]	$V_{DD_HV_REG_2}$	$V_{DD_HV_REG_2}$	I[0]	JCOMP	H[11]	I[1]	F[14]	B[1]	$V_{SS_HV_IO}$	A[4]	F[12]
D	F[5]	F[4]	A[15]	C[6]	$V_{SS_LV_COR}$	$V_{DD_LV_COR}$	F[0]	$V_{DD_HV_IO}$	$V_{SS_HV_IO}$	NC	A[11]	E[13]	F[15]	$V_{DD_HV_IO}$	V_{PP_TEST}	D[14]	G[3]
E	MDO0	F[6]	D[1]	NMI										NC	C[14]	G[2]	I[3]
F	H[1]	G[12]	A[7]	A[8]		$V_{DD_LV_COR}$	$V_{DD_LV_COR}$	$V_{DD_LV_COR}$	$V_{DD_LV_COR}$	$V_{DD_LV_COR}$	$V_{DD_LV_COR}$	$V_{DD_LV_COR}$		NC	C[13]	I[2]	G[4]
G	H[3]	$V_{DD_HV_IO}$	C[5]	A[6]		$V_{DD_LV_COR}$	$V_{SS_LV_COR}$	$V_{SS_LV_COR}$	$V_{SS_LV_COR}$	$V_{SS_LV_COR}$	$V_{SS_LV_COR}$	$V_{DD_LV_COR}$		D[12]	H[13]	H[9]	G[6]
H	G[13]	$V_{SS_HV_IO}$	C[4]	A[5]		V_{DD_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{DD_LV}		V_{SS_LV}	$V_{DD_HV_REG_1}$	$V_{DD_HV_FLA}$	H[6]
J	F[7]	G[15]	$V_{DD_HV_REG_0}$	$V_{DD_HV_REG_0}$		V_{DD_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{DD_LV}		V_{DD_LV}	$V_{DD_HV_REG_1}$	$V_{SS_HV_FLA}$	H[15]
K	F[9]	F[8]		C[7]		V_{DD_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{DD_LV}		NC	H[8]	H[7]	A[3]
L	F[10]	F[11]	D[9]	NC		V_{DD_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{SS_LV}	V_{DD_LV}		NC	TCK	H[4]	B[4]
M	$V_{DD_HV_OSC}$	$V_{DD_HV_IO}$	D[8]	NC		V_{DD_LV}	V_{DD_LV}	V_{DD_LV}	V_{DD_LV}	V_{DD_LV}	V_{DD_LV}	V_{DD_LV}		C[11]	B[5]	TMS	H[5]
N	XTAL	$V_{SS_HV_IO}$	D[5]	$V_{SS_LV_PLL}$										NC	C[12]	A[2]	G[5]
P	$V_{SS_HV_OSC}$	$\overline{\text{RESET}}$	D[6]	$V_{DD_LV_PLL}$	$V_{DD_LV_COR}$	$V_{SS_LV_COR}$	B[8]	NC	$V_{SS_HV_IO}$	$V_{DD_HV_IO}$	B[14]	$V_{DD_LV_COR}$	$V_{SS_LV_COR}$	$V_{DD_HV_IO}$	G[10]	G[8]	G[7]
R	EXTAL	FCCU_F[0]	$V_{SS_HV_IO}$	D[7]	B[7]	E[6]	$V_{DD_HV_ADR0}$	B[10]	$V_{DD_HV_ADR1}$	B[13]	B[15]	C[0]	BCTRL	A[1]	$V_{SS_HV_IO}$	D[11]	G[9]
T	$V_{SS_HV_IO}$	$V_{DD_HV_IO}$	NC	C[1]	E[5]	E[7]	$V_{SS_HV_ADR0}$	B[11]	$V_{SS_HV_ADR1}$	E[9]	E[10]	E[12]	E[0]	A[0]	D[10]	$V_{DD_HV_IO}$	$V_{SS_HV_IO}$
U	$V_{SS_HV_IO}$	$V_{SS_HV_IO}$	NC	E[4]	C[2]	E[2]	B[9]	B[12]	$V_{DD_HV_ADV}$	$V_{SS_HV_ADV}$	E[11]	NC	NC	$V_{DD_HV_PMU}$	G[11]	$V_{SS_HV_IO}$	$V_{SS_HV_IO}$
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

1. NC = Not connected (the pin is physically not connected to anything on the device)

Figure 4. LFBGA257 ballmap

Table 7, *Table 8*, and *Table 9* provide the pin function summaries for the 100-pin, 144-pin, and 257-pin packages, respectively, listing all the signals multiplexed to each pin.

Table 7. LQFP100 pin function summary

Pin #	Port/function	Peripheral	Output function	Input function
1	NMI	—		
2	A[6]	SIUL	GPIO[6]	GPIO[6]
		DSPI_1	SCK	SCK
		SIUL	—	EIRQ[6]
3	D[1]	SIUL	GPIO[49]	GPIO[49]
		eTimer_1	ETC[2]	ETC[2]
		CTU_0	EXT_TGR	—
		FlexRay	—	CA_RX
4	A[7]	SIUL	GPIO[7]	GPIO[7]
		DSPI_1	SOUT	—
		SIUL	—	EIRQ[7]
5	C[4]	SIUL	GPIO[36]	GPIO[36]
		DSPI_0	CS0	CS0
		FlexPWM_0	X[1]	X[1]
		SSCM	DEBUG[4]	—
		SIUL	—	EIRQ[22]
6	A[8]	SIUL	GPIO[8]	GPIO[8]
		DSPI_1	—	SIN
		SIUL	—	EIRQ[8]
7	C[5]	SIUL	GPIO[37]	GPIO[37]
		DSPI_0	SCK	SCK
		SSCM	DEBUG[5]	—
		FlexPWM_0	—	FAULT[3]
		SIUL	—	EIRQ[23]
8	A[5]	SIUL	GPIO[5]	GPIO[5]
		DSPI_1	CS0	CS0
		eTimer_1	ETC[5]	ETC[5]
		DSPI_0	CS7	—
		SIUL	—	EIRQ[5]
9	C[7]	SIUL	GPIO[39]	GPIO[39]
		FlexPWM_0	A[1]	A[1]
		SSCM	DEBUG[7]	—
		DSPI_0	—	SIN

Table 7. LQFP100 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
10	$V_{DD_HV_REG_0}$	—		
11	$V_{SS_LV_COR}$	—		
12	$V_{DD_LV_COR}$	—		
13	$V_{DD_HV_IO}$	—		
14	$V_{SS_HV_IO}$	—		
15	D[9]	SIUL	GPIO[57]	GPIO[57]
		FlexPWM_0	X[0]	X[0]
		LINFlexD_1	TXD	—
16	$V_{DD_HV_OSC}$	—		
17	$V_{SS_HV_OSC}$	—		
18	XTAL	—		
19	EXTAL	—		
20	$\overline{\text{RESET}}$	—		
21	D[8]	SIUL	GPIO[56]	GPIO[56]
		DSPI_1	CS2	—
		eTimer_1	ETC[4]	ETC[4]
		DSPI_0	CS5	—
		FlexPWM_0	—	FAULT[3]
22	D[5]	SIUL	GPIO[53]	GPIO[53]
		DSPI_0	CS3	—
		FlexPWM_0	—	FAULT[2]
23	D[6]	SIUL	GPIO[54]	GPIO[54]
		DSPI_0	CS2	—
		FlexPWM_0	X[3]	X[3]
		FlexPWM_0	—	FAULT[1]
24	$V_{SS_LV_PLL0_PLL1}$	—		
25	$V_{DD_LV_PLL0_PLL1}$	—		
26	D[7]	SIUL	GPIO[55]	GPIO[55]
		DSPI_1	CS3	—
		DSPI_0	CS4	—
		SWG	Analog output	—
27	FCCU_F[0]	FCCU	F[0]	—
28	$V_{DD_LV_COR}$	—		
29	$V_{SS_LV_COR}$	—		

Table 7. LQFP100 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
30	B[7]	SIUL	—	GPIO[23]
		LINFlexD_0	—	RXD
		ADC_0	—	AN[0]
31	B[8]	SIUL	—	GPIO[24]
		eTimer_0	—	ETC[5]
		ADC_0	—	AN[1]
32	E[2]	SIUL	—	GPIO[66]
		ADC_0	—	AN[5]
33	$V_{DD_HV_ADR0}$	—		
34	$V_{SS_HV_ADR0}$	—		
35	B[9]	SIUL	—	GPIO[25]
		ADC_0	—	AN[11]
		ADC_1		
36	B[10]	SIUL	—	GPIO[26]
		ADC_0	—	AN[12]
		ADC_1		
37	B[11]	SIUL	—	GPIO[27]
		ADC_0	—	AN[13]
		ADC_1		
38	B[12]	SIUL	—	GPIO[28]
		ADC_0	—	AN[14]
		ADC_1		
39	$V_{DD_HV_ADR1}$	—		
40	$V_{SS_HV_ADR1}$	—		
41	$V_{DD_HV_ADV}$	—		
42	$V_{SS_HV_ADV}$	—		
43	B[13]	SIUL	—	GPIO[29]
		LINFlexD_1	—	RXD
		ADC_1	—	AN[0]
44	B[14]	SIUL	—	GPIO[30]
		eTimer_0	—	ETC[4]
		SIUL	—	EIRQ[19]
		ADC_1	—	AN[1]
45	C[0]	SIUL	—	GPIO[32]
		ADC_1	—	AN[3]

Table 7. LQFP100 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
46	E[0]	SIUL	—	GPIO[64]
		ADC_1	—	AN[5]
47	BCTRL	—		
48	$V_{DD_LV_COR}$	—		
49	$V_{SS_LV_COR}$	—		
50	$V_{DD_HV_PMU}$	—		
51	A[0]	SIUL	GPIO[0]	GPIO[0]
		eTimer_0	ETC[0]	ETC[0]
		DSPI_2	SCK	SCK
		SIUL	—	EIRQ[0]
52	A[1]	SIUL	GPIO[1]	GPIO[1]
		eTimer_0	ETC[1]	ETC[1]
		DSPI_2	SOUT	—
		SIUL	—	EIRQ[1]
53	D[10]	SIUL	GPIO[58]	GPIO[58]
		FlexPWM_0	A[0]	A[0]
		eTimer_0	—	ETC[0]
54	D[11]	SIUL	GPIO[59]	GPIO[59]
		FlexPWM_0	B[0]	B[0]
		eTimer_0	—	ETC[1]
55	C[11]	SIUL	GPIO[43]	GPIO[43]
		eTimer_0	ETC[4]	ETC[4]
		DSPI_2	CS2	—
56	C[12]	SIUL	GPIO[44]	GPIO[44]
		eTimer_0	ETC[5]	ETC[5]
		DSPI_2	CS3	—
57	A[2]	SIUL	GPIO[2]	GPIO[2]
		eTimer_0	ETC[2]	ETC[2]
		FlexPWM_0	A[3]	A[3]
		DSPI_2	—	SIN
		MC_RGM	—	ABS[0]
		SIUL	—	EIRQ[2]
58	B[5]	SIUL	GPIO[21]	GPIO[21]
		JTAGC	—	TDI
59	TMS	—		

Table 7. LQFP100 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
60	TCK	—		
61	B[4]	SIUL	GPIO[20]	GPIO[20]
		JTAGC	TDO	—
62	$V_{SS_HV_IO}$	—		
63	$V_{DD_HV_IO}$	—		
64	A[3]	SIUL	GPIO[3]	GPIO[3]
		eTimer_0	ETC[3]	ETC[3]
		DSPI_2	CS0	CS0
		FlexPWM_0	B[3]	B[3]
		MC_RGM	—	ABS[2]
		SIUL	—	EIRQ[3]
65	$V_{DD_LV_COR}$	—		
66	$V_{SS_LV_COR}$	—		
67	$V_{DD_HV_REG_1}$	—		
68	$V_{SS_HV_FLA}$	—		
69	$V_{DD_HV_FLA}$	—		
70	D[12]	SIUL	GPIO[60]	GPIO[60]
		FlexPWM_0	X[1]	X[1]
		LINFlexD_1	—	RXD
71	C[13]	SIUL	GPIO[45]	GPIO[45]
		eTimer_1	ETC[1]	ETC[1]
		CTU_0	—	EXT_IN
		FlexPWM_0	—	EXT_SYNC
72	C[14]	SIUL	GPIO[46]	GPIO[46]
		eTimer_1	ETC[2]	ETC[2]
		CTU_0	EXT_TGR	—
73	D[14]	SIUL	GPIO[62]	GPIO[62]
		FlexPWM_0	B[1]	B[1]
		eTimer_0	—	ETC[3]
74	V_{PP_TEST}(1)	—		

Table 7. LQFP100 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
75	A[4]	SIUL	GPIO[4]	GPIO[4]
		eTimer_1	ETC[0]	ETC[0]
		DSPI_2	CS1	—
		eTimer_0	ETC[4]	ETC[4]
		MC_RGM	—	FAB
		SIUL	—	EIRQ[4]
76	B[0]	SIUL	GPIO[16]	GPIO[16]
		FlexCAN_0	TXD	—
		eTimer_1	ETC[2]	ETC[2]
		SSCM	DEBUG[0]	—
		SIUL	—	EIRQ[15]
77	B[1]	SIUL	GPIO[17]	GPIO[17]
		eTimer_1	ETC[3]	ETC[3]
		SSCM	DEBUG[1]	—
		FlexCAN_0	—	RXD
		FlexCAN_1	—	RXD
		SIUL	—	EIRQ[16]
78	C[10]	SIUL	GPIO[42]	GPIO[42]
		DSPI_2	CS2	—
		FlexPWM_0	A[3]	A[3]
		FlexPWM_0	—	FAULT[1]
79	B[2]	SIUL	GPIO[18]	GPIO[18]
		LINFlexD_0	TXD	—
		SSCM	DEBUG[2]	DEBUG[2]
		SIUL	—	EIRQ[17]
80	B[3]	SIUL	GPIO[19]	GPIO[19]
		SSCM	DEBUG[3]	DEBUG[3]
		LINFlexD_0	—	RXD
81	A[10]	SIUL	GPIO[10]	GPIO[10]
		DSPI_2	CS0	CS0
		FlexPWM_0	B[0]	B[0]
		FlexPWM_0	X[2]	X[2]
		SIUL	—	EIRQ[9]

Table 7. LQFP100 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
82	A[11]	SIUL	GPIO[11]	GPIO[11]
		DSPI_2	SCK	SCK
		FlexPWM_0	A[0]	A[0]
		FlexPWM_0	A[2]	A[2]
		SIUL	—	EIRQ[10]
83	A[12]	SIUL	GPIO[12]	GPIO[12]
		DSPI_2	SOUT	—
		FlexPWM_0	A[2]	A[2]
		FlexPWM_0	B[2]	B[2]
		SIUL	—	EIRQ[11]
84	JCOMP	—	—	JCOMP
85	C[15]	SIUL	GPIO[47]	GPIO[47]
		FlexRay	CA_TR_EN	—
		eTimer_1	ETC[0]	ETC[0]
		FlexPWM_0	A[1]	A[1]
		CTU_0	—	EXT_IN
		FlexPWM_0	—	EXT_SYNC
86	D[0]	SIUL	GPIO[48]	GPIO[48]
		FlexRay	CA_TX	—
		eTimer_1	ETC[1]	ETC[1]
		FlexPWM_0	B[1]	B[1]
87	$V_{DD_HV_IO}$	—		
88	$V_{SS_HV_IO}$	—		
89	D[3]	SIUL	GPIO[51]	GPIO[51]
		FlexRay	CB_TX	—
		eTimer_1	ETC[4]	ETC[4]
		FlexPWM_0	A[3]	A[3]
90	D[4]	SIUL	GPIO[52]	GPIO[52]
		FlexRay	CB_TR_EN	—
		eTimer_1	ETC[5]	ETC[5]
		FlexPWM_0	B[3]	B[3]
91	$V_{DD_HV_REG_2}$	—		
92	$V_{DD_LV_COR}$	—		
93	$V_{SS_LV_COR}$	—		

Table 7. LQFP100 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
94	A[9]	SIUL	GPIO[9]	GPIO[9]
		DSPI_2	CS1	—
		FlexPWM_0	B[3]	B[3]
		FlexPWM_0	—	FAULT[0]
95	A[13]	SIUL	GPIO[13]	GPIO[13]
		FlexPWM_0	B[2]	B[2]
		DSPI_2	—	SIN
		FlexPWM_0	—	FAULT[0]
		SIUL	—	EIRQ[12]
96	B[6]	SIUL	GPIO[22]	GPIO[22]
		MC_CGM	clk_out	—
		DSPI_2	CS2	—
		SIUL	—	EIRQ[18]
97	FCCU_F[1]	FCCU	F[1]	—
98	C[6]	SIUL	GPIO[38]	GPIO[38]
		DSPI_0	SOUT	—
		FlexPWM_0	B[1]	B[1]
		SSCM	DEBUG[6]	—
		SIUL	—	EIRQ[24]
99	A[14]	SIUL	GPIO[14]	GPIO[14]
		FlexCAN_1	TXD	—
		eTimer_1	ETC[4]	ETC[4]
		SIUL	—	EIRQ[13]
100	A[15]	SIUL	GPIO[15]	GPIO[15]
		eTimer_1	ETC[5]	ETC[5]
		FlexCAN_1	—	RXD
		FlexCAN_0	—	RXD
		SIUL	—	EIRQ[14]

1. V_{PP_TEST} should always be tied to ground (V_{SS}) for normal operations.

Table 8. LQFP144 pin function summary

Pin #	Port/function	Peripheral	Output function	Input function
1	$\overline{NMI}$	—		
2	A[6]	SIUL	GPIO[6]	GPIO[6]
		DSPI_1	SCK	SCK
		SIUL	—	EIRQ[6]
3	D[1]	SIUL	GPIO[49]	GPIO[49]
		eTimer_1	ETC[2]	ETC[2]
		CTU_0	EXT_TGR	—
		FlexRay	—	CA_RX
4	F[4]	SIUL	GPIO[84]	GPIO[84]
		NPC	MDO[3]	—
5	F[5]	SIUL	GPIO[85]	GPIO[85]
		NPC	MDO[2]	—
6	$V_{DD_HV_IO}$	—		
7	$V_{SS_HV_IO}$	—		
8	F[6]	SIUL	GPIO[86]	GPIO[86]
		NPC	MDO[1]	—
9	MDO0	—		
10	A[7]	SIUL	GPIO[7]	GPIO[7]
		DSPI_1	SOUT	—
		SIUL	—	EIRQ[7]
11	C[4]	SIUL	GPIO[36]	GPIO[36]
		DSPI_0	CS0	CS0
		FlexPWM_0	X[1]	X[1]
		SSCM	DEBUG[4]	—
		SIUL	—	EIRQ[22]
12	A[8]	SIUL	GPIO[8]	GPIO[8]
		DSPI_1	—	SIN
		SIUL	—	EIRQ[8]
13	C[5]	SIUL	GPIO[37]	GPIO[37]
		DSPI_0	SCK	SCK
		SSCM	DEBUG[5]	—
		FlexPWM_0	—	FAULT[3]
		SIUL	—	EIRQ[23]

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
14	A[5]	SIUL	GPIO[5]	GPIO[5]
		DSPI_1	CS0	CS0
		eTimer_1	ETC[5]	ETC[5]
		DSPI_0	CS7	—
		SIUL	—	EIRQ[5]
15	C[7]	SIUL	GPIO[39]	GPIO[39]
		FlexPWM_0	A[1]	A[1]
		SSCM	DEBUG[7]	—
		DSPI_0	—	SIN
16	$V_{DD_HV_REG_0}$	—		
17	$V_{SS_LV_COR}$	—		
18	$V_{DD_LV_COR}$	—		
19	F[7]	SIUL	GPIO[87]	GPIO[87]
		NPC	MCKO	—
20	F[8]	SIUL	GPIO[88]	GPIO[88]
		NPC	$\overline{\text{MSEO[1]}}$	—
21	$V_{DD_HV_IO}$	—		
22	$V_{SS_HV_IO}$	—		
23	F[9]	SIUL	GPIO[89]	GPIO[89]
		NPC	$\overline{\text{MSEO[0]}}$	—
24	F[10]	SIUL	GPIO[90]	GPIO[90]
		NPC	$\overline{\text{EVTO}}$	—
25	F[11]	SIUL	GPIO[91]	GPIO[91]
		NPC	—	$\overline{\text{EVTI}}$
26	D[9]	SIUL	GPIO[57]	GPIO[57]
		FlexPWM_0	X[0]	X[0]
		LINFlexD_1	TXD	—
27	$V_{DD_HV_OSC}$	—		
28	$V_{SS_HV_OSC}$	—		
29	XTAL	—		
30	EXTAL	—		
31	$\overline{\text{RESET}}$	—		

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
32	D[8]	SIUL	GPIO[56]	GPIO[56]
		DSPI_1	CS2	—
		eTimer_1	ETC[4]	ETC[4]
		DSPI_0	CS5	—
		FlexPWM_0	—	FAULT[3]
33	D[5]	SIUL	GPIO[53]	GPIO[53]
		DSPI_0	CS3	—
		FlexPWM_0	—	FAULT[2]
34	D[6]	SIUL	GPIO[54]	GPIO[54]
		DSPI_0	CS2	—
		FlexPWM_0	X[3]	X[3]
		FlexPWM_0	—	FAULT[1]
35	$V_{SS_LV_PLL0_PLL1}$	—		
36	$V_{DD_LV_PLL0_PLL1}$	—		
37	D[7]	SIUL	GPIO[55]	GPIO[55]
		DSPI_1	CS3	—
		DSPI_0	CS4	—
		SWG	analog output	—
38	FCCU_F[0]	FCCU	F[0]	—
39	$V_{DD_LV_COR}$	—		
40	$V_{SS_LV_COR}$	—		
41	C[1]	SIUL	—	GPIO[33]
		ADC_0	—	AN[2]
42	E[4]	SIUL	—	GPIO[68]
		ADC_0	—	AN[7]
43	B[7]	SIUL	—	GPIO[23]
		LINFlexD_0	—	RXD
		ADC_0	—	AN[0]
44	E[5]	SIUL	—	GPIO[69]
		ADC_0	—	AN[8]
45	C[2]	SIUL	—	GPIO[34]
		ADC_0	—	AN[3]
46	E[6]	SIUL	—	GPIO[70]
		ADC_0	—	AN[4]

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
47	B[8]	SIUL	—	GPIO[24]
		eTimer_0	—	ETC[5]
		ADC_0	—	AN[1]
48	E[7]	SIUL	—	GPIO[71]
		ADC_0	—	AN[6]
49	E[2]	SIUL	—	GPIO[66]
		ADC_0	—	AN[5]
50	$V_{DD_HV_ADR0}$	—		
51	$V_{SS_HV_ADR0}$	—		
52	B[9]	SIUL	—	GPIO[25]
		ADC_0 ADC_1	—	AN[11]
53	B[10]	SIUL	—	GPIO[26]
		ADC_0 ADC_1	—	AN[12]
54	B[11]	SIUL	—	GPIO[27]
		ADC_0 ADC_1	—	AN[13]
55	B[12]	SIUL	—	GPIO[28]
		ADC_0 ADC_1	—	AN[14]
56	$V_{DD_HV_ADR1}$	—		
57	$V_{SS_HV_ADR1}$	—		
58	$V_{DD_HV_ADV}$	—		
59	$V_{SS_HV_ADV}$	—		
60	B[13]	SIUL	—	GPIO[29]
		LINFlexD_1	—	RXD
		ADC_1	—	AN[0]
61	E[9]	SIUL	—	GPIO[73]
		ADC_1	—	AN[7]
62	B[15]	SIUL	—	GPIO[31]
		SIUL	—	EIRQ[20]
		ADC_1	—	AN[2]
63	E[10]	SIUL	—	GPIO[74]
		ADC_1	—	AN[8]

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
64	B[14]	SIUL	—	GPIO[30]
		eTimer_0	—	ETC[4]
		SIUL	—	EIRQ[19]
		ADC_1	—	AN[1]
65	E[11]	SIUL	—	GPIO[75]
		ADC_1	—	AN[4]
66	C[0]	SIUL	—	GPIO[32]
		ADC_1	—	AN[3]
67	E[12]	SIUL	—	GPIO[76]
		ADC_1	—	AN[6]
68	E[0]	SIUL	—	GPIO[64]
		ADC_1	—	AN[5]
69	BCTRL	—		
70	$V_{DD_LV_COR}$	—		
71	$V_{SS_LV_COR}$	—		
72	$V_{DD_HV_PMU}$	—		
73	A[0]	SIUL	GPIO[0]	GPIO[0]
		eTimer_0	ETC[0]	ETC[0]
		DSPI_2	SCK	SCK
		SIUL	—	EIRQ[0]
74	A[1]	SIUL	GPIO[1]	GPIO[1]
		eTimer_0	ETC[1]	ETC[1]
		DSPI_2	SOUT	—
		SIUL	—	EIRQ[1]
75	G[11]	SIUL	GPIO[107]	GPIO[107]
		FlexRay	DBG3	—
		FlexPWM_0	—	FAULT[3]
76	D[10]	SIUL	GPIO[58]	GPIO[58]
		FlexPWM_0	A[0]	A[0]
		eTimer_0	—	ETC[0]
77	G[10]	SIUL	GPIO[106]	GPIO[106]
		FlexRay	DBG2	—
		DSPI_2	CS3	—
		FlexPWM_0	—	FAULT[2]

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
78	D[11]	SIUL	GPIO[59]	GPIO[59]
		FlexPWM_0	B[0]	B[0]
		eTimer_0	—	ETC[1]
79	G[9]	SIUL	GPIO[105]	GPIO[105]
		FlexRay	DBG1	—
		DSPI_1	CS1	—
		FlexPWM_0	—	FAULT[1]
		SIUL	—	EIRQ[29]
80	C[11]	SIUL	GPIO[43]	GPIO[43]
		eTimer_0	ETC[4]	ETC[4]
		DSPI_2	CS2	—
81	G[8]	SIUL	GPIO[104]	GPIO[104]
		FlexRay	DBG0	—
		DSPI_0	CS1	—
		FlexPWM_0	—	FAULT[0]
		SIUL	—	EIRQ[21]
82	C[12]	SIUL	GPIO[44]	GPIO[44]
		eTimer_0	ETC[5]	ETC[5]
		DSPI_2	CS3	—
83	G[7]	SIUL	GPIO[103]	GPIO[103]
		FlexPWM_0	B[3]	B[3]
84	A[2]	SIUL	GPIO[2]	GPIO[2]
		eTimer_0	ETC[2]	ETC[2]
		FlexPWM_0	A[3]	A[3]
		DSPI_2	—	SIN
		MC_RGM	—	ABS[0]
		SIUL	—	EIRQ[2]
85	G[5]	SIUL	GPIO[101]	GPIO[101]
		FlexPWM_0	X[3]	X[3]
		DSPI_2	CS3	—
86	B[5]	SIUL	GPIO[21]	GPIO[21]
		JTAGC	—	TDI
87	TMS	—		
88	TCK	—		

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
89	B[4]	SIUL	GPIO[20]	GPIO[20]
		JTAGC	TDO	—
90	$V_{SS_HV_IO}$	—		
91	$V_{DD_HV_IO}$	—		
92	A[3]	SIUL	GPIO[3]	GPIO[3]
		eTimer_0	ETC[3]	ETC[3]
		DSPI_2	CS0	CS0
		FlexPWM_0	B[3]	B[3]
		MC_RGM	—	ABS[2]
		SIUL	—	EIRQ[3]
93	$V_{DD_LV_COR}$	—		
94	$V_{SS_LV_COR}$	—		
95	$V_{DD_HV_REG_1}$	—		
96	$V_{SS_HV_FLA}$	—		
97	$V_{DD_HV_FLA}$	—		
98	G[6]	SIUL	GPIO[102]	GPIO[102]
		FlexPWM_0	A[3]	A[3]
99	D[12]	SIUL	GPIO[60]	GPIO[60]
		FlexPWM_0	X[1]	X[1]
		LINFlexD_1	—	RXD
100	G[4]	SIUL	GPIO[100]	GPIO[100]
		FlexPWM_0	B[2]	B[2]
		eTimer_0	—	ETC[5]
101	C[13]	SIUL	GPIO[45]	GPIO[45]
		eTimer_1	ETC[1]	ETC[1]
		CTU_0	—	EXT_IN
		FlexPWM_0	—	EXT_SYNC
102	G[2]	SIUL	GPIO[98]	GPIO[98]
		FlexPWM_0	X[2]	X[2]
		DSPI_1	CS1	—
103	C[14]	SIUL	GPIO[46]	GPIO[46]
		eTimer_1	ETC[2]	ETC[2]
		CTU_0	EXT_TGR	—

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
104	G[3]	SIUL	GPIO[99]	GPIO[99]
		FlexPWM_0	A[2]	A[2]
		eTimer_0	—	ETC[4]
105	D[14]	SIUL	GPIO[62]	GPIO[62]
		FlexPWM_0	B[1]	B[1]
		eTimer_0	—	ETC[3]
106	F[12]	SIUL	GPIO[92]	GPIO[92]
		eTimer_1	ETC[3]	ETC[3]
		SIUL	—	EIRQ[30]
107	V_{PP_TEST}(1)	—		
108	A[4]	SIUL	GPIO[4]	GPIO[4]
		eTimer_1	ETC[0]	ETC[0]
		DSPI_2	CS1	—
		eTimer_0	ETC[4]	ETC[4]
		MC_RGM	—	FAB
		SIUL	—	EIRQ[4]
109	B[0]	SIUL	GPIO[16]	GPIO[16]
		FlexCAN_0	TXD	—
		eTimer_1	ETC[2]	ETC[2]
		SSCM	DEBUG[0]	—
		SIUL	—	EIRQ[15]
110	B[1]	SIUL	GPIO[17]	GPIO[17]
		eTimer_1	ETC[3]	ETC[3]
		SSCM	DEBUG[1]	—
		FlexCAN_0	—	RXD
		FlexCAN_1	—	RXD
		SIUL	—	EIRQ[16]
111	C[10]	SIUL	GPIO[42]	GPIO[42]
		DSPI_2	CS2	—
		FlexPWM_0	A[3]	A[3]
		FlexPWM_0	—	FAULT[1]
112	F[13]	SIUL	GPIO[93]	GPIO[93]
		eTimer_1	ETC[4]	ETC[4]
		SIUL	—	EIRQ[31]

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
113	F[15]	SIUL	GPIO[95]	GPIO[95]
		LINFlexD_1	—	RXD
114	B[2]	SIUL	GPIO[18]	GPIO[18]
		LINFlexD_0	TXD	—
		SSCM	DEBUG[2]	—
		SIUL	—	EIRQ[17]
115	F[14]	SIUL	GPIO[94]	GPIO[94]
		LINFlexD_1	TXD	—
116	B[3]	SIUL	GPIO[19]	GPIO[19]
		SSCM	DEBUG[3]	—
		LINFlexD_0	—	RXD
117	E[13]	SIUL	GPIO[77]	GPIO[77]
		eTimer_0	ETC[5]	ETC[5]
		DSPI_2	CS3	—
		SIUL	—	EIRQ[25]
118	A[10]	SIUL	GPIO[10]	GPIO[10]
		DSPI_2	CS0	CS0
		FlexPWM_0	B[0]	B[0]
		FlexPWM_0	X[2]	X[2]
		SIUL	—	EIRQ[9]
119	E[14]	SIUL	GPIO[78]	GPIO[78]
		eTimer_1	ETC[5]	ETC[5]
		SIUL	—	EIRQ[26]
120	A[11]	SIUL	GPIO[11]	GPIO[11]
		DSPI_2	SCK	SCK
		FlexPWM_0	A[0]	A[0]
		FlexPWM_0	A[2]	A[2]
		SIUL	—	EIRQ[10]
121	E[15]	SIUL	GPIO[79]	GPIO[79]
		DSPI_0	CS1	—
		SIUL	—	EIRQ[27]

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
122	A[12]	SIUL	GPIO[12]	GPIO[12]
		DSPI_2	SOUT	—
		FlexPWM_0	A[2]	A[2]
		FlexPWM_0	B[2]	B[2]
		SIUL	—	EIRQ[11]
123	JCOMP	—	—	JCOMP
124	C[15]	SIUL	GPIO[47]	GPIO[47]
		FlexRay	CA_TR_EN	—
		eTimer_1	ETC[0]	ETC[0]
		FlexPWM_0	A[1]	A[1]
		CTU_0	—	EXT_IN
		FlexPWM_0	—	EXT_SYNC
125	D[0]	SIUL	GPIO[48]	GPIO[48]
		FlexRay	CA_TX	—
		eTimer_1	ETC[1]	ETC[1]
		FlexPWM_0	B[1]	B[1]
126	$V_{DD_HV_IO}$	—		
127	$V_{SS_HV_IO}$	—		
128	D[3]	SIUL	GPIO[51]	GPIO[51]
		FlexRay	CB_TX	—
		eTimer_1	ETC[4]	ETC[4]
		FlexPWM_0	A[3]	A[3]
129	D[4]	SIUL	GPIO[52]	GPIO[52]
		FlexRay	CB_TR_EN	—
		eTimer_1	ETC[5]	ETC[5]
		FlexPWM_0	B[3]	B[3]
130	$V_{DD_HV_REG_2}$	—		
131	$V_{DD_LV_COR}$	—		
132	$V_{SS_LV_COR}$	—		
133	F[0]	SIUL	GPIO[80]	GPIO[80]
		FlexPWM_0	A[1]	A[1]
		eTimer_0	—	ETC[2]
		SIUL	—	EIRQ[28]

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
134	A[9]	SIUL	GPIO[9]	GPIO[9]
		DSPI_2	CS1	—
		FlexPWM_0	B[3]	B[3]
		FlexPWM_0	—	FAULT[0]
135	$V_{DD_LV_COR}$	—		
136	A[13]	SIUL	GPIO[13]	GPIO[13]
		FlexPWM_0	B[2]	B[2]
		DSPI_2	—	SIN
		FlexPWM_0	—	FAULT[0]
		SIUL	—	EIRQ[12]
137	$V_{SS_LV_COR}$	—		
138	B[6]	SIUL	GPIO[22]	GPIO[22]
		MC_CGM	clk_out	—
		DSPI_2	CS2	—
		SIUL	—	EIRQ[18]

Table 8. LQFP144 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
139	F[3]	SIUL	GPIO[83]	GPIO[83]
		DSPI_0	CS6	—
140	D[2]	SIUL	GPIO[50]	GPIO[50]
		eTimer_1	ETC[3]	ETC[3]
		FlexPWM_0	X[3]	X[3]
		FlexRay	—	CB_RX
141	FCCU_F[1]	FCCU	F[1]	—
142	C[6]	SIUL	GPIO[38]	GPIO[38]
		DSPI_0	SOUT	—
		FlexPWM_0	B[1]	B[1]
		SSCM	DEBUG[6]	—
		SIUL	—	EIRQ[24]
143	A[14]	SIUL	GPIO[14]	GPIO[14]
		FlexCAN_1	TXD	—
		eTimer_1	ETC[4]	ETC[4]
		SIUL	—	EIRQ[13]
144	A[15]	SIUL	GPIO[15]	GPIO[15]
		eTimer_1	ETC[5]	ETC[5]
		FlexCAN_1	—	RXD
		FlexCAN_0	—	RXD
		SIUL	—	EIRQ[14]

1. V_{PP_TEST} should always be tied to ground (V_{SS}) for normal operations.

Table 9. LFBGA257 pin function summary

Pin #	Port/function	Peripheral	Output function	Input function
A1	$V_{SS_HV_IO_RING}$	—		
A2	$V_{SS_HV_IO_RING}$	—		
A3	$V_{DD_HV_IO_RING}$	—		
A4	H[2]	SIUL	GPIO[114]	GPIO[114]
		NPC	MDO[5]	—
A5	H[0]	SIUL	GPIO[112]	GPIO[112]
		NPC	MDO[7]	—
A6	G[14]	SIUL	GPIO[110]	GPIO[110]
		NPC	MDO[9]	—

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
A7	D[3]	SIUL	GPIO[51]	GPIO[51]
		FlexRay	CB_TX	—
		eTimer_1	ETC[4]	ETC[4]
		FlexPWM_0	A[3]	A[3]
A8	C[15]	SIUL	GPIO[47]	GPIO[47]
		FlexRay	CA_TR_EN	—
		eTimer_1	ETC[0]	ETC[0]
		FlexPWM_0	A[1]	A[1]
		CTU_0	—	EXT_IN
		FlexPWM_0	—	EXT_SYNC
A9	$V_{DD_HV_IO_RING}$	—		
A10	A[12]	SIUL	GPIO[12]	GPIO[12]
		DSPI_2	SOUT	—
		FlexPWM_0	A[2]	A[2]
		FlexPWM_0	B[2]	B[2]
		SIUL	—	EIRQ[11]
A11	H[10]	SIUL	GPIO[122]	GPIO[122]
		FlexPWM_1	X[2]	X[2]
		eTimer_2	ETC[2]	ETC[2]
A12	H[14]	SIUL	GPIO[126]	GPIO[126]
		FlexPWM_1	A[3]	A[3]
		eTimer_2	ETC[4]	ETC[4]
A13	A[10]	SIUL	GPIO[10]	GPIO[10]
		DSPI_2	CS0	CS0
		FlexPWM_0	B[0]	B[0]
		FlexPWM_0	X[2]	X[2]
		SIUL	—	EIRQ[9]
A14	B[2]	SIUL	GPIO[18]	GPIO[18]
		LINFlexD_0	TXD	—
		SSCM	DEBUG[2]	—
		SIUL	—	EIRQ[17]
A15	C[10]	SIUL	GPIO[42]	GPIO[42]
		DSPI_2	CS2	—
		FlexPWM_0	A[3]	A[3]
		FlexPWM_0	—	FAULT[1]

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
A16	$V_{SS_HV_IO_RING}$	—		
A17	$V_{SS_HV_IO_RING}$	—		
B1	$V_{SS_HV_IO_RING}$	—		
B2	$V_{SS_HV_IO_RING}$	—		
B3	B[6]	SIUL	GPIO[22]	GPIO[22]
		MC_CGM	clk_out	—
		DSPI_2	CS2	—
		SIUL	—	EIRQ[18]
B4	A[14]	SIUL	GPIO[14]	GPIO[14]
		FlexCAN_1	TXD	—
		eTimer_1	ETC[4]	ETC[4]
		SIUL	—	EIRQ[13]
B5	F[3]	SIUL	GPIO[83]	GPIO[83]
		DSPI_0	CS6	—
B6	A[9]	SIUL	GPIO[9]	GPIO[9]
		DSPI_2	CS1	—
		FlexPWM_0	B[3]	B[3]
		FlexPWM_0	—	FAULT[0]
B7	D[4]	SIUL	GPIO[52]	GPIO[52]
		FlexRay	CB_TR_EN	—
		eTimer_1	ETC[5]	ETC[5]
		FlexPWM_0	B[3]	B[3]
B8	D[0]	SIUL	GPIO[48]	GPIO[48]
		FlexRay	CA_TX	—
		eTimer_1	ETC[1]	ETC[1]
		FlexPWM_0	B[1]	B[1]
B9	$V_{SS_HV_IO_RING}$	—		
B10	H[12]	SIUL	GPIO[124]	GPIO[124]
		FlexPWM_1	B[2]	B[2]
B11	E[15]	SIUL	GPIO[79]	GPIO[79]
		DSPI_0	CS1	—
		SIUL	—	EIRQ[27]
B12	E[14]	SIUL	GPIO[78]	GPIO[78]
		eTimer_1	ETC[5]	ETC[5]
		SIUL	—	EIRQ[26]

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
B13	B[3]	SIUL	GPIO[19]	GPIO[19]
		SSCM	DEBUG[3]	—
		LINFlexD_0	—	RXD
B14	F[13]	SIUL	GPIO[93]	GPIO[93]
		eTimer_1	ETC[4]	ETC[4]
		SIUL	—	EIRQ[31]
B15	B[0]	SIUL	GPIO[16]	GPIO[16]
		FlexCAN_0	TXD	—
		eTimer_1	ETC[2]	ETC[2]
		SSCM	DEBUG[0]	—
		SIUL	—	EIRQ[15]
B16	$V_{DD_HV_IO_RING}$	—		
B17	$V_{SS_HV_IO_RING}$	—		
C1	$V_{DD_HV_IO_RING}$	—		
C2	Not connected	—		
C3	$V_{SS_HV_IO_RING}$	—		
C4	FCCU_F[1]	FCCU	F[1]	—
C5	D[2]	SIUL	GPIO[50]	GPIO[50]
		eTimer_1	ETC[3]	ETC[3]
		FlexPWM_0	X[3]	X[3]
		FlexRay	—	CB_RX
C6	A[13]	SIUL	GPIO[13]	GPIO[13]
		FlexPWM_0	B[2]	B[2]
		DSPI_2	—	SIN
		FlexPWM_0	—	FAULT[0]
		SIUL	—	EIRQ[12]
C7	$V_{DD_HV_REG_2}$	—		
C8	$V_{DD_HV_REG_2}$	—		
C9	I[0]	SIUL	GPIO[128]	GPIO[128]
		eTimer_2	ETC[0]	ETC[0]
		DSPI_0	CS4	—
		FlexPWM_1	—	FAULT[0]
C10	JCOMP	—	—	JCOMP
C11	H[11]	SIUL	GPIO[123]	GPIO[123]
		FlexPWM_1	A[2]	A[2]

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
C12	I[1]	SIUL	GPIO[129]	GPIO[129]
		eTimer_2	ETC[1]	ETC[1]
		DSPI_0	CS5	—
		FlexPWM_1	—	FAULT[1]
C13	F[14]	SIUL	GPIO[94]	GPIO[94]
		LINFlexD_1	TXD	—
C14	B[1]	SIUL	GPIO[17]	GPIO[17]
		eTimer_1	ETC[3]	ETC[3]
		SSCM	DEBUG[1]	—
		FlexCAN_0	—	RXD
		FlexCAN_1	—	RXD
		SIUL	—	EIRQ[16]
C15	$V_{SS_HV_IO_RING}$	—		
C16	A[4]	SIUL	GPIO[4]	GPIO[4]
		eTimer_1	ETC[0]	ETC[0]
		DSPI_2	CS1	—
		eTimer_0	ETC[4]	ETC[4]
		MC_RGM	—	FAB
		SIUL	—	EIRQ[4]
C17	F[12]	SIUL	GPIO[92]	GPIO[92]
		eTimer_1	ETC[3]	ETC[3]
		SIUL	—	EIRQ[30]
D1	F[5]	SIUL	GPIO[85]	GPIO[85]
		NPC	MDO[2]	—
D2	F[4]	SIUL	GPIO[84]	GPIO[84]
		NPC	MDO[3]	—
D3	A[15]	SIUL	GPIO[15]	GPIO[15]
		eTimer_1	ETC[5]	ETC[5]
		FlexCAN_1	—	RXD
		FlexCAN_0	—	RXD
		SIUL	—	EIRQ[14]

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
D4	C[6]	SIUL	GPIO[38]	GPIO[38]
		DSPI_0	SOUT	—
		FlexPWM_0	B[1]	B[1]
		SSCM	DEBUG[6]	—
		SIUL	—	EIRQ[24]
D5	$V_{SS_LV_CORE_RING}$	—		
D6	$V_{DD_LV_CORE_RING}$	—		
D7	F[0]	SIUL	GPIO[80]	GPIO[80]
		FlexPWM_0	A[1]	A[1]
		eTimer_0	—	ETC[2]
		SIUL	—	EIRQ[28]
D8	$V_{DD_HV_IO_RING}$	—		
D9	$V_{SS_HV_IO_RING}$	—		
D10	Not connected	—		
D11	A[11]	SIUL	GPIO[11]	GPIO[11]
		DSPI_2	SCK	SCK
		FlexPWM_0	A[0]	A[0]
		FlexPWM_0	A[2]	A[2]
		SIUL	—	EIRQ[10]
D12	E[13]	SIUL	GPIO[77]	GPIO[77]
		eTimer_0	ETC[5]	ETC[5]
		DSPI_2	CS3	—
		SIUL	—	EIRQ[25]
D13	F[15]	SIUL	GPIO[95]	GPIO[95]
		LINFlexD_1	—	RXD
D14	$V_{DD_HV_IO_RING}$	—		
D15	V_{PP_TEST}(1)	—		
D16	D[14]	SIUL	GPIO[62]	GPIO[62]
		FlexPWM_0	B[1]	B[1]
		eTimer_0	—	ETC[3]
D17	G[3]	SIUL	GPIO[99]	GPIO[99]
		FlexPWM_0	A[2]	A[2]
		eTimer_0	—	ETC[4]
E1	MDO0	—		

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
E2	F[6]	SIUL	GPIO[86]	GPIO[86]
		NPC	MDO[1]	—
E3	D[1]	SIUL	GPIO[49]	GPIO[49]
		eTimer_1	ETC[2]	ETC[2]
		CTU_0	EXT_TGR	—
		FlexRay	—	CA_RX
E4	$\overline{\text{NMI}}$	—		
E14	Not connected	—		
E15	C[14]	SIUL	GPIO[46]	GPIO[46]
		eTimer_1	ETC[2]	ETC[2]
		CTU_0	EXT_TGR	—
E16	G[2]	SIUL	GPIO[98]	GPIO[98]
		FlexPWM_0	X[2]	X[2]
		DSPI_1	CS1	—
E17	I[3]	SIUL	GPIO[131]	GPIO[131]
		eTimer_2	ETC[3]	ETC[3]
		DSPI_0	CS7	—
		CTU_0	EXT_TGR	—
		FlexPWM_1	—	FAULT[3]
F1	H[1]	SIUL	GPIO[113]	GPIO[113]
		NPC	MDO[6]	—
F2	G[12]	SIUL	GPIO[108]	GPIO[108]
		NPC	MDO[11]	—
F3	A[7]	SIUL	GPIO[7]	GPIO[7]
		DSPI_1	SOUT	—
		SIUL	—	EIRQ[7]
F4	A[8]	SIUL	GPIO[8]	GPIO[8]
		DSPI_1	—	SIN
		SIUL	—	EIRQ[8]
F6	$V_{DD_LV_CORE_RING}$	—		
F7	$V_{DD_LV_CORE_RING}$	—		
F8	$V_{DD_LV_CORE_RING}$	—		
F9	$V_{DD_LV_CORE_RING}$	—		
F10	$V_{DD_LV_CORE_RING}$	—		
F11	$V_{DD_LV_CORE_RING}$	—		

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
F12	$V_{DD_LV_CORE_RING}$	—		
F14	Not connected	—		
F15	C[13]	SIUL	GPIO[45]	GPIO[45]
		eTimer_1	ETC[1]	ETC[1]
		CTU_0	—	EXT_IN
		FlexPWM_0	—	EXT_SYNC
F16	I[2]	SIUL	GPIO[130]	GPIO[130]
		eTimer_2	ETC[2]	ETC[2]
		DSPI_0	CS6	—
		FlexPWM_1	—	FAULT[2]
F17	G[4]	SIUL	GPIO[100]	GPIO[100]
		FlexPWM_0	B[2]	B[2]
		eTimer_0	—	ETC[5]
G1	H[3]	SIUL	GPIO[115]	GPIO[115]
		NPC	MDO[4]	—
G2	$V_{DD_HV_IO_RING}$	—		
G3	C[5]	SIUL	GPIO[37]	GPIO[37]
		DSPI_0	SCK	SCK
		SSCM	DEBUG[5]	—
		FlexPWM_0	—	FAULT[3]
		SIUL	—	EIRQ[23]
G4	A[6]	SIUL	GPIO[6]	GPIO[6]
		DSPI_1	SCK	SCK
		SIUL	—	EIRQ[6]
G6	$V_{DD_LV_CORE_RING}$	—		
G7	$V_{SS_LV_CORE_RING}$	—		
G8	$V_{SS_LV_CORE_RING}$	—		
G9	$V_{SS_LV_CORE_RING}$	—		
G10	$V_{SS_LV_CORE_RING}$	—		
G11	$V_{SS_LV_CORE_RING}$	—		
G12	$V_{DD_LV_CORE_RING}$	—		
G14	D[12]	SIUL	GPIO[60]	GPIO[60]
		FlexPWM_0	X[1]	X[1]
		LINFlexD_1	—	RXD

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
G15	H[13]	SIUL	GPIO[125]	GPIO[125]
		FlexPWM_1	X[3]	X[3]
		eTimer_2	ETC[3]	ETC[3]
G16	H[9]	SIUL	GPIO[121]	GPIO[121]
		FlexPWM_1	B[1]	B[1]
		DSPI_0	CS7	—
G17	G[6]	SIUL	GPIO[102]	GPIO[102]
		FlexPWM_0	A[3]	A[3]
H1	G[13]	SIUL	GPIO[109]	GPIO[109]
		NPC	MDO[10]	—
H2	$V_{SS_HV_IO_RING}$	—		
H3	C[4]	SIUL	GPIO[36]	GPIO[36]
		DSPI_0	CS0	CS0
		FlexPWM_0	X[1]	X[1]
		SSCM	DEBUG[4]	—
		SIUL	—	EIRQ[22]
H4	A[5]	SIUL	GPIO[5]	GPIO[5]
		DSPI_1	CS0	CS0
		eTimer_1	ETC[5]	ETC[5]
		DSPI_0	CS7	—
		SIUL	—	EIRQ[5]
H6	V_{DD_LV}	—		
H7	V_{SS_LV}	—		
H8	V_{SS_LV}	—		
H9	V_{SS_LV}	—		
H10	V_{SS_LV}	—		
H11	V_{SS_LV}	—		
H12	V_{DD_LV}	—		
H14	V_{SS_LV}	—		
H15	$V_{DD_HV_REG_1}$	—		
H16	$V_{DD_HV_FLA}$	—		
H17	H[6]	SIUL	GPIO[118]	GPIO[118]
		FlexPWM_1	B[0]	B[0]
		DSPI_0	CS5	—

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
J1	F[7]	SIUL	GPIO[87]	GPIO[87]
		NPC	MCKO	—
J2	G[15]	SIUL	GPIO[111]	GPIO[111]
		NPC	MDO[8]	—
J3	$V_{DD_HV_REG_0}$	—		
J4	$V_{DD_HV_REG_0}$	—		
J6	V_{DD_LV}	—		
J7	V_{SS_LV}	—		
J8	V_{SS_LV}	—		
J9	V_{SS_LV}	—		
J10	V_{SS_LV}	—		
J11	V_{SS_LV}	—		
J12	V_{DD_LV}	—		
J14	V_{DD_LV}	—		
J15	$V_{DD_HV_REG_1}$	—		
J16	$V_{SS_HV_FLA}$	—		
J17	H[15]	SIUL	GPIO[127]	GPIO[127]
		FlexPWM_1	B[3]	B[3]
		eTimer_2	ETC[5]	ETC[5]
K1	F[9]	SIUL	GPIO[89]	GPIO[89]
		NPC	$\overline{\text{MSEO[0]}}$	—
K2	F[8]	SIUL	GPIO[88]	GPIO[88]
		NPC	$\overline{\text{MSEO[1]}}$	—
K3	$\overline{\text{RDY}}$	NPC	$\overline{\text{RDY}}$	—
		SIUL	GPIO[132]	GPIO[132]
K4	C[7]	SIUL	GPIO[39]	GPIO[39]
		FlexPWM_0	A[1]	A[1]
		SSCM	DEBUG[7]	—
		DSPI_0	—	SIN
K6	V_{DD_LV}	—		
K7	V_{SS_LV}	—		
K8	V_{SS_LV}	—		
K9	V_{SS_LV}	—		
K10	V_{SS_LV}	—		
K11	V_{SS_LV}	—		

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
K12	V_{DD_LV}	—		
K14	Not connected	—		
K15	H[8]	SIUL	GPIO[120]	GPIO[120]
		FlexPWM_1	A[1]	A[1]
		DSPI_0	CS6	—
K16	H[7]	SIUL	GPIO[119]	GPIO[119]
		FlexPWM_1	X[1]	X[1]
		eTimer_2	ETC[1]	ETC[1]
K17	A[3]	SIUL	GPIO[3]	GPIO[3]
		eTimer_0	ETC[3]	ETC[3]
		DSPI_2	CS0	CS0
		FlexPWM_0	B[3]	B[3]
		MC_RGM	—	ABS[2]
		SIUL	—	EIRQ[3]
L1	F[10]	SIUL	GPIO[90]	GPIO[90]
		NPC	$\overline{\text{EVTO}}$	—
L2	F[11]	SIUL	GPIO[91]	GPIO[91]
		NPC	—	$\overline{\text{EVTI}}$
L3	D[9]	SIUL	GPIO[57]	GPIO[57]
		FlexPWM_0	X[0]	X[0]
		LINFlexD_1	TXD	—
L4	Not connected	—		
L6	V_{DD_LV}	—		
L7	V_{SS_LV}	—		
L8	V_{SS_LV}	—		
L9	V_{SS_LV}	—		
L10	V_{SS_LV}	—		
L11	V_{SS_LV}	—		
L12	V_{DD_LV}	—		
L14	Not connected	—		
L15	TCK	—		
L16	H[4]	SIUL	GPIO[116]	GPIO[116]
		FlexPWM_1	X[0]	X[0]
		eTimer_2	ETC[0]	ETC[0]

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
L17	B[4]	SIUL	GPIO[20]	GPIO[20]
		JTAGC	TDO	—
M1	$V_{DD_HV_OSC}$	—		
M2	$V_{DD_HV_IO_RING}$	—		
M3	D[8]	SIUL	GPIO[56]	GPIO[56]
		DSPI_1	CS2	—
		eTimer_1	ETC[4]	ETC[4]
		DSPI_0	CS5	—
		FlexPWM_0	—	FAULT[3]
M4	Not connected	—		
M6	V_{DD_LV}	—		
M7	V_{DD_LV}	—		
M8	V_{DD_LV}	—		
M9	V_{DD_LV}	—		
M10	V_{DD_LV}	—		
M11	V_{DD_LV}	—		
M12	V_{DD_LV}	—		
M14	C[11]	SIUL	GPIO[43]	GPIO[43]
		eTimer_0	ETC[4]	ETC[4]
		DSPI_2	CS2	—
M15	B[5]	SIUL	GPIO[21]	GPIO[21]
		JTAGC	—	TDI
M16	TMS	—		
M17	H[5]	SIUL	GPIO[117]	GPIO[117]
		FlexPWM_1	A[0]	A[0]
		DSPI_0	CS4	—
N1	XTAL	—		
N2	$V_{SS_HV_IO_RING}$	—		
N3	D[5]	SIUL	GPIO[53]	GPIO[53]
		DSPI_0	CS3	—
		FlexPWM_0	—	FAULT[2]
N4	$V_{SS_LV_PLL0_PLL1}$	—		
N14	Not connected	—		

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
N15	C[12]	SIUL	GPIO[44]	GPIO[44]
		eTimer_0	ETC[5]	ETC[5]
		DSPI_2	CS3	—
N16	A[2]	SIUL	GPIO[2]	GPIO[2]
		eTimer_0	ETC[2]	ETC[2]
		FlexPWM_0	A[3]	A[3]
		DSPI_2	—	SIN
		MC_RGM	—	ABS[0]
		SIUL	—	EIRQ[2]
N17	G[5]	SIUL	GPIO[101]	GPIO[101]
		FlexPWM_0	X[3]	X[3]
		DSPI_2	CS3	—
P1	$V_{SS_HV_OSC}$	—		
P2	$\overline{RESET}$	—		
P3	D[6]	SIUL	GPIO[54]	GPIO[54]
		DSPI_0	CS2	—
		FlexPWM_0	X[3]	X[3]
		FlexPWM_0	—	FAULT[1]
P4	$V_{DD_LV_PLL0_PLL1}$	—		
P5	$V_{DD_LV_CORE_RING}$	—		
P6	$V_{SS_LV_CORE_RING}$	—		
P7	B[8]	SIUL	—	GPIO[24]
		eTimer_0	—	ETC[5]
		ADC_0	—	AN[1]
P8	Not connected	—		
P9	$V_{SS_HV_IO_RING}$	—		
P10	$V_{DD_HV_IO_RING}$	—		
P11	B[14]	SIUL	—	GPIO[30]
		eTimer_0	—	ETC[4]
		SIUL	—	EIRQ[19]
		ADC_1	—	AN[1]
P12	$V_{DD_LV_CORE_RING}$	—		
P13	$V_{SS_LV_CORE_RING}$	—		
P14	$V_{DD_HV_IO_RING}$	—		

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
P15	G[10]	SIUL	GPIO[106]	GPIO[106]
		FlexRay	DBG2	—
		DSPI_2	CS3	—
		FlexPWM_0	—	FAULT[2]
P16	G[8]	SIUL	GPIO[104]	GPIO[104]
		FlexRay	DBG0	—
		DSPI_0	CS1	—
		FlexPWM_0	—	FAULT[0]
		SIUL	—	EIRQ[21]
P17	G[7]	SIUL	GPIO[103]	GPIO[103]
		FlexPWM_0	B[3]	B[3]
R1	EXTAL	—		
R2	FCCU_F[0]	FCCU	F[0]	—
R3	$V_{SS_HV_IO_RING}$	—		
R4	D[7]	SIUL	GPIO[55]	GPIO[55]
		DSPI_1	CS3	—
		DSPI_0	CS4	—
		SWG	analog output	—
R5	B[7]	SIUL	—	GPIO[23]
		LINFlexD_0	—	RXD
		ADC_0	—	AN[0]
R6	E[6]	SIUL	—	GPIO[70]
		ADC_0	—	AN[4]
R7	$V_{DD_HV_ADR0}$	—		
R8	B[10]	SIUL	—	GPIO[26]
		ADC_0 ADC_1	—	AN[12]
R9	$V_{DD_HV_ADR1}$	—		
R10	B[13]	SIUL	—	GPIO[29]
		LINFlexD_1	—	RXD
		ADC_1	—	AN[0]
R11	B[15]	SIUL	—	GPIO[31]
		SIUL	—	EIRQ[20]
		ADC_1	—	AN[2]

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
R12	C[0]	SIUL	—	GPIO[32]
		ADC_1	—	AN[3]
R13	BCTRL	—		
R14	A[1]	SIUL	GPIO[1]	GPIO[1]
		eTimer_0	ETC[1]	ETC[1]
		DSPI_2	SOUT	—
		SIUL	—	EIRQ[1]
R15	$V_{SS_HV_IO_RING}$	—		
R16	D[11]	SIUL	GPIO[59]	GPIO[59]
		FlexPWM_0	B[0]	B[0]
		eTimer_0	—	ETC[1]
R17	G[9]	SIUL	GPIO[105]	GPIO[105]
		FlexRay	DBG1	—
		DSPI_1	CS1	—
		FlexPWM_0	—	FAULT[1]
		SIUL	—	EIRQ[29]
T1	$V_{SS_HV_IO_RING}$	—		
T2	$V_{DD_HV_IO_RING}$	—		
T3	Not connected	—		
T4	C[1]	SIUL	—	GPIO[33]
		ADC_0	—	AN[2]
T5	E[5]	SIUL	—	GPIO[69]
		ADC_0	—	AN[8]
T6	E[7]	SIUL	—	GPIO[71]
		ADC_0	—	AN[6]
T7	$V_{SS_HV_ADR0}$	—		
T8	B[11]	SIUL	—	GPIO[27]
		ADC_0 ADC_1	—	AN[13]
T9	$V_{SS_HV_ADR1}$	—		
T10	E[9]	SIUL	—	GPIO[73]
		ADC_1	—	AN[7]
T11	E[10]	SIUL	—	GPIO[74]
		ADC_1	—	AN[8]

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
T12	E[12]	SIUL	—	GPIO[76]
		ADC_1	—	AN[6]
T13	E[0]	SIUL	—	GPIO[64]
		ADC_1	—	AN[5]
T14	A[0]	SIUL	GPIO[0]	GPIO[0]
		eTimer_0	ETC[0]	ETC[0]
		DSPI_2	SCK	SCK
		SIUL	—	EIRQ[0]
T15	D[10]	SIUL	GPIO[58]	GPIO[58]
		FlexPWM_0	A[0]	A[0]
		eTimer_0	—	ETC[0]
T16	$V_{DD_HV_IO_RING}$	—		
T17	$V_{SS_HV_IO_RING}$	—		
U1	$V_{SS_HV_IO_RING}$	—		
U2	$V_{SS_HV_IO_RING}$	—		
U3	Not connected	—		
U4	E[4]	SIUL	—	GPIO[68]
		ADC_0	—	AN[7]
U5	C[2]	SIUL	—	GPIO[34]
		ADC_0	—	AN[3]
U6	E[2]	SIUL	—	GPIO[66]
		ADC_0	—	AN[5]
U7	B[9]	SIUL	—	GPIO[25]
		ADC_0 ADC_1	—	AN[11]
U8	B[12]	SIUL	—	GPIO[28]
		ADC_0 ADC_1	—	AN[14]
U9	$V_{DD_HV_ADV}$	—		
U10	$V_{SS_HV_ADV}$	—		
U11	E[11]	SIUL	—	GPIO[75]
		ADC_1	—	AN[4]
U12	Not connected	—		
U13	Not connected	—		
U14	$V_{DD_HV_PMU}$	—		

Table 9. LFBGA257 pin function summary (continued)

Pin #	Port/function	Peripheral	Output function	Input function
U15	G[11]	SIUL	GPIO[107]	GPIO[107]
		FlexRay	DBG3	—
		FlexPWM_0	—	FAULT[3]
U16	$V_{SS_HV_IO_RING}$	—		
U17	$V_{SS_HV_IO_RING}$	—		

1. V_{PP_TEST} should always be tied to ground (V_{SS}) for normal operations.

4.2 Supply pins

Table 10. Supply pins

Supply		Pin #		
Symbol	Description	100 pkg	144 pkg	257 pkg
VREG control and power supply pins				
BCTRL	Voltage regulator external NPN ballast base control pin	47	69	R13
$V_{DD_LV_COR}$	Core logic supply	48	70	VDD_LV(1)
$V_{SS_LV_COR}$	Core regulator ground	49	71	VSS_LV(2)
$V_{DD_HV_PMU}$	Voltage regulator supply	50	72	U14
ADC_0/ADC_1 reference voltage and ADC supply				
$V_{DD_HV_ADR0}$	ADC_0 high reference voltage	33	50	R7
$V_{SS_HV_ADR0}$	ADC_0 low reference voltage	34	51	T7
$V_{DD_HV_ADR1}$	ADC_1 high reference voltage	39	56	R9
$V_{SS_HV_ADR1}$	ADC_1 low reference voltage	40	57	T9
$V_{DD_HV_ADV}$	ADC voltage supply for ADC_0 and ADC_1	41	58	U9
$V_{SS_HV_ADV}$	ADC ground for ADC_0 and ADC_1	42	59	U10
Power supply pins (3.3 V)				
$V_{DD_HV_IO}$	3.3 V Input/Output supply voltage	—	6	VDD_HV(3)
$V_{SS_HV_IO}$	3.3 V Input/Output ground	—	7	VSS_HV(4)
$V_{DD_HV_REG_0}$	VDD_HV_REG_0	10	16	J3
$V_{DD_HV_IO}$	3.3 V Input/Output supply voltage	13	21	VDD_HV(3)
$V_{SS_HV_IO}$	3.3 V Input/Output ground	14	22	VSS_HV(4)
$V_{DD_HV_OSC}$	Crystal oscillator amplifier supply voltage	16	27	M1
$V_{SS_HV_OSC}$	Crystal oscillator amplifier ground	17	28	P1
$V_{SS_HV_IO}$	3.3 V Input/Output ground	62	90	VSS_HV(4)
$V_{DD_HV_IO}$	3.3 V Input/Output supply voltage	63	91	VDD_HV(3)

Table 10. Supply pins (continued)

Supply		Pin #		
Symbol	**Description**	**100 pkg**	**144 pkg**	**257 pkg**
$V_{DD_HV_REG_1}$	VDD_HV_REG_1	67	95	H15
$V_{SS_HV_FLA}$	VSS_HV_FLA	68	96	J16
$V_{DD_HV_FLA}$	VDD_HV_FLA	69	97	H16
$V_{DD_HV_IO}$	VDD_HV_IO	87	126	VDD_HV(3)
$V_{SS_HV_IO}$	VSS_HV_IO	88	127	VSS_HV(4)
$V_{DD_HV_REG_2}$	VDD_HV_REG_2	91	130	C7
Power supply pins (1.2 V)				
$V_{SS_LV_COR}$	VSS_LV_COR Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{DD_LV_COR}$ pin.	11	17	VSS_HV(2)
$V_{DD_LV_COR}$	VDD_LV_COR Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{SS_LV_COR}$ pin.	12	18	VDD_LV(1)
V_{SS} 1V2	VSS_LV_PLL0_PLL1 / 1.2 V Decoupling pins for on-chip FMPLL modules. Decoupling capacitor must be connected between this pin and $V_{DD_LV_PLL}$.	24	35	N4
V_{DD} 1V2	VDD_LV_PLL0_PLL1 Decoupling pins for on-chip FMPLL modules. Decoupling capacitor must be connected between this pin and $V_{SS_LV_PLL}$.	25	36	P4
$V_{DD_LV_COR}$	VDD_LV_COR Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{SS_LV_COR}$ pin.	28	39	VDD_LV(1)
$V_{SS_LV_COR}$	VSS_LV_COR Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{DD_LV_COR}$ pin.	29	40	VSS_LV(2)
$V_{DD_LV_COR}$	VDD_LV_COR Decoupling pins for core logic and Regulator feedback. Decoupling capacitor must be connected between this pins and $V_{SS_LV_REGCOR}$.	—	70	VDD_LV(1)
$V_{SS_LV_COR}$	VSS_LV_REGCOR0 Decoupling pins for core logic and Regulator feedback. Decoupling capacitor must be connected between this pins and $V_{DD_LV_REGCOR}$.	—	71	VSS_LV(2)
$V_{DD_LV_COR}$	VDD_LV_COR Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{SS_LV_COR}$ pin.	65	93	VDD_LV(1)
$V_{SS_LV_COR}$	VSS_LV_COR / 1.2 V Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{DD_LV_COR}$ pin.	66	94	VSS_LV(2)
V_{DD} 1V2	VDD_LV_COR Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{DD_LV_COR}$ pin.	92	131	VDD_LV(1)

Table 10. Supply pins (continued)

Supply		Pin #		
Symbol	**Description**	**100 pkg**	**144 pkg**	**257 pkg**
V_{SS} 1V2	VSS_LV_COR Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{DD_LV_COR}$ pin.	93	132	VSS_LV(2)
V_{DD} 1V2	VDD_LV_COR / Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{DD_LV_COR}$ pin.	—	135	VDD_LV(1)
V_{SS} 1V2	VSS_LV_COR / Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{DD_LV_COR}$ pin.	—	137	VSS_LV(2)

1. VDD_LV balls are tied together on the LFBGA257 substrate.
2. VSS_LV balls are tied together on the LFBGA257 substrate.
3. VDD_HV balls are tied together on the LFBGA257 substrate.
4. VSS_HV balls are tied together on the LFBGA257 substrate.

4.3 System pins

Table 11. System pins

Symbol	Description	Direction	Pin #		
			100 pkg	**144 pkg**	**257 pkg**
Dedicated pins					
MDO0(1)	Nexus Message Data Output — line	Output only	—	9	E1
$\overline{\text{NMI}}$(2)	Non Maskable Interrupt	Input only	1	1	E4
XTAL	Input for oscillator amplifier circuit and internal clock generator	Input only	18	29	N1
EXTAL(3)	Oscillator amplifier output	Input/Output (4)	19	30	R1
TMS(2)	JTAG state machine control	Input only	59	87	M16
TCK(2)	JTAG clock	Input only	60	88	L15
JCOMP(5)	JTAG compliance select	Input only	84	123	C10
Reset pin					
$\overline{\text{RESET}}$	Bidirectional reset with Schmitt-Trigger characteristics and noise filter. This pin has medium drive strength. Output drive is open drain and must be terminated by an external resistor of value 1KOhm.(6)	Bidirectional	20	31	P2
Test pin					
VPP TEST	Pin for testing purpose only. To be tied to ground in normal operating mode.		74	107	D15

1. This pad is configured for Fast (F) pad speed.

2. This pad contains a weak pull-up.
3. EXTAL is an "Output" in "crystal" mode, and is an "Input" in "ext clock" mode.
4. In XOSC Bypass Mode, the analog portion of crystal oscillator (amplifier) is disabled. An external clock can be applied at EXTAL as an input. In XOSC Normal Mode, EXTAL is an output
5. This pad contains a weak pull-down.
6. RESET output shall be considered valid only after the 3.3V supply reaches its stable value.

Note: None of system pins (except RESET) provides an open drain output.

4.4 Pin muxing

Table 12 defines the pin list and muxing for this device.

Each entry of *Table 12* shows all the possible configurations for each pin, via the alternate functions. The default function assigned to each pin after reset is indicated by ALT0.

Note: Pins labeled "NC" are to be left unconnected. Any connection to an external circuit or voltage may cause unpredictable device behavior or damage.

Pins labeled "Reserved" are to be tied to ground. Not doing so may cause unpredictable device behavior.

Table 12. Pin muxing

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
Port A												
A[0]	PCR[0]	SIUL	GPIO[0]	ALT0	GPIO[0]	—	—	M	S	51	73	T14
		eTimer_0	ETC[0]	ALT1	ETC[0]	PSMI[35]; PADSEL=0						
		DSPI_2	SCK	ALT2	SCK	PSMI[1]; PADSEL=0						
		SIUL	—	—	EIRQ[0]	—						
A[1]	PCR[1]	SIUL	GPIO[1]	ALT0	GPIO[1]	—	—	M	S	52	74	R14
		eTimer_0	ETC[1]	ALT1	ETC[1]	PSMI[36]; PADSEL=0						
		DSPI_2	SOUT	ALT2	—	—						
		SIUL	—	—	EIRQ[1]	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
A[2]	PCR[2]	SIUL	GPIO[2]	ALT0	GPIO[2]	—	Pull down	M	S	57	84	N16
		eTimer_0	ETC[2]	ALT1	ETC[2]	PSMI[37]; PADSEL=0						
		FlexPWM_0	A[3]	ALT3	A[3]	PSMI[23]; PADSEL=0						
		DSPI_2	—	—	SIN	PSMI[2]; PADSEL=0						
		MC_RGM	—	—	ABS[0]	—						
		SIUL	—	—	EIRQ[2]	—						
A[3]	PCR[3]	SIUL	GPIO[3]	ALT0	GPIO[3]	—	Pull down	M	S	64	92	K17
		eTimer_0	ETC[3]	ALT1	ETC[3]	PSMI[38]; PADSEL=0						
		DSPI_2	CS0	ALT2	CS0	PSMI[3]; PADSEL=0						
		FlexPWM_0	B[3]	ALT3	B[3]	PSMI[27]; PADSEL=0						
		MC_RGM	—	—	ABS[2]	—						
		SIUL	—	—	EIRQ[3]	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
A[4]	PCR[4]	SIUL	GPIO[4]	ALT0	GPIO[4]	—	Pull down	M	S	75	108	C16
		eTimer_1	ETC[0]	ALT1	ETC[0]	PSMI[9]; PADSEL=0						
		DSPI_2	CS1	ALT2	—	—						
		eTimer_0	ETC[4]	ALT3	ETC[4]	PSMI[7]; PADSEL=0						
		MC_RGM	—	—	FAB	—						
		SIUL	—	—	EIRQ[4]	—						
A[5]	PCR[5]	SIUL	GPIO[5]	ALT0	GPIO[5]	—	—	M	S	8	14	H4
		DSPI_1	CS0	ALT1	CS0	—						
		eTimer_1	ETC[5]	ALT2	ETC[5]	PSMI[14]; PADSEL=0						
		DSPI_0	CS7	ALT3	—	—						
		SIUL	—	—	EIRQ[5]	—						
A[6]	PCR[6]	SIUL	GPIO[6]	ALT0	GPIO[6]	—	—	M	S	2	2	G4
		DSPI_1	SCK	ALT1	SCK	—						
		SIUL	—	—	EIRQ[6]	—						
A[7]	PCR[7]	SIUL	GPIO[7]	ALT0	GPIO[7]	—	—	M	S	4	10	F3
		DSPI_1	SOUT	ALT1	—	—						
		SIUL	—	—	EIRQ[7]	—						
A[8]	PCR[8]	SIUL	GPIO[8]	ALT0	GPIO[8]	—	—	M	S	6	12	F4
		DSPI_1	—	—	SIN	—						
		SIUL	—	—	EIRQ[8]	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
A[9]	PCR[9]	SIUL	GPIO[9]	ALT0	GPIO[9]	—	—	M	S	94	134	B6
		DSPI_2	CS1	ALT1	—	—						
		FlexPWM_0	B[3]	ALT3	B[3]	PSMI[27]; PADSEL=1						
		FlexPWM_0	—	—	FAULT[0]	PSMI[16]; PADSEL=0						
A[10]	PCR[10]	SIUL	GPIO[10]	ALT0	GPIO[10]	—	—	M	S	81	118	A13
		DSPI_2	CS0	ALT1	CS0	PSMI[3]; PADSEL=1						
		FlexPWM_0	B[0]	ALT2	B[0]	PSMI[24]; PADSEL=0						
		FlexPWM_0	X[2]	ALT3	X[2]	PSMI[29]; PADSEL=0						
		SIUL	—	—	EIRQ[9]	—						
A[11]	PCR[11]	SIUL	GPIO[11]	ALT0	GPIO[11]	—	—	M	S	82	120	D11
		DSPI_2	SCK	ALT1	SCK	PSMI[1]; PADSEL=1						
		FlexPWM_0	A[0]	ALT2	A[0]	PSMI[20]; PADSEL=0						
		FlexPWM_0	A[2]	ALT3	A[2]	PSMI[22]; PADSEL=0						
		SIUL	—	—	EIRQ[10]	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
A[12]	PCR[12]	SIUL	GPIO[12]	ALT0	GPIO[12]	—	—	M	S	83	122	A10
		DSPI_2	SOUT	ALT1	—	—						
		FlexPWM_0	A[2]	ALT2	A[2]	PSMI[22]; PADSEL=1						
		FlexPWM_0	B[2]	ALT3	B[2]	PSMI[26]; PADSEL=0						
		SIUL	—	—	EIRQ[11]	—						
A[13]	PCR[13]	SIUL	GPIO[13]	ALT0	GPIO[13]	—	—	M	S	95	136	C6
		FlexPWM_0	B[2]	ALT2	B[2]	PSMI[26]; PADSEL=1						
		DSPI_2	—	—	SIN	PSMI[2]; PADSEL=1						
		FlexPWM_0	—	—	FAULT[0]	PSMI[16]; PADSEL=1						
		SIUL	—	—	EIRQ[12]	—						
A[14]	PCR[14]	SIUL	GPIO[14]	ALT0	GPIO[14]	—	—	M	S	99	143	B4
		FlexCAN_1	TXD	ALT1	—	—						
		eTimer_1	ETC[4]	ALT2	ETC[4]	PSMI[13]; PADSEL=0						
		SIUL	—	—	EIRQ[13]	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
A[15]	PCR[15]	SIUL	GPIO[15]	ALT0	GPIO[15]	—	—	M	S	100	144	D3
		eTimer_1	ETC[5]	ALT2	ETC[5]	PSMI[14]; PADSEL=1						
		FlexCAN_1	—	—	RXD	PSMI[34]; PADSEL=0						
		FlexCAN_0	—	—	RXD	PSMI[33]; PADSEL=0						
		SIUL	—	—	EIRQ[14]	—						
Port B												
B[0]	PCR[16]	SIUL	GPIO[16]	ALT0	GPIO[16]	—	—	M	S	76	109	B15
		FlexCAN_0	TXD	ALT1	—	—						
		eTimer_1	ETC[2]	ALT2	ETC[2]	PSMI[11]; PADSEL=0						
		SSCM	DEBUG[0]	ALT3	—	—						
		SIUL	—	—	EIRQ[15]	—						
B[1]	PCR[17]	SIUL	GPIO[17]	ALT0	GPIO[17]	—	—	M	S	77	110	C14
		eTimer_1	ETC[3]	ALT2	ETC[3]	PSMI[12]; PADSEL=0						
		SSCM	DEBUG[1]	ALT3	—	—						
		FlexCAN_0	—	—	RXD	PSMI[33]; PADSEL=1						
		FlexCAN_1	—	—	RXD	PSMI[34]; PADSEL=1						
		SIUL	—	—	EIRQ[16]	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
B[2]	PCR[18]	SIUL	GPIO[18]	ALT0	GPIO[18]	—	—	M	S	79	114	A14
		LINFlexD_0	TXD	ALT1	—	—						
		SSCM	DEBUG[2]	ALT3	—	—						
		SIUL	—	—	EIRQ[17]	—						
B[3]	PCR[19]	SIUL	GPIO[19]	ALT0	GPIO[19]	—	—	M	S	80	116	B13
		SSCM	DEBUG[3]	ALT3	—	—						
		LINFlexD_0	—	—	RXD	PSMI[31]; PADSEL=0						
B[4](2)	PCR[20]	SIUL	GPIO[20]	ALT0	GPIO[20]	—	—	F	S	61	89	L17
		JTAGC	TDO	ALT1	—	—						
B[5]	PCR[21]	SIUL	GPIO[21]	ALT0	GPIO[21]	—	Pull up	M	S	58	86	M15
		JTAGC	—	—	TDI	—						
B[6]	PCR[22]	SIUL	GPIO[22]	ALT0	GPIO[22]	—	—	F	S	96	138	B3
		MC_CGM	clk_out	ALT1	—	—						
		DSPI_2	CS2	ALT2	—	—						
		SIUL	—		EIRQ[18]	—						
B[7]	PCR[23]	SIUL	—	ALT0	GPI[23]	—	—	—	—	30	43	R5
		LINFlexD_0	—	—	RXD	PSMI[31]; PADSEL=1						
		ADC_0	—	—	AN[0](3)	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
B[8]	PCR[24]	SIUL	—	ALT0	GPI[24]	—	—	—	—	31	47	P7
		eTimer_0	—	—	ETC[5]	PSMI[8]; PADSEL=2						
		ADC_0	—	—	AN[1](3)	—						
B[9]	PCR[25]	SIUL	—	ALT0	GPI[25]	—	—	—	—	35	52	U7
		ADC_0 ADC_1	—	—	AN[11](3)	—						
B[10]	PCR[26]	SIUL	—	ALT0	GPI[26]	—	—	—	—	36	53	R8
		ADC_0 ADC_1	—	—	AN[12](3)	—						
B[11]	PCR[27]	SIUL	—	ALT0	GPI[27]	—	—	—	—	37	54	T8
		ADC_0 ADC_1	—	—	AN[13](3)	—						
B[12]	PCR[28]	SIUL	—	ALT0	GPI[28]	—	—	—	—	38	55	U8
		ADC_0 ADC_1	—	—	AN[14](3)	—						
B[13]	PCR[29]	SIUL	—	ALT0	GPI[29]	—	—	—	—	43	60	R10
		LINFlexD_1	—	—	RXD	PSMI[32]; PADSEL=0						
		ADC_1	—	—	AN[0](3)	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
B[14]	PCR[30]	SIUL	—	ALT0	GPI[30]	—	—	—	—	44	64	P11
		eTimer_0	—	—	ETC[4]	PSMI[7]; PADSEL=2						
		SIUL	—	—	EIRQ[19]	—						
		ADC_1	—	—	AN[1](3)	—						
B[15]	PCR[31]	SIUL	—	ALT0	GPI[31]	—	—	—	—	—	62	R11
		SIUL	—	—	EIRQ[20]	—						
		ADC_1	—	—	AN[2](3)	—						
Port C												
C[0]	PCR[32]	SIUL	—	ALT0	GPI[32]	—	—	—	—	45	66	R12
		ADC_1	—	—	AN[3](3)	—						
C[1]	PCR[33]	SIUL	—	ALT0	GPI[33]	—	—	—	—	—	41	T4
		ADC_0	—	—	AN[2](3)	—						
C[2]	PCR[34]	SIUL	—	ALT0	GPI[34]	—	—	—	—	—	45	U5
		ADC_0	—	—	AN[3](3)	—						
C[4]	PCR[36]	SIUL	GPIO[36]	ALT0	GPIO[36]	—	—	M	S	5	11	H3
		DSPI_0	CS0	ALT1	CS0	—						
		FlexPWM_0	X[1]	ALT2	X[1]	PSMI[28]; PADSEL=0						
		SSCM	DEBUG[4]	ALT3	—	—						
		SIUL	—	—	EIRQ[22]	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
C[5]	PCR[37]	SIUL	GPIO[37]	ALT0	GPIO[37]	—	—	M	S	7	13	G3
		DSPI_0	SCK	ALT1	SCK	—						
		SSCM	DEBUG[5]	ALT3	—	—						
		FlexPWM_0	—	—	FAULT[3]	PSMI[19]; PADSEL=0						
		SIUL	—	—	EIRQ[23]	—						
C[6]	PCR[38]	SIUL	GPIO[38]	ALT0	GPIO[38]	—	—	M	S	98	142	D4
		DSPI_0	SOUT	ALT1	—	—						
		FlexPWM_0	B[1]	ALT2	B[1]	PSMI[25]; PADSEL=0						
		SSCM	DEBUG[6]	ALT3	—	—						
		SIUL	—	—	EIRQ[24]	—						
C[7]	PCR[39]	SIUL	GPIO[39]	ALT0	GPIO[39]	—	—	M	S	9	15	K4
		FlexPWM_0	A[1]	ALT2	A[1]	PSMI[21]; PADSEL=0						
		SSCM	DEBUG[7]	ALT3	—	—						
		DSPI_0	—	—	SIN	—						
C[10]	PCR[42]	SIUL	GPIO[42]	ALT0	GPIO[42]	—	—	M	S	78	111	A15
		DSPI_2	CS2	ALT1	—	—						
		FlexPWM_0	A[3]	ALT3	A[3]	PSMI[23]; PADSEL=1						
		FlexPWM_0	—	—	FAULT[1]	PSMI[17]; PADSEL=0						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
C[11]	PCR[43]	SIUL	GPIO[43]	ALT0	GPIO[43]	—	—	M	S	55	80	M14
		eTimer_0	ETC[4]	ALT1	ETC[4]	PSMI[7]; PADSEL=1						
		DSPI_2	CS2	ALT2	—	—						
C[12]	PCR[44]	SIUL	GPIO[44]	ALT0	GPIO[44]	—	—	M	S	56	82	N15
		eTimer_0	ETC[5]	ALT1	ETC[5]	PSMI[8]; PADSEL=0						
		DSPI_2	CS3	ALT2	—	—						
C[13]	PCR[45]	SIUL	GPIO[45]	ALT0	GPIO[45]	—	—	M	S	71	101	F15
		eTimer_1	ETC[1]	ALT1	ETC[1]	PSMI[10]; PADSEL=0						
		CTU_0	—	—	EXT_IN	PSMI[0]; PADSEL=0						
		FlexPWM_0	—	—	EXT_SYNC	PSMI[15]; PADSEL=0						
C[14]	PCR[46]	SIUL	GPIO[46]	ALT0	GPIO[46]	—	—	M	S	72	103	E15
		eTimer_1	ETC[2]	ALT1	ETC[2]	PSMI[11]; PADSEL=1						
		CTU_0	EXT_TGR	ALT2	—	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
C[15]	PCR[47]	SIUL	GPIO[47]	ALT0	GPIO[47]	—	—	SYM	S	85	124	A8
		FlexRay	CA_TR_EN	ALT1	—	—						
		eTimer_1	ETC[0]	ALT2	ETC[0]	PSMI[9]; PADSEL=1						
		FlexPWM_0	A[1]	ALT3	A[1]	PSMI[21]; PADSEL=1						
		CTU_0	—	—	EXT_IN	PSMI[0]; PADSEL=1						
		FlexPWM_0	—	—	EXT_SYNC	PSMI[15]; PADSEL=1						
Port D												
D[0]	PCR[48]	SIUL	GPIO[48]	ALT0	GPIO[48]	—	—	SYM	S	86	125	B8
		FlexRay	CA_TX	ALT1	—	—						
		eTimer_1	ETC[1]	ALT2	ETC[1]	PSMI[10]; PADSEL=1						
		FlexPWM_0	B[1]	ALT3	B[1]	PSMI[25]; PADSEL=1						
D[1]	PCR[49]	SIUL	GPIO[49]	ALT0	GPIO[49]	—	—	M	S	3	3	E3
		eTimer_1	ETC[2]	ALT2	ETC[2]	PSMI[11]; PADSEL=2						
		CTU_0	EXT_TGR	ALT3	—	—						
		FlexRay	—	—	CA_RX	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
D[2]	PCR[50]	SIUL	GPIO[50]	ALT0	GPIO[50]	—	—	M	S	—	140	C5
		eTimer_1	ETC[3]	ALT2	ETC[3]	PSMI[12]; PADSEL=1						
		FlexPWM_0	X[3]	ALT3	X[3]	PSMI[30]; PADSEL=0						
		FlexRay	—	—	CB_RX	—						
D[3]	PCR[51]	SIUL	GPIO[51]	ALT0	GPIO[51]	—	—	SYM	S	89	128	A7
		FlexRay	CB_TX	ALT1	—	—						
		eTimer_1	ETC[4]	ALT2	ETC[4]	PSMI[13]; PADSEL=1						
		FlexPWM_0	A[3]	ALT3	A[3]	PSMI[23]; PADSEL=2						
D[4]	PCR[52]	SIUL	GPIO[52]	ALT0	GPIO[52]	—	—	SYM	S	90	129	B7
		FlexRay	CB_TR_EN	ALT1	—	—						
		eTimer_1	ETC[5]	ALT2	ETC[5]	PSMI[14]; PADSEL=2						
		FlexPWM_0	B[3]	ALT3	B[3]	PSMI[27]; PADSEL=2						
D[5]	PCR[53]	SIUL	GPIO[53]	ALT0	GPIO[53]	—	—	M	S	22	33	N3
		DSPI_0	CS3	ALT1	—	—						
		FlexPWM_0	—	—	FAULT[2]	PSMI[18]; PADSEL=0						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
D[6]	PCR[54]	SIUL	GPIO[54]	ALT0	GPIO[54]	—	—	M	S	23	34	P3
		DSPI_0	CS2	ALT1	—	—						
		FlexPWM_0	X[3]	ALT3	X[3]	PSMI[30]; PADSEL=1						
		FlexPWM_0	—	—	FAULT[1]	PSMI[17]; PADSEL=1						
D[7]	PCR[55]	SIUL	GPIO[55]	ALT0	GPIO[55]	—	—	M	S	26	37	R4
		DSPI_1	CS3	ALT1	—	—						
		DSPI_0	CS4	ALT3	—	—						
		SWG	analog output	—	—	—						
D[8]	PCR[56]	SIUL	GPIO[56]	ALT0	GPIO[56]	—	—	M	S	21	32	M3
		DSPI_1	CS2	ALT1	—	—						
		eTimer_1	ETC[4]	ALT2	ETC[4]	PSMI[13]; PADSEL=2						
		DSPI_0	CS5	ALT3	—	—						
		FlexPWM_0	—	—	FAULT[3]	PSMI[19]; PADSEL=1						
D[9]	PCR[57]	SIUL	GPIO[57]	ALT0	GPIO[57]	—	—	M	S	15	26	L3
		FlexPWM_0	X[0]	ALT1	X[0]	—						
		LINFlexD_1	TXD	ALT2	—	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
D[10]	PCR[58]	SIUL	GPIO[58]	ALT0	GPIO[58]	—	—	M	S	53	76	T15
		FlexPWM_0	A[0]	ALT1	A[0]	PSMI[20]; PADSEL=1						
		eTimer_0	—	—	ETC[0]	PSMI[35]; PADSEL=1						
D[11]	PCR[59]	SIUL	GPIO[59]	ALT0	GPIO[59]	—	—	M	S	54	78	R16
		FlexPWM_0	B[0]	ALT1	B[0]	PSMI[24]; PADSEL=1						
		eTimer_0	—	—	ETC[1]	PSMI[36]; PADSEL=1						
D[12]	PCR[60]	SIUL	GPIO[60]	ALT0	GPIO[60]		—	M	S	70	99	G14
		FlexPWM_0	X[1]	ALT1	X[1]	PSMI[28]; PADSEL=1						
		LINFlexD_1	—	—	RXD	PSMI[32]; PADSEL=1						
D[14]	PCR[62]	SIUL	GPIO[62]	ALT0	GPIO[62]	—	—	M	S	73	105	D16
		FlexPWM_0	B[1]	ALT1	B[1]	PSMI[25]; PADSEL=2						
		eTimer_0	—	—	ETC[3]	PSMI[38]; PADSEL=1						
Port E												
E[0]	PCR[64]	SIUL	—	ALT0	GPI[64]	—	—	—	—	46	68	T13
		ADC_1	—	—	AN[5](3)	—						
E[2]	PCR[66]	SIUL	—	ALT0	GPI[66]	—	—	—	—	32	49	U6
		ADC_0	—	—	AN[5](3)	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
E[4]	PCR[68]	SIUL	—	ALT0	GPI[68]	—	—	—	—	—	42	U4
		ADC_0	—	—	AN[7](3)	—						
E[5]	PCR[69]	SIUL	—	ALT0	GPI[69]	—	—	—	—	—	44	T5
		ADC_0	—	—	AN[8](3)	—						
E[6]	PCR[70]	SIUL	—	ALT0	GPI[70]	—	—	—	—	—	46	R6
		ADC_0	—	—	AN[4](3)	—						
E[7]	PCR[71]	SIUL	—	ALT0	GPI[71]	—	—	—	—	—	48	T6
		ADC_0	—	—	AN[6](3)	—						
E[9]	PCR[73]	SIUL	—	ALT0	GPI[73]	—	—	—	—	—	61	T10
		ADC_1	—	—	AN[7](3)	—						
E[10]	PCR[74]	SIUL	—	ALT0	GPI[74]	—	—	—	—	—	63	T11
		ADC_1	—	—	AN[8](3)	—						
E[11]	PCR[75]	SIUL	—	ALT0	GPI[75]	—	—	—	—	—	65	U11
		ADC_1	—	—	AN[4](3)	—						
E[12]	PCR[76]	SIUL	—	ALT0	GPI[76]	—	—	—	—	—	67	T12
		ADC_1	—	—	AN[6](3)	—						
E[13]	PCR[77]	SIUL	GPIO[77]	ALT0	GPIO[77]	—	—	M	S	—	117	D12
		eTimer_0	ETC[5]	ALT1	ETC[5]	PSMI[8]; PADSEL=1						
		DSPI_2	CS3	ALT2	—	—						
		SIUL	—	—	EIRQ[25]	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
E[14]	PCR[78]	SIUL	GPIO[78]	ALT0	GPIO[78]	—	—	M	S	—	119	B12
		eTimer_1	ETC[5]	ALT1	ETC[5]	PSMI[14]; PADSEL=3						
		SIUL	—	—	EIRQ[26]	—						
E[15]	PCR[79]	SIUL	GPIO[79]	ALT0	GPIO[79]	—	—	M	S	—	121	B11
		DSPI_0	CS1	ALT1	—	—						
		SIUL	—	—	EIRQ[27]	—						
Port F												
F[0]	PCR[80]	SIUL	GPIO[80]	ALT0	GPIO[80]	—	—	M	S	—	133	D7
		FlexPWM_0	A[1]	ALT1	A[1]	PSMI[21]; PADSEL=2						
		eTimer_0	—	—	ETC[2]	PSMI[37]; PADSEL=1						
		SIUL	—	—	EIRQ[28]	—						
F[3]	PCR[83]	SIUL	GPIO[83]	ALT0	GPIO[83]	—	—	M	S	—	139	B5
		DSPI_0	CS6	ALT1	—	—						
F[4]	PCR[84]	SIUL	GPIO[84]	ALT0	GPIO[84]	—	—	F	S	—	4	D2
		NPC	MDO[3]	ALT2	—	—						
F[5]	PCR[85]	SIUL	GPIO[85]	ALT0	GPIO[85]	—	—	F	S	—	5	D1
		NPC	MDO[2]	ALT2	—	—						
F[6]	PCR[86]	SIUL	GPIO[86]	ALT0	GPIO[86]	—	—	F	S	—	8	E2
		NPC	MDO[1]	ALT2	—	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
F[7]	PCR[87]	SIUL	GPIO[87]	ALT0	GPIO[87]	—	—	F	S	—	19	J1
		NPC	MCKO	ALT2	—	—						
F[8]	PCR[88]	SIUL	GPIO[88]	ALT0	GPIO[88]	—	—	F	S	—	20	K2
		NPC	$\overline{\text{MSEO[1]}}$	ALT2	—	—						
F[9]	PCR[89]	SIUL	GPIO[89]	ALT0	GPIO[89]	—	—	F	S	—	23	K1
		NPC	$\overline{\text{MSEO[0]}}$	ALT2	—	—						
F[10]	PCR[90]	SIUL	GPIO[90]	ALT0	GPIO[90]	—	—	F	S	—	24	L1
		NPC	$\overline{\text{EVTO}}$	ALT2	—	—						
F[11]	PCR[91]	SIUL	GPIO[91]	ALT0	GPIO[91]	—	—	M	S	—	25	L2
		NPC	—	ALT2	$\overline{\text{EVTI}}$	—						
F[12]	PCR[92]	SIUL	GPIO[92]	ALT0	GPIO[92]	—	—	M	S	—	106	C17
		eTimer_1	ETC[3]	ALT1	ETC[3]	PSMI[12]; PADSEL=2						
		SIUL	—	—	EIRQ[30]	—						
F[13]	PCR[93]	SIUL	GPIO[93]	ALT0	GPIO[93]	—	—	M	S	—	112	B14
		eTimer_1	ETC[4]	ALT1	ETC[4]	PSMI[13]; PADSEL=3						
		SIUL	—	—	EIRQ[31]	—						
F[14]	PCR[94]	SIUL	GPIO[94]	ALT0	GPIO[94]	—	—	M	S	—	115	C13
		LINFlexD_1	TXD	ALT1	—	—						
F[15]	PCR[95]	SIUL	GPIO[95]	ALT0	GPIO[95]	—	—	M	S	—	113	D13
		LINFlexD_1	—	—	RXD	PSMI[32]; PADSEL=2						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
FCCU												
FCCU_F[0]	—	FCCU	F[0]	ALT0	—	—	—	S	S	27	38	R2
FCCU_F[1]	—	FCCU	F[1]	ALT0	—	—	—	S	S	97	141	C4
Port G												
G[2]	PCR[98]	SIUL	GPIO[98]	ALT0	GPIO[98]	—	—	M	S	—	102	E16
		FlexPWM_0	X[2]	ALT1	X[2]	PSMI[29]; PADSEL=1						
		DSPI_1	CS1	ALT2	—	—						
G[3]	PCR[99]	SIUL	GPIO[99]	ALT0	GPIO[99]	—	—	M	S	—	104	D17
		FlexPWM_0	A[2]	ALT1	A[2]	PSMI[22]; PADSEL=2						
		eTimer_0	—	—	ETC[4]	PSMI[7]; PADSEL=3						
G[4]	PCR[100]	SIUL	GPIO[100]	ALT0	GPIO[100]	—	—	M	S	—	100	F17
		FlexPWM_0	B[2]	ALT1	B[2]	PSMI[26]; PADSEL=2						
		eTimer_0	—	—	ETC[5]	PSMI[8]; PADSEL=3						
G[5]	PCR[101]	SIUL	GPIO[101]	ALT0	GPIO[101]	—	—	M	S	—	85	N17
		FlexPWM_0	X[3]	ALT1	X[3]	PSMI[30]; PADSEL=2						
		DSPI_2	CS3	ALT2	—	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
G[6]	PCR[102]	SIUL	GPIO[102]	ALT0	GPIO[102]	—	—	M	S	—	98	G17
		FlexPWM_0	A[3]	ALT1	A[3]	PSMI[23]; PADSEL=3						
G[7]	PCR[103]	SIUL	GPIO[103]	ALT0	GPIO[103]		—	M	S	—	83	P17
		FlexPWM_0	B[3]	ALT1	B[3]	PSMI[27]; PADSEL=3						
G[8]	PCR[104]	SIUL	GPIO[104]	ALT0	GPIO[104]	—	—	M	S	—	81	P16
		FlexRay	DBG0	ALT1	—	—						
		DSPI_0	CS1	ALT2	—	—						
		FlexPWM_0	—	—	FAULT[0]	PSMI[16]; PADSEL=2						
		SIUL	—	—	EIRQ[21]	—						
G[9]	PCR[105]	SIUL	GPIO[105]	ALT0	GPIO[105]	—	—	M	S	—	79	R17
		FlexRay	DBG1	ALT1	—	—						
		DSPI_1	CS1	ALT2	—	—						
		FlexPWM_0	—	—	FAULT[1]	PSMI[17]; PADSEL=2						
		SIUL	—	—	EIRQ[29]	—						
G[10]	PCR[106]	SIUL	GPIO[106]	ALT0	GPIO[106]	—	—	M	S	—	77	P15
		FlexRay	DBG2	ALT1	—	—						
		DSPI_2	CS3	ALT2	—	—						
		FlexPWM_0	—	—	FAULT[2]	PSMI[18]; PADSEL=1						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
G[11]	PCR[107]	SIUL	GPIO[107]	ALT0	GPIO[107]	—	—	M	S	—	75	U15
		FlexRay	DBG3	ALT1	—	—						
		FlexPWM_0	—	—	FAULT[3]	PSMI[19]; PADSEL=2						
G[12]	PCR[108]	SIUL	GPIO[108]	ALT0	GPIO[108]	—	—	F	S	—	—	F2
		NPC	MDO[11]	ALT2	—	—						
G[13]	PCR[109]	SIUL	GPIO[109]	ALT0	GPIO[109]	—	—	F	S	—	—	H1
		NPC	MDO[10]	ALT2	—	—						
G[14]	PCR[110]	SIUL	GPIO[110]	ALT0	GPIO[110]	—	—	F	S	—	—	A6
		NPC	MDO[9]	ALT2	—	—						
G[15]	PCR[111]	SIUL	GPIO[111]	ALT0	GPIO[111]	—	—	F	S	—	—	J2
		NPC	MDO[8]	ALT2	—	—						
Port H												
H[0]	PCR[112]	SIUL	GPIO[112]	ALT0	GPIO[112]	—	—	F	S	—	—	A5
		NPC	MDO[7]	ALT2	—	—						
H[1]	PCR[113]	SIUL	GPIO[113]	ALT0	GPIO[113]	—	—	F	S	—	—	F1
		NPC	MDO[6]	ALT2	—	—						
H[2]	PCR[114]	SIUL	GPIO[114]	ALT0	GPIO[114]	—	—	F	S	—	—	A4
		NPC	MDO[5]	ALT2	—	—						
H[3]	PCR[115]	SIUL	GPIO[115]	ALT0	GPIO[115]	—	—	F	S	—	—	G1
		NPC	MDO[4]	ALT2	—	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
H[4]	PCR[116]	SIUL	GPIO[116]	ALT0	GPIO[116]	—	—	M	S	—	—	L16
		FlexPWM_1	X[0]	ALT1	X[0]	—						
		eTimer_2	ETC[0]	ALT2	ETC[0]	PSMI[39]; PADSEL=0						
H[5]	PCR[117]	SIUL	GPIO[117]	ALT0	GPIO[117]	—	—	M	S	—	—	M17
		FlexPWM_1	A[0]	ALT1	A[0]	—						
		DSPI_0	CS4	ALT3	—	—						
H[6]	PCR[118]	SIUL	GPIO[118]	ALT0	GPIO[118]	—	—	M	S	—	—	H17
		FlexPWM_1	B[0]	ALT1	B[0]	—						
		DSPI_0	CS5	ALT3	—	—						
H[7]	PCR[119]	SIUL	GPIO[119]	ALT0	GPIO[119]	—	—	M	S	—	—	K16
		FlexPWM_1	X[1]	ALT1	X[1]	—						
		eTimer_2	ETC[1]	ALT2	ETC[1]	PSMI[40]; PADSEL=0						
H[8]	PCR[120]	SIUL	GPIO[120]	ALT0	GPIO[120]	—	—	M	S	—	—	K15
		FlexPWM_1	A[1]	ALT1	A[1]	—						
		DSPI_0	CS6	ALT3	—	—						
H[9]	PCR[121]	SIUL	GPIO[121]	ALT0	GPIO[121]	—	—	M	S	—	—	G16
		FlexPWM_1	B[1]	ALT1	B[1]	—						
		DSPI_0	CS7	ALT3	—	—						
H[10]	PCR[122]	SIUL	GPIO[122]	ALT0	GPIO[122]	—	—	M	S	—	—	A11
		FlexPWM_1	X[2]	ALT1	X[2]	—						
		eTimer_2	ETC[2]	ALT2	ETC[2]	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
H[11]	PCR[123]	SIUL	GPIO[123]	ALT0	GPIO[123]	—	—	M	S	—	—	C11
		FlexPWM_1	A[2]	ALT1	A[2]	—						
H[12]	PCR[124]	SIUL	GPIO[124]	ALT0	GPIO[124]	—	—	M	S	—	—	B10
		FlexPWM_1	B[2]	ALT1	B[2]	—						
H[13]	PCR[125]	SIUL	GPIO[125]	ALT0	GPIO[125]	—	—	M	S	—	—	G15
		FlexPWM_1	X[3]	ALT1	X[3]	—						
		eTimer_2	ETC[3]	ALT2	ETC[3]	PSMI[42]; PADSEL=0						
H[14]	PCR[126]	SIUL	GPIO[126]	ALT0	GPIO[126]	—	—	M	S	—	—	A12
		FlexPWM_1	A[3]	ALT1	A[3]	—						
		eTimer_2	ETC[4]	ALT2	ETC[4]	—						
H[15]	PCR[127]	SIUL	GPIO[127]	ALT0	GPIO[127]	—	—	M	S	—	—	J17
		FlexPWM_1	B[3]	ALT1	B[3]	—						
		eTimer_2	ETC[5]	ALT2	ETC[5]	—						
Port I												
I[0]	PCR[128]	SIUL	GPIO[128]	ALT0	GPIO[128]	—	—	M	S	—	—	C9
		eTimer_2	ETC[0]	ALT1	ETC[0]	PSMI[39]; PADSEL=1						
		DSPI_0	CS4	ALT2	—	—						
		FlexPWM_1	—	—	FAULT[0]	—						

Table 12. Pin muxing (continued)

Port name	PCR	Peripheral	Alternate output function	Output mux sel	Input functions	Input mux select	Weak pull config during reset	Pad speed(1)		Pin #		
								SRC = 1	SRC = 0	100 pkg	144 pkg	257 pkg
I[1]	PCR[129]	SIUL	GPIO[129]	ALT0	GPIO[129]	—	—	M	S	—	—	C12
		eTimer_2	ETC[1]	ALT1	ETC[1]	PSMI[40]; PADSEL=1						
		DSPI_0	CS5	ALT2	—	—						
		FlexPWM_1	—	—	FAULT[1]	—						
I[2]	PCR[130]	SIUL	GPIO[130]	ALT0	GPIO[130]	—	—	M	S	—	—	F16
		eTimer_2	ETC[2]	ALT1	ETC[2]	PSMI[41]; PADSEL=1						
		DSPI_0	CS6	ALT2	—	—						
		FlexPWM_1	—	—	FAULT[2]	—						
I[3]	PCR[131]	SIUL	GPIO[131]	ALT0	GPIO[131]	—	—	M	S	—	—	E17
		eTimer_2	ETC[3]	ALT1	ETC[3]	PSMI[42]; PADSEL=1						
		DSPI_0	CS7	ALT2	—	—						
		CTU_0	EXT_TGR	ALT3	—	—						
		FlexPWM_1	—	—	FAULT[3]	—						
$\overline{\text{RDY}}$	PCR[132]	SIUL	GPIO[132]	ALT0	GPIO[132]	—	—	F	S	—	—	K3
		NPC	$\overline{\text{RDY}}$	ALT2	—	—						

1. Programmable via the SRC (Slew Rate Control) bit in the respective Pad Configuration Register; S = Slow, M = Medium, F = Fast, SYM = Symmetric (for FlexRay)
2. The default function of this pin out of reset is ALT1 (TDO).
3. Analog

Note: Open Drain can be configured by the PCRn for all pins used as output (except FCCU_F[0] and FCCU_F[1]).

4.5 Mapping of ports to PGPDO/I registers

Table 13. Mapping of ports to PGPDO registers

Port	PGPDO field(1)	Absolute address
A	PPDO[0]	0xC3F9_0C00
B	PPDO[1]	0xC3F9_0C02
C	PPDO[2]	0xC3F9_0C04
D	PPDO[3]	0xC3F9_0C06
E	PPDO[4]	0xC3F9_0C08
F	PPDO[5]	0xC3F9_0C0A
G	PPDO[6]	0xC3F9_0C0C
H	PPDO[7]	0xC3F9_0C0E
I	—(2)	0xC3F9_0680(2)

1. All fields are 16 bits.
2. Port I does not have an associated PGPDO register or field. To write data to port I, execute a 32-bit write access to GPDO128_131 at the absolute address shown.

Table 14. Mapping of ports to PGPDI registers

Port	PGPDI field(1)	Absolute address
A	PPDI[0]	0xC3F9_0C40
B	PPDI[1]	0xC3F9_0C42
C	PPDI[2]	0xC3F9_0C44
D	PPDI[3]	0xC3F9_0C46
E	PPDI[4]	0xC3F9_0C48
F	PPDI[5]	0xC3F9_0C4A
G	PPDI[6]	0xC3F9_0C4C
H	PPDI[7]	0xC3F9_0C4E
I	—(2)	0xC3F9_0880(2)

1. All fields are 16 bits.
2. Port I does not have an associated PGPDI register or field. To read data from port I, execute a 32-bit read access from GPDI128_131 at the absolute address shown.

5 Operating Modes

5.1 Overview

SPC56ELxx devices can operate in two modes of operation:

- Lock Step Mode (LSM)
- Decoupled Parallel Mode (DPM)

One of these two operating modes is statically selected at power-up (see *Section 5.4, Selecting LSM or DPM*). The selected operating mode may be changed only when going through a full power-on reset.

Both operating mode support a number of chip modes which are controlled by the Mode Entry Module (MC_ME). These chip modes differ from one another in:

- Which peripherals are enabled
- How the pins are configured
- How the clocks are configured
- Their relative safety status
- Their relative power consumption

See *Section 33.1, Introduction*, for a complete description of the chip modes.

5.2 Lock Step Mode (LSM)

This operating mode takes its name from the execution of the same commands by both cores in synchronicity (lock step). It has been implemented to allow reaching SIL3 with minimal software overhead and is the only operating mode of SPC56ELxx for which a SIL3 capability certificate has been planned.

In LSM, the Sphere of Replication (SoR) plays a major role. It contains all hardware elements which have been replicated for safety reasons resulting in the SoR being a collection of pairs. Each member of such a pair will execute the same operations or transactions as its partner resulting in lock step behavior. The compliance with this behavior expectation is checked only on the boundary of the SoR, minimizing checker effort.

This boundary check is based on a modified version of the fault isolation concept. Fault isolation requires that a fault must not cause failures outside a marked area, in this case the SoR. A failure in the SoR, as long as it does not propagate to the outside of the SoR and potentially cause a fault there, does not influence the effective operability of the periphery (and so the ECU). Thus it can not cause a hazard.

For example, an error in the ALU can cause wrong calculation results but as long as these results only influence core register values, they are not a hazard to the operation of the system. Also, propagation inside the SoR is of no immediate consequence, e.g. if the wrong register value is written to the INTC, this — in itself — will not change the overall behavior of the system. But once the registers are written somewhere external or used as addresses, or once the badly changed interrupt triggers, this 'safeness' changes because the failure now propagates to the outside of the SoR.

The Redundancy Control Checker Units (RCCU) at the outputs of the SoR to the periphery bus, to the Flash subsystem and to the SRAM subsystem detect such propagating failures due to data on the external busses being inconsistent between both processing units. Thus

the RCCUs implement the modified fault isolation in that they detect but not prevent the propagation of a non-common cause failure at the point where the two redundant channels are merged into a single actuator or recipient. Isolation of the overall system is then achieved by the Fault Collection and Control Unit (FCCU) signaling an error, thereby allowing the device or application to react appropriately.

Several MPC5643L e200 z4d core registers power up in a random state and therefore need to be initialized with known values before they can be used in the LSM configuration. The MPC5643L SRAM also powers up in a random state and also therefore needs to be initialized with known values before it can be used in an LSM configuration. The software is responsible to perform this special initialization.

The following MPC5643L e200z4 core registers require software initialization in LSM mode[e]:

General Registers — CR, CTR, LR, GPR0-GPR31, ACC

Debug Registers — DBCNT, DVC1-DVC2

Exception Handling — SPRG0-SPRG7, SPRG8-SPRG9, USPRG0, SRR0-SRR1, CSRR0-CSRR1, DSRR0-DSRR1, MCSRR0-MCSRR1, IVPR, IVOR0-IVOR15, IVOR32-IVOR34, MCAR, DEAR

Timers — DEC, DECAR, TBL, TBU

Memory Management Registers — MAS0-MAS4, MAS6

5.3 Decoupled Parallel Mode (DPM)

In this operating mode, each CPU core and connected channel runs independently from the other core, and the redundancy checkers (RCCU) are disabled.

At a given frequency, operating the chip in DPM offers a performance increase of approximately 1.6× over operating the chip in LSM.

In this operating mode, the chip boots with Core_0 enabled and Core_1 disabled directly after most resets. After a short external or short 'functional' reset, core1 is immediately enabled if it was enabled prior to the reset. Software running on Core_0 can enable Core_1 at any time, and once core1 has been enabled, it cannot be disabled by software. While Core_1 is disabled, it is not clocked, thus minimizing the chip's overall current consumption during this time.

See the tables in *Chapter 3: Memory Map*, for information on how memory is configured and accessed in DPM.

5.4 Selecting LSM or DPM

The operating mode (LSM or DPM) on SPC56ELxx is determined by the LSM_DPM user option bit in the shadow block of the flash memory. This user option bit is described in *Section 24.1.7, User option bits,* and is physically accessed using the UOPS[UOPT] field in the SSCM (see *Section 49.3.1.7, User Option Status Register (UOPS)*).

e. Refer to the e200z4 Core RM, Section 2.6, Table 2-16 “Reset Settings”

5.4.1 Entering LSM

By default, SPC56ELxx is configured to start in LSM (LSM_DPM = 1).

5.4.2 Entering DPM

5.4.2.1 Dual-core boot concepts

Entering DPM implies a dual-core boot. The key concept to a dual-core boot is that it is nothing more than a typical single-core boot, except that it starts another single-core boot. The initialization of interrupts, stack, and other parameters needs to be performed on each core. In other words, it is a single-core boot performed twice.

Figure 5 shows a simplification of this boot process.

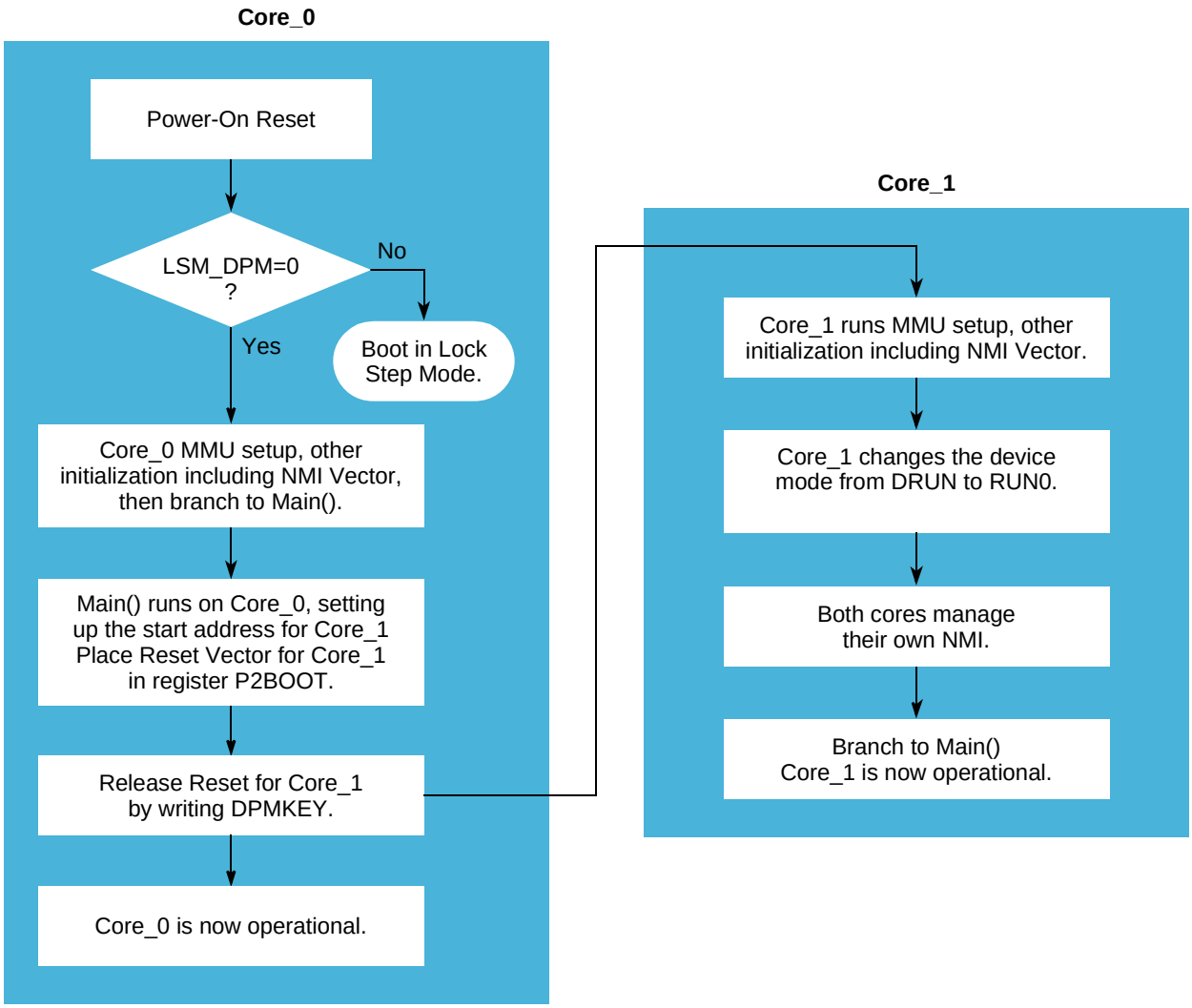


Figure 5. Simplified boot process in DPM

At power-on reset (POR), Core_0 begins operation while Core_1 remains held in reset. At this time, Core_0 must initialize its set of peripherals, set up its environment (including the

NMI routine), then branch to main. At this point, Core_0 is essentially fully operational. Now Core_0 provides the reset vector and writes the DPMKEY, thus releasing Core_1 from reset.

Core_1 begins its execution. The first thing that it must do is initialize its set of peripherals and set up its environment, including its NMI routine.

Core_0 then moves the chip from DRUN mode to RUN0 mode. Each core must service its own NMI routines. It is recommended to always use Core_0 to control the chip modes in order to avoid conflicting or inconsistent chip mode configurations and change requests.

5.4.2.2 Software setup

During the boot sequence, Core_0 runs from reset and executes code that sets up and enables Core_1. At that time, the system then begins operating in DPM.

To make the second core operational, you must configure the following two registers in the SSCM:

- DPMBOOT (see *Section 49.3.1.5, DPM Boot Register (DPMBOOT)*)
- DPMKEY (see *Section 49.3.1.6, Boot Key Register (DPMKEY)*)

Follow this sequence to enable DPM:

- Write the reset vector into DPMBOOT[P2BOOT].
- Set DPMBOOT[DVLE] to indicate that the second core will be executing in VLE mode. (Otherwise, the core will operate in Power Architecture mode.)
- Write 0x5AF0 to DPMKEY[KEY].
- Write 0xA50F to DPMKEY[KEY].

After the second write to DPMKEY[KEY], Core_1 starts execution from its reset vector. Repeating this sequence after Core_1 is running will have no effect on Core_1 until the next reset, and only then if the reset is a short external or short “functional” reset.

6 Device Boot Modes

6.1 Boot mode functionality

The device supports the following boot modes:

- Single Chip (SC) - the device will boot from the first bootable section of the flash memory main array
- Serial Boot (SBL) - the device will download boot code from either SCI or CAN interface and then execute it

If booting is not possible with the selected configuration (e.g. if no Boot ID is found in the selected boot location) then the device will enter static mode (see *Section 6.4.6, Static Mode*).

6.2 Hardware configuration

The device will detect the boot mode based on external pins and device status. The following sequence applies:

- If the FAB (Force Alternate Boot Mode) pin is set to boot in serial mode, the device can be forced into an Alternate Boot Loader Mode. The type of alternate boot mode is selected according to the ABS (Alternate Boot Selector) pins (see *Table 15*). For details of the serial boot modes please see *Chapter 10: Boot Assist Module (BAM)*.
- If the device identifies a flash memory sector with a valid boot signature, it will boot from the lowest sector. (See *Figure 6*.)
- If none of the flash memory sectors contains a valid boot signature, the device will go into static mode.

Table 15. Hardware configuration

FAB	ABS 2,0	Standby-RAM Boot Flag	Boot ID	Boot Mode
1	00	0	—	Serial Boot SCI
1	01	0	—	Serial Boot CAN
0	—	0	valid	SC (Single Chip)
0	—	0	not found	Static Mode

6.2.1 Single Chip boot mode

The SSCM performs a sequential search of each bootable sector (starting at sector 0) for a valid BOOT_ID within the RCHW. If a valid BOOT_ID is found in the RCHW of such a sector , the SSCM reads the VLE bit and the boot vector address. If a valid BOOT_ID is not found, the microcontroller is put into static mode.

In order for the e200z4d core to be able to access its program space memory , a valid MMU TLB entry has to be created. The SSCM does this automatically by reading the reset vector (the word after the RCHW) and modifying TLB entry 0 to create a 4 KB page containing the reset vector address. The corresponding MMU page must be properly aligned , which is achieved by setting the 12 LSB's of the reset vector address stored into this entry to 0.

The MMU VLE bit of this TLB entry is set depending on the status of the VLE bit within the RCHW. 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 the remaining program space, SRAM and peripherals
- Perform standard system initialization tasks (initialize 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.

6.2.1.1 Boot and alternate boot

Some applications require an alternate boot sector so that the main boot can be erased and reprogrammed in the field.

When an alternate boot is needed, user can create two bootable sectors; the lowest sector shall be the main boot sector and the highest shall be the alternate boot sector. The alternate boot sector does not need to be consecutive to the main boot sector.

This scheme ensures that there is always one active boot sector by erasing one of the boot sectors only:

- Sector is activated (that is, program a valid BOOT_ID instead of 0xFF as initially programmed).
- Sector is deactivated writing to 0 some of the bits BOOT-ID bit field (bit 1 and/or bit 3, and/or bit 4, and/or bit 6).

6.3 Boot-sector search

Note: The Boot-sector search do not run if the reset is programmed to behave as short reset.

For more details on this configuration, please refer Section 42.3.1.6, Functional Event Short Sequence Register (RGM_FESS).

6.3.1 Potential boot sectors

As shown in *Figure 6* in single chip mode the device will search several locations for a valid boot ID. The lowest sector which starts with a valid boot ID will be used to boot the device.

The Flash locations 0x0000_0000, 0x0000_4000, 0x0001_0000, 0x0001_C000, 0x0002_0000 and 0x0003_0000 will be searched.

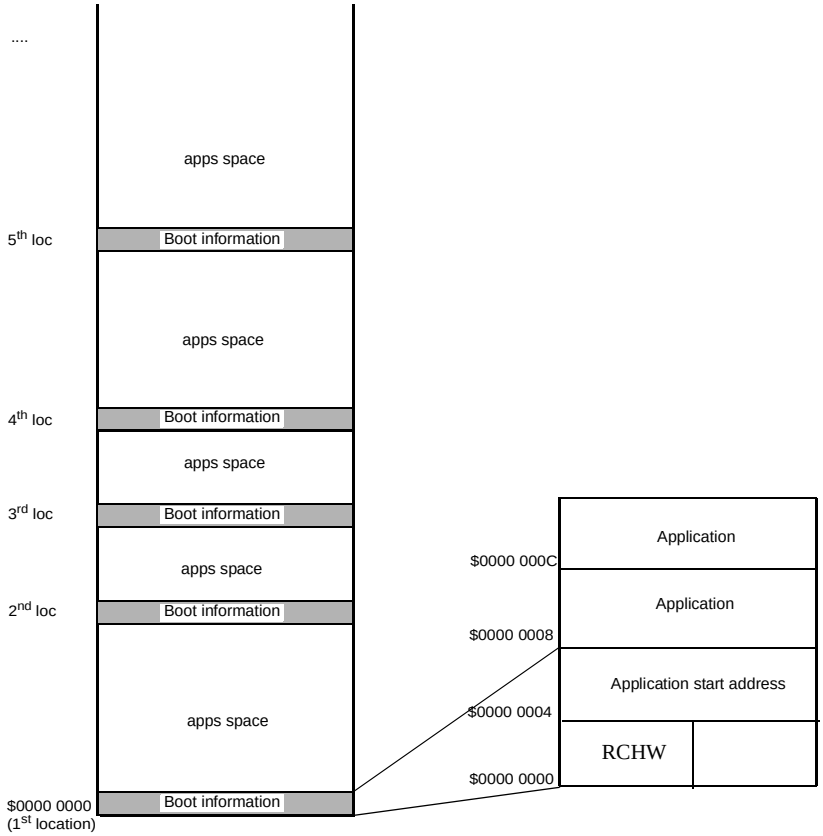


Figure 6. Flash partitioning and RCHW search

6.3.2 Reset Configuration Half-Word

Each boot sector in the flash memory contains at offset 0x00 the Reset Configuration Half-Word (RCHW). If the RCHW field BOOT_ID holds the value 0x5A then the sector is considered bootable. In addition there is a flag which indicates that the code is Book E or VLE code. This flag should be set to match the executable image. All other bits are reserved.

(No BAM code execution in this case, and BAM code will not be visible unless the device is in SBL mode.)

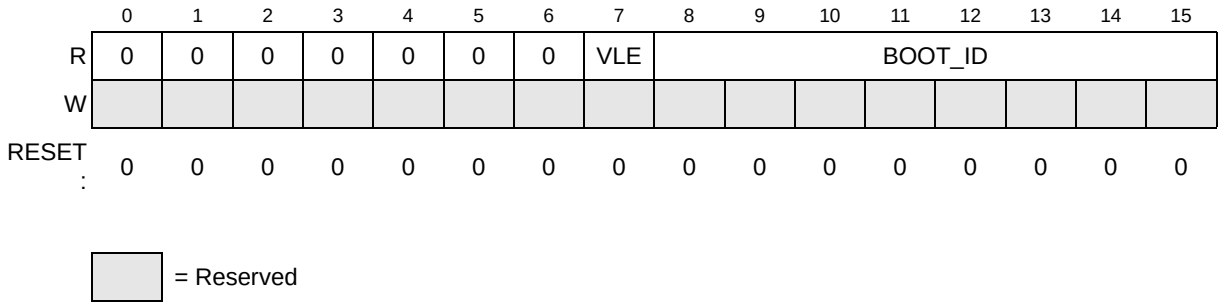


Figure 7. Reset Configuration Half Word (RCHW)

Once the device detects that it needs to boot from Flash, and finds a valid BOOT_ID, it will boot from the application start address at offset 0x04 within the boot sector.

6.3.3 Boot and alternate boot

Some applications require an alternate boot sector so that the main boot can be erased and reprogrammed in the field.

When an alternate boot is needed, the user can create two bootable sectors; the lowest sector shall be the main boot sector and the highest shall be the alternate boot sector. The alternate boot sector does not need to be consecutive to the main boot sector.

This scheme ensures that there is always one active boot sector by erasing one of the boot sectors only:

- Sector is activated (that is, program a valid BOOT_ID instead of 0xFF as initially programmed).
- Sector is deactivated writing to 0 some of the bits BOOT-ID bit field (bit 1 and/or bit 3, and/or bit 4, and/or bit 6).

6.4 Device behavior by boot mode

The following describes which security related device features are available in the device boot modes.

6.4.1 Single Chip Mode — Unsecured

In Normal Single Chip Mode, the system boots from the flash memory main array. The device will boot from the first sector which is marked bootable. The system may be configured to enable the external bus.

- Nexus/JTAG available
- Boot from flash memory main array
  - Shadow Block available

6.4.2 Single Chip Mode — Secured

In Secured Single Chip Mode, the system boots from the flash memory main array. The device will boot from the first sector which is marked bootable.

- No Nexus/JTAG apart from JTAG info for device and mask ID (major and minor) and commands to temporarily unlock the device with a valid Flash password.
- Boot from Flash main array
  - Shadow Block accessible

6.4.3 Serial Boot Loader Mode — Public Password Enabled

- In Serial Boot Loader Mode, if public serial access is allowed, the MCU executes BAM code which will check for a valid public password on the chosen interface. The interface will be selected via FAB and ABS pins.
- Nexus/JTAG available if device is not sensored.
- Boot from BAM
- Flash Main Array and Shadow Block access is disabled if device is censored.

6.4.4 Serial Boot Loader Mode — Flash Memory Password Enabled

It is possible to boot in Serial Boot Loader Mode when serial access with password is selected. The MCU executes BAM code which will check for a valid flash memory password on the chosen interface. If the password is known, full access to the device is enabled.

- Nexus/JTAG available once the correct password has been received.
- Boot from BAM
- Flash Main Array and Shadow Block access possible once the correct password has been received.

6.4.5 Standby Boot Mode

In Standby-Boot Mode, the MCU can be configured to boot from internal flash memory (Single Chip mode) or from Standby RAM. If the MCU boots from Standby RAM the reset sequence can be abridged - in this case the flash memory will not be available. (See RGM block guide for mode selection).

The device security status will be stored in the SSCM Standby area, so debugging will be possible if the device is unsecured.

6.4.6 Static Mode

Static mode means the device enters the SAFE mode and the processor executes a wait instruction. It is needed if the device can not boot in the mode which was selected. Access to Nexus and flash memory is as defined by the boot mode and the security status. Eventually the SWT will cause a reset and restart the boot search.

7 Device Security

7.1 Security

The censorship mechanism implemented on the device is intended to increase the protection against unauthorized access to the device. The mechanisms employed does not provide a guarantee of a secure device. It can only be deemed effective if used in conjunction with an appropriately robust software and documentation infrastructure to assist in guarding against unauthorized access to device resources.

When the flash memory is censored, the MCU is in the Secured state. This feature intends to prevent the unauthorized read and write of memory contents. Which device features are enabled or disabled is determined by the chip mode and security state in effect. Please refer to *Section 6.4, Device behavior by boot mode,* for details.

The user needs to keep in mind that part of the security must lie with the application code. As an extreme example: it would be possible to generate application code that dumps the contents of the internal program - obviously this code would defeat the purpose of security. However there may be good reasons to provide a back door in the application code. Care must be taken when implementing these backdoors, since they allow paths of attack to break security on the device.

7.1.1 Securing the microcontroller

The device can be secured by programming the flash memory NVSCI register (*Figure 8*; also see *Section 24.1.5.15, Nonvolatile System Censoring Information register (NVSCI)*). This non-volatile register will keep the device secured even when it is reset or powered down.

Figure 8. Nonvolatile System Censorship Information Register (NVSCI) for cut1

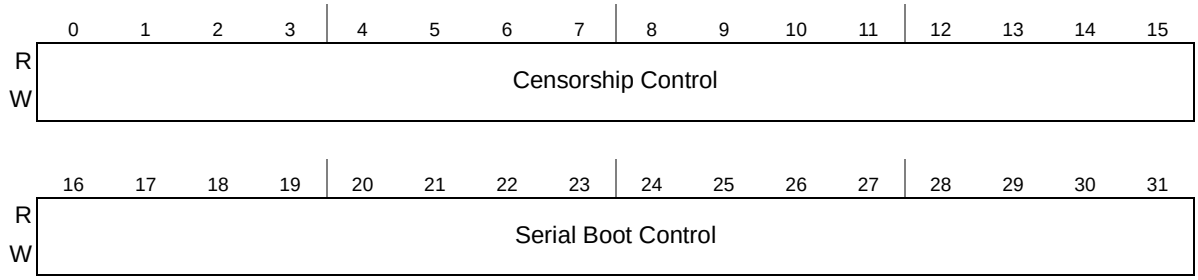
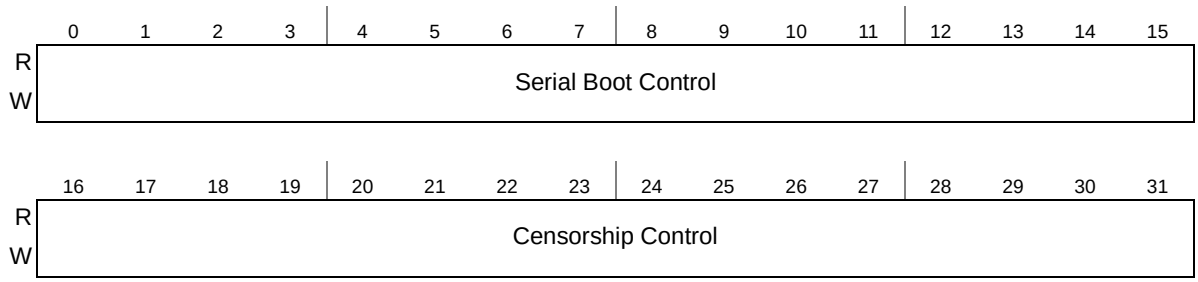


Figure 9. Nonvolatile System Censorship Information Register (NVSCI) for cut2/3



A value of 0x55AA in the censorship control word of the NVSCI register determines that the device is unsecured, any other value determines that the device is secured. The factory default of the NVSCI holds the value 0x55AA55AA. (For the function of the Serial Boot Control word see *Section 7.2, Serial access*.)

The device supports a backdoor to unsecure the device via the 64 bit password register (NVPWD) in the flash memory block. The flash memory password can be programmed or modified as long as the device is unsecured.

In order to modify an already programmed password the shadow block needs to be erased first, and all other configuration bits re-programmed as well.

To protect against voltage manipulations, each 16-bit halfword needs to contain both 1's and 0's. Below are examples for legal and illegal passwords:

Illegal passwords

```
0x0000_0000_0000_0000
0xFFFF_FFFF_FFFF_FFFF
0xFFFF_0000_FFFF_FFFF
0x0000_0000_0000_FFFF
0xAAAA_AAAA_AAAA_0000
```

Legal passwords

```
0x0001_0010_0100_1000
0xFFFE_FFFE_FFFE_FFFE
0xFFF0_000F_0FFF_0FFF
0x1000_1000_1000_FFFE
0xAAAA_AAAA_AAAA_0001
```

To deactivate the flash memory password ("swallowing the key") the register can be programmed with the value 0x0000_0000_0000_0000. However once this is done neither the manufacturer nor the user can unlock device security again. Of course, application code can still implement a different backdoor scheme if there are special requirements which are not covered by the available mechanism.

7.1.2 Unsecuring the microcontroller

To unsecure a secured device, the flash memory password needs to be provided. This can be done by booting in SBL mode and providing the flash memory password via the serial bootloader protocol (see refer to the BAM chapter) or via JTAG command.

If SBL mode was used in conjunction with the flash memory password, the password comparison result will only be known after a delay - this is in order to avoid brute-force attacks. The data can be downloaded into SRAM during this delay.

Note: The scheme used here does not prohibit a malicious listener from capturing the password of a valid serial download, since the key is not in encrypted form. The user must take appropriate measures to protect against access by third parties during a valid download.

If the password is correct, the device is temporarily unsecured. In this state the a new flash memory password can be programmed into the NVPWD registers or new application data can be programmed into the main flash memory array. (In order to modify an already programmed password the shadow block needs to be erased first, and all other configuration bits re-programmed as well.)

It is also possible to unlock the device via JTAG. For this the device needs to be held in reset by pulling the external reset input, once the flash memory has completed its internal sequence, the JTAG register CENSOR_CTRL can be written with the password in bits 63:0 and with bit 64 set to 1. The password comparator will compare the password and unsecure the device if it is correct, if serial access with the flash memory password is allowed, and if the device hasn't swallowed the key. (Only one transition on bit 64 from 0 to 1 is allowed.) The debugger needs to wait until the device is unlocked, after that a breakpoint can be set if desired, and the debugger can release the reset. The device will remain unsecured until the next reset event occurs.

7.1.2.1 Software unsecure

Since the security state of the device is determined solely by a user programmable location in the Shadow Block, any application may choose to implement a software unsecure method, through any interface.

7.2 Serial access

The device can be accessed via a serial interface, if it is booted in SBL mode (see *Section 6.2, Hardware configuration*). It is possible to either use the public password or the flash memory password — this is decided by programming the Serial Boot Control word in the NVSCI register (see *Figure 8*). If it contains the value 0x55AA, the flash memory password must be used, if it contains any other value the public password must be used.

Access to the flash memory depends both on the Serial Boot Control field and the Censorship Control Field in the NVSCI. An unsecured device always allows access to the flash memory regardless of the Serial Boot Control field. For a secured device if the public password access is used, the flash memory will not be visible, if flash memory password access is used then device security is disabled and the flash memory is visible.

The application may wish to prohibit access via serial line - this can be accomplished by programming NVSCI to 0x55AA which mandates the use of the flash memory password, and programming NVPWD to 0x0000_0000_0000_0000 which makes the flash memory password invalid.

However it should be carefully evaluated whether this scenario is desirable. If programmed like this there is no longer the possibility to allow the manufacturer to run diagnostic on a returned device. Similarly it won't be possible anymore to update the application. This does not apply if the user decides to implement an alternative backdoor scheme in software (see *Section 7.1.2.1, Software unsecure*).

8 Functional Safety

8.1 Overview

This device offers a set of features to support using it for applications which need to fulfill functional safety requirements as defined by safety integrity levels SIL3 of IEC61508 or ASILD of ISO26262. Also, the development processes and documentation of this device target these safety standards. This device is considered a Type B subsystem (“complex”, see IEC 61508-2, section 7.4.3.1. Architectural constraints on hardware safety integrity) and is assumed to be used in a “High-Demand / Continuous mode of operation“ safety mode (see IEC 61508-1, section 7.6.2, “Requirements”).

8.2 Redundancy

The main approach used to achieve functional safety requirements is redundancy. Redundancy is applied in different ways for different modules of this device:

- Processing cores: When used for a safety critical application, the two redundant cores must be used in lock-step mode. Any difference between the outputs of the cores indicates a fault and triggers the according reaction to prevent propagation of the fault and to put the device into a Fail-Safe mode.
- Replicated peripherals, if safety critical for the application, have to be used in a redundant way by the application software. Details are specified in the Safety Application Guide.
- Non-replicated input peripherals, if safety critical for the application and not self-tested, have to be read twice by the application software. Details are specified in the Safety Application Guide.
- Non-replicated output peripherals, if safety critical for the application and not read-back, have to be written twice by the application software. Details are specified in the Safety Application Guide.

8.3 Built-In Self-Test (BIST)

8.3.1 BIST during boot

A device BIST is performed every time a destructive or external reset (long) occurs. The device provides full reset conditions to the outside world while BIST is executed. The BIST is performed transparently for the application while the device is still under reset. In case the BIST fails, the Fault Collection and Control Unit (FCCU) puts the device into the configured Fail-Safe state accordingly. Application software can only start to run when BIST finished successfully without detecting a fault. The boot time BIST comprises:

- Memory BIST for all RAMs and ROM
- Scan-based Logic BIST for three partitions of digital logic (for the contents of the individual partitions see hierarchy definition).

8.3.2 Software-triggered BIST during operation

For some modules of this device it is required to run BIST during operation, because testing every 10 hours (Trip Time) is not sufficient for the targeted safety integrity level. These BIST runs need to be triggered by application software once within each Process Safety Time (10 ms). Details are specified in the Safety Application Guide.

Modules which require BIST runs during application are:

- ADC: Software triggers several self-tests implemented in hardware
- Flash memory ECC logic test

8.3.3 Software-triggered self-tests after boot

In order to ensure integrity of flash memory for a safety application, hardware based flash self-test needs to be triggered by software every time the device is booted. Details are specified in the Safety Application Guide.

8.4 Memory error detection and correction

RAMs are protected against soft errors by a 7-bit/32 SEC/DED ECC code computed over address and data at each memory access. Detected faults are reported to the FCCU.

NVM Flash is protected by 8-bit/64 parity SEC/DED.

8.5 Monitoring

All monitoring features react within the so-called Process Safety Time of 10 ms or faster.

Core voltage is equipped with a low voltage detector and a high voltage detector to indicate if voltage is out of the specified range. IO voltage is monitored by a low voltage detector. The voltage detectors themselves have a hardware based self-checking feature.

This device is equipped with two temperature sensors.

Four clocks are monitored:

- Internal oscillator clock
- FMPLL-generated system clock
- FlexRay clock
- SMC clock

A Memory Protection Unit prevents incorrect memory accesses based on 16 possibly overlapping physical memory regions.

8.6 Software measures

The Safety Application Guide specifies several software measures required to achieve safety integrity for this device. Software has to trigger these test features at least every 10 ms (Process Safety Time). These are simple checks. No software based self-test routine library is necessary for this device.

The following shows some examples for software checks. The complete list of software measures is defined in the Safety Application Guide.

Example 1: Modules which require CRC checks during operation are:

- SIUL: System configuration registers checked
- ADC: Configuration registers checked
- eTimer: Configuration registers checked

Example 2: The CTU unit requires the following checks:

- Have all triggers been generated and served (supported by hardware, faults to be handled in SW)?
- Do trigger times match expected behavior?
- Is there a trigger buffer overrun (supported by hardware, faults to be handled in SW)?
- Does the channel number sequence match expected behavior?
- Are the issued commands valid (supported by hardware, faults to be handled in SW)?

8.7 Fault reaction

All faults detected by hardware measures like the redundancy checkers, self-test features, ECC, voltage and clock monitors are reported to the central Fault Collection and Control Unit (FCCU). Depending on the particular fault, the FCCU puts the device into the according configured Fail-Safe state. This prevents fault propagation to system level. By definition, the Safe State of this chip is either of the following:

- I/Os in tristate when the device is in shutdown or reset
- Device flagging an Error Out for critical errors

The continuous switch between a standard operation state and the reset state without any shut down is not considered a Safe State.

Safety critical IOs are kept in tri-state when the device is in a Fail-Safe state.

8.8 External measures

This chip requires several external measures to allow safe operation:

- External power supply and monitor
- External watchdog timer
- Error out monitor to handle situations where this device indicates an internal fault
- PWM output monitor that monitors and corrects the PWM outputs

9 Analog-to-Digital Converter (ADC)

9.1 Introduction

The Analog to Digital Converter (ADC) is a 12 bit Successive Approximation register (SAR) ADC with a mixed capacitive/resistive DAC. Sampling and conversion durations are software programmable, as are other features such as the conversion characteristics offset. Maximum input frequency is 500KHz, but to reach a SINAD of 65dB an input frequency of 10KHz maximum is required. The ADC architecture allows input channel multiplexing. The maximum operating frequency depends on the ADC settings; a clock of 20MHz can be applied regardless of the configuration values. Larger clock frequencies can be applied for some specific setting values and provided the sampling error is smaller than the requested precision. The ADC also implements self-test features to detect if the ADC is not operating properly.

This device includes two ADC modules, referred to as ADC_0 and ADC_1.

External ADC channels:

- 9 external channels on ADC_0 (channels 0..8)
- 9 external channels on ADC_1 (channels 0..8)
- 4 external channels shared between ADC_0 and ADC_1 (channels 11..14)

The internal connections of peripherals to ADC channels are as follows:

- TSENS_0 = ADC_0 channel 15
- TSENS_1 = ADC_1 channel 15
- VREG_1.2V = ADC_0 channel 10 and ADC_1 channel 10

Note: ADC_0/1 channel 9 is reserved for factory test only.

9.2 Features

The ADC contains advanced features for normal or injected conversion. A conversion can be triggered by software or hardware (CTU).

The ADC on SPC56ELxx supports 16 internally-multiplexed precision channels (ANS), all of which have the same accuracy.

The mask registers present within the ADC can be programmed to configure which channel has to be converted.

Conversion timing registers are used to configure different sampling and conversion times. On SPC56ELxx, two such registers are provided — one for configuring the precision channels (0–14) and one for the temperature sensor channel.

Analog watchdogs allow continuous hardware monitoring.

Features:

- 12 bit resolution
- Reference voltage: from 3V to 5.5V
- Supply voltage: from 3V to 3.6V
- Maximum clock frequency: from 20MHz to 60MHz depending on configuration settings
- Minimum clock frequency: from 3MHz to 12MHz depending on configuration settings
- Sampling time @60MHz: programmable from 383ns to 4.25us(f)
- Conversion time@60MHz: programmable from 1us to 4.87us (sampling time included)[1]
- Conversion characteristic shift: 0, 1/2 and 1 Lsb
- Software controlled Power-down
- Self test feature.

9.3 Memory map and register descriptions

Table 16. Bit access descriptions

Access type	Description
read/write (rw)	Software can read and write to these bits.
read-only (r)	Software can only read these bits.
write-only (w)	Software can only write to these bits.
write 1 to clear (w1c)	Software can clear bits by writing ‘1’.

9.3.1 Memory map

Table 17. ADC memory map

Address offset	Register	Location
0x000	Main Configuration Register (MCR)	*on page 183*
0x004	Main Status Register (MSR)	*on page 185*
0x008–0x00C	Reserved	
0x010	Interrupt Status Register (ISR)	*on page 186*
0x014	Channel Pending Register 0 (CEOCFR0)	*on page 187*
0x018–0x01C	Reserved	
0x020	Interrupt Mask Register (IMR)	*on page 188*
0x024	Channel Interrupt Mask Register 0 (CIMR0)	*on page 189*
0x028–0x02C	Reserved	
0x030	Watchdog Threshold Interrupt Register (WTISR)	*on page 189*
0x034	Watchdog Threshold Interrupt Mask Register (WTIMR)	*on page 190*

f. These timings do not consider any variation in the clock frequency such as spread spectrum PLL, jitter controlled oscillators, etc.

Table 17. ADC memory map (continued)

Address offset	Register	Location
0x038–0x03C	Reserved	
0x040	DMA Enable Register (DMAE)	*on page 191*
0x044	DMA Channel Select Register 0 (DMAR0)	*on page 191*
0x048–0x05C	Reserved	
0x060	Threshold Register 0 (THRHLR0)	*on page 192*
0x064	Threshold Register 1 (THRHLR1)	*on page 192*
0x068	Threshold Register 2 (THRHLR2)	*on page 192*
0x06C	Threshold Register 3 (THRHLR3)	*on page 192*
0x070–0x07C	Reserved	
0x080	Presampling Control Register (PSCR)	*on page 192*
0x084	Presampling Register 0 (PSR0)	*on page 193*
0x088–0x090	Reserved	
0x094	Conversion Timing Register 0 (CTR0)	*on page 194*
0x098	Conversion Timing Register 1 (CTR1)	*on page 195*
0x09C–0x0A0	Reserved	
0x0A4	Normal Conversion Mask Register 0 (NCMR0)	*on page 196*
0x0A8–0x0B0	Reserved	
0x0B4	Injected Conversion Mask Register 0 (JCMR0)	*on page 196*
0x0B8–0x0C4	Reserved	
0x0C8	Power Down Exit Delay Register (PDEDR)	*on page 197*
0x0CC–0x0FC	Reserved	
0x100	Channel 0 Data Register (CDR0)	*on page 197*
0x104	Channel 1 Data Register (CDR1)	*on page 197*
0x108	Channel 2 Data Register (CDR2)	*on page 197*
0x10C	Channel 3 Data Register (CDR3)	*on page 197*
0x110	Channel 4 Data Register (CDR4)	*on page 197*
0x114	Channel 5 Data Register (CDR5)	*on page 197*
0x118	Channel 6 Data Register (CDR6)	*on page 197*
0x11C	Channel 7 Data Register (CDR7)	*on page 197*
0x120	Channel 8 Data Register (CDR8)	*on page 197*
0x124	Channel 9 Data Register (CDR9)	*on page 197*
0x128	Channel 10 Data Register (CDR10)	*on page 197*
0x12C	Channel 11 Data Register (CDR11)	*on page 197*
0x130	Channel 12 Data Register (CDR12)	*on page 197*
0x134	Channel 13 Data Register (CDR13)	*on page 197*

Table 17. ADC memory map (continued)

Address offset	Register	Location
0x138	Channel 14 Data Register (CDR14)	on page 197
0x13C	Channel 15 Data Register (CDR15)	on page 197
0x140–0x280	Reserved	
0x280	Threshold Register 4 (THRHLR4)	on page 192
0x284	Threshold Register 5 (THRHLR5)	on page 192
0x288	Threshold Register 6 (THRHLR6)	on page 192
0x28C	Threshold Register 7 (THRHLR7)	on page 192
0x290	Threshold Register 8 (THRHLR8)	on page 192
0x294	Threshold Register 9 (THRHLR9)	on page 192
0x298	Threshold Register 10 (THRHLR10)	on page 192
0x29C	Threshold Register 11 (THRHLR11)	on page 192
0x2A0	Threshold Register 12 (THRHLR12)	on page 192
0x2A4	Threshold Register 13 (THRHLR13)	on page 192
0x2A8	Threshold Register 14 (THRHLR14)	on page 192
0x2AC	Threshold Register 15 (THRHLR15)	on page 192
0x2B0	Channel Watchdog Selection Register 0 (CWSEL0)	on page 198
0x2B4	Channel Watchdog Selection Register 1 (CWSEL1)	on page 198
0x2B8–0x2DC	Reserved	
0x2E0	Channel Watchdog Enable Register 0 (CWENR0)	on page 199
0x2E4–0x2EC	Reserved	
0x2F0	Analog Watchdog Out of Range Register 0 (AWORR0)	on page 200
0x2F4–0x33C	Reserved	
0x340	Self Test Configuration Register 1 (STCR1)	on page 200
0x344	Self Test Configuration Register 2 (STCR2)	on page 201
0x348	Self Test Configuration Register 3 (STCR3)	on page 203
0x34C	Self Test Baud Rate Register (STBRR)	on page 204
0x350	Self Test Status Register 1 (STSR1)	on page 205
0x354	Self Test Status Register 2 (STSR2)	on page 207
0x358	Self Test Status Register 3 (STSR3)	on page 208
0x35C	Self Test Status Register 4 (STSR4)	on page 208
0x360–0x36C	Reserved	
0x370	Self Test Data Register 1 (STDR1)	on page 209
0x374	Self Test Data Register 2 (STDR2)	on page 210
0x378–0x37C	Reserved	
0x380	Self Test Analog Watchdog Register 0 (STAW0R)	on page 210

Table 17. ADC memory map (continued)

Address offset	Register	Location
0x384	Self Test Analog Watchdog Register 1A (STAW1AR)	*on page 211*
0x388	Self Test Analog Watchdog Register 1B (STAW1BR)	*on page 212*
0x38C	Self Test Analog Watchdog Register 2 (STAW2R)	*on page 212*
0x390	Self Test Analog Watchdog Register 3 (STAW3R)	*on page 213*
0x394	Self Test Analog Watchdog Register 4 (STAW4R)	*on page 213*
0x398	Self Test Analog Watchdog Register 5 (STAW5R)	

9.3.2 Control logic registers

9.3.2.1 Main Configuration Register (MCR)

The Main Configuration Register (MCR) provides configuration settings for the ADC.

Figure 10. Main Configuration Register (MCR)

Address: Base + 0x000 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	REF_RANGE_EXP	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	STCL	0	0	0	0	0	0	ADCLK SEL	ABORT CHAIN	ABORT	ACKO	0	0	0	0	PWDN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 18. MCR field descriptions

Field	Description
OWREN	Overwrite enable This bit enables or disables the functionality to overwrite unread converted data. 0 Prevents overwrite of unread converted data; new result is discarded. 1 Enables converted data to be overwritten by a new conversion.
WLSIDE	Write left/right-aligned 0 The conversion data is written right-aligned. 1 Data is left-aligned (from 15 to (15 – resolution + 1)).
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.

Table 18. MCR field descriptions (continued)

Field	Description
NSTART	Normal Start conversion Setting this bit starts the chain or scan conversion. Clearing 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 (injected conversion cannot be started using an external signal). 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. Clearing this bit has no effect, as the injected chain conversion cannot be interrupted.
REF_RANGE_EXP	This is a control bit programmed by the user, which specifies the expected value of MSR[REF_RANGE]. If the expected value does not match with the actual value, an ISR bit is set.
CTUEN	Cross Trigger Unit Enable 0 The cross triggering unit is disabled and the triggered injected conversion cannot occur. 1 The cross triggering unit is enabled and the triggered injected conversion can occur.
STCL	Self Testing Configuration Lock 0 No lock 1 The self-testing configuration is locked i.e. STCR1, STCR2, STCR3, STBRR, STAW0R, STAW1AR, STAW1BR, STAW2R, STAW3R, STAW4R, STAW5R are write-protected. It can be used only in CPU and SCAN mode. The lock bit is cleared only by a peripheral reset.
ADCLKSEL	Analog clock frequency selector If this bit is set, the AD_clk frequency is equal to the motor control clock frequency. Otherwise, it is half of the motor control clock frequency. This bit can be written in power-down mode only.
ABORTCHAIN	Abort Chain When this bit is set, the ongoing chain conversion is aborted. This bit is cleared 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 cleared by hardware as soon as a new conversion is invoked. 0 Conversion is not affected. 1 Aborts the ongoing conversion.

Table 18. MCR field descriptions (continued)

Field	Description
ACKO	Auto-clock-off enable If set, this bit enables the Auto clock off feature. 0 Auto clock off is disabled. 1 Auto clock off is enabled.
PWDN	Power-down enable When this bit is set, the analog module is requested to enter Power Down mode. When ADC status is PWDN, clearing this bit starts ADC transition to idle mode. 0 ADC is in normal mode. 1 ADC has been requested to power down.

9.3.2.2 Main Status Register (MSR)

The Main Status Register (MSR) provides status bits for the ADC.

Figure 11. Main Status Register (MSR)

Address: Base + 0x004 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	REF_RANGE	SELF_TEST_S	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	ACKO	0	0	ADCSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 19. 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.
REF_RANGE	This bit defines the voltage range for operation of the ADC. It is provided as an output by the ADC, along with data, after every conversion. 0 Reference voltage is less than or equal to 3.6 V 1 Reference voltage is greater than 4.5 V *Note: If the reference voltage is outside of the ranges specified here, the value of REF_RANGE can be 0 or 1 depending on the supply, process, and temperature.*

Table 19. MSR field descriptions (continued)

Field	Description
SELF_TEST_S	This Status bit signals that self test conversion is ongoing. 0 Self test conversion is complete. 1 Self test conversion is going on.
CTUSTART	This status bit is used to signal that a CTU conversion is ongoing. This bit is set when a CTU trigger pulse is received and the CTU conversion starts. Otherwise, if Control Mode is enabled this bit is reset when the CTU is disabled (CTUEN set to '0').
CHADDR	Channel under measure address This status bit is used to signal which channel is under measure.
ACKO	Auto-clock-off enable This status bit is used to signal if the Auto-clock-off feature is on.
ADCSTATUS	The value of this parameter depends on ADC status: 000 Idle 001 Power-down 010 Wait state 011 — 100 Sample 101 — 110 Conversion 111 —

9.3.3 Interrupt registers

9.3.3.1 Interrupt Status Register (ISR)

The Interrupt Status Register (ISR) contains interrupt status bits for the ADC.

Figure 12. Interrupt Status Register (ISR)

Address: Base + 0x010 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	REF_RANGE	0	0	0	0	0	0	0	0	0	0	EOCTU	JEOC	JECH	EOC	ECH
W	w1c											w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Interrupt signals EOCTU, JEOC, JECH EOC, and ECH (along with interrupt signals WDG_EOA_S, WDG_EOA_RC and WDG_EOA_C in the STSR1 register) are routed to the INTC as the ADC_EOC interrupt source. Interrupt signal REF_RANGE (along with interrupt

signals WDSERR, WDTERR, ERR_C, ERR_RC, ERR_S2, ERR_S1, ERR_S0 in the STSR1 register) is routed to the INTC as the ADC_ER interrupt source.

Table 20. ISR field descriptions

Field	Description
REF_RANGE	This bit is set if REF_RANGE output from the ADC does not match with the expected value programmed in the MCR.
EOCTU	End of CTU Conversion interrupt flag. It is the interrupt of the digital end of conversion for the CTU channel; active when set. When this bit is set, an EOCTU interrupt has occurred.
JEOC	End of Injected Channel Conversion interrupt flag. It is the interrupt of the digital end of conversion for the injected channel; active when set. When this bit is set, a JEOC interrupt has occurred.
JECH	End of Injected Chain Conversion interrupt flag. It is the interrupt of the digital end of chain conversion for the injected channel; active when set. When this bit is set, a JECH interrupt has occurred.
EOC	End of Channel Conversion interrupt flag. It is the interrupt of the digital end of conversion. When this bit is set, an EOC interrupt has occurred.
ECH	End of Chain Conversion interrupt flag. It is the interrupt of the digital end of chain conversion. When this bit is set, an ECH interrupt has occurred.

9.3.3.2 Channel Pending Register 0 (CEOCFR0)

Figure 13. Channel Pending Register 0 (CEOCFR0)

Address: Base + 0x014 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EOC_CH15	EOC_CH14	EOC_CH13	EOC_CH12	EOC_CH11	EOC_CH10	EOC_CH9	EOC_CH8	EOC_CH7	EOC_CH6	EOC_CH5	EOC_CH4	EOC_CH3	EOC_CH2	EOC_CH1	EOC_CH0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 21. CEOCFR0 field descriptions

Field	Description
EOC_CH*n*	This field indicates the end of conversion. 0 The measure of channel *n* is not complete. 1 The measure of channel *n* is complete.

9.3.3.3 Interrupt Mask Register (IMR)

The Interrupt Mask Register (IMR) contains the interrupt enable bits for the ADC.

Figure 14. Interrupt Mask Register (IMR)

Address: Base + 0x020 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	MSK_REF_RANGE	0	0	0	0	0	0	0	0	0	0	MSKEOCTU	MSKJEOC	MSKJECH	MSKEOC	MSKECH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 22. IMR field descriptions

Field	Description
MSK_REF_RANGE	When set interrupt corresponding to REF_RANGE bit in ISR is enabled.
MSKEOCTU	Mask bit for EOCTU When set, the EOCTU interrupt is enabled.
MSKJEOC	Mask bit for JEOC When set, the JEOC interrupt is enabled.
MSKJECH	Mask bit for JECH When set, the JECH interrupt is enabled.
MSKEOC	Mask bit for EOC When set, the EOC interrupt is enabled.
MSKECH	Mask bit for ECH When set, the ECH interrupt is enabled.

9.3.3.4 Channel Interrupt Mask Register 0 (CIMR0)

Figure 15. Channel Interrupt Mask Register 0 (CIMR0)

Address: Base + 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	CIM 15	CIM 14	CIM 13	CIM 12	CIM 11	CIM 10	CIM 9	CIM 8	CIM 7	CIM 6	CIM 5	CIM 4	CIM 3	CIM 2	CIM 1	CIM 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 23. CIMR0 field descriptions

Field	Description
CIM*n*	This field enables the interrupt for channel *n*. 0 Interrupt for channel *n* is disabled. 1 Interrupt for channel *n* is enabled.

9.3.4 Watchdog Threshold Interrupt Status Register (WTISR)

Figure 16. Watchdog Threshold Interrupt Status Register (WTISR)

Address: Base + 0x030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WDG 15H	WDG 15L	WDG 14H	WDG 14L	WDG 13H	WDG 13L	WDG 12H	WDG 12L	WDG 11H	WDG 11L	WDG 10H	WDG 10L	WDG 9H	WDG 9L	WDG 8H	WDG 8L
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	WDG 7H	WDG 7L	WDG 6H	WDG 6L	WDG 5H	WDG 5L	WDG 4H	WDG 4L	WDG 3H	WDG 3L	WDG 2H	WDG 2L	WDG 1H	WDG 1L	WDG 0H	WDG 0L
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 24. WTISR field descriptions

Field	Description
WDG*n*L	Reading this field indicates whether an interrupt has been generated due to the converted value being lower than the programmed lower threshold. 0 Converted value is higher or equal to the programmed lower threshold (no interrupt generated) 1 Converted value is lower than the programmed lower threshold (interrupt is generated)
WDG*n*H	Reading this field indicates whether an interrupt has been generated due to the converted value being higher than the programmed higher threshold. 0 Converted value is lower or equal to the programmed higher threshold (no interrupt generated) 1 Converted value is higher than the programmed higher threshold (interrupt is generated)

9.3.5 Watchdog Threshold Interrupt Mask Register (WTIMR)

Figure 17. Watchdog Threshold Interrupt Mask Register (WTIMR)

Address: Base + 0x034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MSKWDG15H	MSKWDG15L	MSKWDG14H	MSKWDG14L	MSKWDG13H	MSKWDG13L	MSKWDG12H	MSKWDG12L	MSKWDG11H	MSKWDG11L	MSKWDG10H	MSKWDG10L	MSKWDG9H	MSKWDG9L	MSKWDG8H	MSKWDG8L
Reset	0	0	0	0	0	0	0	0	0	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	MSKWDG7H	MSKWDG7L	MSKWDG6H	MSKWDG6L	MSKWDG5H	MSKWDG5L	MSKWDG4H	MSKWDG4L	MSKWDG3H	MSKWDG3L	MSKWDG2H	MSKWDG2L	MSKWDG1H	MSKWDG1L	MSKWDG0H	MSKWDG0L
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

All Watchdog interrupt signals are routed to the INTC as the ADC_WD interrupt source.

Table 25. WTIMR field descriptions

Field	Description
MSKWDG*n*L	Mask bit for the interrupt generated due to the converted value being lower than the programmed lower threshold. 0 Interrupt is not enabled 1 Interrupt is enabled
MSKWDG*n*H	Mask bit for the interrupt generated due to the converted value being higher than the programmed higher threshold. 0 Interrupt is not enabled 1 Interrupt is enabled

9.3.6 DMA Enable Register (DMAE)

Figure 18. DMA Enable Register (DMAE)

Address: Base + 0x040 Access: User read/write

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

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

Table 26. DMAE field descriptions

Field	Description
DCLR	DMA clear sequence enable 0 DMA request is cleared by an acknowledge from DMA controller 1 DMA request is cleared when a read of data registers occurs
DMAEN	DMA global enable 0 The DMA feature is disabled 1 The DMA feature is enabled

9.3.7 DMA Channel Select Register 0 (DMAR0)

Figure 19. DMA Channel Select Register 0 (DMAR0)

Address: Base + 0x044 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	DMA 15	DMA 14	DMA 13	DMA 12	DMA 11	DMA 10	DMA 9	DMA 8	DMA 7	DMA 6	DMA 5	DMA 4	DMA 3	DMA 2	DMA 1	DMA 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 27. DMAR0 field descriptions

Field	Description
DMA*n*	DMA enable 0 DMA transfer for channel *n* is disabled 1 Channel *n* is enabled to transfer data in DMA mode

9.3.8 Threshold Registers (THRHLR*n*)

Figure 20. Threshold Registers (THRHLR*n*)

Address: See *Table 17* 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

Table 28. THRHLR*n* field descriptions

Field	Description
THRH	High threshold value for channel n (see *Section 9.4.6, Programmable analog watchdog*)
THRL	Low threshold value for channel n (see *Section 9.4.6, Programmable analog watchdog*)

9.3.9 Presampling registers

9.3.9.1 Presampling Control Register (PSCR)

Figure 21. Presampling Control Register (PSCR)

Address: Base + 0x080 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	PREVAL1		PREVAL0		PRECONV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 29. PSCR field descriptions

Field	Description
PREVAL1	Internal voltage selection for presampling of internal channels 9, 10, and 15 (see *Table 30*) 00 Select the V0 internal presampling voltage 01 Select the V1 internal presampling voltage 10 Reserved 11 Reserved
PREVAL0	Internal voltage selection for presampling of external channels 0-8, and 11-14 (see *Table 30*) 00 Select the V0 internal presampling voltage 01 Select the V1 internal presampling voltage 10 Reserved 11 Reserved
PRECONV	Convert presampled value 0 The ADC will perform a sampling followed by a conversion 1 The ADC will perform a presampling followed by a conversion (the sampling is bypassed, so the conversion result is that of the presampled value)

Table 30. Presampling voltages

PSCR[PREVAL0/1] voltage label	Voltage for ADC_0	Voltage for ADC_1
V0	VDD_HV_ADR0	VDD_HV_ADR1
V1	VSS_HV_ADR0	VSS_HV_ADR1

9.3.9.2 Presampling Register 0 (PSR0)

Figure 22. Presampling Register 0 (PSR0)

Address: Base + 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	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	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

Table 31. PSR0 field descriptions

Field	Description
PRES*n*	Presampling enable 0 Presampling for channel *n* is disabled 1 Presampling for channel *n* is enabled

9.3.10 Conversion timing registers

9.3.10.1 Conversion Timing Register 0 (CTR0)

This register configures the conversion timing for external channels 0–8 and 11-14.

Figure 23. Conversion Timing Register 0 (CTR0)

Address: Base + 0x094 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	INP LATCH	0	OFFSHIFT		0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1

Table 32. CTR0 field descriptions

Field	Description
INPLATCH	Configuration bit for latching phase duration (see *Figure 24*)
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 (see *Table 33*)
INPSAMP	Configuration bits for sampling phase duration (see *Figure 24*)

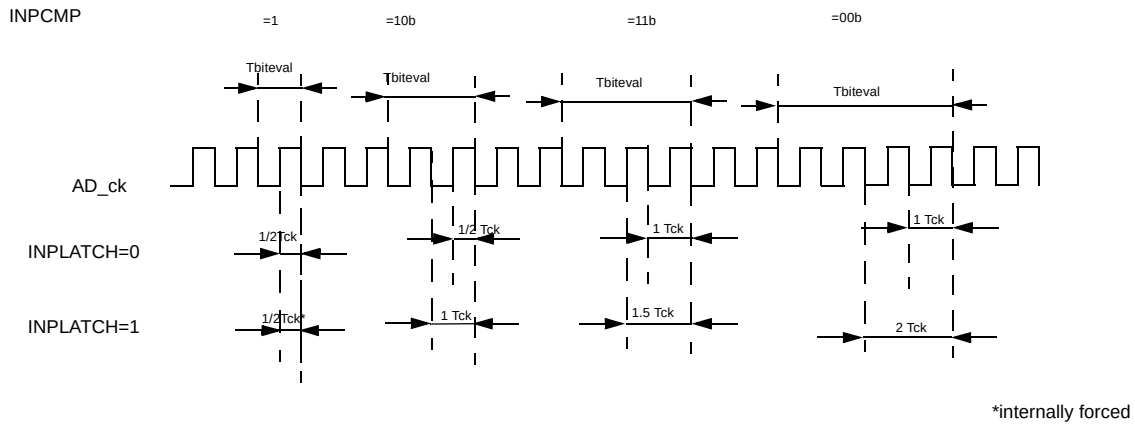


Figure 24. Conversion timing

Table 33. Minimum AD_ck frequency

INPCMP	AD_ck min. freq (MHz)
01	3
10	6
11	9
00	12

9.3.10.2 Conversion Timing Register 1 (CTR1)

This register configures the conversion timings for internal channels 9 (for factory test use only), 10 (VREG_1.2V), and 15 (the TSENS channel).

Figure 25. Conversion Timing Register 1 (CTR1)

Address: Base + 0x098 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

Table 34. CTR1 field descriptions

Field	Description
INPLATCH	Configuration bit for latching phase duration (see *Figure 24*)
OFFSHIFT	Configuration for offset shift characteristic is fixed to 0 in CTR1 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.
INPCMP	Configuration bits for comparison phase duration (see *Table 33*)
INPSAMP	Configuration bits for sampling phase duration (see *Figure 24*) When converting the TSENS internal channel 15 using ADC0, the LSB of this field CTR1[31] selects the operating mode of the TSENS module. CTR1[31] is part of the values of both TSENSOR_SEL and INPSAMP.

9.3.11 Mask registers

These registers are used to program which of the input channels must be converted during normal and injected conversion.

9.3.11.1 Normal Conversion Mask Register 0 (NCMR0)

Figure 26. Normal Conversion Mask Register 0 (NCMR0)

Address: Base + 0x0A4 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

Table 35. NCMR0 field descriptions

Field	Description
CH*n*	Sampling enable 0 Sampling is disabled for channel *n* 1 Sampling is enabled for channel *n*

9.3.11.2 Injected Conversion Mask Register 0 (JCMR0)

Figure 27. Injected Conversion Mask Register 0 (JCMR0)

Address: Base + 0x0B4 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

Table 36. JCMR0 field descriptions

Field	Description
CH*n*	Sampling enable 0 Sampling is disabled for channel *n* 1 Sampling is enabled for channel *n*

9.3.12 Power Down Exit Delay Register (PDEDR)

Figure 28. Power-down Exit Delay Register (PDEDR)

Address: Base + 0x0C8 Access: User read/write

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

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

Table 37. PDEDR field descriptions

Field	Description
PDED	The 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)

9.3.13 Channel Data Registers (CDR*n*)

Figure 29. Channel Data Register *n* (CDR*n*)

Address: See *Table 17*. 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[0:1]	
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	CDATA (MCR[WLSIDE] = 0)											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CDATA (MCR[WLSIDE] = 1)												0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 38. CDR*n* field descriptions

Field	Description
VALID	Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.

Table 38. CDR*n* field descriptions (continued)

Field	Description
OVERW	Overwrite data This bit signals that the previous converted data has been overwritten by a new conversion. This functionality depends on the value of MCR[OWREN]: – When OWREN = 0, then OVERW is frozen to 0 and CDATA field is protected against being overwritten until being read. – When OWREN = 1, then OVERW flags the CDATA field overwrite status. 0 Converted data has not been overwritten 1 Previous converted data has been overwritten before having been read
RESULT	This field 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 *n* converted data. Depending on the value of the MCR[WLSIDE] bit, the position of this bitfield can be changed as shown in *Figure 29*

9.3.14 Channel Watchdog Selection Registers (CWSEL*n*)

Figure 30. Channel Watchdog Selection Register 0 (CWSEL0)

Address: Base + 0x2B0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WSEL_CH7				WSEL_CH6				WSEL_CH5				WSEL_CH4			
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH3				WSEL_CH2				WSEL_CH1				WSEL_CH0			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31. Channel Watchdog Selection Register 1 (CWSEL1)

Address: Base + 0x2B4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WSEL_CH15				WSEL_CH14				WSEL_CH13				WSEL_CH12			
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH11				WSEL_CH10				WSEL_CH9				WSEL_CH8			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 39. CWSEL*n* field descriptions

Field	Description
WSEL_CH*n*	Selects the threshold register that provides the values to be used for upper and lower thresholds for channel *n* 0000 THRHLR0 register is selected 0001 THRHLR1 register is selected ... 1110 THRHLR14 register is selected 1111 THRHLR15 register is selected

9.3.15 Channel Watchdog Enable Register 0 (CWENR0)

Figure 32. Channel Watchdog Enable Register 0 (CWENR0)

Address: Base + 0x2E0 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

Table 40. CWENR0 field descriptions

Field	Description
CWEN*n*	Enables the watchdog feature for channel *n* 0 The watchdog feature for channel *n* is disabled 1 The watchdog feature for channel *n* is enabled

9.3.16 Analog Watchdog Out of Range Register 0 (AWORR0)

Figure 33. Analog Watchdog Out of Range Register 0 (AWORR0)

Address: Base + 0x2F0 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

Table 41. AWORR0 field descriptions

Field	Description
AWOR_CH*n*	Out of range indicator 0 Channel *n* converted data is in range 1 Channel *n* converted data is out of range

9.3.17 Self test registers

9.3.17.1 Self Test Configuration Register 1 (STCR1)

Figure 34. Self Test Configuration Register 1 (STCR1)

Address: Base + 0x340 Access: User read/write

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

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

Table 42. STCR1 field descriptions

Field	Description
INPSAMP_C	Sampling phase duration for the test conversions related to the algorithm C. Valid self-test values for this field are given below. Minimum: 0x18 Maximum: 0xFF
INPSAMP_RC	Sampling phase duration for the test conversions related to the algorithm RC. Valid self-test values for this field are given below. Minimum: 0x60 Maximum: 0xFF
INPSAMP_S	Sampling phase duration for the test conversions related to the algorithm S. Valid self-test values for this field are given below. Minimum: 0xFF Maximum: 0xFF
ST_INPCMP	Configuration bits for comparison phase duration for self test channel (as in *Table 33* for normal conversion)
ST_INPLATCH	Configuration bits for latching phase duration for self test channel (as in *Figure 24* for normal conversion)

9.3.17.2 Self Test Configuration Register 2 (STCR2)

Figure 35. Self Test Configuration Register 2 (STCR2)

Address: Base + 0x344 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	MSKWDSERR	0	MSKWDTERR	0	MSKST_EOC	0	0	0	0	MSKWDG_EOA_C	MSKWDG_EOA_RC	MSKWDG_EOA_S
W						SERR										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MSKERR_C	MSKERR_RC	MSKERR_S2	MSKERR_S1	MSKERR_S0	0	0	0	EN	0	0	FMA_WDSERR	FMA_WDTERR	FMA_C	FMA_RC	FMA_S
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Table 43. STCR2 field descriptions

Field	Description
MSKWDSERR	Interrupt enable (STSR1[WDSERR] status bit) 0 Interrupt disabled 1 Enables the STSR1[WDSERR] status bit to generate an interrupt
SERR	Error fault injection control. Setting this bit causes the STSR1[ERR*n*] status bits to be set.
MSKWDTERR	Interrupt enable (STSR1[WDTERR] status bit) 0 Interrupt disabled 1 Enables the STSR1[WDTERR] status bit to generate an interrupt
MSKST_EOC	Interrupt Enable bit for STSR1[ST_EOC] 0 Interrupt disabled 1 If IMR[MSKEOC] = 1, enables the STSR1[ST_EOC] status bit to generate an interrupt indication
MSKWDG_EOA_C	Interrupt enable (STSR1[WDG_EOA_C] status bit) 0 Interrupt disabled 1 Enables the STSR1[WDG_EOA_C] status bit to generate an interrupt
MSKWDG_EOA_RC	Interrupt enable (STSR1[WDG_EOA_RC] status bit) 0 Interrupt disabled 1 Enables the STSR1[WDG_EOA_RC] status bit to generate an interrupt
MSKWDG_EOA_S	Interrupt enable (STSR1[WDG_EOA_S] status bit) 0 Interrupt disabled 1 Enables the STSR1[WDG_EOA_S] status bit to generate an interrupt
MSKERR_C	Interrupt enable (STSR1[ERR_C] status bit) 0 Interrupt disabled 1 Enables the STSR1[ERR_C] status bit to generate an interrupt
MSKERR_RC	Interrupt enable (STSR1[ERR_RC] status bit) 0 Interrupt disabled 1 Enables the STSR1[ERR_RC] status bit to generate an interrupt
MSKERR_S2	Interrupt enable (STSR1[ERR_S2] status bit) 0 Interrupt disabled 1 Enables the STSR1[ERR_S2] status bit to generate an interrupt
MSKERR_S1	Interrupt enable (STSR1[ERR_S1] status bit) 0 Interrupt disabled 1 Enables the STSR1[ERR_S1] status bit to generate an interrupt
MSKERR_S0	Interrupt enable (STSR1[ERR_S0] status bit) 0 Interrupt disabled 1 Enables the STSR1[ERR_S0] status bit to generate an interrupt
EN	Self-test channel enable bit. It enables the TEST channel only in CPU mode. In CTU trigger/control mode the enable is provided directly by CTU. This bit should be set before starting the normal conversion and should not be changed while conversion is ongoing. This bit should be cleared only after end of conversion for the last self test channel has been received. 0 Test conversions are disabled 1 Test conversions are enabled

Table 43. STCR2 field descriptions (continued)

Field	Description
FMA_WDSERR	Fault mapping for the Watchdog Sequence error. 0 NCF mapping 1 CF mapping
FMA_WDTERR	Fault mapping for the Watchdog Timer error. 0 NCF mapping 1 CF mapping
FMA_C	Fault mapping for the algorithm C. 0 NCF mapping 1 CF mapping
FMA_RC	Fault mapping for the algorithm RC. 0 NCF mapping 1 CF mapping
FMA_S	Fault mapping for the algorithm S. 0 NCF mapping 1 CF mapping

9.3.17.3 Self Test Configuration Register 3 (STCR3)

Figure 36. Self Test Configuration Register 3 (STCR3)

Address: Base + 0x348 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	ALG		0	0	0	MSTEP				
W																
Reset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

Table 44. STCR3 field descriptions

Field	Description
ALG	Algorithm scheduling. This field has different functionality depending on the ADC mode (one-shot or scan). **For one-shot mode:** 00 Algorithm S (single step = MSTEP) 01 Algorithm RC (single step = MSTEP) 10 Algorithm C (single step = MSTEP) 11 Algorithm S For test/debug purposes **For scan mode:** 00 Algorithm S 01 Algorithm RC 10 Algorithm C 11 Algorithm S + algorithm RC + algorithm C The baud rate for the execution of the selected algorithm is defined by the STBRR register.
MSTEP	For one-shot mode, defines the current step for algorithms S, RC, and C as follows: – For algorithm S: MSTEP = 0 to 2 – For algorithm C: MSTEP = 0 to 16 – For algorithm RC: MSTEP = 0 to 18 For scan mode, this field is not used and should be programmed to 0b00. This is because in scan mode performs only single-step execution (interleaved mode).

9.3.17.4 Self Test Baud Rate Register (STBRR)

Figure 37. Self Test Baud Rate Register (STBRR)

Address: Base + 0x34C 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	WDT		
W																
Reset	0	0	0	0	0	0	0	0	0	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	BR							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 45. STBRR field descriptions

Field	Description
WDT	Watchdog timer value. This value is used to monitor that the algorithm sequence is correctly executed within the safe time period. The self testing watchdog is enabled by setting the STAW*n*R[WDTE] control bits. A fixed pre-scaler runs on the ADC clock (120 MHz). 000 0.1 ms 001 0.5 ms 010 1 ms 011 2 ms 100 5 ms 101 10 ms 110 20 ms 111 50 ms
BR	Baud rate for the selected algorithm in scan mode (MCR[MODE] = 1). You should program this field before enabling the self test channel. 0x00 Maximum scheduling rate (nominal rate) ... 0xFF Minimum scheduling rate (nominal rate scaled by 255)

9.3.17.5 Self Test Status Register 1 (STSR1)

Figure 38. Self Test Status Register 1 (STSR1)

Address: Base + 0x350 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	WDSERR	0	WDTERR	OVERWR	ST_EOC	0	0	0	0	WDG_EOA_C	WDG_EOA_RC	WDG_EOA_S
W					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	ERR_C	ERR_RC	ERR_S2	ERR_S1	ERR_S0	0	STEP_C					STEP_RC				
W	w1c	w1c	w1c	w1c	w1c											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Self Test End of Algorithm interrupt signals WDG_EOA_S, WDG_EOA_RC and WDG_EOA_C (along with interrupt signals EOCTU, JEOC, JECH EOC, and ECH in the ISR register) are routed to the INTC as the ADC_EOC interrupt source. Interrupt signals WDSERR, WDTERR, ERR_C, ERR_RC, ERR_S2, ERR_S1, ERR_S0 (along with REF_RANGE in the ISR register) are routed to the INTC as the ADC_ER interrupt source.

Table 46. STSR1 field descriptions

Field	Description
WDSERR	Watchdog sequence error of the ADC sub-system (check for algorithm step sequence). It generates an interrupt if enabled (STCR2[MSKWDSERR] = 1). It provides the fault indication to the FCCU, asserting CF or NCF according to the STCR2[FMA_WDSERR] mapping. 0 no failure 1 failure occurred
WDTERR	Watchdog timer error of the ADC sub-system (algorithm check for completion within safe time). It generates an interrupt if enabled (STCR2[MSKWDTERR] = 1). It provides the fault indication to the FCCU, asserting CF or NCF according to the STCR2[FMA_WDTERR] mapping. 0 no failure 1 failure occurred
OVERWR	Overwrite error. Used to notify when the STSR1[ERR*n*] bit is overwritten by a newer one. The new error status is written or discarded according to the MCR[OWREN] bit value. To avoid OVERWR indication, the ERR*n* status bit must be cleared (via SW).
ST_EOC	Self Test EOC Bit. If IMR[MSKEOC] = 1, this bit is set along with EOC bit when end_of_conversion signal is received from ADC analog for self test channel. It generates an interrupt if enabled by STCR2[MSKST_EOC].
WDG_EOA_C	This bit indicates that Algorithm C has been completed. This bit is set after the last step of the algorithm is executed. It generates an interrupt if enabled (STCR2[MSKWDG_EOA_C] = 1). This bit is set only if STAW4R[WDTE] = 1. For CTU conversions, this bit is significant only for Burst mode of operation.
WDG_EOA_RC	This bit indicates that Algorithm RC has been completed. This bit is set after the last step of the algorithm is executed. It generates an interrupt if enabled (STCR2[MSKWDG_EOA_RC] = 1). This bit is set only if STAW3R[WDTE] = 1. For CTU conversions, this bit is significant only for Burst mode of operation.
WDG_EOA_S	This bit indicates that Algorithm S has been completed. This bit is set after the last step of algorithm S is executed. It generates an interrupt if enabled (STCR2[MSKWDG_EOA_S] = 1). This bit is set only if STAW0R[WDTE] = 1. For CTU conversions, this bit is significant only for Burst mode of operation.
ERR_C	Indicates an error on the self testing channel (algorithm C). It generates an interrupt if enabled (STCR2[MSKERR_C] = 1). It provides the fault indication to the FCCU, asserting the programmed fault line (STCR2[FMA*n*]) You can also set the ERR_C bit (fault injection) by setting the STCR2[SERR] bit. In this case the CF or NCF lines are asserted according to the STCR2[FMA*n*] mapping. 0 No C-algorithm error has occurred 1 A C-algorithm error has occurred
ERR_RC	Indicates an error on the self testing channel (algorithm RC). It generates an interrupt if enabled (STCR2[MSKERR_RC] = 1). It provides the fault indication to the FCCU, asserting the programmed fault line (STCR2[FMA*n*]) You can also set the ERR_RC bit (fault injection) by setting the STCR2[SERR] bit. In this case the CF or NCF lines are asserted according to the STCR2[FMA*n*] mapping. 0 No RC-algorithm error has occurred 1 An RC-algorithm error has occurred

Table 46. STSR1 field descriptions (continued)

Field	Description
ERR_S2	Indicates an error on the self testing channel (algorithm SUPPLY, step 2). It generates an interrupt if enabled (STCR2[MSKERR_S2] = 1). It provides the fault indication to the FCCU, asserting the programmed fault line (STCR2[FMA*n*]) You can also set the ERR_S2 bit (fault injection) by setting the STCR2[SERR] bit. In this case the CF or NCF lines are asserted according to the STCR2[FMA*n*] mapping. 0 No error has occurred on the sampled signal 1 An error has occurred on the sampled signal
ERR_S1	Indicates an error on the self testing channel (algorithm SUPPLY, step 1). It generates an interrupt if enabled (STCR2[MSKERR_S1] = 1). It provides the fault indication to the FCCU, asserting the programmed fault line (STCR2[FMA*n*]) You can also set the ERR_S1 bit (fault injection) by setting the STCR2[SERR] bit. In this case the CF or NCF lines are asserted according to the STCR2[FMA*n*] mapping. 0 No VDD error has occurred 1 A VDD error has occurred
ERR_S0	Indicates an error on the self testing channel (algorithm SUPPLY, step 0). It generates an interrupt if enabled (STCR2[MSKERR_S0] = 1). It provides the fault indication to the FCCU, asserting the programmed fault line (STCR2[FMA*n*]) You can also set the ERR_S0 bit (fault injection) by setting the STCR2[SERR] bit. In this case the CF or NCF lines are asserted according to the STCR2[FMA*n*] mapping. 0 No VREF error has occurred 1 A VREF error has occurred
STEP_C	Step of the algorithm C when an ERR_C has occurred. 0.. (NUM_C_STEPS-1) => algorithm C
STEP_RC	Step of the algorithm RC when an ERR_RC has occurred. 0.. (NUM_RC_STEPS-1) => algorithm RC

9.3.17.6 Self Test Status Register 2 (STSR2)

Figure 39. Self Test Status Register 2 (STSR2)

Address: Base + 0x354 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OVFL	0	0	0	DATA1											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	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	DATA0											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 47. STSR2 field descriptions

Field	Description
OVFL	Overflow bit. This bit is set when the divisor is zero. If this happens, the STSR1[ERR_S1] bit is also set.
DATA1	Test channel converted data when the ERR_S1 has occurred. algorithm S (step1) => fractional part of the ratio TEST(step1)/TEST (step0) = VDD/VBGAP
DATA0	Test channel converted data when the ERR_S1 has occurred. - algorithm S (step1) => integer part of the ratio TEST(step1)/TEST (step0) = VDD/VBGAP

9.3.17.7 Self Test Status Register 3 (STSR3)

Figure 40. Self Test Status Register 3 (STSR3)

Address: Base + 0x358 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	DATA1											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	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	DATA0											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 48. STSR3 field descriptions

Field	Description
DATA1	Test channel converted data when the ERR_S2 has occurred. - algorithm S (step2) => TEST channel data = VREF/VREF
DATA0	Test channel converted data when the ERR_S0 has occurred. - algorithm S (step0) => TEST channel data = VBGAP/VREF

9.3.17.8 Self Test Status Register 4 (STSR4)

Figure 41. Self Test Status Register 4 (STSR4)

Address: Base + 0x35C 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	DATA1											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	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	DATA0											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 49. STSR4 field descriptions

Field	Description
DATA1	Test channel converted data when the ERR_C has occurred. - algorithm C => TEST channel data
DATA0	Test channel converted data when the ERR_RC has occurred. - algorithm RC => TEST channel data

9.3.17.9 Self Test Data Register 1 (STDR1)

Figure 42. Self Test Data Register 1 (STDR1)

Address: Base + 0x370 Access: User read only

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

Table 50. STDR1 field descriptions

Field	Description
VALID	Valid data. Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERWR	Overwrite data. Used to notify when a conversion data is overwritten by a newer result. The new data is written or discarded according to the MCR[OWREN] bit value.
TCDATA	Test channel converted data

9.3.17.10 Self Test Data Register 2 (STDR2)

Figure 43. Self Test Data Register 2 (STDR2)

Address: Base + 0x374 Access: User read only

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

Table 51. STDR2 field descriptions

Field	Description
FDATA	Fractional part of the ratio TEST(step1)/TEST (step0) = VDD/VBGAP for the algorithm S.
VALID	Valid data. Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERWR	Overwrite data. Used to notify when a conversion data is overwritten by a newer result. The new data is written or discarded according to the MCR[OWREN] bit value.
IDATA	Integer part of the ratio TEST(step1)/TEST (step0) = VDD/VBGAP for the algorithm S

9.3.17.11 Self Test Analog Watchdog Register 0 (STAW0R)

Figure 44. Self Test Analog Watchdog Register 0 (STAW0R)

Address: Base + 0x380 Access: User read/write

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

Table 52. STAW0R field descriptions

Field	Description
AWDE	Analog watchdog enable 0 The analog watchdog related to the algorithm S (step 0) is disabled 1 The analog watchdog related to the algorithm S (step 0) is enabled
WDTE	Watchdog timer enable. The watchdog timer verifies: – Correct sequence of the algorithm (step sequence) – Execution of the algorithm within the safe time period as defined by STBRR[WDT] As soon as the watchdog timer is enabled the algorithm starting must be detected within the safe time period. The watchdog timer is reset each time the algorithm restarts. *Note: This bit should be set only in scan mode.* 0 The watchdog timer related to the algorithm S is disabled 1 The watchdog timer related to the algorithm S is enabled
THRH	High threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR*n*] status bit is set if STDR1[TCDATA] > THRH.
THRL	Low threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR*n*] status bit is set if STDR1[TCDATA] < THRH..

9.3.17.12 Self Test Analog Watchdog Register 1A (STAW1AR)

Figure 45. Self Test Analog Watchdog Register 1A (STAW1AR)

Address: Base + 0x384 Access: User read/write

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

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

Table 53. STAW1AR field descriptions

Field	Description
AWDE	Analog watchdog enable 0 The analog watchdog related to the algorithm S (step 1) is disabled 1 The analog watchdog related to the algorithm S (step 1) is enabled
THRH	High threshold value (integer part) for test channel for algorithm S (step 1) (unsigned coding)
THRL	Low threshold value (integer part) for test channel for algorithm S (step 1) (unsigned coding)

9.3.17.13 Self Test Analog Watchdog Register 1B (STAW1BR)

Figure 46. Self Test Analog Watchdog Register 1B (STAW1BR)

Address: Base + 0x388 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	0	0	1	1	1	1	1	0	1	0	0	0

	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	1	1	1	0	1	1	0	1	0	0	0	0

Table 54. STAW1BR field descriptions

Field	Description
THRH	High threshold value (fractional part) for test channel for algorithm S (step 1)(unsigned coding)
THRL	Low threshold value (fractional part) for test channel for algorithm S (step 1) (unsigned coding)

9.3.17.14 Self Test Analog Watchdog Register 2 (STAW2R)

Figure 47. Self Test Analog Watchdog Register 2 (STAW2R)

Address: Base + 0x38C Access: User read/write

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

Table 55. STAW2R field descriptions

Field	Description
AWDE	Analog watchdog enable 0 The analog watchdog related to the algorithm S (step 2) is disabled 1 The analog watchdog related to the algorithm S (step 2) is enabled
THRL	Low threshold value for channel *n* (unsigned coding). If the analog watchdog is enabled, the STSR1[ERR_S2] status bit is set if STDR1[TCDATA] <THRL.

9.3.17.15 Self Test Analog Watchdog Register 3 (STAW3R)

Figure 48. Self Test Analog Watchdog Register 3 (STAW3R)

Address: Base + 0x390 Access: User read/write

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

Table 56. STAW3R field descriptions

Field	Description
AWDE	Analog watchdog enable 0 The analog watchdog related to the algorithm RC is disabled 1 The analog watchdog related to the algorithm RC is enabled
WDTE	Watchdog timer enable. The watchdog timer verifies: – Correct sequence of the algorithm (step sequence) – Execution of the algorithm within the safe time period as defined by STBRR[WDT] As soon as the watchdog timer is enabled the algorithm starting must be detected within the safe time period. The watchdog timer is reset each time the algorithm restarts. *Note: This bit should be set only in scan mode.* 0 The watchdog timer related to the algorithm RC is disabled 1 The watchdog timer related to the algorithm RC is enabled
THRH	High threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR*n*] status bit is set if STDR1[TCDATA] > THRH.
THRL	Low threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR*n*] status bit is set if STDR1[TCDATA] < THRH.

9.3.17.16 Self Test Analog Watchdog Register 4 (STAW4R)

Figure 49. Self Test Analog Watchdog Register 4 (STAW4R)

Address: Base + 0x394 Access: User read/write

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

	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	1	1	1	1	1	0	1	1	0	0	0

Table 57. STAW4R field descriptions

Field	Description
AWDE	Analog watchdog enable 0 The analog watchdog related to the algorithm C is disabled 1 The analog watchdog related to the algorithm C is enabled
WDTE	Watchdog timer enable. The watchdog timer verifies: – Correct sequence of the algorithm (step sequence) – Execution of the algorithm within the safe time period as defined by STBRR[WDT] As soon as the watchdog timer is enabled the algorithm starting must be detected within the safe time period. The watchdog timer is reset each time the algorithm restarts. *Note: This bit should be set only in scan mode.* 0 The watchdog timer related to the algorithm C is disabled 1 The watchdog timer related to the algorithm C is enabled
THRH	High threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR*n*] status bit is set if STDR1[TCDATA] > THRH. *Note: This value is valid only for the step 0 (refer to STAW5R register for the other Algorithm C steps).*
THRL	Low threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR*n*] status bit is set if STDR1[TCDATA] < THRH. *Note: This value is valid only for the step 0 (refer to STAW5R register for the other Algorithm C steps).*

9.3.17.17 Self Test Analog Watchdog Register 5 (STAW5R)

Figure 50. Self Test Analog Watchdog Register 5 (STAW5R)

Address: Base + 0x398 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	0	0	0	0	0	0	0	1	0	0	0	0

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

Table 58. STAW5R field descriptions

Field	Description
THRH	High threshold value (unsigned coding) for the algorithm C (step1 to step CS-1). If the analog watchdog is enabled (STAW4R[AWDE] = 1), the STSR1[ERR_C] status bit is set if STDR1[TCDATA{Stepn}] - STDR1[TCDATA {algC-step0}] > THRH.
THRL	Low threshold value (unsigned coding) for the algorithm C (step1 to step CS-1). If the analog watchdog is enabled (STAW4R[AWDE] = 1), the STSR1[ERR_C] status bit is set if STDR1[TCDATA {algC-step0}] - STDR1[TCDATA{Stepn}] > THRL.

9.4 Functional description

9.4.1 Inter-module communication

Figure 51 shows the modules that communicate with the ADC. In the figure, “ADCD” and “ADCA” refer to the digital and analog components of the ADC, respectively.

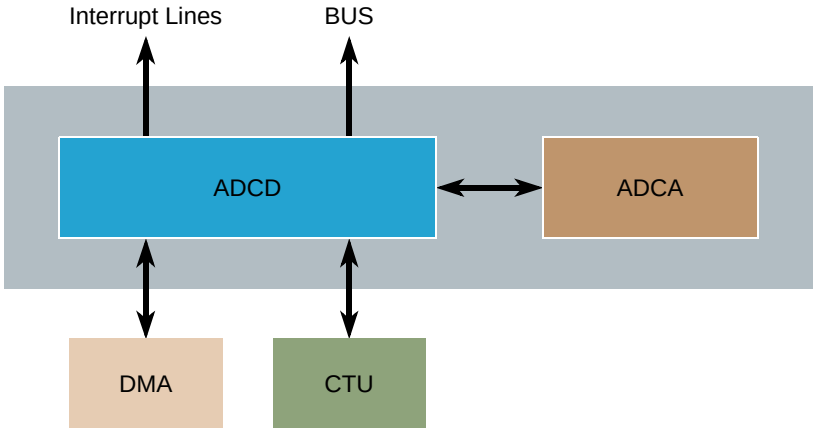


Figure 51. ADC interaction with other modules

Figure 52 shows the CTU / ADC interface. Each ADC can be controlled by the CPU (CPU Control Mode) or by the CTU (CTU Control Mode). The CTU can control the ADC sending an ADC command only when the ADC is in CTU Control Mode. The control mode is selected via a configuration bit. You can dynamically switch this mode without generating a reset or a power-down; in this case, however, you must not switch the mode while a conversion is in progress. During the CTU Control Mode, the CPU is able to write in the ADC registers but it can not start a new conversion.

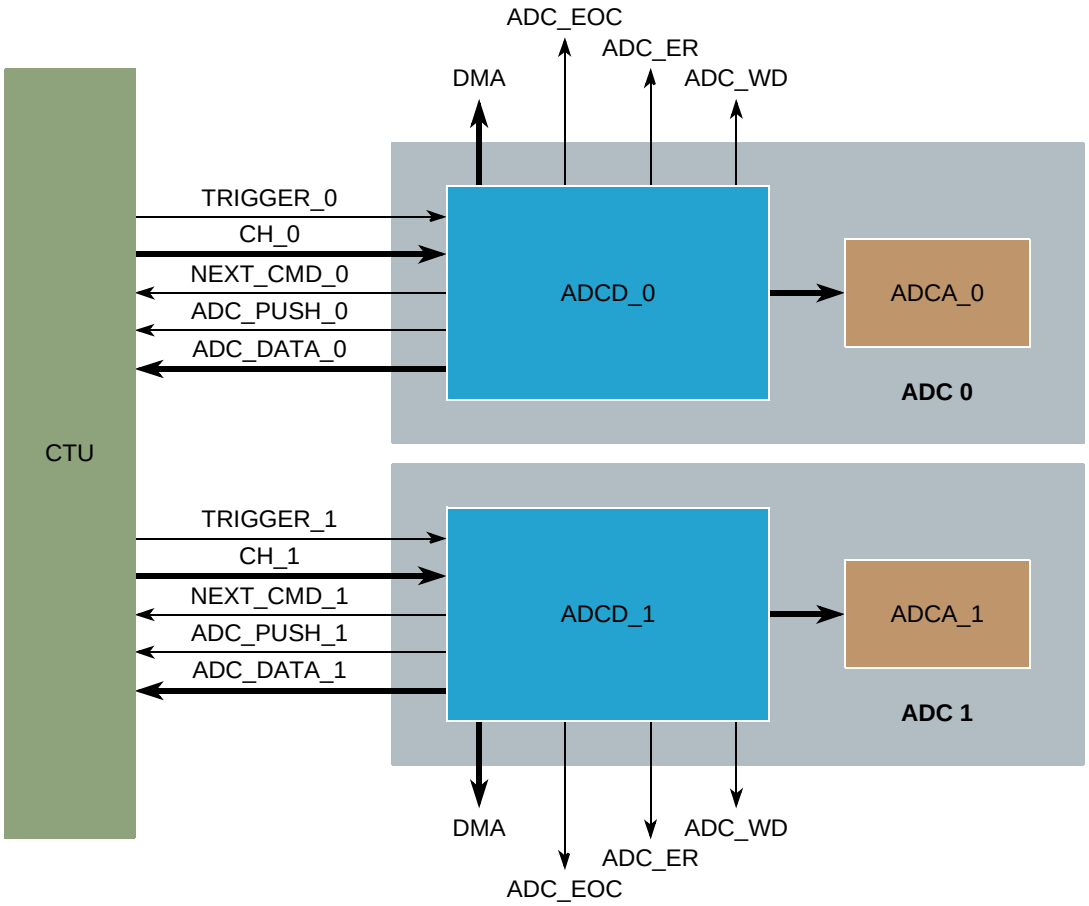


Figure 52. CTU/ADC interface

9.4.2 Analog channel conversion

Two conversion modes are available within the ADCDig:

- Normal conversion
- Injected conversion

9.4.2.1 Normal conversion

This is the normal conversion that the user programs by configuring the normal conversion mask registers (NCMRs). Each channel can be individually enabled by setting ‘1’ in the corresponding NCMR field. Mask registers must be programmed before starting the conversion and cannot be changed until the conversion of all the selected channels ends (MSR[NSTART] is cleared).

9.4.2.2 Start of normal conversion

The MSR[NSTART] status bit is automatically set when the normal conversion starts. At the same time the MCR[NSTART] bit is cleared, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

If the content of all the normal conversion mask registers is zero (that is, no channel is selected) the conversion operation is considered completed and the interrupt ECH (see *Section 9.4.8, Interrupts*) is immediately issued after the start of conversion.

9.4.2.3 Normal conversion operating modes

Two operating modes are available for the normal conversion:

- One Shot
- Scan

To enter one of these modes, you must program the MCR[MODE] bit. The first phase of the conversion process involves sampling the analog channel and the next phase involves the conversion phase when the sampled analog value is converted to digital as shown in *Figure 53*.

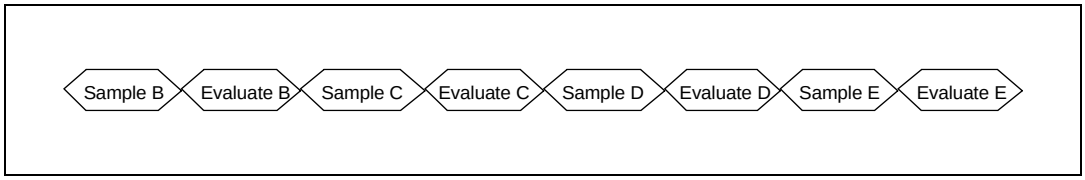


Figure 53. Normal conversion flow

In One Shot Mode (MCR[MODE] = 0) a sequential conversion specified in the NCMRs 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 (MCR[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. MCR[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 MSR[NSTART] status bit is automatically set when the normal conversion starts. At the same time the MCR[NSTART] bit cleared, 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 NCMRs is continuously performed. As in the previous case, at the end of each conversion the digital result of the conversion is stored in the corresponding data register.

The MSR[NSTART] status bit is automatically set when the normal conversion starts. Unlike One Shot Mode, the MCR[NSTART] bit is not cleared. It can be cleared 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 clears the MSR[NSTART] bit. Indeed, an additional chain is executed after NSTART is cleared.

Example 2 Scan Mode (MCR[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. MCR[MODE] = 1 is set for Scan Mode. Conversion starts from the channel B followed by conversion of the channels D-E. At the end of conversion of channel E the scanning of channel B starts followed by conversion of the

channels D-E. This sequence repeats itself till the MCR[NSTART] bit is cleared by software.

At the end of each conversion an End Of Conversion interrupt is issued (if enabled by the corresponding mask bit) and at the end of the conversion sequence an End Of Chain interrupt is issued (if enabled by the corresponding mask bit). Both ISR.EOC and the channel specific CEOCFR.EOCCHx shall be cleared at the end of the ISR.

9.4.2.4 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 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 stopped as shown in *Figure 54*.

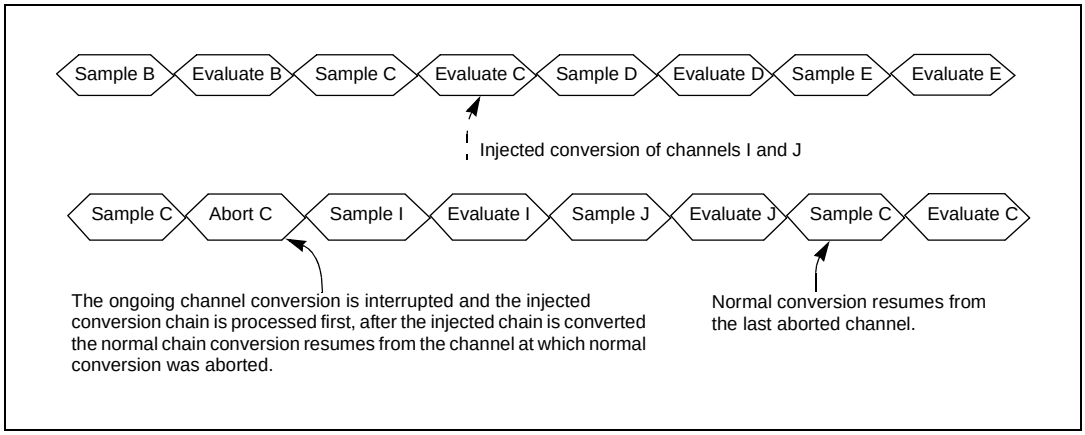


Figure 54. Injected sample/conversion sequence

The MSR[JSTART] status bit is automatically set when the injected conversion starts. At the same time the MCR[JSTART] bit is cleared, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running injected conversion is completed.

At the end of each injected conversion, an End Of Injected Conversion (JEOC) interrupt is issued (if enabled by the corresponding mask bit) and at the end of the sequence an End Of Injected Chain (JECH) interrupt is issued (if enabled by the corresponding mask bit).

If the content of all the injected conversion mask registers is zero (that is, no channel is selected) the interrupt JECH is immediately issued after the start of conversion.

Once started, injected chain conversion cannot be interrupted by any other conversion type. It can, however, be aborted; see *Section 9.4.2.5, Abort conversion.*

9.4.2.5 Abort conversion

Two different abort functions are provided.

- The user can abort the ongoing conversion by setting the MCR[ABORT] bit. The current conversion is aborted and the conversion of the next channel of the chain is

immediately started (generating a new start pulse to the Analog ADC). In the case of an abort operation, the NSTART/JSTART bit remains set and the ABORT bit is cleared after the conversion of the next channel starts. The EOC corresponding to the aborted channel is not generated. This behavior is true for normal or triggered/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 MCR[MODE] bit. In fact, if scan mode is disabled, the NSTART bit is automatically cleared together with the ABORTCHAIN bit. Otherwise, if the MODE bit is set, a new chain conversion is started. The EOC 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 cleared).

9.4.3 Analog clock generator and conversion timings

The clock frequency can be selected by programming the MCR[ADCLKSEL] bit. When this bit is set, the ADC clock has the same frequency as the system clock. Otherwise, the ADC clock is half of the system clock frequency. The MCR[ADCLKSEL] bit can be written only in power-down mode.

9.4.4 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.

When a conversion is started, the ADC connects the internal sampling capacitor to the respective analog input pin, allowing the capacitor to charge up to the input voltage value. The time to load the capacitor is referred to as sampling time. After completion of the sampling phase, the evaluation phase starts and all the bits corresponding to the resolution of the ADC are estimated to provide the conversion result.

The conversion times are programmed via the bit fields of the CTR. Bit fields INPLATCH, INPCMP and INPSAMP are used to define the sampling rate ($f_s = 1/T_{SR}$) and the partition between sampling phase duration (T_{sample}) and total evaluation phase duration (T_{eval}).

In the following equations, the unit T_{ck} refers to the reciprocal of the system clock which is then modified by the value of the MCR.ADCCLKSEL field:

$$T_{ck} = (2 - MCR.ADCCLKSEL) \bullet T_{sysck}$$

The minimum sampling phase duration is:

$$T_{sample} = 7 \bullet T_{ck}$$

For INPSAMP > 8, the sampling phase duration is:

$$T_{sample} = (INPSAMP - 1) \bullet T_{ck}$$

INPSAMPLE must be greater than or equal to 8 (hardware requirement). If INPSAMPLE is less than or equal to 7, the sampling time remains as if INPSAMPLE = 8.

The total evaluation phase duration is:

$$T_{eval} = 12 \bullet T_{biteval}$$

Note: the Tsample and Teval must respect the minimum value specified in Data Sheet to allow ADC to reach TUE performance.

1. For Tbiteval explanation, please refer to *Figure 24*.

The total conversion duration T_{conv} is (not including external multiplexing) the time to perform the total evaluation:

$$T_{conv} = T_{sample} + T_{eval} + T_{ck} + T_{digsync} + T_{ctusync}$$

where;

- $T_{digsync} = T_{ck}$, if MCR.ADCLKSEL= 0 or $T_{digsync} = 2 \bullet T_{ck}$, if MCR.ADCLKSEL=1
- $T_{ctusync} = 0$, if MCR.CTUEN=0 else, $T_{ctusync} = 0.5 \bullet T_{ck}$, if MCR.CTUEN=1 and MCR.ADCLKSEL=0, or $T_{ctusync} = 2 \bullet T_{ck}$, if MCR.CTUEN=1 and MCR.ADCLKSEL=1

9.4.5 Presampling

Presampling allows to precharge or discharge the ADC internal capacitor before it starts sampling/conversion of the analog input coming from pads. This is useful for resetting information regarding the last converted data. During presampling, the analog ADC samples the internally generated voltage while in the sampling the analog ADC samples analog input coming from pads.

Presampling can be enabled/disabled on a channel basis by setting the corresponding bits in the PSRs.

After enabling the presampling for a channel, normal sequence of operation will be Presampling+Sampling+Conversion for that channel. Sampling of the channel can be bypassed by setting the PSCR[PRECONV] bit. When sampling of a channel is bypassed, the sampled data of internal voltage in the Presampling State will be converted (see *Figure 55* and *Figure 56*).

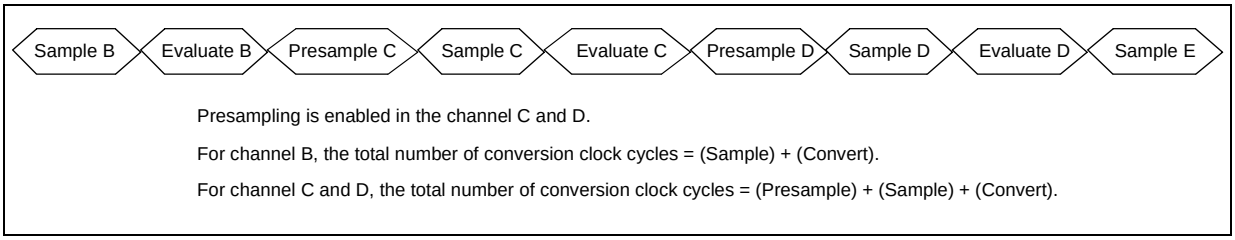


Figure 55. Presampling sequence

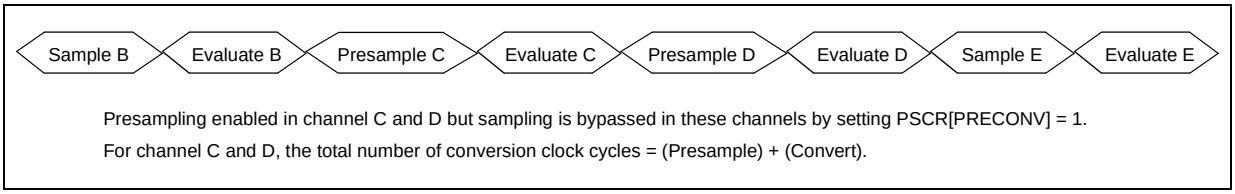


Figure 56. Presampling sequence with PRECONV = 1

The presampling channels implemented on this device are:

- For ADC_0:
  - VREFH ADC_0 = presampling channel 0
  - VREFL ADC_0 = presampling channel 1
- For ADC_1:
  - VREFH ADC_1 = presampling channel 0
  - VREFL ADC_1 = presampling channel 1

Note: Presampling cannot be used to detect open faults which can rise outside the MCU. Do not use this feature to detect open faults on the analog input.

9.4.6 Programmable analog watchdog

9.4.6.1 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 57*) specified by an upper and a lower threshold value named THRH and THRL respectively.

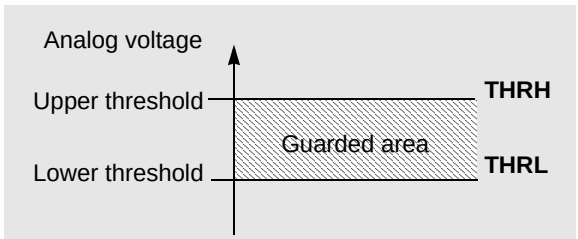


Figure 57. Guarded area

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 WDGxH and WDGxL bits in the WTISR as explained in *Table 59*. Depending on the mask bits MSKWDGxL and MSKWDGxH in the WTIMR, an interrupt is generated on threshold violation.

Table 59. 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

Note: Avoid the situation where THRH < THRL. In this case, a WDGxH or WDGxL interrupt will always be generated, and this could lead to misinterpretation of the watchdog interrupts.

9.4.7 DMA functionality

A Direct Memory Access (DMA) request can be programmed after the conversion of every channel, by setting the respective masking bit in the DMAR0 register. The DMA masking registers must be programmed before starting any conversion.

The DMA transfers can be enabled using the DMAE[DMAEN] bit. When the DMAE[DCLR] bit is set, the DMA request is cleared on the reading of the register for which DMA transfer has been enabled.

9.4.8 Interrupts

The ADC generates the following maskable interrupt signals:

- EOC (end of conversion) interrupt request
- ECH (end of chain) interrupt request
- JEOC (end of injected conversion) interrupt request
- JECH (end of injected chain) interrupt request
- WDGxL and WDGxH (watchdog threshold) interrupt requests
- REF_RANGE (reference voltage comparison) interrupt request
- Self test interrupts

Interrupts are generated during the conversion process to signal events such as End Of Conversion as explained in *Section 9.3.3.2, Channel Pending Register 0 (CEOCFR0)*. Two registers named CEOCFR (Channel Pending Registers) and IMR (Interrupt Mask Register) are provided in order to check and enable the interrupt requests.

Interrupts can be individually enabled on a channel by channel base by programming the CIMR (Channel Interrupt Mask Register).

Several Channel Interrupt Pending Registers are also provided in order to signal which of the channels' measurement has been completed.

The analog watchdog interrupts are managed by two registers:

- Watchdog Threshold Interrupt Status Register (WTISR)
- Watchdog Threshold Interrupt Mask Register (WTIMR)

The watchdog interrupt source sets two 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').

End of conversion interrupts for self test channel is similar to normal conversion channel. Same EOC and ECH bits for normal conversion are set for self test channel also. Similar to other channels, self test channel has Channel interrupt pending bit (ST_EOC) present in STSR1 and channel mask bit (MSKST_EOC) in STCR2 (corresponding to bits for every channel in CEOCFR and CIMR registers). In addition, End of Algorithm interrupts are also present which are handled by STSR1 and STCR2 registers.

Error interrupts related to Self Testing are handled by status bits in STSR1 and mask bits in STCR2 registers. If an error is generated due to analog watchdog monitoring or sequence checking of algorithm or internal watchdog timer timeout, corresponding error bit is set in STSR1. The mask bits for these error bits are present in STCR2 register.

9.4.9 Power-down mode

The analog part of the ADC can be put in a low-power mode, called “power-down mode,” by setting the MCR[PWDN] bit. 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 clearing the appropriate bit in the MCR.

The power-down mode can be requested at any time by setting the MCR[PWDN] bit. If a conversion is ongoing, the ADC hard macrocell cannot immediately enter 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 clearing the NSTART bit and using the ABORTCHAIN bit.

Bit ADCSTATUS[0] in the MSR is set only when ADC enters power-down mode.

After the power-down phase is completed, the process that was occurring before the power-down phase must be restarted manually (by setting the appropriate START bit).

After an exit from power-down mode, the first conversion can be started after 5 µs, otherwise the result can be affected by the improper setting of the ADC analog operating point.

Caution: You must not clear the MCR[PWDN] and set the MCR[NSTART] or MCR[JSTART] bits during the same cycle.

See also *Section 9.3.12, Power Down Exit Delay Register (PDEDR).*

9.4.10 Auto-clock-off mode

To reduce power consumption during the IDLE mode of operation (without going into power-down mode), an “auto-clock-off” feature can be enabled by setting the MCR[ACKO] bit. When enabled, the analog clock is automatically switched off when no operation is ongoing, that is, no conversion is programmed by the user.

9.4.11 Self testing

9.4.11.1 General operation

For safety devices used for very critical applications, it is important to check at regular intervals if the ADC is functioning correctly. For this purpose, Self Testing feature has been incorporated inside the ADC.

The self-tests use analog watchdogs to verify the result of self-test conversions. The threshold of these watchdogs is saved in the test flash. Before running the self test, you must copy these values from the test flash to the STAWxR registers.

Three types of Self Testing algorithms have been implemented inside ADC analog.

- Supply Self test: Algorithm S. It includes the conversion of the ADC internal bandgap voltage, ADC supply voltage, and ADC reference voltage. It includes a sequence of 3 test conversions (steps). The supply test conversions must be an atomic operation (no functional conversions interleaved).
- Resistive-Capacitive Self test: Algorithm RC. It includes a sequence of 19 test conversions (steps) by setting the ADC internal resistive digital-to-analog converter (DAC).
- Capacitive Self test: Algorithm C. It includes a sequence of 17 test conversions (steps) by setting the capacitive elements comprising the sampling capacitor/ capacitive DAC.

The ADC implements an additional test channel dedicated for Self testing. It also provides signals to schedule self testing algorithms using Configuration registers, monitors the converted data using analog watchdog registers, flags the error to FCCU in case some failure occurs in any of the algorithms.

Test channel can be activated in CPU or CTU mode as described in *Table 60*.

Table 60. Test channel activation

ADC mode	Test channel (ADC)	Test channel (CTU)
CPU mode (MCR[CTUEN] = 0)	Yes (one shot mode and scan mode)	No
CTU control mode	No	Yes

9.4.11.2 CPU mode

In this case, test channel works similar to normal conversion. In CPU mode, test channel is enabled by setting STCR2[EN].

The Self testing channel conversions are carried along with the functional conversions. The sequencing of steps of the selected algorithm for test channel depends on the operating mode of normal conversions interleaved, selected by the MCR[MODE] bit.

In One shot mode, if test channel is enabled, only one step of selected Self Testing algorithm is executed at the end of the chain. The step number and algorithm to be executed is programmed in STCR3 register. So, in one shot mode the sequence will be as follows:

- Program NCMR0 to select channels to be converted for normal conversion.
- Program MCR[MODE] = 0 to select one shot mode.
- Program sampling duration values in STCR1[INPSAMP*n*] field.
- Select the Self Testing algorithm in STCR3.ALG. Default is Algorithm S.
- Enable self testing channel by setting EN bit in the STCR2 register.
- Start the normal conversion by setting NSTART bit in the MCR.
- All normal conversions are executed as usual.
- At the end of all the normal conversions, Step Number programmed in STCR3.MSTEP field of Self testing algorithm selected by STCR3.ALG is executed similar to a normal functional channel.
- On receiving end of conversion for test channel, the digital result is written in STDR1.TCDATA and STDR1.VALID bit is set. Also, EOC and ECH bits are set in the ISR and ST_EOC bit is set in STSR1.
- State Machine returns to IDLE state.

Example 3

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. The result is shown in *Figure 58*. Conversion for channels B-D-E are done. After channel E test channel conversion is done and ISR.ECH and ISR.EOC are set.

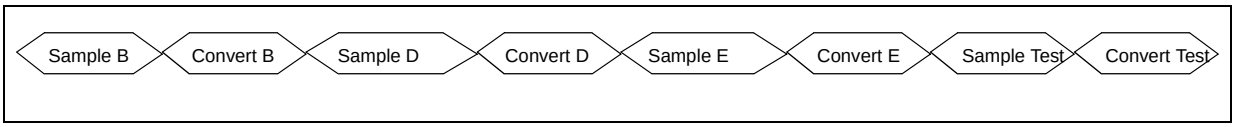


Figure 58. Test channel conversion example

The NSTART status bit of MSR is automatically set when the Normal conversion starts and is reset at the end of conversion for test channel.

In Scan Mode, consecutive steps of selected self test algorithm are converted continuously at the end of each chain of normal conversions. The number of channels converted at the end of each chain is 1 (except for Algorithm S, in which all the steps are performed at once without any functional conversion interleaved). So, in scan mode the sequence will be as follows.

- Program NCMR0 to select channels to be converted for normal conversion.
- Program MODE = 1 in the MCR to select scan mode.
- Program sampling duration values in STCR1.INPSAMPx field.
- Select the Self Testing algorithm in STCR3.ALG. By default, all three algorithms are selected i.e. all algorithms will be executed step by step one after the other.
- Enable self testing channel by setting EN bit in STCR2 register.
- Start the normal conversion by setting MCR[NSTART].
- All normal conversions are executed as usual.
- At the end of chain of the normal conversions, (assuming default value of STCR3.ALG) all steps of Algorithm S are performed (as Algorithm S is always atomic). MSR.SELF_TEST_S and STSR1.WDG_EOA_S is set.
- On receiving end of conversion of test channel for last step of Algorithm S, the digital result is written in STDR1.TCDATA and STDR1.VALID bit is set. At the same time, MSR.SELF_TEST_S is reset. For Step 1, the integral part and fractional part are written in STDR2.IDATA and STDR2.FDATA. Also, EOC and ECH bits are set in the ISR and ST_EOC is set in STSR2.
- Then the next chain of normal conversion starts.
- At end of the normal conversion chain, Step0 of RC algorithm is executed.
- On receiving end of conversion of test channel for Step0 of RC algorithm, the digital result is stored in STDR1.TCDATA and STDR1.VALID bit is set (if MCR.OVERWR bit is set). Also, EOC and ECH bits are set in the ISR and ST_EOC is set (if ceocfr_exist =1) in STSR2.
- Then the next chain of normal conversion starts.
- At end of the normal conversion chain, Step1 of RC algorithm is executed.
- After the last step of Algorithm RC is executed, STSR1.WDG_EOA_RC is set.
- This process continues for all the Steps of all three algorithms. After the last step of Algorithm C is executed, STSR1.WDG_EOA_C is set.
- State Machine returns to IDLE state when MCR[NSTART] is cleared.

Instead of starting normal conversion by software (by setting MCR[NSTART]), if it is started by external trigger, the test channel behavior will remain same.

In case of Injected conversions, test channel conversion is not performed. It is performed only during Normal conversions.

If during a test channel conversion, injection conversion arrives, then the test channel is aborted (just as a normal functional channel) and injected conversions are done. After injected conversions are completed, the test channel resumes from the Step at which it was aborted. In this case, the MSR.SELF_TEST_S remains high during the injected conversion.

For self testing, Mode bit should be programmed (atleast one cycle) BEFORE setting MCR.NSTART bit and should not be changed thereafter until conversion is ongoing.

9.4.11.3 CTU mode

The CTU mode is enabled by setting MCR[CTUEN]. With this bit set, the CTU operates in control mode.

If CTUEN is set, the test channel conversion can be started only by CTU interface and software cannot start it. The EN bit in STCR2 register does not have any effect in CTU mode. The CTUEN bit should not be changed while normal conversion is ongoing.

The interface between CTU and ADC (for Self Test) is shown in *Figure 59*.

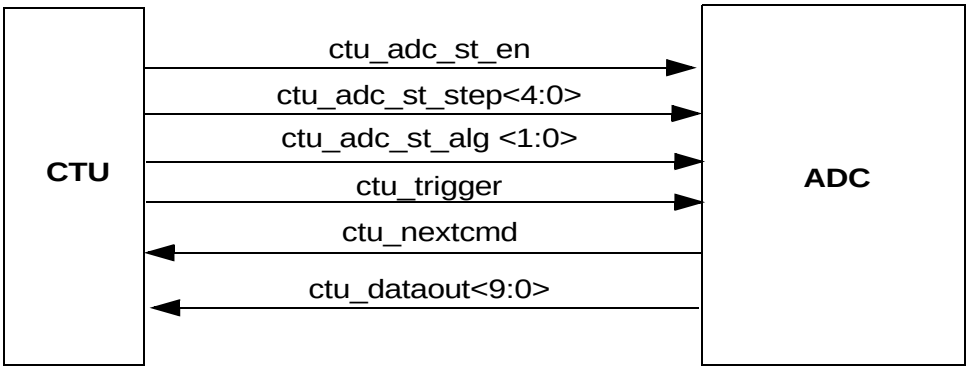


Figure 59. Interface between ADC and CTU to manage self test

For self testing conversions in CTU mode, CTU asserts ctu_adc_st_en signal along with ctu_trigger. The algorithm and the step number to be executed is put on ctu_adc_st_alg and ctu_adc_st_step respectively. The other signals ctu_nextcmd, ctu_trigger and ctu_dataout have same meaning as for normal CTU functional conversion.

For algorithm S, the three steps must be atomic. In CTU mode, this is managed by CTU itself i.e. CTU has to send three triggers (one for each step) asserting ctu_adc_st_en and updating ctu_adc_st_step for each step.

Channel Conversion Command Registers (CLRx) in CTU allow to setup a self test command, choosing also among the algorithms available. See *Table 61* and *Table 62*.

Table 61. CTU-CLRx-ST1, ST0 meaning

ST1 bit	ST0 bit	Command meaning
0	0	No self test Single conversion
0	1	Self test command

Table 61. CTU-CLRx-ST1, ST0 meaning

ST1 bit	ST0 bit	Command meaning
1	0	No self test command Dual conversion
1	1	No self test Dual conversion

Table 62. CTU-CLRx-ALG1, ALG0 meaning

ALG1	ALG0	Algorithm meaning
0	0	Algorithm S
0	1	Algorithm RC
1	0	Algorithm C
1	1	Algorithm FULL

The algorithm S must be executed as an atomic test without interleaved user conversions.

Step 0:

- VBGAP/VREF test
- Step1: VDD/VREF test
- Step2: VREF/VREF test

CTU sends three triggers (one for each step) asserting ctu_adc_st_en and updating ctu_adc_st_step for each step.

The RC and C algorithm can be executed, programming the CLRx registers, in one of the following schemes:

- Burst mode: when the CTU schedules the execution of the CLRx register configured for the self testing, both the algorithms RC and C are executed in burst mode (step0-algRC, step1-algRC, ...stepN-algRC, step0-algC, step1-algC, ... stepM-algC).
- Interleaved mode: when the CTU schedules the execution of the CLRx register configured for the self testing, a single step of the algorithm RC or C is executed according to an internal counter. In this mode the ADC self testing procedure is distributed and the functional conversions are not stalled for a long time.

The burst mode and interleaved mode are not applicable to the S algorithm that can be executed only as an atomic conversion.

When the FULL algorithm is enabled the ctu will execute an S algorithm followed by an RC algorithm and a C algorithm.

The enabling of the trigger configured for the ADC self-testing can be performed according to the following schemes:

- Triggered mode: according to the current CTU implementation
- Sequential mode: according to the current CTU implementation

9.4.11.4 Abort and abort chain for self testing channel

Setting MCR[ABORT] during self test channel has no effect.

In One Shot Mode, if MCR[ABORTCHAIN] is set when test channel conversion is ongoing, the test channel is aborted and ECH is set. In this case, EOC for test channel is not generated.

For zero baud rate in Scan Mode, if MCR[ABORTCHAIN] bit is set when test channel STEP N is ongoing, the test channel STEP N is aborted and next chain conversion starts. At the end of this chain, STEP N conversion is performed again. (In case of Algorithm S, full algorithm is executed again).

The case of non-zero baud rate is described in *Section 9.4.11.7.1, Abort chain when baud rate is non-zero.*

9.4.11.5 Self test analog watchdog

The ADC also provides a monitor (watchdog) for the values returned by its analog portion for Self Test algorithms. The analog watchdogs are used to determine whether the result of conversion for self test algorithms lie in a particular guard area. For this purpose, separate Self test analog watchdog registers have been provided for each algorithm.

After the conversion of each Step of an algorithm, a comparison is performed between the converted value and the threshold values if Analog watchdog feature is enabled by setting STAWxR.AWDE bit. If the converted value does not lie between the upper and lower threshold values specified by Analog Watchdog Register of the particular algorithm, corresponding error bit STSR1.ERR_x is set and Step Number in which error occurred is updated in STSR1.STEP_x (in case of C or RC algorithm). Also, erroneous data is written in STSR4.DATAx field. The STSR1.ERR_x bits will generate an interrupt if enabled by corresponding Mask bit in STCR2 register. The fault indication is also given to FCCU via CF and NCF, so that necessary action can be taken at SOC level.

Analog Watchdog feature works differently in case of Algorithm S. As already mentioned, Algorithm S is always an atomic operation. So, separate error bits are provided in STSR1 for each STEP of Algorithm S to avoid overwrite in case error occurs in more than one STEP. Hence, there are separate Mask bit for each step in STCR1. For the same reason, separate fields exist in Status registers (STSR2 and STSR3) to store erroneous data for each step.

For Algorithm S STEP1, a fixed point division has to be performed outside ADC analog hardmacro (it takes around 26 cycles after EOC for STEP1 to get divided value) and it is the divided value on which analog watchdog checks are applied. So, the value to be compared for STEP1 contains integer as well as fractional part and thus, two registers (STAW1AR and STAW1BR) are provided. The comparison is done first for integer part using the threshold values programmed in STAW1AR. If integer part does not lie in the range, the fractional part comparison is skipped otherwise it is compared with the values programmed in STAW1BR. *Table 63* summarizes this feature for STEP1.

Table 63. Algorithm S (STEP1) threshold comparison

STDR2[IDATA] (integer part)	STDR2[FDATA] (fractional part)	STSR1[ERR_S1]
> STAW1AR[THRH]	Any value	Set
< STAW1AR[THRL]	Any value	Set
== STAW1AR[THRH]	> STAW1BR[THRH]	Set
== STAW1AR[THRL]	< STAW1BR[THRL]	Set

For Algorithm S STEP2, (VREF/VREF) is measured in order to check the integrity of sampling signal. For this particular conversion, no higher threshold value is required as the ideal value is 0xFFF. Only lower threshold value is programmed in STAW2R.

For Algorithm C, a separate register is provided for Step0. In Step0 an offset for other steps is measured. The converted data is compared with the threshold values provided by STAW5R if STAW4R[AWDE] is set. For other steps, this offset is subtracted from converted data before performing watchdog checks.

9.4.11.6 Watchdog timer

The watchdog timer is an additional check which monitors the sequence of the self testing algorithm implemented and also that the algorithm is completed within a safe time period. The Watchdog timers can be enabled for CPU as well CTU conversions. Each algorithm has a different watchdog timer which runs independently of the other. The watchdog timer for a particular algorithm can be enabled by setting STAW*n*R[WDTE] bit. The safe time value can be programmed in STBRR[WDT] field (default value is 10 ms assuming a 120 MHz clock).

The safe time is measured starting from Step0 of the algorithm (including all normal chain conversions in between) to the point where Step0 of the same algorithm starts again.

The sequence is as follows:

- Program NCMR0 to select channels to be converted for normal conversion in scan mode (MODE = 1).
- Select the Self Testing algorithm in STCR3.ALG. By default, all three algorithms are selected i.e. all algorithms will be executed step by step one after the other.
- Enable self testing channel by setting EN bit in STCR2 register.
- Program safe period value in STBRR.WDT field.
- Enable watchdog timer by setting STAWxR.WDTE bit. Assume setting of WDTE bit to be ‘t0’. (It is important to do all the programming first and then enable WDTE bit as the safe time check is also performed between setting of WDTE bit and start of STEP0 to check that algorithm has started within the safe time).
- Start the normal conversion by setting MCR[NSTART].
- After first chain conversion ends, STEP0 of algorithm S is executed. Assume the start of STEP0 to be ‘t1’.
- After this STEP1 and STEP2 of algorithm S are executed.
- Then, next chain conversions are performed.
- When chain conversion completes, STEP0 of RC algorithm is performed.
- After each chain conversion, consecutive STEP of RC algorithm is performed. Similar sequence follows for C algorithm.
- After the last step of C algorithm is performed, another chain conversion is executed. At the end of this chain conversion, STEP0 of algorithm S is started repeating the whole sequence. Lets assume this time (starting of STEP0) to be ‘t2’.
- For S algorithm, if (t1 – t0) > Safe Period or (t2 – t1) > Safe Period, Watchdog timer flags an error and STSR1.WDTERR bit is set. Crictical fault is asserted and interrupt is also generated if enabled by STCR2.MSKWDTERR bit. Otherwise, watchdog timer counter is reset and starts again to monitor the same for the next sequence.
- Similar sequence is followed for watchdog timers for RC and C algorithms.

As CTU does not incorporate any safe period checking mechanism, the Watchdog timers can be enabled for CTU conversions also.

Note: You must not enable the watchdog timer for the algorithm which is not to be executed.

9.4.11.6.1 Watchdog sequence checking

The watchdog timer also incorporates sequence checking features which checks that the steps of a particular algorithm are in correct order. If steps are not in order, then error is flagged by setting STSR1[WDSERR]. Crictical fault is asserted and interrupt is also generated if enabled by STCR2[MSKWDSERR].

Watchdog sequence error is flagged in following cases:

- If steps of any algorithm are not executed in proper order.
- If abort chain occurs during test channel conversion, that step has to be repeated at the end of next chain. This will give a sequence error as soon as test channel conversion starts again.
  Exception : If abort chain occurs during last step of Alg-S, then sequence error is not flagged as the whole algorithm has to be repeated again.
- If, for CTU conversions, Step Numbers provided by CTU are not in order. Watchdog sequence checking is significant for CTU burst mode only.

If injected conversion occurs during the test channel, Watchdog sequence error is NOT flagged although the ongoing Step number is aborted and is repeated again.

Watchdog Timer feature is applicable only for scan mode of operation and not for one shot mode.

9.4.11.7 Baud rate control for test channel

This control defines the scheduling of test channel between the normal conversions. The scheduling rate is specified by STBRR[BR].

By default, if test channel is enabled, one Step of selected algorithm is executed after every chain of normal conversion. The bandwidth consumed by test channel depends on the number of channels in normal chain. e.g. if we have 100 normal conversions in a chain, then test channel consumes only 1% of total bandwidth, which is very small. But in case the number decreases to just 4 channels, then the bandwidth consumed by test channel is 25%, which is significant (and may not be desirable as it slows down the normal conversion rate).

STBRR[BR] field provides flexibility by scheduling the test channel conversion to be performed not at the end of every chain but at the end of BR+1 number of chains. e.g. if BR = 5, a single Step of selected Algorithm for test channel is performed after 6 chain conversions, then next Step is performed at end of next 6 chain conversions and so on. By default, the value of BR is 0.

To use the baud rate feature in scan mode, the NCMR should have a non-zero value.

Note: This feature is applicable only for scan mode of operation and not for one shot mode. The STBRR.BR should be set to zero for one shot mode.

9.4.11.7.1 Abort chain when baud rate is non-zero

As already described, for non-zero value of STBRR[BR] field, the test channel conversion is performed at the end of (BR+1) number of chains. If abort chain occurs during the chain in which test channel is scheduled to be converted, then test channel is converted after next (BR+1) number of chains.

For example, if STBRR.BR field is programmed to 2, the sequence will be two normal chains (without any test channel conversion) followed by chain with test channel converted at the end. Now, if abort chain occurs during first two chains it is treated as normal chain abort and test channel is converted at the end of 3rd chain only (as the case without any abort chain). But if abort chain occurs during the 3rd chain in which test channel is scheduled to be converted, then test channel is converted after next three chains — that is, at the end of 6th chain (counting from the beginning).

10 Boot Assist Module (BAM)

10.1 Overview

The Boot Assist Module (BAM) is a block of read-only memory containing VLE code which is executed according to the boot mode of the device. The code stored in the BAM is not executed when booting in Single Chip mode (see *Chapter 6: Device Boot Modes*), except when entering the "Static mode" in case no valid bootable sector has been found.

The BAM downloads code into internal SRAM through the following serial protocols and executes it afterwards:

- FlexCAN
- LINFlex-UART

For cut2/3, dependent on the selected boot mode (see *Section 10.4.1, Entering boot modes*), any download is performed either with a fixed baud rate or after running a short sequence to measure the selected baud rate (with “autobaud”). See *Section 10.4.5, Boot with Autobaud feature [cut2/3 only]*, for further information about the autobaud feature.

Additionally, for cut2/3, the BAM provides a short code sequence to retrieve factory settings from the Test Flash as convenience software. Please refer to *Section 10.4.6, Reading from Test Flash [cut2/3only]*, for further information.

10.2 Features

The BAM provides the following features:

- Programmable 64-bit password protection for serial boot mode
- Serial boot loads the application boot code from a FlexCAN or LINFlex bus into internal SRAM
- Censorship protection for internal flash module
- Detection of the selected baud rate in autobaud mode

10.3 Memory map

The BAM code resides in 8 KB of ROM mapped from address 0xFFFF_C000. The address space and memory used by the BAM is shown in *Table 64*.

Table 64. BAM memory organization

Entity	Address
BAM entry point	0xFFFF_C000
RAM area used by the BAM code (do not use)	0x4000_0000–0x4000_00FF
Downloaded code base address	0x4000_0100

The RAM location where to download the code can be any 4 byte-aligned location in the SRAM starting from the address 0x4000_0100.

Caution: Do not use the RAM area used by the BAM code as indicated in *Table 64*.

10.4 Functional description

10.4.1 Entering boot modes

The SPC56ELxx detects the boot mode based on external pins and device status. The following sequence applies (see *Figure 60*):

1. To boot either from FlexCAN or LINFlex, the device must be forced into an Alternate Boot Loader Mode via the FAB (Force Alternate Boot Mode) pin which must be asserted before initiating the reset sequence. The type of alternate boot mode is selected according to the ABS (Alternate Boot Selector) pins (see *Table 65*).
2. If FAB is not asserted, the device boots from the first flash memory sector which contains a valid boot signature.
3. If no flash memory sector contains a valid boot signature, the device will go into static mode.

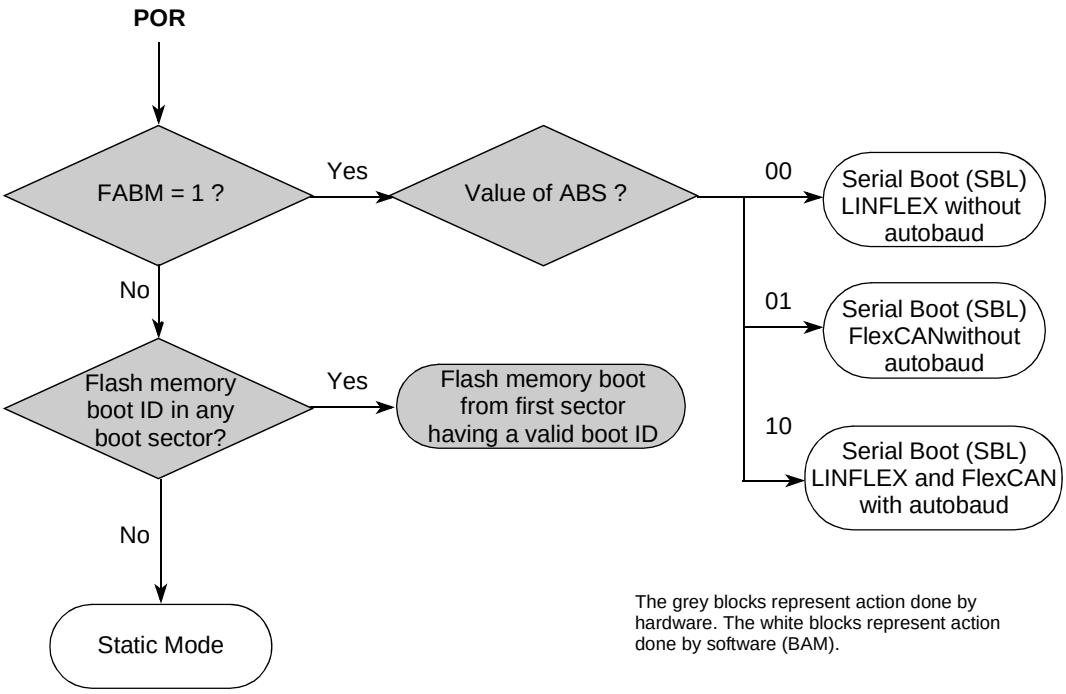


Figure 60. Boot mode selection

Table 65. Hardware configuration to select boot mode

FAB	ABS[2,0]	Standby-RAM Boot Flag	Boot ID	Boot Mode
1	00	0	—	LINFlex without autobaud
1	01	0	—	FlexCAN without autobaud
1	10	0	—	Scan of both serial interfaces (FlexCAN and LINFlex) with autobaud

10.4.2 Boot through BAM

10.4.2.1 Executing BAM

Single chip boot mode (see *Section 6.2.1, Single Chip boot mode*) is managed by hardware and BAM doesn't participate in it.

BAM is executed only on one or both of the following cases:

- serial boot mode has been selected by FABM pin;
- hardware hasn't found a valid Boot ID in any flash memory boot locations.

If one of these conditions is true, the device fetches code at location 0xFFFF_C000 and the BAM application starts.

10.4.2.2 BAM software flow

Figure 61 describes the BAM logic flow.

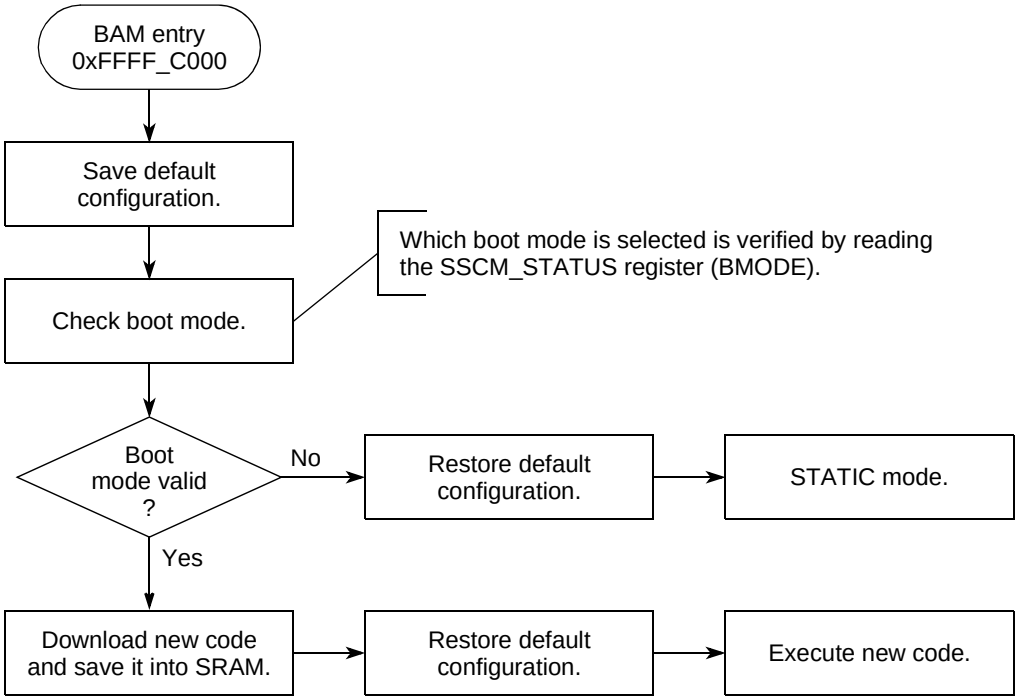


Figure 61. BAM logic flow

The first action is to save the initial device configuration. In this way it is possible to restore the initial configuration after downloading the new code but before executing it. This allows the new code to be executed as the device was just coming out of reset.

The SSCM_STATUS[BMODE] field (see *Section 49.3.1.1, System Status Register (STATUS)*) indicates which boot has to be executed (see *Table 66*).

If BMODE field shows either a single chip value (011) or the reserved values, the boot mode is not considered valid and the BAM pushes the device into static mode.

In all other cases data is downloaded in serial boot mode and saved into the correct SRAM location.

Table 66. Fields of SSCM STATUS register read by BAM to detect the chosen boot mode

Field	Description
BMODE	BMODE Device Boot Mode. 000 Serial Boot Loader (FlexCAN or LINFlex) with autobaud 001 FlexCAN Serial Boot Loader (without autobaud) 010 LINFlex Serial Boot Loader (without autobaud) 011 Single Chip other values are reserved

Then, the initial device configuration is restored and the code jumps to the address provided for the downloaded code. At this point BAM has just finished its task.

If there is any error (that is, communication error, wrong boot selected, etc.), BAM restores the default configuration and puts the device into static mode. Static mode means the device enters the low power mode SAFE and the processor executes a wait instruction. It is needed if the device can not boot in the mode which was selected. During BAM execution and after, the mode reported by the field S_CURRENT_MODE of the register ME_GS in the module MC_ME Module is “DRUN”.

10.4.2.3 BAM resources

BAM uses/initializes the following MCU resources:

- MMU is programmed to support either VLE or Power Architecture Technology (dependent on downloaded code)
- Cache is disabled
- MC_ME and MC_CGM modules to initialize mode and clock sources
- CAN_0, LINFlex_0 and their pins when performing serial boot mode
- SWT is disabled in case of a serial boot mode or when entering static mode
- PIT for time measurement when performing serial boot mode
- SSCM to check the boot mode and during password check (see *Table 66* and *Figure 62*)
- External oscillator (XOSC)
- SIUL to perform programming of the required GPIO pins

On cut2/3, the following hardware resources are used only when autobaud feature is selected:

- STM to measure the baud rate
- CMU to measure the external clock frequency related to the internal IRCOSC clock source
- FMPLL to work with the system clock near the maximum allowed frequency (for higher resolution during baud rate measurement).

The initial configuration is restored before executing the downloaded code.

For cut2/3, when the autobaud feature is disabled, the system clock is selected directly from the external oscillator. Thus the frequency of the external oscillator defines the baud rate for serial interfaces used to download the user application. For a LINFlex transmission, the selected baud rate is f_{XOSC} / 833. For a FlexCAN transmission, the selected baud rate is f_{XOSC} / 40.

10.4.2.4 Download and execute the new code

From high level perspective, the download protocol follows steps:

0. (only in autobaud mode)
   transmit message for autobaud measurement and subsequent baud rate selection
1. Transmit 64 bits password
2. Transmit start address, size of downloded code in bytes and VLE bit
3. Transmit download data
4. Execute code from start address.

Each step must be completed before the next step starts.

The communication is done in half duplex manner, any transmission from host is followed by the MCU transmission:

- Host sends data to MCU and starts waiting
- MCU echoes to host the data received
- Host verifies if echoes are correct
  - if data is correct, the host can continue to send data
  - if data is not correct, the host stops to transmit and MCU need to be reset.

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

A more detailed description of these steps follows.

10.4.2.5 Download 64-bit password and password check

The first 64 bits received represent the password. This password is sent to the Password Check procedure which verifies if it is correct.

Password check data flow is shown in *Figure 62* where:

- SSCM_STATUS.PUB = 1 specifies serial boot mode with public password
- SSCM_STATUS.SEC = 1 specifies that the flash memory is secured

In case of serial boot mode with public password, the received password is compared with the public password 0xFEED_FACE_CAFE_BEEF.

If public access is not allowed but the flash is not secured, the received password is compared with the 64 bit value saved in the NVPWD0 and NVPWD1 locations in the shadow flash.

In both cases, comparison is done by the BAM code. If it fails, BAM pushes the device into static mode.

If the public password is not allowed and the flash memory is secured, the received password is compared by hardware against the password stored in NVPWD0 and NVPWD1. Only in this case (no public password and secured flash), the words of the provided password must be swapped (NVPWD1|NVPWD0).

After a fixed time waiting:

- If the correct password was supplied, flash is now unsecured and BAM continues its task
- If an incorrect password was supplied, flash is still secured; BAM puts the device into static mode

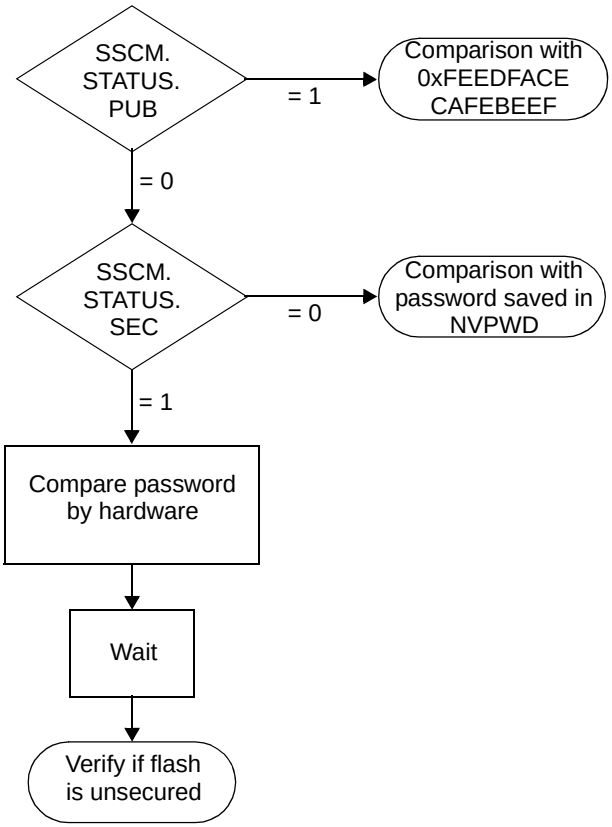


Figure 62. Password check flow

10.4.2.6 Download start address, VLE bit and code size

The next 8 bytes received by the MCU contain a 32-bit Start Address, the VLE mode bit and a 31-bit code Length as shown in *Figure 63*.

The VLE bit (Variable Length Instruction) is used to indicate for which instruction set the code has been compiled. The BAM supports the download of VLE code (VLE bit = 1) and Power Architecture Technology (VLE bit = 0). In case of Power Architecture Technology, the MMU page for the RAM space (0x4000_0000 - 0x7FFF_FFFF) will be modified to select Power Architecture. Otherwise the MMU page for this page will continue to select VLE code.

Note: Power Architecture Technology is supported in cut2 and beyond only.

The Start Address defines where the received data will be stored and where the MCU will branch after the download is finished. The two LSB bits of the Start Address are ignored by the BAM program, such that the loaded code should be 32-bit word aligned.

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

START_ADDRESS[31:16]

START_ADDRESS[15:0]

VLE	CODE_LENGTH[30:16]

CODE_LENGTH[15:0]

Figure 63. Start address, VLE bit, and download size in bytes

10.4.2.7 Download data

Each byte of data received is stored into device's SRAM, starting from the address specified in the previous protocol step. It is not verified whether the provided address is a valid address in SRAM or is writable.

The address increments until the number of bytes of data received matches the number of bytes specified in the previous protocol step.

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

Then a "dummy" word (0x0000_0000) is written to avoid a possible ECC error during core prefetch.

10.4.2.8 Execute code

The BAM program waits for the last echo message transmission being completed.

Then it restores the initial MCU configuration and jumps to the code loaded at Start Address which was received in step 2 of the protocol.

At this point BAM has finished its tasks and MCU is controlled by new code executing from SRAM.

10.4.3 UART Boot — autobaud disabled

10.4.3.1 Configuration

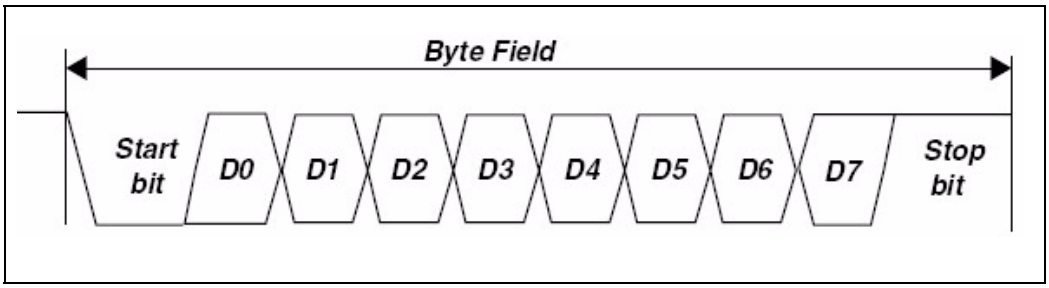
Boot using the UART protocol is implemented by LINFlex_0 module. Pins used are:

- LINFlex_TX corresponds to pin B[2]
- LINFlex_RX corresponds to pin B[3].

The LINFlex controller is configured to operate at a baud rate of f_{XOSC} / 833, using 8 bit data frame without parity bit and 1 stop bit. For cut2/3, when autobaud feature is disabled, the

system clock is driven by the external oscillator. For cut1, the system clock is driven by the internal RC oscillator (16 MHz), resulting in a baud rate of f_{XOSC} / 833 = 19200.

Table 67. LINFlex bit timing in UART mode



10.4.3.2 Protocol

Table 68 summarizes the protocol and BAM action during this boot mode.

Table 68. UART boot mode download protocol (autobaud disabled)

Protocol step	Host sent message	BAM response message	Action
1	64-bit password (MSB first)	64-bit password	Password checked for validity and compared against stored password.
2	32-bit store address	32-bit store address	Load address is stored for future use.
3	VLE bit + 31-bit number of bytes (MSB first)	VLE bit + 31-bit number of bytes (MSB first)	Size of download is stored for future use. Verify VLE bit.
4	8 bits of raw binarydata	8 bits of raw binarydata	4 x 8 bits of data are packed into 32-bit words. These words are saved into SRAM starting from the “Load address”. “Load address” incrementes until the number of data received and stored matches the size as specified in the previous step.
5	none	none	Branch to downloaded code

10.4.4 CAN Boot — autobaud disabled

10.4.4.1 Configuration

Boot using the CAN protocol is implemented by FlexCAN_0 module. Pins used are:

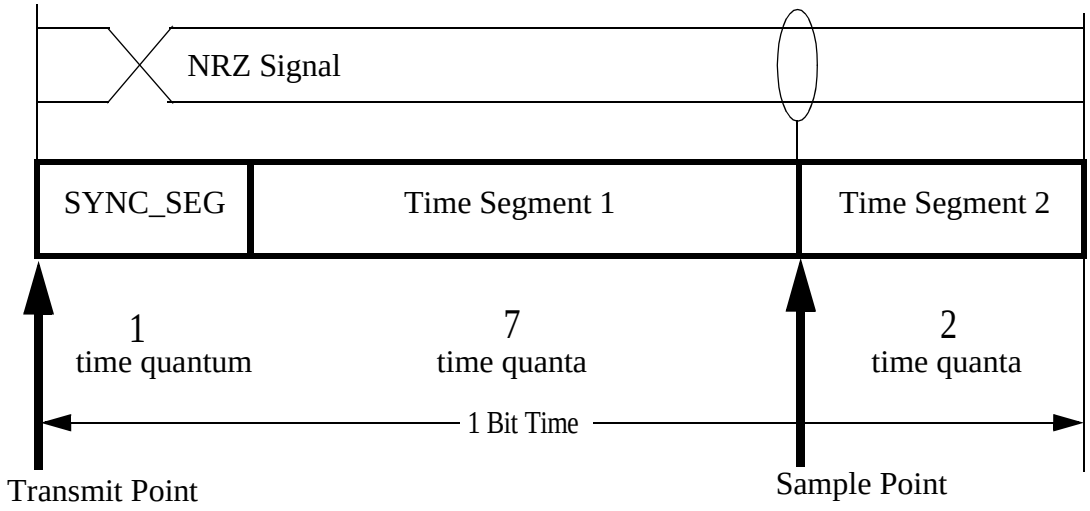
- CAN_TX corresponds to pin B[0]
- CAN_RX corresponds to pin B[1].

Boot from FlexCAN with autobaud disabled uses the system clock driven by the external oscillator.

The FlexCAN controller is configured to operate at a baud rate = system clock frequency/40.

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 64*.



1 time Quantum = 4 system clock periods

Figure 64. FlexCAN bit timing

10.4.4.2 Protocol

Table 69 summarizes the protocol and BAM action during this boot mode. All data is transmitted byte wise.

Table 69. FlexCAN boot mode download protocol (autobaud disabled)

Protocol step	Host sent message	BAM response message	Action
1	FlexCAN ID 0x011+ 64-bit password	FlexCAN ID 0x001+ 64-bit password	Password checked for validity and compared against stored password.
2	FlexCAN ID 0x012+VLE bit+ 31-bit number of bytes	FlexCAN ID 0x002+ 32-bit store address+ VLE bit+ 31-bit number of bytes	Load address is stored for future use. Size of download are stored for future use. Verify VLE bit. 32-bit store address+
3	FlexCAN ID 0x013+ 8 to 64 bits of raw binary data	FlexCAN ID 0x003+ 8 to 64 bits of raw binary data	4 x 8 bits of data are packed into 32-bit words. These words are saved into SRAM starting from the “Load address”. “Load address” increments until the number of data received and stored matches the size as specified in the previous step.
4	none	none	Branch to downloaded code

10.4.5 Boot with Autobaud feature [cut2/3 only]

The purpose of Autobaud is to allow boot operation with a wide range of baud rates independent of the external oscillator frequency.

Note: The baud rates achievable by the autobaud feature are limited by the software based measurement method and the effects resulting from different clocking schemes used by the host and the target.

The maximum baud rate for a CAN transmission allowing a stable transmission is in the range of 125 kBaud, assuming that the bit sampling point is programmed to be in the middle of a bit time and a maximum resynchronization jump width is selected. This requires to select the CAN parameters PROP_SEG, PHASE_SEG1, PHASE_SEG2 accordingly. The maximum baud rate for an UART transmission allowing a stable transmission is in the range of 48 kBaud when using an external oscillator frequency of maximum 40MHz, a slower external clock will scale the achievable baud rate accordingly. Transmissions at higher baud rates are not recommended.

Note: Autobaud is supported on cut3 only on unsecured devices; it is not available on devices with a censored flash.

10.4.5.1 Configuration and detection/measurement flow

Baud rate measurement is using the System Timer Module (STM) which is driven by the system clock. Measurement itself is performed by software polling the related inputs as general purpose IO's, resulting in a detection granularity that is directly related to the execution speed of the software.

One main difference of the autobaud feature is that the system clock is not driven directly by the external oscillator, but it is driven by the FMPLL output. The reason is that to have an optimum resolution for baud rate measurement, the system clock needs to be nearer to the maximum allowed device's frequency.

This is achieved with the following two steps:

1. using the Clock Monitor Unit (CMU) and the internal RC oscillator (IRC), the external frequency is measured using the IRC as reference to determine this frequency.
2. based on the result of this measurement, the FMPLL is programmed to generate a system clock that is configured to be near, but lower, to the maximum allowed frequency.

After setting up the system clock, the BAM code continously searches for a falling edge on either the CAN RX input or the UART RX input; a falling edge of the CAN RX input takes precedence. For this purpose. the CAN RX and UART RX inputs are both configured as GPIO inputs. In case a falling edge is detected on any input, the corresponding autobaud measurement functionality is started:

- a falling edge on CAN RX (corresponds to pin B[1]) starts the CAN autobaud measurement and then sets up the FlexCAN baud rate accordingly
- a falling edge on UART RX (corresponds to pin B[3]) starts the UART autobaud measurement and then sets up the LINFlex baud rate accordingly

After performing the autobaud measurement and setting up the baud rate, the corresponding RX input is reconfigured and the related standard download process is started; in case of a detected CAN transmission a download using the CAN protocol as described in *Section 10.4.4, CAN Boot — autobaud disabled*, and in case of a detected UART transmission a download using the UART protocol as described in *Section 10.4.3,*

UART Boot — autobaud disabled.

The following *Figure 65* identifies the corresponding flow and steps.

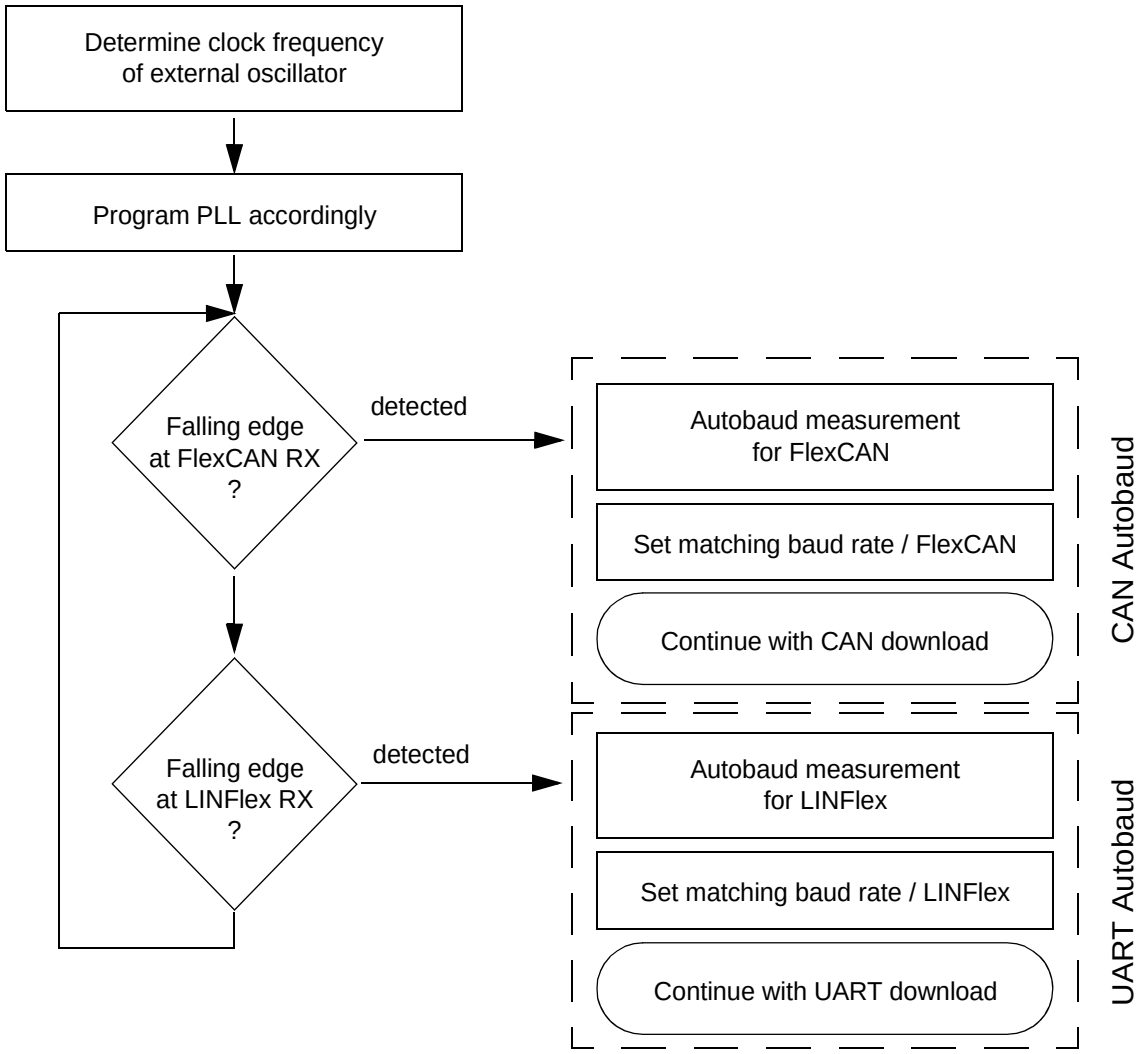


Figure 65. Autobaud configuration and detection/measurement flow

10.4.5.2 Autobaud measurement for LINFlex / UART protocol

Enabling the autobaud feature when using the UART protocol assumes that first an additional byte is sent from host to MCU. Its value is 0x00, resulting in a high - low - high transition visible on the UART RX pin.

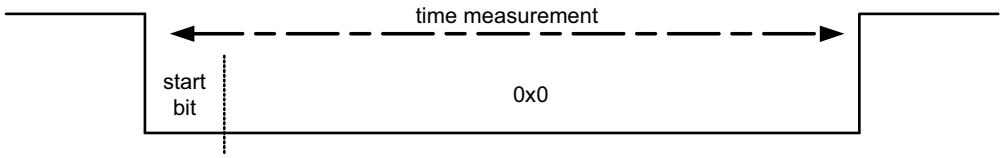


Figure 66. Autobaud measurement / UART protocol

Initially the UART RX pin is configured as GPIO input and the BAM code waits polling for the first falling edge. Upon detection of this edge, the STM starts. Subsequently, the UART RX pin is again polled, waiting for a rising edge. Upon detection of this edge, the STM is stopped and from the measured time, the used baud rate is computed. The error introduced due to this polling will be small, but might be visible.

Higher Baud rates may be used, but customers will be required to ensure they fall within acceptable error ranges. This is shown in *Figure 67* which shows the effect of quantization error on the baud rate selection.

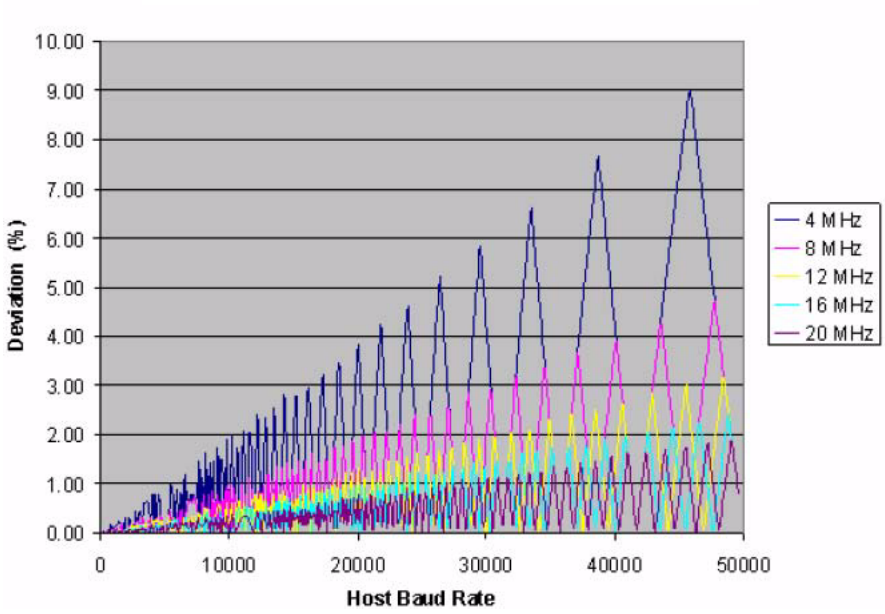


Figure 67. Baud rate Deviation Between Host and SPC56ELxx

This additional, first byte is used to measure the transmission time from the falling edge until the rising edge. Its length will be equivalent to a start bit (low), followed by eight data bits (low), followed by a stop bit (high); therefore the low transmission time should be equivalent to 9 bit times.

From the measured time it is possible to determine the used baud rate in relation to the current system clock selection. The following equation gives the relation between baud rate and LINFlex register configuration:

$$LDIV = \frac{F_{cpu}}{16 \cdot baudrate}$$

LDIV is an unsigned fixed point number and its value is reflected in the related LINFlex's registers (LINIBRR, LINFBRR).

Upon reception of this zero byte and the configuration of the corresponding baud rate, an acknowledge byte is returned by the BAM code to the host using the selected baud rate. This acknowledge byte is the ASCII char "Y" (0x59). From this point BAM follows the normal UART mode boot protocol.

10.4.5.3 Autobaud measurement for FlexCAN / CAN protocol

Enabling the autobaud feature when using the CAN protocol assumes that the following additional CAN frame is sent initially from the host to the MCU for baud measurement purposes:

- Standard identifier = 0x0,
- Data Length Code (DLC) = 0x0.

As all the bits to be transmitted are dominant bits, there is a succession of 5 dominant bits and 1 stuff bit on the FlexCAN network (see *Figure 68*).

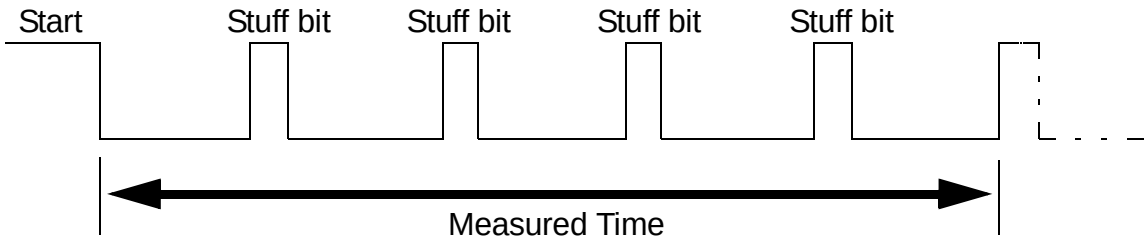


Figure 68. Autobaud measurement / CAN protocol

Initially the CAN RX pin is configured as GPIO input and the BAM code waits polling for the first falling edge. Upon detection of this edge, the STM starts. Subsequently, the CAN RX pin is again polled, waiting for a series of low-high transitions. Upon detection of this sequence, the STM is stopped and from the measured time, the used baud rate is computed. The error introduced due to this polling will be small, but might be visible.

The calculation of the FlexCAN baud rate allows the operation of the boot loader with a wide range of baudrates. However, the upper and lower limits of related parameters have to be kept in mind, in order to ensure proper data transfer. The maximum permissible baud rate that can be supported is 1Mbit/s.

From this additional frame being transmitted initially, the MCU calculates the corresponding baud rate factor with respect to the current CPU clock and initializes the FlexCAN control register (especially the fields PRESDIV, RJW, PSEG1, PSEG2, and PROPSEG) accordingly.

After measuring a predefined number of bit times, the resulting transmission time can be determined from the STM time base; this value is then used to select PRESDIV and the desired number of time quanta. The number of time quanta in a FlexCAN bit time is given by:

$$BitTime = SYNCSEG + TSEG1 + TSEG2$$

where

- SYNCSEG = Exactly one time quantum.
- TSEG1 = PROGPSEG + PSEG1 +2
- TSEG2 = PSEG2 + 1

FlexCAN protocol specifies that the FlexCAN bit timing should comprise a minimum of 8 time quanta and a maximum of 25 time quanta. Therefore available range is:

$$8 \leq (1 + TSEG1 + TSEG2) \leq 25$$

All other values defining transmission parameters are derived from the number of desired time quantas. To help compensate for any error in the calculated baud rate, the resynchronization jump width will be increased from its default value of 1. TSEG1 and TSEG2 times have been chosen to preserve a sample time in the second half of the bit time.

Upon reception of this initial CAN frame, the corresponding baud rate is configured by the BAM code. From this point BAM follows the normal CAN mode boot protocol.

10.4.5.4 Shadow Flash Code Improvements and Extensions (for cut2 only)

The original BAM ROM code delivers minimal support of the Autobaud feature with the following limitations:

- Only external oscillator frequencies larger than the IRC frequency (~ 16 MHz) are supported, lower frequencies will cause a critical fault during measuring the external oscillator frequency
- measurement inaccuracies for both autobaud measurements (CAN and UART) may result in large quantization errors. Subsequently the calculated baud rate might be not deterministic and fail to match the baud rate selected by the host
- usage of the CAN protocol is not possible with the autobaud feature

To fully support the autobaud feature, an extension of the related BAM code is provided in the Shadow Flash, which corrects the above issues and significantly improves the measurement accuracy.

Note: The corresponding code can only be executed on unsecured devices, on devices with a censored Flash the Shadow Flash can not be accessed in serial boot modes.

The function called by the READ_FROM_TF macro is VLE code. Therefore the MMU programming for the memory range occupied by the BAM (which includes this function) needs to select VLE code, otherwise calling this function will fail.

Since the corresponding code will also be erased when the Shadow Flash is erased, the corresponding code image is available to allow re-programming the related Shadow Flash locations. This re-programming step is required to preserve the related code improvements in the Shadow Flash. It can be skipped, if the usage of the Autobaud feature is not intended.

10.4.6 Reading from Test Flash [cut2/3only]

There are some factory settings stored during final test in the Test Flash (mostly calibration information), which are needed by application software. The related data is:

a) Temperature Sensor 1 Calibration Word 1 .. 4
b) Temperature Sensor 2 Calibration Word 1 .. 4
c) ADC 0 Calibration Word 1 .. 8
d) ADC 1 Calibration Word 1 .. 8
e) Part ID 1 L/H
f) Part ID 2

However, accessing the Test Flash requires to set the SCTR TFE bit in the SSCM, which

- will replace the “normal” Flash memory space with the Test Flash block (while TFE is set), and
- is only possible one time after a reset of the device

Therefore retrieving these values usually requires several steps when performed by code stored in Flash:

1. copying a corresponding function XYZ( ) to access this data from the “normal” Flash into RAM
2. switching between “normal” Flash and Test Flash access by setting the SCTR TFE bit
3. executing the function XYZ( ) to copy the data from Test Flash into RAM
4. clear the SCTR TFE bit to switch back to “normal” Flash

This has been reported as inconvinient by customers, therefore a supporting function has been added to the BAM, which will perform the corresponding actions. This function will copy the above factory settings into a given memory location, without the need to copy the corresponding code into the RAM. (g)

The location of this function is identified by a vector located in the location at address 0xFFFF_DFF0. This function receives a 32-bit address, which specifies the start address of a 1024 byte wide buffer that will receive the data to be retrieved. This function returns a 32 bit value that denotes either SUCCESS (value 0), an erroneous second attempt to access the Test Flash (value 4), or an erroneous access due to the fact that the Test Flash is not available or can not be accessed due to other reasons (value 8). The provided header file specifies a convenience function call macro **`READ_FROM_TF(<buffer_loc>,<result>)`** that allows to call this function with its parameter <buffer_loc> and provides the return status in <result>. It further specifies some convenience code to access the retrieved information.

Note: Since the “normal” Flash and related interrupt and/or exception handlers defined in this memory space are not available during the runtime of this function, it is the responsibility of the calling code to ensure that no interrupts or exceptions can occur.

10.4.7 Inhibiting BAM operation

Under certain circumstances, you may want to inhibit BAM operation. To do this, set the ERROR[RAE] bit in the SSCM (see *Section 49.3.1.3, Error Configuration (ERROR)*). Any

g. Please note that this function cannot be recommended for use in a safety case as defined by the Safety Application Guide.

attempt to access the memory range occupied by the BAM will then result in an access error.

10.4.8 Interrupt

No interrupts are generated by or are enabled by the BAM.

11 Clock Architecture

11.1 Clock generation

The clock generation for this device is illustrated in *Figure 69*.

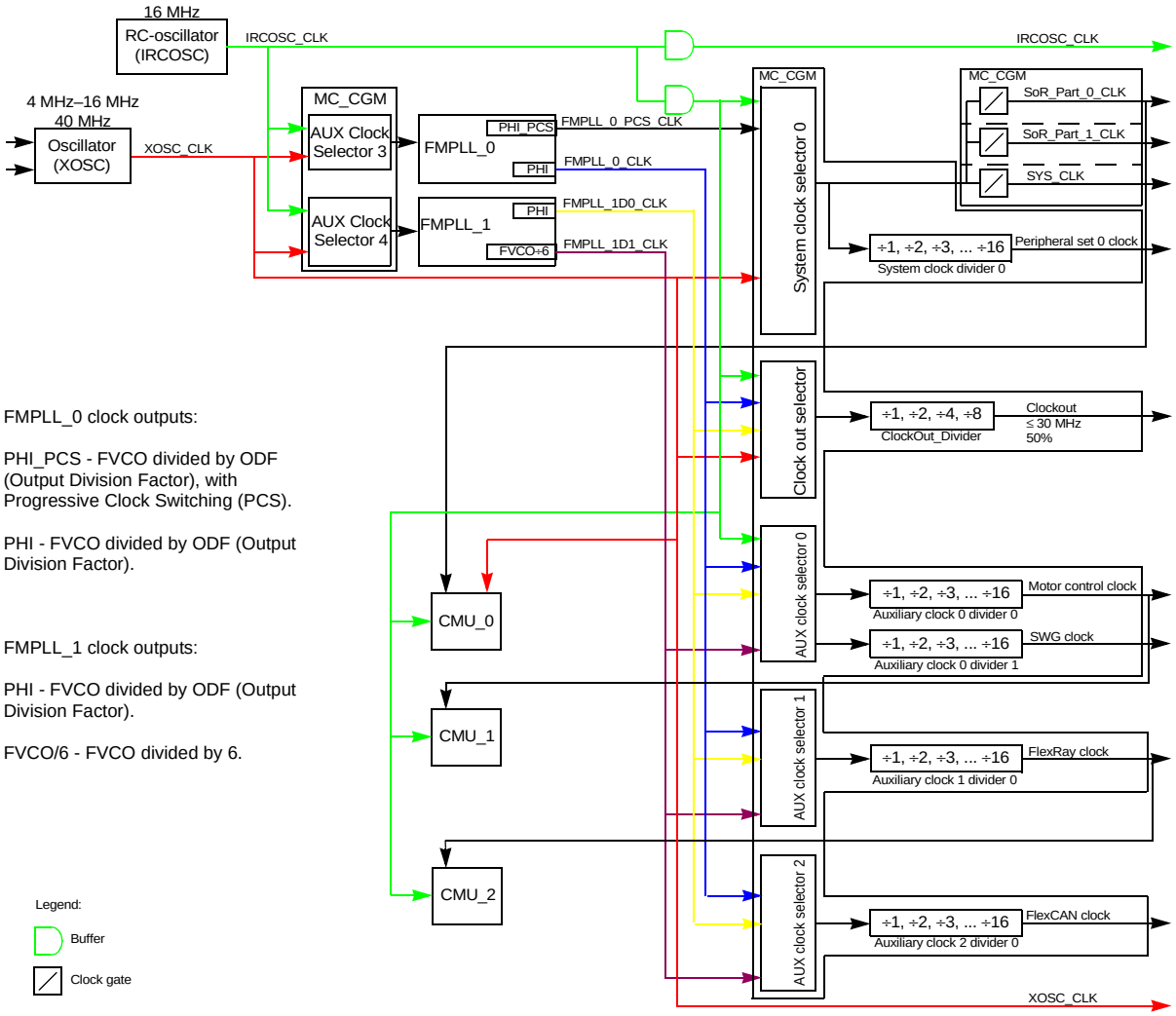


Figure 69. System clock generation

11.2 Clock distribution

Table 70 describes the clock distribution on this chip.

Table 70. Clock distribution

Module name	Clock	Frequency	Supports FM on PLL
ADC_0	motor control	≤ 120 MHz	yes
ADC_1	motor control	≤ 120 MHz	yes
BAM	system	≤ 120 MHz	yes
Core_0	system	≤ 120 MHz	yes
Core_1	system	≤ 120 MHz	yes
CRC	system	≤ 120 MHz	yes
CTU	motor control	≤ 120 MHz	yes
DMA_MUX	system	≤ 120 MHz	yes
DSPI_0	peripheral set 0	≤ 120 MHz	yes
DSPI_1	peripheral set 0	≤ 120 MHz	yes
DSPI_2	peripheral set 0	≤ 120 MHz	yes
ECSM_0	system	≤ 120 MHz	yes
ECSM_1	system	≤ 120 MHz	yes
eDMA_0	system	≤ 120 MHz	yes
eDMA_1	system	≤ 120 MHz	yes
eTimer_0	motor control	≤ 120 MHz	yes
eTimer_1	motor control	≤ 120 MHz	yes
eTimer_2	motor control	≤ 120 MHz	yes
FCCU (REG)	system	≤ 120 MHz	yes
FCCU (FSM)	IRCOSC	16 MHz	—
FLASH0	system	≤ 120 MHz	yes
FlexCAN_0 (BIU and MBM)	peripheral set 0	≤ 120 MHz	yes
FlexCAN_0 (CPI)(1)	FlexCAN	≤ 60 MHz	yes
	XOSC	≤ 40 MHz	—
FlexCAN_1 (BIU and MBM)	peripheral set 0	≤ 120 MHz	yes
FlexCAN_1 (CPI)(2)	FlexCAN	≤ 60 MHz	yes
	XOSC	≤ 40 MHz	—
FlexPWM_0	motor control	≤ 120 MHz	yes
FlexPWM_1	motor control	≤ 120 MHz	yes
FlexRay (CHI)	system	32 MHz and ≤ 120 MHz	yes

Table 70. Clock distribution (continued)

Module name	Clock	Frequency	Supports FM on PLL
FlexRay (PE)[3]	FlexRay	80 MHz	no
	XOSC	40 MHz	—
INTC_0	system	≤ 120 MHz	yes
INTC_1	system	≤ 120 MHz	yes
LINFlexD_0	peripheral set 0	≤ 120 MHz	yes
LINFlexD_1	peripheral set 0	≤ 120 MHz	yes
MC_CGM	system	≤ 120 MHz	yes
MC_ME	system	≤ 120 MHz	yes
MC_PCU	system	≤ 120 MHz	yes
MC_RGM	IRCOSC	16 MHz	—
MPU_0	system	≤ 120 MHz	yes
MPU_1	system	≤ 120 MHz	yes
PBRIDGE_0	system	≤ 120 MHz	yes
PBRIDGE_1	system	≤ 120 MHz	yes
PIT	peripheral set 0	≤ 120 MHz	yes
SEMA4_0	system	≤ 120 MHz	yes
SEMA4_1	system	≤ 120 MHz	yes
SIUL (registers)	system	≤ 120 MHz	yes
SIUL (interrupt filters)	IRCOSC	16 MHz	—
SRAM	system	≤ 120 MHz	yes
SSCM	system	≤ 120 MHz	yes
STCU	system	≤ 120 MHz	yes
STM_0	system	≤ 120 MHz	yes
STM_1	system	≤ 120 MHz	yes
SWG	SWG	≤ 20 MHz	yes
SWT_0	IRCOSC	16 MHz	—
SWT_1	IRCOSC	16 MHz	—
WKPU	system	≤ 120 MHz	yes
XBAR_0	system	≤ 120 MHz	yes
XBAR_1	system	≤ 120 MHz	yes

1. Selection between FlexCAN clock and XOSC clock is done via the CLK_SRC bit in the FlexCAN_0's CTRL register.
2. Selection between FlexCAN clock and XOSC clock is done via the CLK_SRC bit in the FlexCAN_1's CTRL register.
3. Selection between FlexRay clock and XOSC clock is done via the CLKSEL bit in the FlexRay's FR_MCR register. Also note that the FlexRay clock is divided by two when used as source for the FlexRay (PE).

11.3 Detailed module descriptions

Additional details on the clock-related modules on this device are provided in the following chapters:

- *Chapter 12: Clock Generation Module (MC_CGM)*
- *Chapter 13: Clock Monitor Unit (CMU)*
- *Chapter 28: Frequency-Modulated Phase-Locked Loop (FMPLL)*
- *Chapter 36: Oscillators*

12 Clock Generation Module (MC_CGM)

12.1 Introduction

12.1.1 Overview

The clock generation module (MC_CGM) generates reference clocks for all the SoC blocks. The MC_CGM selects one of the system clock sources to supply the system clock. The MC_ME controls the system clock selection (see *Chapter 33: Mode Entry Module (MC_ME)*, for more details). A set of MC_CGM registers controls the clock dividers which are used for divided system and peripheral clock generation. The memory spaces of system and peripheral clock sources which have addressable memory spaces are accessed through the MC_CGM memory space. The MC_CGM also selects and generates an output clock.

Figure 70 depicts the MC_CGM block diagram.

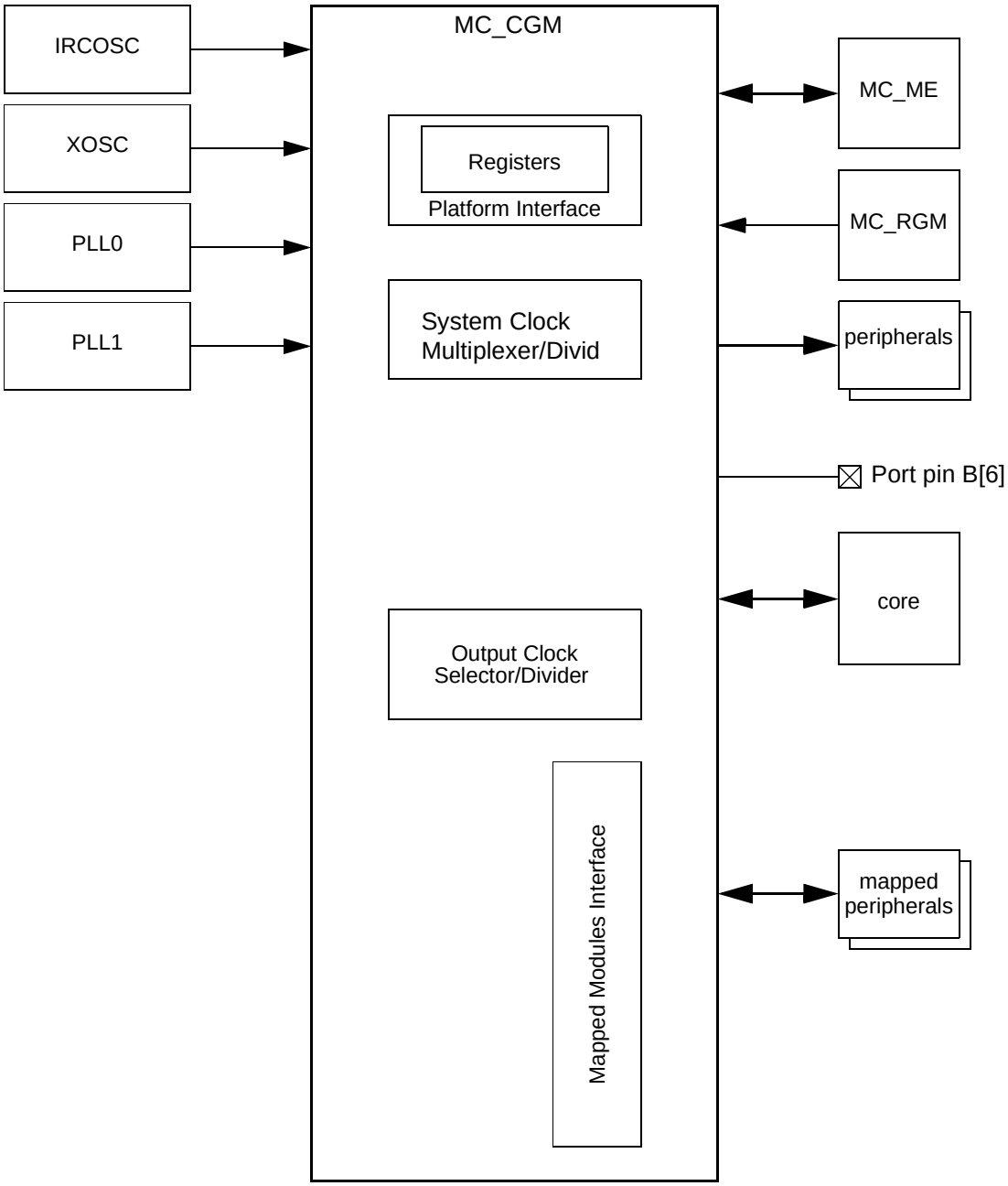


Figure 70. MC_CGM block diagram

12.1.2 Features

The MC_CGM includes the following features:

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

12.2 External signal description

The MC_CGM delivers an output clock to the port pin B[6] for off-chip use and/or observation.

12.3 Memory map and register definition

Table 71. MC_CGM register description

Address	Name	Description	Size	Access		
				User	Supervisor	Test
0xC3FE_0370	CGM_OC_EN	Output Clock Enable	word	read	read/write	read/write
0xC3FE_0374	CGM_OCDS_SC	Output Clock Division Select	byte	read	read/write	read/write
0xC3FE_0378	CGM_SC_SS	System Clock Select Status	byte	read	read	read
0xC3FE_037C	CGM_SC_DC0	System Clock Divider Configuration 0	byte	read	read/write	read/write
0xC3FE_0380	CGM_AC0_SC	Aux Clock 0 Select Control	word	read	read/write	read/write
0xC3FE_0384	CGM_AC0_DC0	Aux Clock 0 Divider Configuration 0	byte	read	read/write	read/write
0xC3FE_0385	CGM_AC0_DC1	Aux Clock 0 Divider Configuration 1	byte	read	read/write	read/write
0xC3FE_0388	CGM_AC1_SC	Aux Clock 1 Select Control	word	read	read/write	read/write
0xC3FE_038C	CGM_AC1_DC0	Aux Clock 1 Divider Configuration 0	byte	read	read/write	read/write
0xC3FE_0390	CGM_AC2_SC	Aux Clock 2 Select Control	word	read	read/write	read/write
0xC3FE_0394	CGM_AC2_DC0	Aux Clock 2 Divider Configuration 0	byte	read	read/write	read/write
0xC3FE_0398	CGM_AC3_SC	Aux Clock 3 Select Control	word	read	read/write	read/write
0xC3FE_03A0	CGM_AC4_SC	Aux Clock 4 Select Control	word	read	read/write	read/write

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 72. MC_CGM memory map

Address	Name	0 16	1 17	2 18	3 19	27 20	5 21	6 22	7 23	8 24	9 25	10 26	11 27	12 28	13 29	14 30	15 31
0xC3FE_0000 … 0xC3FE_001C	XOSC registers																
0xC3FE_0020 … 0xC3FE_003C	reserved																
0xC3FE_0040 … 0xC3FE_005C	reserved																
0xC3FE_0060 … 0xC3FE_007C	IRCOSC registers																
0xC3FE_0080 … 0xC3FE_009C	reserved																
0xC3FE_00A0 … 0xC3FE_00BC	PLL0 registers																
0xC3FE_00C0 … 0xC3FE_00DC	PLL1 registers																
0xC3FE_00E0 … 0xC3FE_00FC	reserved																
0xC3FE_0100 … 0xC3FE_011C	CMU0 registers																
0xC3FE_0120 … 0xC3FE_013C	CMU1 registers																
0xC3FE_0140 … 0xC3FE_015C	CMU2 registers																
0xC3FE_0160 … 0xC3FE_017C	reserved																

Table 72. MC_CGM memory map (continued)

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

Table 72. MC_CGM memory map (continued)

Address	Name		0 / 16	1 / 17	2 / 18	3 / 19	27 / 20	5 / 21	6 / 22	7 / 23	8 / 24	9 / 25	10 / 26	11 / 27	12 / 28	13 / 29	14 / 30	15 / 31
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																
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	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_0380	CGM_AC0_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0384	CGM_AC0_DC0…1	R	DE0	0	0	0	DIV0				DE1	0	0	0	DIV1			
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																

Table 72. MC_CGM memory map (continued)

Address	Name		0 16	1 17	2 18	3 19	27 20	5 21	6 22	7 23	8 24	9 25	10 26	11 27	12 28	13 29	14 30	15 31
0xC3FE_0388	CGM_AC1_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_038C	CGM_AC1_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0390	CGM_AC2_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0394	CGM_AC2_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0398	CGM_AC3_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_03A0	CGM_AC4_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_03A4 … 0xC3FE_3FFC	reserved																	

12.3.1 Register descriptions

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

12.3.1.1 Output Clock Enable Register (CGM_OC_EN)

Figure 71. Output Clock Enable Register (CGM_OC_EN)

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

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

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

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

Table 73. 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

12.3.1.2 Output Clock Division Select Register (CGM_OCDS_SC)

Figure 72. Output Clock Division Select Register (CGM_OCDS_SC)

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

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

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

Table 74. 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 16 MHz int. RC osc. 0001 4-40 MHz crystal osc. 0010 system FMPLL 0011 secondary (80 MHz) FMPLL 0100 reserved 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

12.3.1.3 System Clock Select Status Register (CGM_SC_SS)

Figure 73. System Clock Select Status Register (CGM_SC_SS)

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

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

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

This register provides the current system clock source selection.

Table 75. System Clock Select Status Register (CGM_SC_SS) field descriptions

Field	Description
SELSTAT	**System Clock Source Selection Status** — This value indicates the current source for the system clock. 0000 16 MHz int. RC osc. 0001 reserved 0010 4–40 MHz crystal osc. 0011 reserved 0100 system 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

12.3.1.4 System Clock Divider Configuration Registers (CGM_SC_DC0)

Figure 74. System Clock Divider Configuration Registers (CGM_SC_DC0)

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

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

This register controls the system clock dividers.

Table 76. System Clock Divider Configuration Registers (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 I/O 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 I/O clock remains disabled.

12.3.1.5 Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)

Figure 75. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)

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

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

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

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

- undivided: (unused)
- divided by auxiliary clock 0 divider 0: motor control clock
- divided by auxiliary clock 0 divider 1: sine wave generator clock

Table 77. 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 16 MHz int. RC osc. 0001 reserved 0010 reserved 0011 reserved 0100 system FMPLL 0101 secondary (120 MHz) FMPLL 0110 reserved 0111 reserved 1000 secondary (80 MHz) FMPLL 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

12.3.1.6 Auxiliary Clock 0 Divider Configuration Registers (CGM_AC0_DC0…1)

Figure 76. Auxiliary Clock 0 Divider Configuration Register 0 (CGM_AC0_DC0)

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

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

Figure 77. Auxiliary Clock 0 Divider Configuration Register 1 (CGM_AC0_DC1)

Address0xC3FE_0385 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

These registers control the auxiliary clock 0 dividers.

Table 78. Auxiliary Clock 0 Divider Configuration Registers (CGM_AC0_DC0…1) field descriptions

Field	Description
DE0	**Divider 0 Enable** 0 Disable auxiliary clock 0 divider 0 1 Enable auxiliary clock 0 divider 0
DIV0	**Divider 0 Division Value** — The resultant motor control clock will have a period DIV0 + 1 times that of auxiliary clock 0. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the motor control clock remains disabled.
DE1	**Divider 1 Enable** 0 Disable auxiliary clock 0 divider 1 1 Enable auxiliary clock 0 divider 1
DIV1	**Divider 1 Division Value** — The resultant sine wave generator clock will have a period DIV1 + 1 times that of auxiliary clock 0. If the DE1 is set to 0 (Divider 1 is disabled), any write access to the DIV1 field is ignored and the sine wave generator clock remains disabled.

12.3.1.7 Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)

Figure 78. 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	1	0	0	0	0	0	0	0	0	0	0

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

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

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

Table 79. 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 reserved 0001 reserved 0010 reserved 0011 reserved 0100 system FMPLL 0101 secondary (120 MHz) FMPLL 0110 reserved 0111 reserved 1000 secondary (80 MHz) FMPLL 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

12.3.1.8 Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0)

Figure 79. Auxiliary Clock 1 Divider 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

This register controls the auxiliary clock 1 divider.

Table 80. Auxiliary Clock 1 Divider 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.

12.3.1.9 Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)

Figure 80. Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)

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

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

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

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

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

Table 81. Auxiliary Clock 2 Select Control Register (CGM_AC2_SC) Field Descriptions

Field	Description
SELCTL	**Auxiliary Clock 2 Source Selection Control** — This value selects the current source for auxiliary clock 2. 0000 reserved 0001 reserved 0010 reserved 0011 reserved 0100 system FMPLL 0101 secondary (120 MHz) FMPLL 0110 reserved 0111 reserved 1000 secondary (80 MHz) FMPLL 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

12.3.1.10 Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0)

Figure 81. Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0)

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

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

This register controls the auxiliary clock 2 divider.

Table 82. Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0) field descriptions

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

12.3.1.11 Auxiliary Clock 3 Select Control Register (CGM_AC3_SC)

Figure 82. Auxiliary Clock 3 Select Control Register (CGM_AC3_SC)

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

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

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

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

- undivided: PLL0 reference clock

Table 83. Auxiliary Clock 3 Select Control Register (CGM_AC3_SC) Field Descriptions

Field	Description
SELCTL	**Auxiliary Clock 3 Source Selection Control** — This value selects the current source for auxiliary clock 3. 0000 16 MHz int. RC osc. 0001 4–40 MHz crystal osc. 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

12.3.1.12 Auxiliary Clock 4 Select Control Register (CGM_AC4_SC)

Figure 83. Auxiliary Clock 4 Select Control Register (CGM_AC4_SC)

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

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

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

This register is used to select the current clock source for the FMPLL1 reference clock.

See *Figure 89* for details.

Table 84. Auxiliary Clock 4 Select Control Register (CGM_AC4_SC) Field Descriptions

Field	Description
SELCTL	**Auxiliary Clock 4 Source Selection Control** — This value selects the current source for auxiliary clock 4. 0000 16 MHz int. RC osc. 0001 4-40 MHz crystal osc. 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

12.4 Functional description

12.4.1 System clock generation

Figure 84 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 16 MHz int. RC osc. as the system clock and to ignore the system clock select.

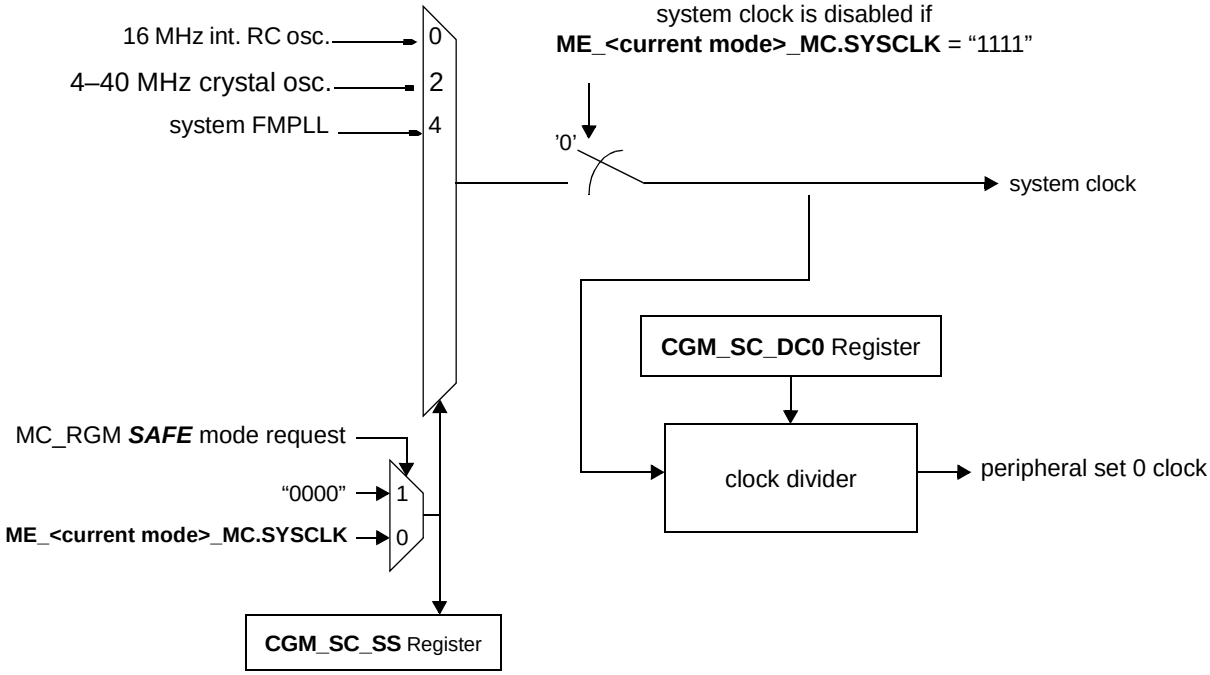


Figure 84. MC_CGM system clock generation overview

12.4.1.1 System clock source selection

During normal operation, the system clock selection is controlled

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

12.4.1.2 System clock disable

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

12.4.1.3 System clock dividers

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

- peripheral I/O clock - controlled by the **CGM_SC_DC0** register

12.4.2 Auxiliary clock generation

Figure 84 shows the block diagram of the auxiliary clock generation logic. See *Section 12.3.1.5, Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)*, *Section 12.3.1.7, Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)*, *Section 12.3.1.9, Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)*, *Section 12.3.1.11, Auxiliary Clock 3 Select Control Register (CGM_AC3_SC)*, and *Section 12.3.1.12, Auxiliary Clock 4 Select Control Register (CGM_AC4_SC)* for auxiliary clock selection control.

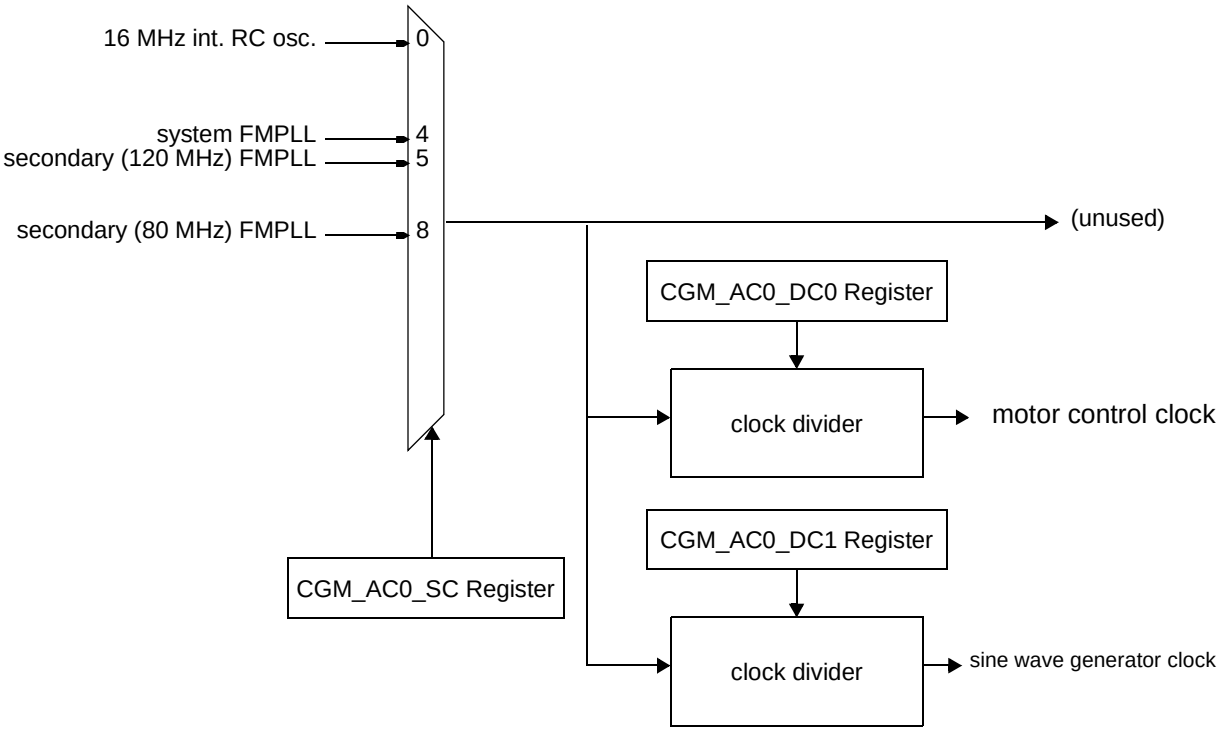


Figure 85. MC_CGM auxiliary clock 0 generation overview

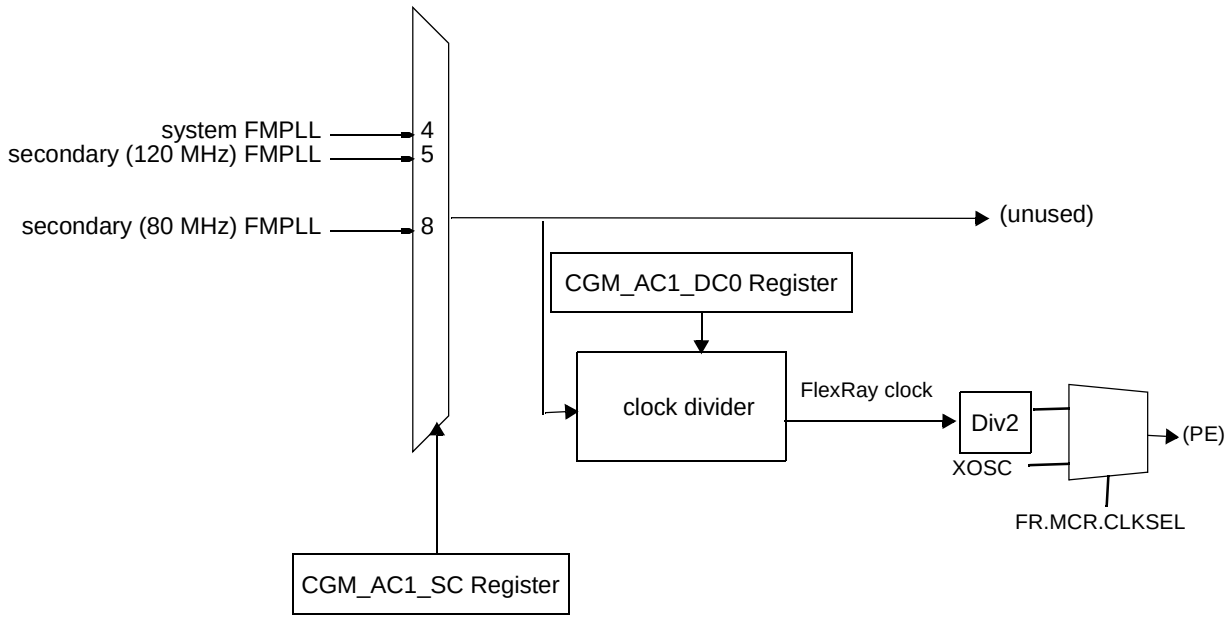


Figure 86. MC_CGM auxiliary clock 1 generation overview

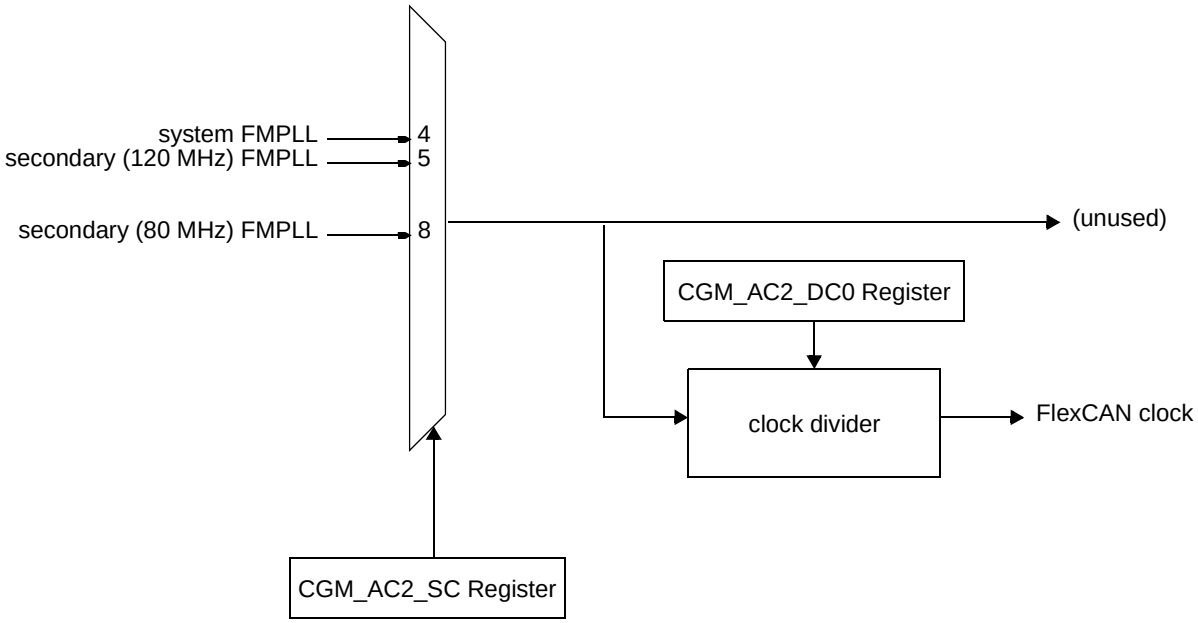


Figure 87. MC_CGM auxiliary clock 2 generation overview

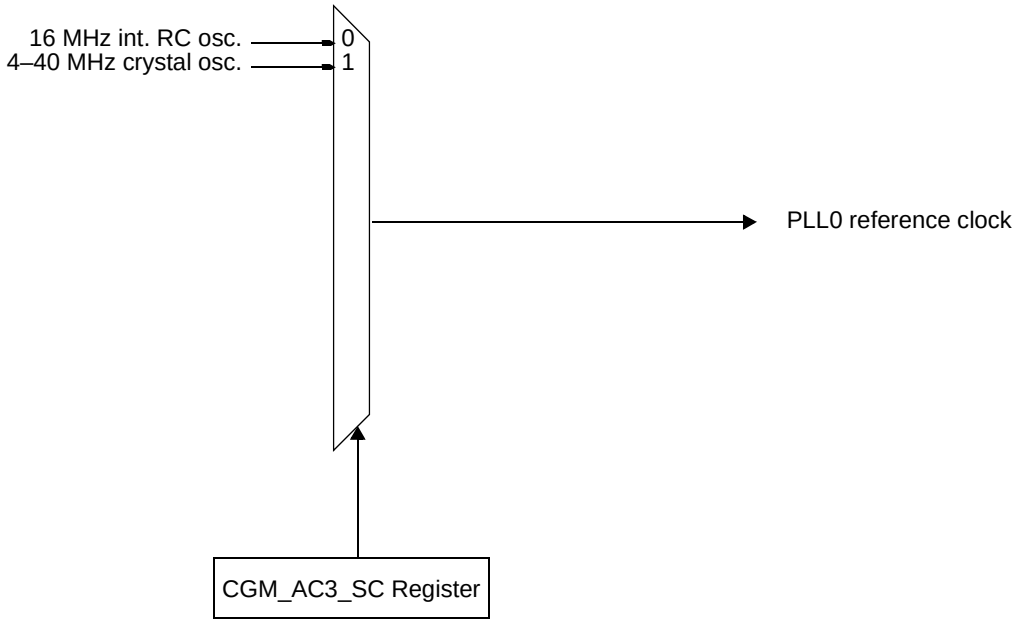


Figure 88. MC_CGM auxiliary clock 3 generation overview

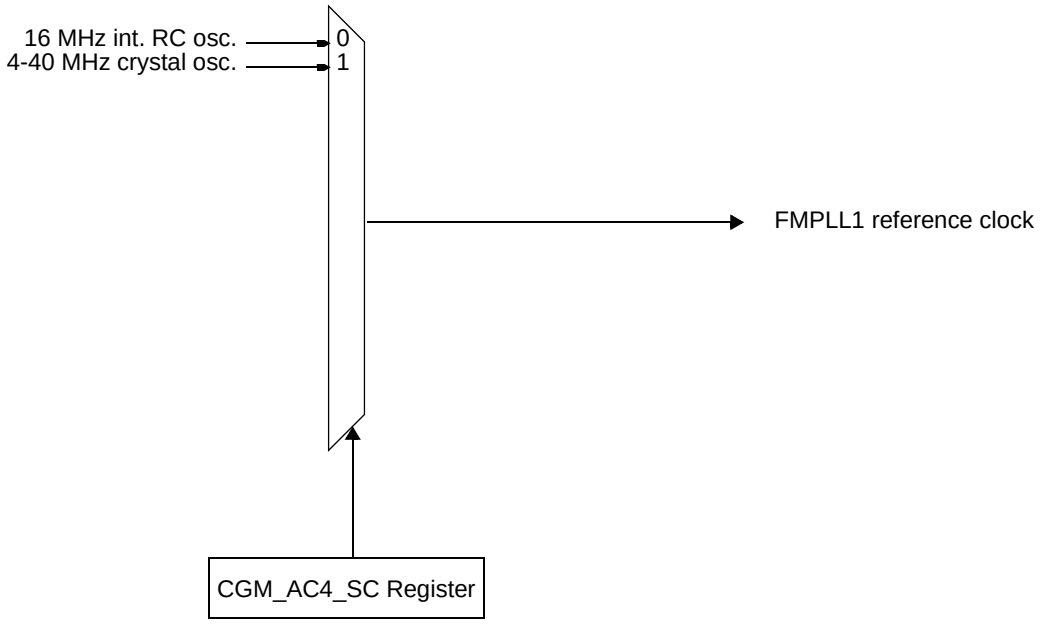


Figure 89. MC_CGM auxiliary clock 4 generation overview

12.4.2.1 Auxiliary clock dividers

The MC_CGM generates the following derived clocks:

- motor control clock - controlled by the **CGM_AC0_DC0** register
- sine wave generator clock - controlled by the **CGM_AC0_DC1** register
- FlexRay clock - controlled by the **CGM_AC1_DC0** register
- FlexCAN clock - controlled by the **CGM_AC2_DC0** register

12.4.3 Functional description of dividers

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 12.3.1.4, System Clock Divider Configuration Registers (CGM_SC_DC0)*
- *Section 12.3.1.6, Auxiliary Clock 0 Divider Configuration Registers (CGM_AC0_DC0…1)*
- *Section 12.3.1.8, Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0)*
- *Section 12.3.1.10, Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_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.

12.4.4 Output clock multiplexing

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

12.4.5 Output Clock Division Selection

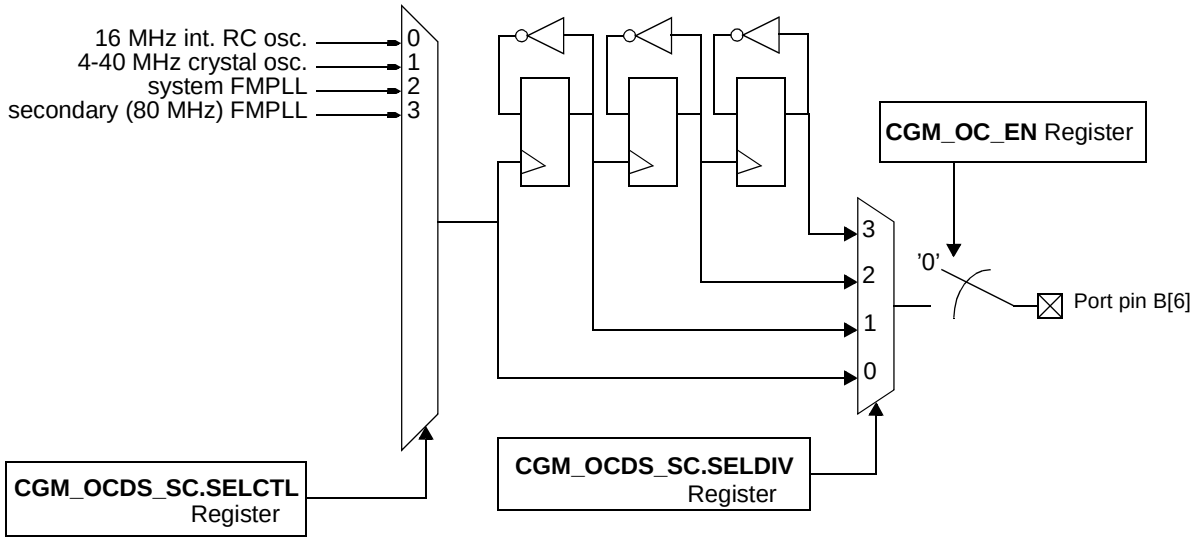


Figure 90. MC_CGM Output Clock Multiplexer and Port Pin B[6] Generation

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

- Port pin B[6] (see *Figure 90*). 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 12.3.1.1, Output Clock Enable Register (CGM_OC_EN)*) which contains the output clock enable/disable control bit.

13 Clock Monitor Unit (CMU)

13.1 Overview

The Clock Monitor Unit (CMU) serves three purposes:

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

When a failure is detected in one of the CMUs, by either the selected clock monitor or the XOSC monitor, the CMU notifies the MC_RGM, the MC_ME, and the FCCU modules. The default behavior is such that a reset occurs and a status bit is set in the MC_RGM. The user also has the option to change the behavior of the action by disabling the reset and selecting an alternate action. The alternate action can be either entering SAFE mode or generating an interrupt.

Table 85. CMU module summary

Module	Monitored clocks
CMU_0	– System clock – XOSC
CMU_1	Motor control clock
CMU_2	FlexRay clock

13.2 Main features

- IRCOSC frequency measurement
- XOSC clock monitoring with respect to (IRCOSC clock) ÷ *n*
- Selected clock frequency monitoring with respect to (IRCOSC clock) ÷ 4
- Event generation for various failures detected inside monitoring unit

13.3 Memory map and register description

The CMU registers are mapped through the MC_CGM (see the memory map in *Chapter 12: Clock Generation Module (MC_CGM)*). The base address for each CMU is shown in *Table 86*.

Table 86. CMU base addresses

Module	Base address
CMU_0	0xC3FE_0100
CMU_1	0xC3FE_0120
CMU_2	0xC3FE_0140

The memory map of each CMU is shown in *Table 87*.

Table 87. CMU memory map

Address offset	Register	Location
0x00	Control status register (CMU_CSR)	*on page 275*
0x04	Frequency display register (CMU_FDR)	*on page 276*
0x08	High-frequency reference register A (CMU_HFREFR_A)	*on page 277*
0x0C	Low-frequency reference register A (CMU_LFREFR_A)	*on page 277*
0x10	Interrupt status register (CMU_ISR)	*on page 278*
0x14	Reserved	
0x18	Measurement duration register (CMU_MDR)	*on page 279*

13.3.1 Control status register (CMU_CSR)

Figure 91. Control status register (CMU_CSR)

Address: Base + 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	SFM	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	CKSEL1		0	0	0	0	0	RCDIV		CME
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Table 88. CMU_CSR field descriptions

Field	Description
SFM	Start frequency measure Set this bit to start a clock frequency measure. The bit is cleared by hardware when the measure is ready in the CMU_FDR register. Software cannot clear this bit. 0 Frequency measurement is completed or not yet started. 1 Frequency measurement is not completed.
CKSEL1	Clock selection This fieldselects the clock to be measured by the frequency meter. For cut1 : 00 IRCOSC_CLK is selected. 01 IRCOSC_CLK is selected. 10 No clock is selected. 11 IRCOSC_CLK is selected. For cut2/3: 00 IRCOSC_CLK is selected. 01 IRCOSC_CLK is selected. 10 IRCOSC_CLK is selected. 11 IRCOSC_CLK is selected.

Table 88. CMU_CSR field descriptions (continued)

Field	Description
RCDIV	IRCOSC clock division factor These bits specify the IRCOSC clock division factor. The output clock is IRCOSC_CLK divided by the factor 2^{RCDIV}. This output clock is compared with XOSC_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	Clock monitor enable 0 Monitor is disabled. 1 Monitor is enabled.

13.3.2 Frequency display register (CMU_FDR)

Figure 92. Frequency display Register (CMU_FDR)

Address: Base + 0x04 Access: User read-only

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

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

Table 89. CMU_FDR field descriptions

Field	Description
FD	Measured frequency bits This register displays the measured frequency f_{IRCOSC_CLK} with respect to f_{XOSC_CLK}. The measured value is given by the following formula: $F_{IRCOSC_CLK} = (F_{XOSC_CLK} \times MD) / n$ where n is the value in CMU_FDR register

13.3.3 High-frequency reference register A (CMU_HFREFR_A)

Figure 93. High-frequency reference register A (CMU_HFREFR_A)

Address: Base + 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	0	0	0	0	HFREF_A											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table 90. CMU_HFREFR_A field descriptions

Field	Description
HFREF_A	High-frequency reference value These bits determine the high reference value for the FMPLL clock. The reference value is given by: (HFREF_A÷16) × (F_{IRCOSC_CLK}÷4).

13.3.4 Low-frequency reference register A (CMU_LFREFR_A)

Figure 94. Low-frequency reference register A (CMU_LFREFR_A)

Address: Base + 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	0	0	0	0	LFREF_A											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 91. CMU_LFREFR_A fields descriptions

Field	Description
LFREF_A	Low-frequency reference value These bits determine the low reference value for the FMPLL clock. The reference value is given by: (LFREF_A÷16) × (F_{IRCOSC_CLK}÷4).

13.3.5 Interrupt status register (CMU_ISR)

Figure 95. Interrupt status register (CMU_ISR)

Address: Base + 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	0	0	0	0	0	0	0	0	0	0	0	0	FLCI_A	FHHI_A	FLLI_A	OLRI
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 92. CMU_ISR field descriptions

Field	Description
FLCI_A	Monitored clock frequency less than reference clock interrupt This bit is set by hardware when both of the following are true: – The monitored clock frequency becomes lower than reference clock frequency (F_{IRCOSC_CLK}÷4) value – The selected clock source is 'ON' (and locked in the case of FMPLL_0 and FMPLL_1). It can be cleared by software by writing 1. 0 No FLC event. 1 FLC event is pending.
FHHI_A	Monitored clock frequency higher than high reference interrupt This bit is set by hardware when both of the following are true: – The monitored frequency becomes higher than HFREF_A value – The selected clock source is 'ON' (and locked in the case of FMPLL_0 and FMPLL_1). It can be cleared by software by writing 1. 0 No FHH event. 1 FHH event is pending.
FLLI_A	Monitored clock frequency less than low reference event This bit is set by hardware when both of the following are true: – The monitored clock frequency becomes lower than LFREF_A value – The selected clock source is 'ON' (and locked in the case of FMPLL_0 and FMPLL_1). It can be cleared by software by writing 1. 0 No FLL event. 1 FLL event is pending.

Table 92. CMU_ISR field descriptions (continued)

Field	Description
OLRI	XOSC clock frequency less than IRCOSC clock frequency event This bit is set by hardware when both of the following are true: – The frequency of XOSC_CLK is less than IRCOSC_CLK/2^{RCDIV} frequency – XOSC_CLK is 'ON' and stable as signaled by the MC_ME. It can be cleared by software by writing 1. 0 No OLR event. 1 OLR event is pending.

13.3.6 Measurement duration register (CMU_MDR)

Figure 96. Measurement duration register (CMU_MDR)

Address: Base + 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	MD[19:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Table 93. CMU_MDR field descriptions

Field	Description
MD	Measurement duration bits This register displays the measured duration in terms of IRCOSC clock cycles. This value is loaded in the frequency meter down-counter. When CMU_CSR[SFM] = 1, the down-counter starts counting.

13.4 Functional description

The names of the clocks involved in this block have the following meaning:

- XOSC_CLK: clock coming from the XOSC
- IRCOSC_CLK: clock coming from the IRCOSC
- CK_PLL: clock coming from the FMPLL
- F_{XOSC_CLK}: frequency of external crystal oscillator clock
- F_{IRCOSC_CLK}: frequency of low frequency internal RC oscillator
- FPLL: frequency of FMPLL clock

13.4.1 XOSC clock monitor

The XOSC clock is monitored by CMU_0. If F_{XOSC_CLK} is smaller than F_{IRCOSC_CLK} divided by $2^{CMU_CSR[RCDIV]}$ and the XOSC is 'ON' and stable as signaled by the MC_ME, then:

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

Note: The XOSC monitor may produce a false event when F_{XOSC_CLK} is less than $2 \times F_{IRCOSC_CLK} \div 2^{CMU_CSR[RCDIV]}$ due to an accuracy limitation of the compare circuitry.

13.4.1.1 System clock monitor

The system clock is monitored by CMU_0. The F_{SYS_CLK} frequency can be monitored by programming CMU_CSR[CME] = 1. SYS_CLK monitoring starts as soon as CMU_CSR[CME] = 1. This monitor can be disabled at any time by writing CME bit to 0.

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

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

If F_{SYS_CLK} is less than a reference clock frequency ($F_{IRCOSC_CLK} \div 4$) and the system clock is enabled, then:

- CMU_ISR[FLCI] is set
- A failure event FLC is signaled to the MC_RGM and Fault Collection Unit, which in turn can generate a 'functional' reset, a SAFE mode request, or an interrupt

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

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

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

13.4.1.2 Motor control clock monitor

The motor control clock is monitored by CMU_1. F_{MOTC_CLK} can be monitored by programming CMU_CSR[CME] = 1. MOTC_CLK monitoring starts as soon as CMU_CSR[CME] = 1. This monitor can be disabled at any time by programming CMU_CSR[CME] = 0.

If F_{MOTC_CLK} is greater than a reference value determined by the CMU_HFREFR_A[HFREF_A] and the currently selected motor control clock source is 'ON' (and locked in the case of FMPLL_0 or FMPLL_1), then:

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

If F_{MOTC_CLK} is less than a reference clock frequency (F_{IRCOSC_CLK}÷4) and the currently selected motor control clock source is 'ON' (and locked in the case of FMPLL_0 or FMPLL_1), then:

- Event pending bit CMU_ISR[FLCI] is set.
- A failure event FLC is signaled to the MC_RGM and FCCU, which in turn can generate a 'functional' reset, a SAFE mode request, or an interrupt.

If F_{MOTC_CLK} is less than a reference value determined by the CMU_LFREFR_A[LFREF_A] bits and the currently selected motor control clock source is 'ON' (and locked in the case of FMPLL_0 or FMPLL_1), then:

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

Note: The motor control clock monitor may produce a false event when F_{MOTC_CLK} is less than $2 \times F_{IRCOSC_CLK} \div 2^{CMU_CSR[RCDIV]}$ due to an accuracy limitation of the compare circuitry.

13.4.1.3 FlexRay clock monitor

The FlexRay clock is monitored by CMU_2. F_{FR_CLK} can be monitored by programming CMU_CSR[CME] = 1. FR_CLK monitoring starts as soon as CMU_CSR[CME] = 1. This monitor can be disabled at any time by programming CMU_CSR[CME] = 0.

If F_{FR_CLK} is greater than a reference value determined by the CMU_HFREFR_A[HFREF_A] bits and the currently selected motor control clock source is 'ON' and locked, then

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

If F_{FR_CLK} is less than a reference clock frequency (F_{IRCOSC_CLK}÷4) and the currently selected motor control clock source is 'ON' and locked, then:

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

If F_{FR_CLK} is less than a reference value determined by the CMU_LFREFR_A[LFREF_A] bits and the currently selected motor control clock source is 'ON' locked, then:

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

Note: The FlexRay clock monitor may produce a false event when F_{FR_CLK} is less than $2 \times F_{IRCOSC_CLK} \div 2^{CMU_CSR[RCDIV]}$ due to an accuracy limitation of the compare circuitry.

13.4.1.4 Frequency meter

The frequency meter, which is part of CMU_0, calibrates the IRCOSC using a known frequency.

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

The reference clock is always the XOSC. A simple frequency meter returns a draft value of IRCOSC_CLK. The measurement starts when CMU_CSR[SFM] is set. The measurement

duration is given by the CMU_MDR register in terms of IRCOSC_CLK cycles with a width of 20 bits. The SFM bit is cleared by the hardware after the frequency measurement is done and the count is loaded in the CMU_FDR. F_{IRCOSC_CLK} can be derived from the value loaded in the CMU_FDR register as follows:

Equation 1 $$F_{IRCOSC_CLK} = (F_{XOSC_CLK} \times MD) \div n$$

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

14 Cross-Triggering Unit (CTU)

14.1 Introduction

In PWM driven systems it is important to schedule the acquisition of the state variables with respect to PWM cycle. State variables are obtained through the following peripherals: ADC, position counter (e.g. quadrature decoder, resolver and sine-cos sensor) and PWM duty cycle decoder.

The cross triggering unit (CTU) is intended to completely avoid CPU involvement in the time acquisitions of state variables during the control cycle that can be the PWM cycle, the half PWM cycle or a number of PWM cycles. In such case the pre-setting of the acquisition times needs to be completed during the previous control cycle, where the actual acquisitions are to be made, and a double-buffered structure for the CTU registers is used, in order to activate the new settings at the beginning of the next control cycle. In addition, there are 4 FIFOs inside the CTU available to store the ADC results.

14.2 Block diagram

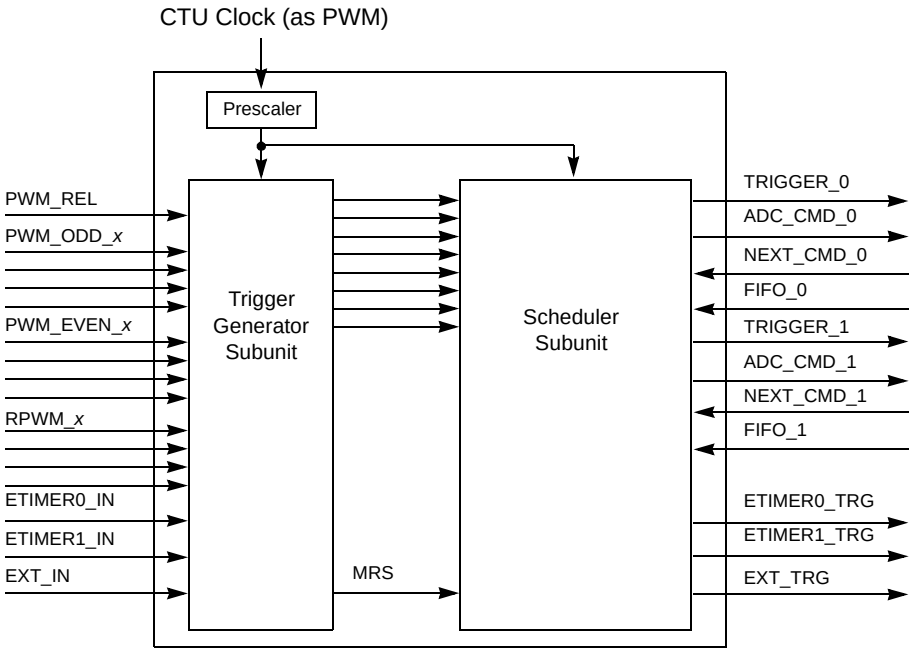


Figure 97. Cross triggering unit block diagram

14.3 CTU overview

The CTU receives various incoming signals from different sources (PWM, timers, position decoder and/or external pins). These signals are then processed to generate up to eight trigger events. An input can be a rising edge, a falling edge or both, edges of each incoming signal. The output can be a pulse or a command (or a stream of consecutive commands for over-sampling support) or both, to one or more peripherals (e.g. ADC, timers and so on).

The CTU interfaces to the following peripherals: (1) PWM (13 inputs); (2) Timers (2 inputs); (3) GPIO, i.e. an external signal (1 input). The 16 input signals are digital signals and the CTU must be able to detect a rising and/or a falling edge for each of them.

The CTU comprises the:

- Input signals interface
- User interface (configuration registers and so on)
- ADC interface
- Timers interface

The block diagram of the CTU is shown in *Figure 97*. The CTU consists of two subunits:

- Trigger generator subunit
- Scheduler subunit

The trigger generator subunit handles incoming signals, selecting for each signal, the active edges to generate the Master Reload signal, and generates up to eight trigger events (signals). The scheduler subunit generates the trigger event output according to the occurred trigger event (signal).

14.4 Functional description

14.4.1 Interaction with other peripherals

Figure 98 shows how the CTU interacts with the following peripherals:

- ADC
- DSPI
- eTimer
- FlexPWM
- FlexRay

The CTU ETMR0_IN, also known as ETIMER0_IN, is an input for the CTU sub module called TGS (Trigger Generator Subunit). Additional information on how to use these inputs is in sections *Section 14.4.4, TGS in triggered mode* and *Section 14.4.5, TGS in sequential mode*. The FlexPWM signals are described in *Section 26.2.5, EXT_FORCE — External Output Force Signal*.

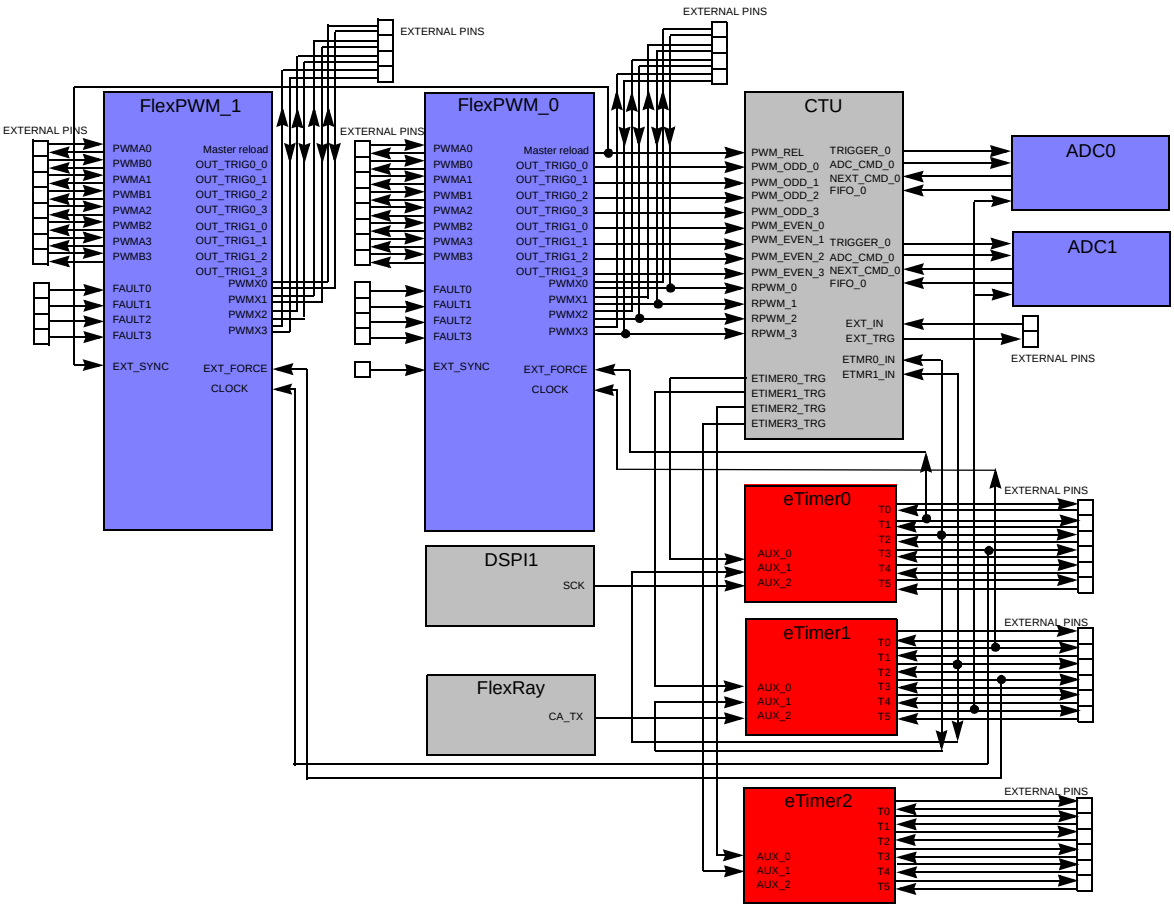


Figure 98. CTU interaction with other peripherals

14.4.2 Trigger events features

The TGS has the capability to generate up to eight trigger events. Each trigger event has the following characteristics:

- The generation of the trigger event is sequential in time.
- The Triggers List uses eight 16-bit double-buffered registers.
- On each Master Reload Signal (MRS), the new Triggers List is loaded.
- The Triggers List is reloaded on a MRS occurrence, only if the reload enable bit is set.
- The minimum time between two CTU triggers is the ADC conversion time plus two CTU clock cycles. If an ADC conversion is not required, then the minimum time between two triggers is a single CTU clock cycle.

14.4.3 Trigger generator subunit (TGS)

The trigger generator subunit has the following two modes:

- **Triggered mode**: each event source for the incoming signals can generate up to eight trigger event outputs. For the ADC, a commands list is entered by the CPU, and each event source can generate up to eight commands or streams of commands.
- **Sequential mode**: each event source for the incoming signals can generate one trigger event output, the next event source generates the next trigger event output, and so on in a predefined sequence. For the ADC, a commands list is entered by the CPU and the sequence of the selected incoming trigger events generate commands or stream of commands.

The TGS Mode is selected using the TGS_M bit in the TGS Control Register.

14.4.4 TGS in triggered mode

The structure of the TGS in Triggered Mode is shown in *Figure 99*.

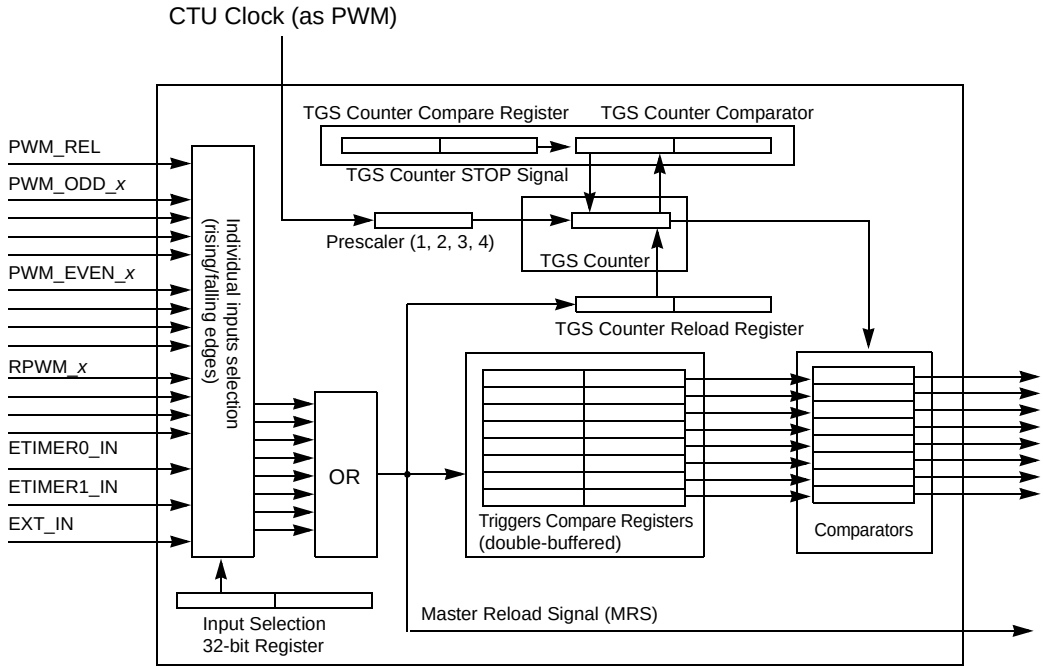


Figure 99. TGS in triggered mode

The TGS has 16 input signals, each of which is selected from the input selection register (TGSISR), selecting the states inactive, rising, falling or both. Depending on the selection, up to 32 input events can be enabled. These signals are OR-ed in order to generate the MRS. The MRS, at the beginning of the control cycle "N" (defined by the MRS occurrence), is used to pre-load the TGS counter register, using the pre-load value written into the double-buffered register (TGSCRR), during the control cycle "N-1", and to reload all the double-buffered registers (Trigger Compare registers, TGSCR, TGSCRR itself etc).

The triggers list registers, consist of 8 compare registers. Each triggers list register is associated with a comparator. On reload (MRS occurrence), the comparators are disabled: 1 TGS clock cycle is necessary to enable them and to start the counting. The MRS is output together with individual trigger signals. The MRS can be performed by hardware or by software. The MRS_SG bit in the CTU control register, if set to 1, generates equivalent

software MRS (i.e. resets/reloads TGS Counter and reloads all double-buffered registers). This bit is cleared by each hardware or software MRS occurrence.

The TGS counter compare register and the TGS counter comparator, are used to stop the TGS counter when it reaches the value stored in the TGS counter compare register, before an MRS occurs.

The prescaler for TGS and SU can be 1,2,3,4 (PRES bits in the TGS Control Register).

An example timing for the TGS in Triggered Mode is shown in *Figure 100*. The red arrows indicate the MRS occurrences, while the black arrows indicate the trigger event occurrences, with the relevant delay in respect to the last MRS occurrence.

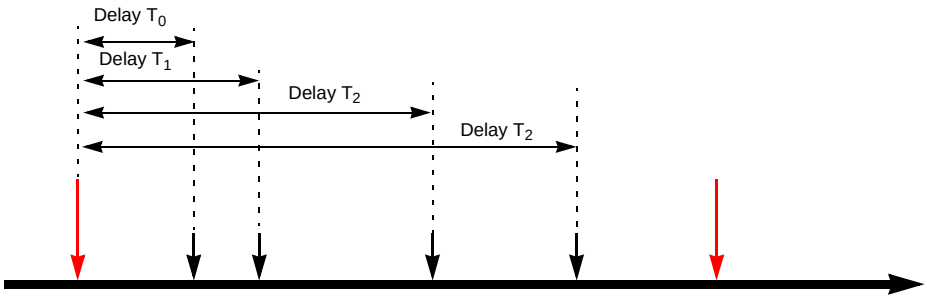


Figure 100. Example timing for TGS in triggered mode

14.4.5 TGS in sequential mode

The structure of the TGS in sequential mode is shown in *Figure 101*.

The 32 input events (16 signals with 2 edges for signal), which can be individually enabled, are OR-ed in order to generate the event signal (ES). The ES is used to enable the reload of the TGS counter register and to pilot the 3-bits counter, in order to select the next active trigger. One of the 32 input events can be selected, through the MRS_SM (master reload selection sequential mode) 5-bits in the TGS control register, to be the MRS, that enables the reload of the triggers list and resets the 3-bits counter (incoming events counter), i.e. the MRS is the signal linked with the control cycle defined as the time window between two consecutive MRSs. In this mode, each incoming event sequentially enables only one trigger event through the 3-bits counter and the MUX. The MUX is a selection switch which enables, according to the number of event signals occurred, only one of the eight trigger signals to the scheduler subunit. Sequences of up to eight trigger events can be supported within this control cycle.

For the other features see the previous paragraphs.

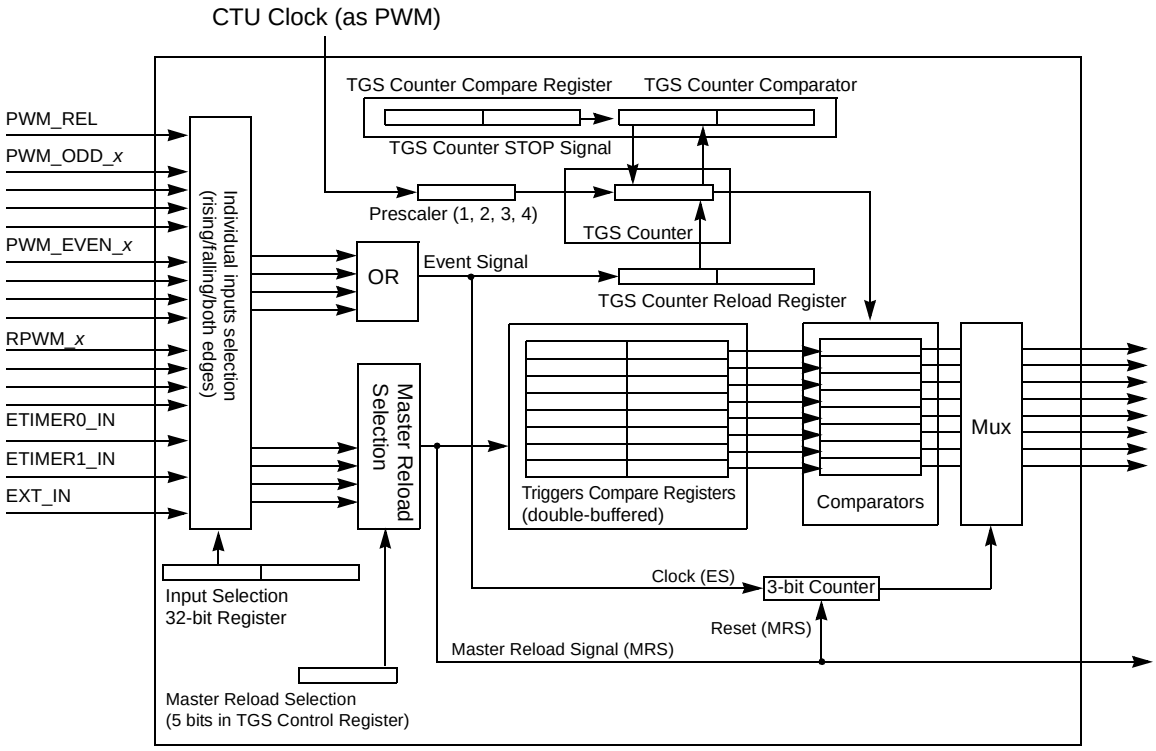


Figure 101. TGS in sequential mode

An example timing diagram for TGS in sequential mode is shown in *Figure 102*. The red arrows indicate the MRS occurrences and ES occurrences, while the black arrows indicate the trigger event occurrences with the relevant delay in respect to the ES occurrence. The first red arrow indicates the first ES occurrence which is also the MRS.

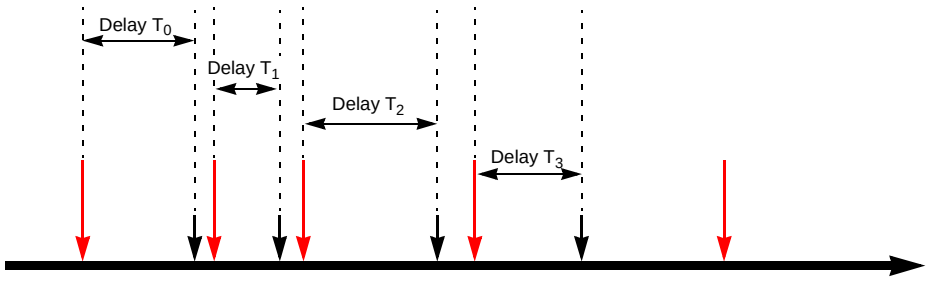


Figure 102. Example timing for TGS in sequential mode

14.4.6 TGS counter

The TGS counter is able to count from negative to positive, i.e. from 0x8000 to 0x7FFF. *Figure 103* shows examples in order to explain the TGS counter counts. The compare operation to stop the TGS counter is not enabled during the first counting cycle, in order to allow the counting, if the value of the TGSCRR is the same as the value of the TGSCCR.

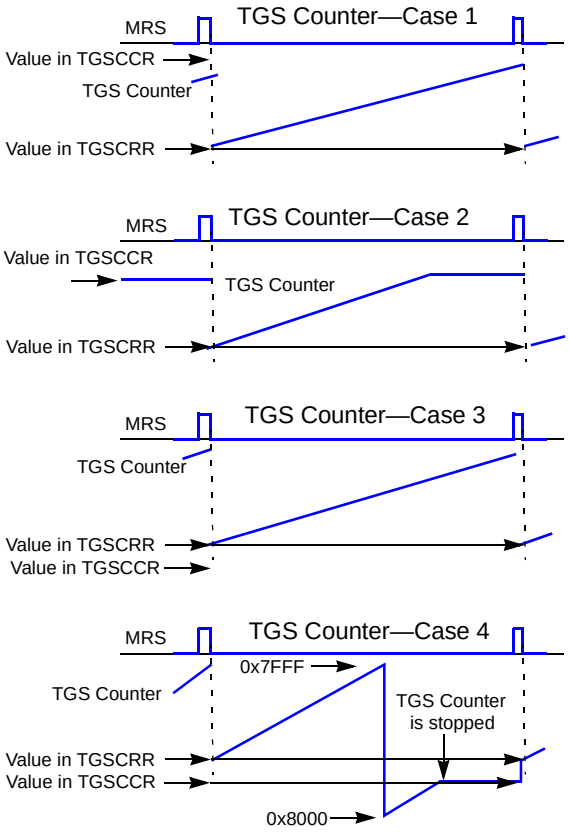


Figure 103. TGS counter cases

14.5 Scheduler subunit (SU)

The structure of the SU is shown in *Figure 104*.

The **SU** generates the trigger event output according to the occurred trigger event, and it has the same functionality in both TGS modes (triggered mode and sequential mode). Each of the 4 SU outputs:

1. ADC command or ADC stream of commands,
2. eTimer1 pulse,
3. eTimer2 pulse and
4. External trigger pulse)

can be linked to any of 8 trigger events by the Trigger Handler block. Each trigger event can be linked to one or more SU outputs.

If two events at the same time are linked to the same output only one output is generated and an error is provided. The output is generated using the trigger with the lowest index. For example if trigger 0 and trigger 1 are linked to the ADC output and they occur together, an error is generated and the output linked with the trigger 0 is generated.

When a trigger is linked to the ADC, an associated ADC command (or stream of commands) is generated. The ADC Commands List Control Register CLCRx sets the assignment to an ADC command or to a stream of commands. When a trigger is linked to a

timer or to the external trigger, a pulse with an active rising edge is generated. Additional features for the external triggers are available:

The external trigger output has:

- Pulse mode
- Toggle mode.

In Toggle Mode, each trigger event is linked to the external trigger, the external trigger pin toggles. The ON-Time for both modes (Pulse Mode and Toggle Mode) of the triggers is defined from a COTR register (Control On Time Register). A guard time is also defined from the same register at the same value of the ON-Time. A new trigger will be generated only if the ON time + Guard Time has past. The ON-Time and the Guard Time are only used for external Triggers.

External signals can be asynchronous with motor control clock. For this reason a programmable digital filter is available. The external signal is considered at 1 if it is latched N time at 1, and is considered at 0 if it is latched N time at 0, where N is a value in the digital filter control register.

Trigger events in the SU can be initiated by hardware or by software, and an additional software control is possible for each trigger event (as for the MRS), so 1 bit for each trigger event in the CTU Control Register is used to generate an equivalent software trigger event. Each of these bits will be cleared by a respective hardware or software trigger event.

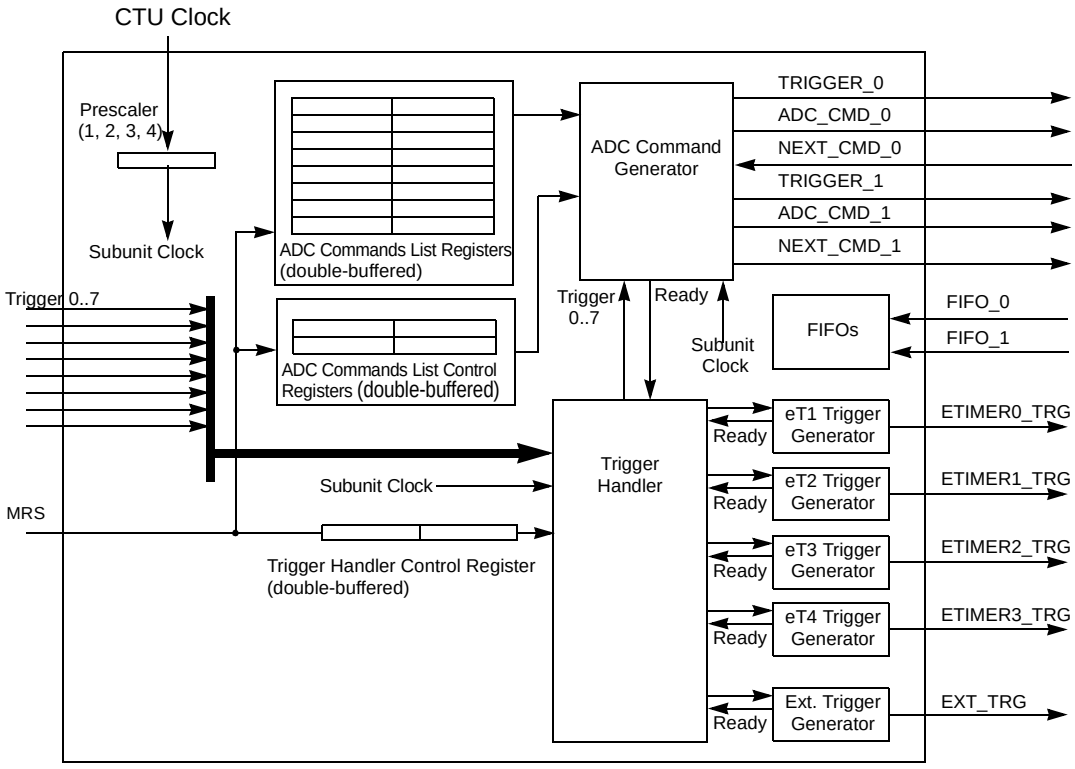


Figure 104. Scheduler subunit

14.5.1 ADC commands list

The ADC can be controlled by the CPU (CPU Control Mode) and by the CTU (CTU Control Mode). The CTU can control the ADC from sending an ADC command only when the ADC is in CTU control mode. During the CTU control mode, the CPU is able to write to the ADC registers but it can not start a new conversion. A control bit is allowed to select from the classic interface of the CTU control mode. Once selected, no change is possible unless a reset occurs.

The SU uses a Commands List in order to select the command to send to the ADC when a trigger event occurs. The commands list can hold 24 16-bits commands (see *Section 14.5.2, ADC commands list format*) and it is double-buffered, i.e. the commands list can be updated at any time between two consecutive MRS, but the changes become workable only after the next MRS occurs, and a correct reload is performed. In order to manage the commands list, 5 bits are available in the CLCRx (ADC Commands List Control Register x), for the position of the first command in the list of commands for each trigger event. The number of commands piloted by the same trigger event is defined directly in the commands list. For each command there is a bit which defines whether it is the first command of a commands list, or not.

14.5.2 ADC commands list format

The two ADCs support the Single Conversion Mode (1 bit in the ADC command format allows selection of the conversion mode), and the Dual Conversion Mode (the sampling phases and the conversion phases, are performed at the same time, the storage of the results are performed in series). The result of each conversion, in both modes, can be stored in one of the 4 available FIFOs. In dual conversion mode, both ADCs must store the result of their conversion in the same FIFO. If the access to the FIFO is in the same clock cycle, the ADC unit 0 has the priority, otherwise the first ADC which ends its conversion, will write as first in the FIFO. The CTU FIFO order is non-deterministic. Please note that CTU FIFO order in dual mode may be in reverse when a dual conversion is followed by single ADC0 with ADC clock divider is enabled. 4 analog channels are shared across the 2 ADCs and the total number of channels is 28 (12 + 12 + 4 shared channels), i.e. 16 channels for each ADC (12 + 4 shared channels). The dual conversion mode on the same physical channel is not allowed, but the dual conversion mode on the same channel number is allowed. According to this, if, in dual conversion mode, the channel number is the same for both the ADCs and the selected channel is one of the shared channels, the CTU will detect an invalid command. In dual conversion mode 4 bits for each ADC are used to select the channel number and the conversion mode selection bit is used to select the dual conversion mode. If the single conversion mode is selected, 5 bits of the 8 bits reserved to select the channels in dual conversion mode are re-used to select the channel (4 bits) and the ADC unit (1 bit). See *Section 14.10.9, Commands list register x (x = 1,...,24) (CLRx)*.

The interrupt request bit is used as an interrupt request to the CPU when ADC will complete the command with this bit set and it is only for CTU internal use. Before the next command to the CTU controls is sent, the value of the first command bit, is checked to see if it is the current command is the first command of a new stream of consecutive commands or not. If

not, the CTU sends the command. According to the previous considerations, the commands in the list will allow to have control on:

- Channel A: number of ADC channel to sample from ADC unit 0 (4 bits);
- Channel B: number of ADC channel to sample from ADC unit 1 (4 bits);
- FIFO selection bits for the ADC unit 0/1 (2 bits);
- Conversion Mode selection bit: 0 Single Conversion Mode - 1 Dual Conversion Mode;
- First command bit (only for CTU internal use);
- Interrupt request bit (only for CTU internal use).

14.5.3 ADC results

ADC results can be stored in the channel relevant standard result register and/or in one of the 4 FIFOs: the different FIFOs allow to dispatch ADC results according to the type of acquisition (ex: phase currents, rotor position, ground-noise, other). Each FIFO has its own interrupt line and DMA request signal (plus an individual overflow error bit in the FIFO status register). The store location is specified in the ADC command, i.e. the FIFOs are available only in CTU Control Mode. Each entry of a FIFO is 32-bits. The size of the FIFOs are the following: FIFO0 & FIFO1: 16 entries (sized to avoid overflow during a full PWM period for current acquisitions); FIFO2 & FIFO3: 4 entries (low acquisition rate FIFOs). Results in each FIFO can be read by 16 bits read transaction (only the result is read in order to minimize the CPU load before computing on results) or by 32 bits read transaction (both the result and the channel number are read in order to avoid blind acquisitions), 5 bits in the upper 16 bits indicate the ADC unit (1 bit) and the channel number (4 bits). The result registers (only for the FIFOs) can be read from 2 different addresses in the ADC memory map. The format of the result depends on the address from which it is read. The available formats are:

- Unsigned right-justified
  (conversion result is unsigned right-justified data, i.e. bits [11:0] are used for 12-bit resolution and bits [15:12] always return zero when read).
- Signed left-justified
  (conversion result is signed left-justified data, i.e. bit [15] is reserved for sign and is always read as zero for this ADC, bits [14:3] are used for 12-bit resolution and bits [2:0] always return zero when read).

14.6 Reload mechanism

Some CTU registers are double-buffered, and the reload is controlled by a reload enable bit, as the TGSISR_RE bit or the DFE bit, but for the most of the double-buffered registers, the reload is controlled by the MRS occurrence, and it is synchronized with the beginning of the CTU control period.

If the MRS occurs while the user is updating some double-buffered registers, eg. some registers of the Triggers List, the new Triggers List will be a mix of the old Triggers List and the new Triggers List, because the user has not ended the update of the Triggers List before the MRS occurrence.

In order to avoid this case, one bit is used to enable the reload operation, i.e. to inform the CTU that the user has ended updates to the double-buffered registers, and the reload can be performed without problems of mixed scenarios. In order to guarantee the coherency, the reload of all double-buffered registers is enabled by setting GRE (General Reload Enable) bit in the CTU Control Register. The user must ensure that all intended double-buffered

registers are updated before a new MRS occurrence. If an MRS occurs before a GRE bit is set (e.g. wrong application timing), the update is not performed, the previous values of all double-buffered registers remain active, the error flag is set (the MRS_RE bit in the CTU Error Flag Register) and, if enabled, CTU performs an interrupt request.

All the double-buffered registers use the same bit (General Reload Enable - GRE) to enable the reload when the MRS occurs. GRE bit is R/S (Read/Set) and if this bit is 1, the reload can be performed, while if this bit is 0, the reload is not performed. A correct reload reset GRE bit. All double-buffered registers cannot be writtento while the GRE bit remains set to 1. The GRE bit can be reset by the occurrence of the next MRS (i.e. a correct reload) or by software setting CGRE bit.

The CGRE is reset by hardware after that GRE bit is reset. If the user sets the CGRE bit and at the same time a MRS occurs, CGRE has the priority so GRE is reset and the reload is not performed. In the same way, the GRE has the priority when compared with the MRS occurrence, and the CGRE has the priority compared with the GRE (the two bits are in the same register so they can be set in the same time). MRS has the priority compared with the re-synchronization bit of the TGSISR.

In order to verify if a reload error occurs, FGRE (Flag GRE bit in the CTU Control Register) bit is used. When one of the double-buffered registers is written, this flag is set to 1 and it is reset by a correct reload. When the MRS occurs while FGRE is 1 and GRE is 1, a correct reload is performed (because all intended registers have been updated before the MRS occurs). If FGRE is 1 and GRE is 0, a reload is not performed, the error flag (MRS_RE) is set and (if enabled) an interrupt for an error is performed (in this case at least one register was written but the update has not ended before the MRS occurrence). If FGRE is 0 it is not necessary to perform a reload because all the double-buffered registers are unchanged (see *Figure 105*).

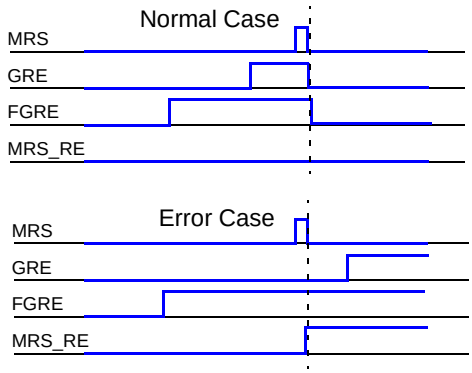


Figure 105. Reload error scenario

14.7 Power safety mode

STOP mode is a mechanism to reduce power consumption.

14.7.1 STOP mode

To reduce power consumption, it is also possible to enable a stop request from the MC_ME. The FIFOs are considered a lot like memory mapped registers. When the clock is started after a stop signal, some mistakes could occur. For example, a wrong trigger could be

provided because it was programmed before the stop signal was performed, and some incorrect write operations into the FIFOs could happen. For this reason after a stop signal the FIFO must be empty. In order to avoid the problems linked to a wrong trigger, the CTU output can be disabled by the CTU_ODIS bit and the ADC interface state machine can be reset by the CRU_ADC_R (see *Section 14.10.14, Cross triggering unit control register (CTUCR)*.

14.8 Interrupts and DMA requests

14.8.1 DMA support

The DMA can be used to configure the CTU registers. One DMA channel is reserved for performing a block transfer, and the MRS can be used as an optional DMA request signal (MRS_DMAE bit in the CTU Interrupt/DMA Register).

Note: If enabled, the DMA request on the MRS occurrence will be performed only if a reload is performed, i.e. only if GRE bit is set.

Moreover, this CTU implementation requires DMA support for reading the data from the FIFOs. One DMA channel is available for each FIFO. Each FIFO can perform a DMA request when the number of words stored in the FIFO exceeds the threshold value. The threshold value must be less than the number of FIFO entries.

14.8.2 CTU faults and errors

Faults and errors which could occur during the programming:

- a MRS occurs while user is updating the double-buffered registers and the MRS_RE bit is set.
- Receiving more than 8 EVs before that the next MRS occurs in TGS sequential mode and the SM_TO bit is set.
- A trigger event occurs during the time when the actions of the previous trigger event are not completed (user ensures no trigger event occurs during another one is processed, but if user makes a mistake and a trigger event occurs when another one is processed, the incoming trigger event will be lost and an error occurs).
  There are 4 overrun flags (one for each type of output). The general mechanism shall be as in *Figure 104*.
  The Trigger Handler, when a trigger event occurs, and the corresponding Ready signal is high, presents the respective trigger signal (one cycle high time + one cycle low time) to the respective generator sub-block (ADC Command Generator, eT0 Trigger Generator, eT1 Trigger Generator or Ext. Trigger Generator). This generator sub-block then generates the requested signal. Until this real signal is generated (including guard time) the Ready signal is kept low.
  In the case of ADC command generator, the Ready signal shall be kept low until the last conversion in the batch is finished. The respective overrun flag is set at the following conditions:
  - Ready signal is low.
  - The rising edge of the respective trigger signal (from Trigger Handler to generator sub-block) occurs.

  This architecture allows user to pre-set, for example, a trigger to the eTimer1 in the middle of an ADC conversion, i.e. the SU will be considered busy only if a request to

perform the same action that the SU is already performing occurs. One of the ADC_OE, T0_OE, T1_OE or ET_OE bit is set.

- Invalid (unrecognized) ADC command and the ICE bit is set.
- The MRS occurs before the enabled trigger events occur and the MRS_O bit is set.
- TGS overrun in sequential mode: a new incoming EV occurs before than the trigger event selected by the previous EV occurs. The incoming EV sets an internal busy flag. The outgoing trigger event (all line are OR-ed) resets this flag to zero. TGS Overrun in the sequential mode shall be generated under the following conditions:
  - TGS is in sequential mode
  - there is an incoming EV while the busy flag is high. the TGS_OSM bit is set.
- The time between a start of conversion (ADCTRIG) and the end of conversion is not in the range specified by the registers CTU_EXP_A, CTU_EXP_B and CTU_CNTRNG.

The faults/errors flags in the CTU error flag register and in the CTU interrupt flag register can be cleared by writing a 1 while writing a 0 has no effect. The CTU does not support a write-protection mechanism.

14.8.3 CTU interrupt/DMA requests

The CTU can perform the following interrupt/DMA requests (15 interrupt lines):

- Error interrupt request (see *Section 14.8.2, CTU faults and errors*) (1 interrupt line);
- ADC command interrupt request (1 interrupt line);
- Interrupt request on MRS occurrence (1 interrupt line);
- Interrupt request on each trigger event occurrence (1 interrupt line for each trigger event).
- FIFOs interrupt requests and/or DMA transfer request (1 interrupt line for each FIFO).
- DMA transfer request on the MRS occurrence if GRE bit is set;

The interrupt flags are shown in *Table 94*.

Table 94. CTU interrupts

Interrupt	Interrupt function
MRS_RE	Master Reload Signal Reload Error
SM_TO	Trigger Overrun (more than 8 EV) in TGS Sequential Mode
ICE	Invalid Command Error
MRS_O	Master Reload Signal Overrun
TGS_OSM	TGS Overrun in Sequential Mode
ADC_OE	ADC command generation Overrun Error
T0_OE	Timer 0 trigger generation Overrun Error
T1_OE	Timer 1 trigger generation Overrun Error
ET_OE	External Trigger generation Overrun Error
Safety_error_B	Conversion time is in the expected range B
Safety_error_A	Conversion time is in the expected range A
ADC_I	ADC command interrupt flag

Table 94. CTU interrupts (continued)

Interrupt	Interrupt function
MRS_I	MRS Interrupt flag
T0_I	Trigger 0 interrupt flag
T1_I	Trigger 1 interrupt flag
T2_I	Trigger 2 interrupt flag
T3_I	Trigger 3 interrupt flag
T4_I	Trigger 4 interrupt flag
T5_I	Trigger 5 interrupt flag
T6_I	Trigger 6 interrupt flag
T7_I	Trigger 7 interrupt flag
FIFO_FULL0	This bit is set to 1 if the FIFO 0 is full.
FIFO_EMPTY0	This bit is set to 1 if the FIFO 0 is empty.
FIFO_OVERFLOW0	This bit is set to 1 if the number of words became higher of the value set in the threshold 0
FIFO_OVERRUN0	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL0 flag is set
FIFO_FULL1	This bit is set to 1 if the FIFO 1 is full.
FIFO_EMPTY1	This bit is set to 1 if the FIFO 1 is empty.
FIFO_OVERFLOW1	This bit is set to 1 if the number of words became higher of the value set in the threshold 1
FIFO_OVERRUN1	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL1 flag is set
FIFO_FULL2	This bit is set to 1 if the FIFO 2 is full.
FIFO_EMPTY2	This bit is set to 1 if the FIFO 2 is empty.
FIFO_OVERFLOW2	This bit is set to 1 if the number of words became higher of the value set in the threshold 2
FIFO_OVERRUN2	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL2 flag is set
FIFO_FULL3	This bit is set to 1 if the FIFO 3 is full.
FIFO_EMPTY3	This bit is set to 1 if the FIFO 3 is empty.
FIFO_OVERFLOW3	This bit is set to 1 if the number of words became higher of the value set in the threshold 3
FIFO_OVERRUN3	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL3 flag is set

Note: In order to reduce the number of interrupt lines, the interupts are combined (OR-ed) on the same line, as follows:

MRS_RE, SM_TO, ICE, MRS_O, TGS_OSM, ADC_OE, T0_OE, T1_OE & ET_OE - ERR_I, FIFO_FULL0, FIFO_EMPTY0, FIFO_OVERFLOW0 and FIFO_OVERRUN0 - FIFO0_I, FIFO_FULL1, FIFO_EMPTY1, FIFO_OVERFLOW1 and FIFO_OVERRUN1 - FIFO1_I,

FIFO_FULL2, FIFO_EMPTY2, FIFO_OVERFLOW2 and FIFO_OVERRUN2 - FIFO2_I, FIFO_FULL3, FIFO_EMPTY3, FIFO_OVERFLOW3 and FIFO_OVERRUN3 - FIFO3_I. According to this, the total number of interrupt lines is 15.

14.9 Conversion time evaluate

Using the registers CTU_EXP_A and CTU_EXP_B, it is possible to check if the time between a start of conversion (ADCTRIG) and the end of conversion is in a specified range

The range is obtained using the expected register and the CTU_CNTRNG.

The ctu range register is used to mask the least significant bit of the CTU_EXP register. If a bit of the CTU_CNTRNG is at one , the correspondent bit of the "expected register" becomes don't care.

If the CTU_RANGE is 00001111 and the CTU_EXP is 10100111. The expected value became 1010---- and the range is from 10100000 - 10101111.

14.10 Memory map

Table 95. TGS registers

Address offset	Register	Double-buffered	Synchronization	Reset value
0x0000	TGSISR — Trigger Generator subunit Input Selection Register	Yes	TGSISR_RE	0x0000 0000
0x0004	TGSCR — Trigger Generator subunit Control Register	Yes	MRS	0x0000
0x0006	T0CR — Trigger 0 Compare Register	Yes	MRS	0x0000
0x0008	T1CR — Trigger 1 Compare Register	Yes	MRS	0x0000
0x000A	T2CR — Trigger 2 Compare Register	Yes	MRS	0x0000
0x000C	T3CR — Trigger 3 Compare Register	Yes	MRS	0x0000
0x000E	T4CR — Trigger 4 Compare Register	Yes	MRS	0x0000
0x0010	T5CR — Trigger 5 Compare Register	Yes	MRS	0x0000
0x0012	T6CR — Trigger 6 Compare Register	Yes	MRS	0x0000
0x0014	T7CR — Trigger 7 Compare Register	Yes	MRS	0x0000
0x0016	TGSCCR — TGS Counter Compare Register	Yes	MRS	0x0000
0x0018	TGSCRR — TGS Counter Reload Register	Yes	MRS	0x0000

Table 96. SU registers

Address offset	Register	Double-buffered	Synchronization	Reset value
0x001C	CLCR1 — Commands List Control Register 1	Yes	MRS	0x0000 0000
0x0020	CLCR2 — Commands List Control Register 2	Yes	MRS	0x0000 0000
0x0024	THCR1 — Trigger Handler Control Register 1	Yes	MRS	0x0000 0000
0x0028	THCR2 — Trigger Handler Control Register 2	Yes	MRS	0x0000 0000
0x002C ... 0x005A	CLRx — Commands List Register x (x = 1,...,24)	Yes	MRS	0x0000

Table 97. FIFO registers

Address offset	Register	Double-buffered	Synchronization	Reset value
0x006C	FDCR — FIFO DMA Control Register	NO	---	0x0F00
0x0070	FCR — FIFO Control Register	NO	---	0x0000 0000
0x0074	FTH — FIFO Threshold	NO	---	0x0000 0000
0x007C	FST — FIFO Status Register	NO	---	0x0000 0000
0x0080	FR0 — FIFO Right aligned data 0	NO	---	0x0000 0000
0x0084	FR1 — FIFO Right aligned data 1	NO	---	0x0000 0000
0x0088	FR2 — FIFO Right aligned data 2	NO	---	0x0000 0000
0x008C	FR3 — FIFO Right aligned data 3	NO	---	0x0000 0000
0x00A0	FL0 — FIFO Left aligned data 0	NO	---	0x0000 0000
0x00A4	FL1 — FIFO Left aligned data 1	NO	---	0x0000 0000
0x00A8	FL2 — FIFO Left aligned data 2	NO	---	0x0000 0000
0x00AC	FL3 — FIFO Left aligned data 3	NO	---	0x0000 0000

Table 98. Other CTU registers

Address offset	Register	Double-buffered	Synchronization	Reset value
0x00C0	CTUEFR — Cross Triggering Unit Error Flag Register	NO	---	0x0000
0x00C2	CTUIFR — Cross Triggering Unit Interrupt Flag Register	NO	---	0x0000
0x00C4	CTUIR — Cross Triggering Unit Interrupt Register	NO	---	0x0000
0x00C6	COTR — Control ON-Time Register	Yes	MRS	0x0000
0x00C8	CTUCR — cross triggering unit control register	NO	----	0x0000
0x00CA	CTUDF — Cross Triggering Unit Digital Filter	Yes	DFE	0x0000
0x00CC	CTU_EXP_A — Cross Triggering Unit Expected Value A	NO	----	0xFFFF

Table 98. Other CTU registers (continued)

Address offset	Register	Double-buffered	Synchronization	Reset value
0x00CE	CTU_EXP_B — Cross Triggering Unit Expected Value A	NO	----	0xFFFF
0x00D2	CTU_CNTRNG — Cross Triggering Unit Counter Range	NO	----	0x0000

14.10.1 Trigger generator subunit input selection register (TGSISR)

Figure 106. Trigger generator subunit input sel. register(TGSISR)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	I15_FE	I15_RE	I14_FE	I14_RE	I13_FE	I13_RE	I12_FE	I12_RE	I11_FE	I11_RE	I10_FE	I10_RE	I9_FE	I9_RE	I8_FE	I8_RE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	I7_FE	I7_RE	I6_FE	I6_RE	I5_FE	I5_RE	I4_FE	I4_RE	I3_FE	I3_RE	I2_FE	I2_RE	I1_FE	I1_RE	I0_FE	I0_RE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 99. TGSISR Field descriptions

Filed	Description
0 I15_FE	Input 15 ext_signals Falling edge Enable (0: disabled - 1: enabled)
1 I15_RE	Input 15 ext_signals Rising edge Enable (0: disabled - 1: enabled)
2 I14_FE	Input 14 eTimers1 ETIMER1_IN Falling edge Enable (0: disabled - 1: enabled)
3 I14_RE	Input 14 eTimers1 ETIMER1_IN Rising edge Enable (0: disabled - 1: enabled)
4 I13_FE	Input 13 eTimers0 ETIMER0_IN Falling edge Enable (0: disabled - 1: enabled)
5 I13_RE	Input 13 eTimers0 ETIMER0_IN Rising edge Enable (0: disabled - 1: enabled)
6 I12_FE	Input 12 Real PWM ch.3 Falling edge Enable (0: disabled - 1: enabled)
7 I12_RE	Input 12 Real PWM ch.3 Rising edge Enable (0: disabled - 1: enabled)
8 I11_FE	Input 11 Real PWM ch.2 Falling edge Enable (0: disabled - 1: enabled)

Table 99. TGSISR Field descriptions (continued)

Filed	Description
9 I11_RE	Input 11 Real PWM ch.2 Rising edge Enable (0: disabled - 1: enabled)
10 I10_FE	Input 10 Real PWM ch.1 Falling edge Enable (0: disabled - 1: enabled)
11 I10_RE	Input 10 Real PWM ch.1 Rising edge Enable (0: disabled - 1: enabled)
12 I9_FE	Input 9 Real PWM ch.0 Falling edge Enable (0: disabled - 1: enabled)
13 I9_RE	Input 9 Real PWM ch.0 Rising edge Enable (0: disabled - 1: enabled)
14 I8_FE	Input 8 PWM ch.3 even Falling edge Enable (0: disabled - 1: enabled)
15 I8_RE	Input 8 PWM ch.3 even Rising edge Enable (0: disabled - 1: enabled)
16 I7_FE	Input 7 PWM ch.2 even Falling edge Enable (0: disabled - 1: enabled)
17 I7_RE	Input 7 PWM ch.2 even Rising edge Enable (0: disabled - 1: enabled)
18 I6_FE	Input 6 PWM ch.1 even Falling edge Enable (0: disabled - 1: enabled)
19 I6_RE	Input 6 PWM ch.1 even Rising edge Enable (0: disabled - 1: enabled)
20 I5_FE	Input 5 PWM ch.0 even Falling edge Enable (0: disabled - 1: enabled)
21 I5_RE	Input 5 PWM ch.0 even Rising edge Enable (0: disabled - 1: enabled)
22 I4_FE	Input 4 PWM ch.3 odd Falling edge Enable (0: disabled - 1: enabled)
23 I4_RE	Input 4 PWM ch.3 odd Rising edge Enable (0: disabled - 1: enabled)
24 I3_FE	Input 3 PWM ch.2 odd Falling edge Enable (0: disabled - 1: enabled)
25 I3_RE	Input 3 PWM ch.2 odd Rising edge Enable (0: disabled - 1: enabled)
26 I2_FE	Input 2 PWM ch.1 odd Falling edge Enable (0: disabled - 1: enabled)
27 I2_RE	Input 2 PWM ch.1 odd Rising edge Enable (0: disabled - 1: enabled)

Table 99. TGSISR Field descriptions (continued)

Filed	Description
28 I1_FE	Input 1 PWM ch.0 odd Falling edge Enable (0: disabled - 1: enabled)
29 I1_RE	Input 1 PWM ch.0 odd Rising edge Enable (0: disabled - 1: enabled)
30 I0_FE	Input 0 PWM Reload Falling edge Enable (0: disabled - 1: enabled)
31 I0_RE	Input 0 PWM Reload Rising edge Enable (0: disabled - 1: enabled)

14.10.2 Trigger generator subunit control register (TGSCR)

Figure 107. Trigger generator subunit control register (TGSCR)

Address: Base + 0x0004 Access: User read/write

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

Table 100. TGSCR field descriptions

Field	Description
0-6	Reserved
7 ET_TM	This bit is used for enable toggle mode for external trigger
8-9 PRES	TGS and SU prescaler selection bits (0x00: 1 - 0x01: 2 - 0x02: 3 - 0x03: 4)
10-14 MRS_SM	MRS Selection in Sequential Mode (5 bits to select one of the 32 inputs shown in *Table 99*)
15 TGS_M	Trigger Generator subunit Mode (0: Triggered Mode - 1: Sequential Mode)

14.10.3 TxCR - Trigger x compare register (x = 0,...,7)

Figure 108. TxCR - Trigger x compare register (x = 0,...,7)

Address: Base + 0x0006 (T0CR), Base + 0x0008 (T1CR), Base + 0x000A (T2CR), Base + 0x000C (T3CR), Base + 0x000E (T4CR), Base + 0x0010 (T5CR), Base + 0x0012 (T6CR), Base + 0x0014 (T7CR)

Access: User read/write

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

Table 101. TxCR field descriptions

Field	Description
0-15 TxCRV	Trigger x Compare Register Value

14.10.4 TGS counter compare register (TGSCCR)

Figure 109. TGS counter compare register (TGSCCR)

Address: Base + 0x0016

Access: User read/write

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

Table 102. TGSCCR field format

Field	Description
0-15 TGSCCV	TGS Counter Compare Value

14.10.5 TGS counter reload register (TGSCRR)

Figure 110. TGS counter reload register (TGSCRR)

Address: Base + 0x0018

Access: User read/write

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

Table 103. TGSCRR field descriptions

Field	Description
TGSCRV	TGS Counter Reload Value

14.10.6 Commands list control register 1 (CLCR1)

Figure 111. Commands list control register 1 (CLCR1)

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T3_INDEX					0	0	0	T2_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T1_INDEX					0	0	0	T0_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 104. CLCR1 field description

Field	Description
0-2	Reserved
3-7 T3_INDEX	Trigger 3 Commands List 1st command address
8-10	Reserved
11-15 T2_INDEX	Trigger 2 Commands List 1st command address
16-18	Reserved
19-23 T1_INDEX	Trigger 1 Commands List 1st command address
24-26	Reserved
27-31 T0_INDEX	Trigger 0 Commands List 1st command address

14.10.7 Commands list control register 2 (CLCR2)

Figure 112. Commands list control register 2 (CLCR2)

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

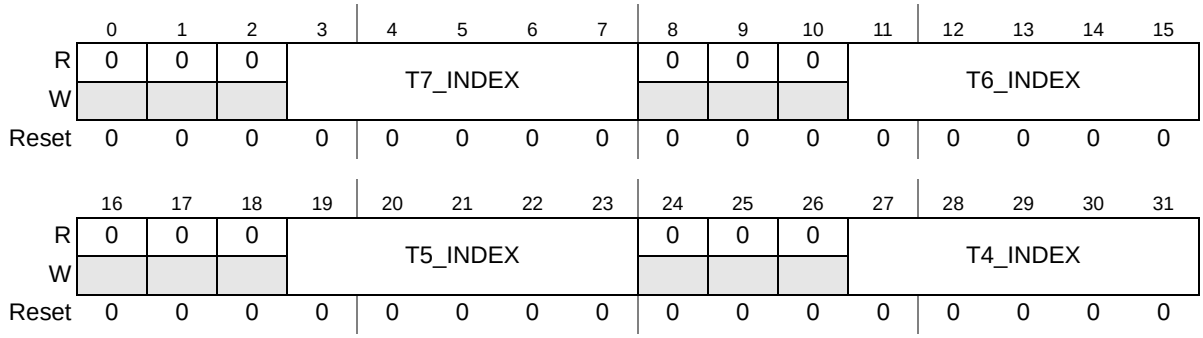


Table 105. CLCR2 field description

Field	Description
0-2	Reserved
3-7 T7_INDEX	Trigger 7 Commands List 1st command address
8-10	Reserved
11-15 T6_INDEX	Trigger 6 Commands List 1st command address
16-18	Reserved
19-23 T5_INDEX	Trigger 5 Commands List 1st command address
24-26	Reserved
27-31 T4_INDEX	Trigger 4 Commands List 1st command address

14.10.8 Trigger handler control registers (THCR1 and THCR2)

Figure 113. Trigger handler control register 1 (THCR1)

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	T3_E	T3_ETE	T3_T4E	T3_T3E	T3_T2E	T3_T1E	T3_ADCE	0	T2_E	T2_ETE	T2_T4E	T2_T3E	T2_T2E	T2_T1E	T2_ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	T1_E	T1_ETE	T1_T4E	T1_T3E	T1_T2E	T1_T1E	T1_ADCE	0	T0_E	T0_ETE	T0_T4E	T0_T3E	T0_T2E	T0_T1E	T0_ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 114. Trigger handler control register 2 (THCR2)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	T7_E	T7_ETE	T7_T4E	T7_T3E	T7_T2E	T7_T1E	T7_ADCE	0	T6_E	T6_ETE	T6_T4E	T6_T3E	T6_T2E	T6_T1E	T6_ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	T5_E	T5_ETE	T5_T4E	T5_T3E	T5_T2E	T5_T1E	T5_ADCE	0	T4_E	T4_ETE	T4_T4E	T4_T3E	T4_T2E	T4_T1E	T4_ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 106. THCR1 and THCR2 field descriptions

Field	Description
T*n*_E	Trigger *n* enable (0: disabled - 1: enabled)
T*n*_ETE	Trigger *n* External Trigger output enable (0: disabled - 1: enabled)
T*n*_T*m*E	Trigger *n* Timer *m* output enable (0: disabled - 1: enabled)
T*n*_ADCE	Trigger *n* ADC command output enable (0: disabled - 1: enabled)

14.10.9 Commands list register x (x = 1,...,24) (CLRx)

14.10.9.1 CLRx for ADC single-conversion mode commands (CMS = 0)

Figure 115. Commands list register x (x = 1,...,24) (CMS=0)

Address: See *Table 110*. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	LC(1)	CMS	FIFO			ST	0	0	0	SU	0	CH			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This bit may be called FC or LC depending on the device, but the functionality is identical.

Table 107. CLRx (CMS = 0) field descriptions

Field	Description
CIR	Command Interrupt Request bit (0: disabled - 1: enabled)
LC	Last command bit (0: not last - 1: last)
CMS	Conversion mode selection (0: single conversion mode - 1: dual conversion mode)
FIFO	FIFO for ADC unit 0/1
ST	Self-Test bit (0: normal mode - 1: self-test mode)
SU	Selection Unit bit (0: ADC unit 0 selected - 1: ADC unit 1 selected)
CH	ADC unit channel number

14.10.9.2 CLRx for ADC dual-conversion mode commands (CMS = 1)

Figure 116. Commands list register x (x = 1,...,24) (CMS=1)

Address: See *Table 110*. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	LC(1)	CMS	FIFO			CH_B					0	CH_A			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This bit may be called FC or LC depending on the device, but the functionality is identical.

Table 108. CLRx (CMS = 1) field descriptions

Field	Description
CIR	Command Interrupt Request bit (0: disabled - 1: enabled)
LC	Last command bit (0: not last - 1: last)
CMS	Conversion mode selection (0: single conversion mode - 1: dual conversion mode)
FIFO	FIFO for ADC unit 0/1
CH_B	ADC unit 1 channel number
CH_A	ADC unit 0 channel number

14.10.9.3 CLRx for self-test commands (ST1 = 0)

Figure 117. Commands list register x (x = 1,...,24) (ST1 = 0)

Address: See *Table 110*. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	LC(1)	ST1	CMS	SU	0	ST0	ALG		BSIZE						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This bit may be called FC or LC depending on the device, but the functionality is identical.

Table 109. CLRx (ST1 = 0) field descriptions

Field	Description
CIR	Command Interrupt Request bit (0: disabled - 1: enabled)
LC	Last command bit 0: not last 1: last
ST1	Self test mode. ST1 must equal 0 if you want to send a self test command.
CMS	Conversion Mode Selection 0: Single Conversion Mode 1: Dual Conversion Mode
SU	Selection unit bit, usable only if CMS is at zero. 0: ADC unit A 1: ADC unit B
ST0	Self test mode. ST0 must equal 1 if you are considering this register configuration.
ALG	Algorithm scheduled: 00: Algorithm S 01: Algorithm RC 10: Algorithm C 11: Full algorithm (S + RC + C)
BSIZE	Burst size of the algorithm iteration. 0: Single step execution N: N+1 step execution with one trigger

Table 110. CLR*x* addresses(1)

Register	Address	Register	Address	Register	Address	Register	Address
CLR1	0x002C	CLR7	0x0038	CLR13	0x0044	CLR19	0x0050
CLR2	0x002E	CLR8	0x003A	CLR14	0x0046	CLR20	0x0052
CLR3	0x0030	CLR9	0x003C	CLR15	0x0048	CLR21	0x0054
CLR4	0x0032	CLR10	0x003E	CLR16	0x004A	CLR22	0x0056
CLR5	0x0034	CLR11	0x0040	CLR17	0x004C	CLR23	0x0058
CLR6	0x0036	CLR12	0x0042	CLR18	0x004E	CLR24	0x005A

1. Offset from CTU module base.

14.10.10 Cross triggering unit error flag register (CTUEFR)

Figure 118. Cross triggering unit error flag register (CTUEFR)

Address: Base + 0x00C0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	CS	ET_OE	ERROR_compare	T4_OE	T3_OE	T2_OE	T1_OE	ADC_OE	TGS_OSM	MRS_O	ICE	SM_TO	MRS_RE
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

Table 111. CTUEFR field description

Field	Description
CS	CS Counter status. This bit resets the self-test counter. 0 All the counter are at reset value. 1One of the counter used in self-test mode is not in the reset state.
ET_OE	External trigger generation overrun error 0 Error has not occurred. 1Error has occurred.
ERR_CMP	This bit is set to one if the counter reaches the TGSCCR register value. It is cleared by writing a one to this field.
T*n*_OE	Timer *n* trigger generation Overrun Error 0 Error has not occurred. 1Error has occurred.
ADC_OE	ADC command generation Overrun Error 0 Error has not occurred. 1Error has occurred.
TGS_OSM	TGS Overrun in Sequential Mode 0 Error has not occurred. 1Error has occurred.
MRS_O	Master Reload Signal Overrun 0 Overrun has not occurred. 1Error has occurred.
ICE	Invalid Command Error 0 Error has not occurred. 1Error has occurred.

Table 111. CTUEFR field description

Field	Description
SM_TO	Trigger Overrun (more than 8 EV) in TGS Sequential Mode 0 Overrun has not occurred. 1Overrun has occurred.
MRS_RE	Master Reload Signal Reload Error 0 Error has not occurred. 1Error has occurred.

14.10.11 Cross triggering unit interrupt flag register (CTUIFR)

Figure 119. Cross triggering unit interrupt flag register (CTUIFR)

Address: Base + 0x00C2 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	Safety_error_B	Safety_error_A	ADC_I	T7_I	T6_I	T5_I	T4_I	T3_I	T2_I	T1_I	T0_I	MRS_I
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

Table 112. CTUIFR field descriptions

Field	Description
Safety_error_B	If this bit is set means that the slice time between the start of conversion and the end of conversion is in the expected range
Safety_error_A	If this bit is set means that the slice time between the start of conversion and the end of conversion is in the expected range
ADC_I	ADC command interrupt flag 0 Interrupt has not occurred. 1Interrupt has occurred.
T*n*_I	Trigger *n* interrupt flag 0 Interrupt has not occurred. 1Interrupt has occurred.
MRS_I	MRS Interrupt flag 0 Interrupt has not occurred. 1Interrupt has occurred.

14.10.12 Cross triggering unit interrupt/DMA register (CTUIR)

Figure 120. Cross triggering unit interrupt/DMA register (CTUIR)

Address: Base + 0x00C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	T7_IE	T6_IE	T5_IE	T4_IE	T3_IE	T2_IE	T1_IE	T0_IE	0	0	Saf_cnt_b_en	Saf_cnt_a_en	DMA_DE	MRS_DMAE	MRS_IE	IEE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 113. CTUIR field description

Field	Description
T*n*_IE	Trigger *n* Interrupt Enable 0 Interrupt is not enabled. 1Interrupt is enabled.
Saf_cnt_b_en	If this bit is set the counter used to check the conversion time is enabled
Saf_cnt_a_en	If this bit is set the counter used to check the conversion time is enabled
DMA_DE	If this bit is set, a dma done is like a write in the gre bit. If it is at 0 you have to write the gre bit (if something is changed in the double buffered register) otherwise you will have an error on the next master reload.
MRS_DMAE	DMA transfer Enable on MRS occurrence if GRE bit is set
MRS_IE	MRS Interrupt Enable 0 Interrupt is not enabled. 1Interrupt is enabled.
IEE	Interrupt Error Enable 0 Interrupt is not enabled. 1Interrupt is enabled.

14.10.13 Control ON time register (COTR)

Figure 121. Control On time register (COTR)

Address: Base + 0x00C6 Access: User read/write

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

Table 114. COTR field description

Field	Description
0-7	Reserved
8-15 COTR	Control ON-Time and Guard Time for external trigger. This bit field decides cycle numbers of MC clock when EXT_TRG is high.

14.10.14 Cross triggering unit control register (CTUCR)

Figure 122. Cross triggering unit control Register (CTUCR)

Address: Base + 0x00C8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	T7_SG	T6_SG	T5_SG	T4_SG	T3_SG	T2_SG	T1_SG	T0_SG	CTU_ADC_R	CTU_ODIS	DFE	CGRE	FGRE	MRS_SG	GRE	TGSISR_RE
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c					w1c		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 115. CTUCR field descriptions

Field	Description
0 T7_SG	Trigger 7 Software Generated 0Trigger has not been generated. 1Trigger has been generated.
1 T6_SG	Trigger 6 Software Generated 0Trigger has not been generated. 1Trigger has been generated.
2 T5_SG	Trigger 5 Software Generated 0Trigger has not been generated. 1Trigger has been generated.
3 T4_SG	Trigger 4 Software Generated 0Trigger has not been generated. 1Trigger has been generated.
4 T3_SG	Trigger 3 Software Generated 0Trigger has not been generated. 1Trigger has been generated.
5 T2_SG	Trigger 2 Software Generated 0Trigger has not been generated. 1Trigger has been generated.
6 T1_SG	Trigger 1 Software Generated 0Trigger has not been generated. 1Trigger has been generated.

Table 115. CTUCR field descriptions

Field	Description
7 T0_SG	Trigger 0 Software Generated 0Trigger has not been generated. 1Trigger has been generated.
8 CTU_ADC_R	CTU/ADC state machine Reset 0Reset has not been generated. 1Reset has been generated.
9 CTU_ODIS	CTU Output Disable CTU_ODISCTU output disable 0CTU output is enabled. 1CTU output is disabled.
10 DFE	Digital Filter Enable 0Digital filter is disabled. 1Digital filter is enabled.
11 CGRE	Clear GRE. Writing 1 clears GRE bit in this register. This bit will be cleard after GRE is cleared.
12 FGRE	Flag GRE 0General Reload has not occurred. 1General Reload has occurred.
13 MRS_SG	MRS Software Generated. Writing 1 generates MRS signal by software. This bit will be cleared after MRS is generated.
14 GRE	General Reload Enable 0General reload is disabled. 1General reload is enabled.
15 TGSISR_RE	TGS Input Selection Register Reload Enable 0Register reload is disabled. 1Register reload is enabled.

14.10.15 Cross triggering unit digital filter (CTUDF)

Figure 123. Cross triggering unit digital filter (CTUDF)

Address: Base + 0x00CA Access: User read/write

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

Table 116. CTUDF field descriptions

Field	Description
0-7	Reserved
8-15 FILTERVALUE	Digital Filter value (the external signal is considered at 1 if it is latched N time at 1 and is considered at 0 if it is latched N time at 0)

14.10.16 Cross triggering unit expected value A (CTU_EXP_A)

This register is used to configure the number of the expected clock cycles needed for the conversion time of ADC_0. Please refer *Section 14.9, Conversion time evaluate* for more details.

Figure 124. Cross triggering unit expected value A (CTU_EXPECTED_A)

Address: Base + 0x00CC Access: User read/write

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

Table 117. CTU_EXP_A field descriptions

Field	Description
VALUE	This value is the number of system clock cycle needed for the conversion time.(ADC 0)

14.10.17 Cross triggering unit expected value B (CTU_EXP_B)

This register is used to configure the number of the expected clock cycles needed for the conversion time of ADC_1. Please refer *Section 14.9, Conversion time evaluate* for more details.

Figure 125. Cross triggering unit expected value B (CTU_EXPECTED_B)

Address: Base + 0x00CE Access: User read/write

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

Table 118. CTU_EXP_B field descriptions

Field	Description
VALUE	This value is the number of system clock cycle needed for the conversion time.(ADC 1)

14.10.18 Cross triggering unit counter range (CTU_CNTRNG)

This register is used to mask the less significant bits in the expected clock cycles (i.e. CTU_EXP_A and CTU_EXP_B). In this way it is possible to define an acceptance range

around these values.Please refer *Section 14.9, Conversion time evaluate* for more details

Figure 126. Cross triggering unit count range (CTU_CNTRNG)

Address: Base + 0x00D2 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	VALUE							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 119. CTU_CNTRNG field descriptions

Field	Description
VALUE	Setting at one this bit you mask the same bit of the expected counter and of the internal counter. In this way you can set a range for the expected conversion time.

14.10.19 FIFO DMA control register (FDCR)

Figure 127. FIFO DMA control register (FDCR)

Address: Base + 0x006C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	DE3	DE2	DE1	DE0
W																
Reset	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

Table 120. FDCR field description

Name	Description
0-11	Reserved
12-15 DEx	This bit enables DMA for the FIFOx

14.10.20 FIFO control register (FCR)

Figure 128. FIFO control register (FCR)

Address: Base + 0x0070 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OR_EN3	OF_EN3	EMPTY_EN3	FULL_EN3	OR_EN2	OF_EN2	EMPTY_EN2	FULL_EN2	OR_EN1	OF_EN1	EMPTY_EN1	FULL_EN1	OR_EN0	OF_EN0	EMPTY_EN0	FULL_EN0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 121. FCR field descriptions

Field	Description
0-15	Reserved
16 OR_EN3	FIFO 3 Overrun interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
17 OF_EN3	FIFO 3 threshold Overflow interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
18 EMPTY_EN3	FIFO 3 Empty interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
19 FULL_EN3	FIFO 3 Full interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
20 OR_EN2	FIFO 2 Overrun interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
21 OF_EN2	FIFO 2 threshold Overflow interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
22 EMPTY_EN2	FIFO 2 Empty interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
23 FULL_EN2	FIFO 2 Full interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.

Table 121. FCR field descriptions

Field	Description
24 OR_EN1	FIFO 1 Overrun interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
25 OF_EN1	FIFO 1 threshold Overflow interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
26 EMPTY_EN1	FIFO 1 Empty interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
27 FULL_EN1	FIFO 1 Full interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
28 OR_EN0	FIFO 0 Overrun interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
29 OF_EN0	FIFO 0 threshold Overflow interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
30 EMPTY_EN0	FIFO 0 Empty interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.
31 FULL_EN0	FIFO 0 Full interrupt enable 0 Interrupt is disabled. 1Interrupt is enabled.

14.10.21 FIFO threshold (FTH)

Figure 129. FIFO threshold (FTH)

Address: Base + 0x0074 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TH3								TH2							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TH1								TH0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Description
0-7 TH3	FIFO 3 Threshold. Maximum value of 3, as the threshold value must be less than the number of FIFO 3 entries.
8-15 TH2	FIFO 2 Threshold. Maximum value of 3, as the threshold value must be less than the number of FIFO 2 entries.
16-23 TH1	FIFO 1 Threshold. Maximum value of 15, as the threshold value must be less than the number of FIFO 1 entries.
24-31 TH0	FIFO 0 Threshold. Maximum value of 15, as the threshold value must be less than the number of FIFO 0 entries.

14.10.22 FIFO status register (FST)

Figure 130. FIFO status (FST)

Address: Base + 0x007C Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OR3	OF3	EMP3	FULL3	OR2	OF2	EMP2	FULL2	OR1	OF1	EMP1	FULL1	OR0	OF0	EMP0	FULL0
W	w1c				w1c				w1c				w1c			
Reset	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0

Table 122. FST field descriptions

Field	Description
0-15	Reserved
16 OR3	FIFO 3 Overrun interrupt flag 0 Condition has not occurred. 1Condition has occurred.
17 OF3	FIFO 3 threshold Overflow interrupt flag 0 Condition has not occurred. 1Condition has occurred.
18 EMP3	FIFO 3 Empty interrupt flag 0 Condition has not occurred. 1Condition has occurred.
19 FULL3	FIFO 3 Full interrupt flag 0 Condition has not occurred. 1Condition has occurred.

Table 122. FST field descriptions (continued)

Field	Description
20 OR2	FIFO 2 Overrun interrupt flag 0 Condition has not occurred. 1Condition has occurred.
21 OF2	FIFO 2 threshold Overflow interrupt flag 0 Condition has not occurred. 1Condition has occurred.
22 EMP2	FIFO 2 Empty interrupt flag 0 Condition has not occurred. 1Condition has occurred.
23 FULL2	FIFO 2 Full interrupt flag 0 Condition has not occurred. 1Condition has occurred.
24 OR1	FIFO 1 Overrun interrupt flag 0 Condition has not occurred. 1Condition has occurred.
25 OF1	FIFO 1 threshold Overflow interrupt flag 0 Condition has not occurred. 1Condition has occurred.
26 EMP1	FIFO 1 Empty interrupt flag 0 Condition has not occurred. 1Condition has occurred.
27 FULL1	FIFO 1 Full interrupt flag 0 Condition has not occurred. 1Condition has occurred.
28 OR0	FIFO 0 Overrun interrupt flag 0 Condition has not occurred. 1Condition has occurred.
29 OF0	FIFO 0 threshold Overflow interrupt flag 0 Condition has not occurred. 1Condition has occurred.
30 EMP0	FIFO 0 Empty interrupt flag 0 Condition has not occurred. 1Condition has occurred.
31 FULL0	FIFO 0 Full interrupt flag 0 Condition has not occurred. 1Condition has occurred.

14.10.23 FIFO Right aligned data x (x = 0,...,3) (FRx)

Figure 131. FIFO Right aligned data x

Address: Base + 0x0080 (FR0), Base + 0x0084 (FR1), Base + 0x0088 (FR2), Base + 0x008C (FR3) — Access: User read-only

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

Table 123. FRx filed descriptions

Field	Description
ADC	This bit is used for indicate from which ADC the data is coming from. 1: The data is coming from ADC_0 0: The data is coming from ADC_1
N_CH	Number of stored channel
DATA	Data of stored channel

14.10.24 FIFO signed Left aligned data x (x = 0,...,3) (FLx)

Figure 132. FIFO signed left aligned data x

Address: Base + 0x00A0 (FL0), Base + 0x00A4 (FL1), Base + 0x00A8 (FL2), Base + 0x00AC (FL3) — Access: User read-only

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

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

Table 124. FLx field description

Field	Description
ADC	This bit is used for indicate from which ADC the data is coming from. 1: The data is coming from ADC_0 0: The data is coming from ADC_1
N_CH	Number of stored channel
DATA	Data of stored channel

15 Crossbar Switch (XBAR)

15.1 Information specific to this device

This section presents device-specific parameterization, customization, and feature availability information not specifically referenced in the remainder of this chapter.

15.1.1 Register availability

Not all registers listed in *Table 128* are available on this device. Specifically, this device includes only registers for:

- Slaves 0, 2, and 7
- Masters 0, 1, 2, 3, 5, and 6

15.1.2 MPR reset value

The reset value of the MPR register on this device is 0x0540_3210.

15.1.3 max_halt signal unavailable

The max_halt signal is unavailable on this device.

15.1.4 Logical master IDs

Table 125 defines the logical master ID used for the chip masters in Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM). The logical master IDs for the two cores are different in DPM so that they both can access the same XBAR.

Table 125. Logical master IDs

Master	Logical master ID	
	LSM	DPM
Core_0 Instruction port Core_0 Load/Store port	0	0
Core_1 Instruction port Core_1 Load/Store port		1
eDMA_0	2	2
eDMA_1		6(1)
Core_0 Nexus	8(2)	8(2)
Core_1 Nexus		9(2)
FlexRay	3	3

1. DMA_1 is not enabled in DPM. LMID assigned here for completeness and future enhancements.
2. The MPU cannot differentiate between core and Nexus accesses to slave ports.

Note: Logical Master IDs 4 and 5 are not used.

15.1.5 Master port allocation

Table 126 defines the XBAR master port allocation for this device.

Table 126. XBAR master port allocation

XBAR master port	LSM		DPM	
	XBAR_0 module	XBAR_1 module	XBAR_0 module	XBAR_1 module
M0	z4d_0 core complex Instruction port	z4d_1 core complex Instruction port	z4d_0 core complex Instruction port	z4d_1 core complex Instruction port
M1	z4d_0 core complex Load/Store port + Nexus port	z4d_1 core complex Load/Store port + Nexus port	z4d_0 core complex Load/Store port + Nexus port	z4d_1 core complex Load/Store port + Nexus port
M2	DMA_0	DMA_1	DMA_0	DMA_1(1)
M3	FlexRay	FlexRay	FlexRay	FlexRay
M4	—	—	—	—
M5	—	—	z4d_1 core complex Instruction port	z4d_0 core complex Instruction port
M6	—	—	z4d_1 core complex Load/Store port + Nexus port	z4d_0 core complex Load/Store port + Nexus port
M7	—	—	—	—

1. DMA_1 is not enabled in DPM. XBAR Master Port assigned here for completeness and future enhancements.

15.1.6 Slave port allocation

Table 127 defines the XBAR slave port allocation for this device.

Table 127. XBAR slave port allocation

XBAR slave port	LSM		DPM	
	XBAR_0 module	XBAR_1 module	XBAR_0 module	XBAR_1 module
S0	PFLASH_0_PORT_0	PFLASH_1_PORT_1	PFLASH_0_PORT_0 PFLASH_1_PORT_0	PFLASH_0_PORT_1 PFLASH_1_PORT_1
S1	—	—	—	—
S2	SRAMC_0	SRAMC_1	SRAMC_0	SRAMC_1
S3	—	—	—	—
S4	—	—	—	—
S5	—	—	—	—
S6	—	—	—	—
S7	PBRIDGE_0	PBRIDGE_1	PBRIDGE_0	PBRIDGE_1

15.2 Introduction

15.2.1 Overview

This section provides an overview of the generic multi-layer AHB crossbar switch (XBAR[h]). The purpose of the XBAR is to concurrently support up to eight simultaneous connections between master ports and slave ports. The XBAR supports a 32-bit address bus width and a 64-bit data bus width at all master and slave ports.

15.2.2 Features

The XBAR has the ability to gain control of all the slave ports and prevent any masters from making accesses to the slave ports. This feature is useful when the user wishes to turn off the clocks to the system and needs to ensure that no bus activity will be interrupted.

The XBAR can put each slave port into a low power park mode so that slave port will not dissipate any power transitioning address, control or data signals when not being actively accessed by a master port.

Each slave port can also support multiple master priority schemes. Each slave port has a hardware input which selects the master priority scheme so the user can dynamically change master priority levels on a slave port by slave port basis.

The XBAR will allow for concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic will select the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port will stalled until the higher priority master completes its transactions.

15.2.3 Limitations

The XBAR routes bus transactions initiated on the master ports to the appropriate slave ports. There is no provision included to route transactions initiated on the slave ports to other slave ports or to master ports. Simply put, the slave ports do not support the bus request/bus grant protocol, the XBAR assumes it is the sole master of each slave port.

Since the XBAR does not support the bus request/bus grant protocol, if multiple masters are to be connected to a single master port an external arbiter will need to be used. In the case of a single master connecting to a master port the single master’s bus grant signal must be tied off in the asserted state.

Each master and slave port is fully AHB-Lite + AMBA V6 extensions compliant. The ports are not fully AHB compliant because the XBAR does not support SPLITs or RETRYs.

15.2.4 General Operation

When a master makes an access to the XBAR the access will be immediately taken by the XBAR. If the targeted slave port of the access is available then the access will be immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR. If the targeted slave port of the access is busy or parked on a

h. An alternate abbreviation for this module is MAX.

different master port the requesting master will simply see wait states inserted (**hready** held negated) until the targeted slave port can service the master's request. The latency in servicing the request will depend on each master's priority level and the responding peripheral's access time.

Since the XBAR appears to be just another slave to the master device, the master device will have no knowledge of whether or not it actually owns the slave port it is targeting. While the master does not have control of the slave port it is targeting it will simply be wait stated.

A master will be given control of the targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has an outstanding request to one slave port that has a long response time, has a pending access to a different slave port, and a lower priority master is also making a request to the same slave port as the pending access of the higher priority master.

Once the master has control of the slave port it is targeting the master will remain in control of that slave port until it gives up the slave port by running an IDLE cycle or by leaving that slave port for its next access. The master could also lose control of the slave port if another higher priority master makes a request to the slave port; however, if the master is running a locked or fixed length burst transfer it will retain control of the slave port until that transfer is completed. Based on the AULB bit in the MGPCR (Master General Purpose Control Register) the master will either retain control of the slave port when doing undefined length incrementing burst transfers or will lose the bus to a higher priority master.

The XBAR will terminate all master IDLE transfers (as opposed to allowing the termination to come from one of the slave busses). Additionally, when no master is requesting access to a slave port the XBAR will drive IDLE transfers onto the slave bus, even though a default master may be granted access to the slave port. When the XBAR is controlling the slave bus (that is, during low power park or halt mode) the **hmaster** field will indicate 4'b0000.

When a slave bus is being IDLEd by the XBAR it can park the slave port on the master port indicated by the PARK bits in the SGPCR (Slave General Purpose Control Register). This can be done in an attempt to save the initial clock of arbitration delay that would otherwise be seen if the master had to arbitrate to gain control of the slave port. The slave port can also be put into low power park mode in attempt to save power.

15.3 XBAR registers

This section provides information on XBAR registers.

15.3.1 Register summary

There are four registers that reside in each slave port of the XBAR and one register that resides in each master port of the XBAR. These registers are IP bus compliant registers. Read and write transfers both require two IP bus clock cycles. The registers can only be read from and written to in supervisor mode. Additionally, these registers can only be read from or written to by 32-bit accesses.

The registers are fully decoded and an error response is returned if an unimplemented location is accessed within the XBAR.

The slave registers also feature a bit, which when written with a 1, will prevent the registers from being written to again. The registers will still be readable, but future write attempts will have no effect on the registers and will be terminated with an error response.

The memory map for the XBAR program-visible registers is shown in *Table 128*. *Table 129* shows the XBAR register summary.

Table 128. XBAR register configuration summary

XBAR base offset	Register	Use
0x000	MPR0	Master Priority Register for Slave port 0
0x010	SGPCR0	General Purpose Control Register for Slave port 0
0x100	MPR1	Master Priority Register for Slave port 1
0x110	SGPCR1	General Purpose Control Register for Slave port 1
0x200	MPR2	Master Priority Register for Slave port 2
0x210	SGPCR2	General Purpose Control Register for Slave port 2
0x300	MPR3	Master Priority Register for Slave port 3
0x310	SGPCR3	General Purpose Control Register for Slave port 3
0x400	MPR4	Master Priority Register for Slave port 4
0x410	SGPCR4	General Purpose Control Register for Slave port 4
0x500	MPR5	Master Priority Register for Slave port 5
0x510	SGPCR5	General Purpose Control Register for Slave port 5
0x600	MPR6	Master Priority Register for Slave port 6
0x610	SGPCR6	General Purpose Control Register for Slave port 6
0x700	MPR7	Master Priority Register for Slave port 7
0x710	SGPCR7	General Purpose Control Register for Slave port 7
0x800	MGPCR0	General Purpose Control Register for Master port 0
0x900	MGPCR1	General Purpose Control Register for Master port 1
0xA00	MGPCR2	General Purpose Control Register for Master port 2
0xB00	MGPCR3	General Purpose Control Register for Master port 3
0xC00	MGPCR4	General Purpose Control Register for Master port 4
0xD00	MGPCR5	General Purpose Control Register for Master port 5
0xE00	MGPCR6	General Purpose Control Register for Master port 6
0xF00	MGPCR7	General Purpose Control Register for Master port 7

Figure 133. Key to Register Fields

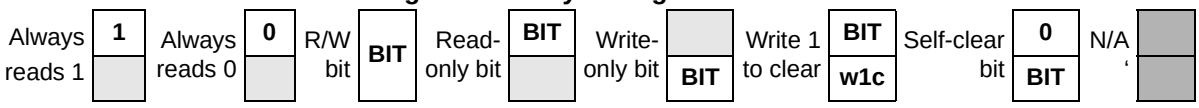


Table 129. XBAR register summary

Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MPRn ($BASE + 0x000 + n*0x100)	R	0	MSTR_7			0	MSTR_6			0	MSTR_5			0	MSTR_4		
	W																
	R	0	MSTR_3			0	MSTR_2			0	MSTR_1			0	MSTR_0		
	W																

Table 129. XBAR register summary (continued)

Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
SGPCRn ($BASE + 0x010 + n*0x100)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	RO	HLP							HPE7	HPE6	HPE5	HPE4	HPE3	HPE2	HPE1	HPE0
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W							ARB				PCTL			PARK		
MGPCRn ($BASE + 0x800 + n*0x100	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W														AULB		

Note: for n = 0 to 7

15.3.2 XBAR register descriptions

The following paragraphs provide detailed descriptions of the various XBAR registers.

Table 130 provides a key to the terms found in XBAR registers.

Table 130. Register terms

Term	Description
Gray bit	Unimplemented bit; always reads as zero;writing has no effect
Access	
S	Supervisor mode only
-	Supervisor or user mode
Type	
r	Read only; writing to this bit has no effect
w	Write only
rw	Standard read/write bit. Only software can change a bit's value (other than a hardware reset).
rwm	A read/write bit that may be modified by hardware in some fashion other than reset.
w1c	A status bit that can be read and cleared by writing a logic 1
slfclr	Self-clearing bit. Writing a 1 has some effect on module, but it always reads as a 0.
Reset	
0	Resets to a logic 0
1	Resets to a logic 1

Table 130. Register terms (continued)

Term	Description
u	Unaffected by reset
?	Reset state is unknown.

15.3.2.1 Master Priority Register

The Master Priority Register (MPR) sets the priority of each master port on a per slave port basis and resides in each slave port.

MPRn — **Master Priority Register n** — **Addr $BASE + 0x000 + n*100, Wait State: 0, Access: S**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		MSTR_7				MSTR_6				MSTR_5				MSTR_4		
TYPE:	r	rw	rw	rw	r	rw	rw	rw	r	rw	rw	rw	r	rw	rw	rw
RESET:	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
		MSTR_3				MSTR_2				MSTR_1				MSTR_0		
TYPE:	r	rw	rw	rw	r	rw	rw	rw	r	rw	rw	rw	r	rw	rw	rw
RESET:	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

Note: for n = 0 to 7

Figure 134. Master Priority Register n

Table 131. Master Priority Register descriptions

Name	Description	Settings
Bit 0	**Master Priority Register Reserved —** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_7	**Master 7 Priority —** These bits set the arbitration priority for master port 7 on the associated slave port. These bits are initialized by hardware reset. The reset value is 111	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 4	**Master Priority Register Reserved —** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA

Table 131. Master Priority Register descriptions (continued)

Name	Description	Settings
MSTR_6	**Master 6 Priority —** These bits set the arbitration priority for master port 6 on the associated slave port. These bits are initialized by hardware reset. The reset value is 110	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 8	**Master Priority Register Reserved —** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_5	**Master 5 Priority —** These bits set the arbitration priority for master port 5 on the associated slave port. These bits are initialized by hardware reset. The reset value is 101	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 12	**Master Priority Register Reserved —** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_4	**Master 4 Priority —** These bits set the arbitration priority for master port 4 on the associated slave port. These bits are initialized by hardware reset. The reset value is 100	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 16	**Master Priority Register Reserved —** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_3	**Master 3 Priority —** These bits set the arbitration priority for master port 3 on the associated slave port. These bits are initialized by hardware reset. The reset value is 011	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 20	**Master Priority Register Reserved —** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_2	**Master 2 Priority —** These bits set the arbitration priority for master port 2 on the associated slave port. These bits are initialized by hardware reset. The reset value is 010	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 24	**Master Priority Register Reserved —** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_1	**Master 1 Priority —** These bits set the arbitration priority for master port 1 on the associated slave port. These bits are initialized by hardware reset. The reset value is 001	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.

Table 131. Master Priority Register descriptions (continued)

Name	Description	Settings
Bit 28	**Master Priority Register Reserved —** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_0	**Master 0 Priority —** These bits set the arbitration priority for master port 0 on the associated slave port. These bits are initialized by hardware reset. The reset value is 000	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.

The Master Priority Register can only be accessed in supervisor mode with 32-bit accesses. Once the RO (Read Only) bit has been set in the slave General Purpose Control Register the Master Priority Register can only be read from, attempts to write to it will have no effect on the MPR and result in an error response.

Note: No two available master ports may be programmed with the same priority level. Attempts to program two or more available masters with the same priority level will result in an error response and the MPR will not be updated.

15.3.2.2 Slave General Purpose Control Register

The Slave General Purpose Control Register (SGPCR) controls several features of each slave port.

The Read Only (RO) bit will prevent any registers associated with this slave port from being written to once set. This bit may be written with 0 as many times as the user desires, but once it is written to a 1 only a reset condition will allow it to be written again.

The Halt Low Priority (HLP) bit will set the priority of the **max_halt_request** input to the lowest possible priority for initial arbitration of the slave ports. By default it is the highest priority. Setting this bit will not effect the **max_halt_request** from attaining highest priority once it has control of the slave ports.

The PCTL bits determine how the slave port will park when no master is actively making a request. The available options are to park on the master defined by the PARK bits, park on the last master to use the slave port, or go into a low power park mode which will force all the outputs of the slave port to inactive states when no master is requesting an access. The low power park feature can result in an overall power savings if a the slave port is not saturated; however, it will force an extra clock of latency whenever any master tries to access it when it is not in use because it will not be parked on any master.

The PARK bits determine which master the slave will park on when no master is making an active request and the **max_halt_request** input is negated. Please use caution to only select master ports that are actually present in the design. If the user programs the PARK bits to a master not present in the current design implementation undefined behavior will result.

SGPCRn	Slave General Purpose Control Register n	Addr $BASE + 0x010 + n*100 Wait State: 0 Access: S

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	RO	HLP							HPE 7	HPE 6	HPE 5	HPE 4	HPE 3	HPE 2	HPE 1	HPE 0
TYPE:	rw	rw	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Note:	Once the RO bit is written to a 1, only hardware reset will return it to a 0.															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
							ARB				PCTL			PARK		
TYPE:	r	r	r	r	r	r	rw	rw	r	r	rw	rw	r	rw	rw	rw
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Note:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Note: for n = 0 to 7

Figure 135. Slave General Purpose Control Register n

Table 132. Slave General Purpose Control Register Descriptions

Name	Description	Setting
RO	**Read Only —** This bit is used to force all of a slave port's registers to be read only. Once written to 1 it can only be cleared by hardware reset. This bit is initialized by hardware reset. The reset value is 0	0All this slave port's registers can be written. 1All this slave port's registers are read only and cannot be written (attempted writes have no effect and result in an error response).
HLP	**Halt Low Priority —** This bit is used to set the initial arbitration priority of the max_halt_request input. This bit is initialized by hardware reset. The reset value is 0	0The max_halt_request input has the highest priority for arbitration on this slave port 1The max_halt_request input has the lowest initial priority for arbitration on this slave port.
Bits 2–7	**Slave General Purpose Control Register Reserved —** These bits are reserved for future expansion. They read as zero and should be written with zero for upward compatibility.	NA
HPEx	**High Priority Enable —** These bits are used to enable the mX_high_priority inputs for the respective master. These bits are initialized by hardware reset. The reset value is 0	0The mX_high_priority input is disabled on this slave port 1The mX_high_priority input is enabled on this slave port.

Table 132. Slave General Purpose Control Register Descriptions (continued)

Name	Description	Setting
Bits 16–21	**Slave General Purpose Control Register Reserved —** These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.	NA
ARB	**Arbitration Mode —** These bits are used to select the arbitration policy for the slave port. These bits are initialized by hardware reset. The reset value is 00	00Fixed Priority. 01Round Robin (rotating) Priority. 10Reserved 11Reserved
Bits 24–25	**Slave General Purpose Control Register Reserved —** These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.	NA
PCTL	**Parking Control —** These bits determine the parking control used by this slave port. These bits are initialized by hardware reset. The reset value is 00.	00When no master is making a request the arbiter will park the slave port on the master port defined by the PARK bit field. 01When no master is making a request the arbiter will park the slave port on the last master to be in control of the slave port. 10When no master is making a request the arbiter will park the slave port on no master and will drive all outputs to a constant safe state. 11Reserved
Bit 28	**Slave General Purpose Control Register Reserved —** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
PARK	**PARK —** These bits are used to determine which master port this slave port parks on when no masters are actively making requests and the PCTL bits are set to 00. These bits are initialized by hardware reset. The reset value is 000	000Park on Master Port 0 001Park on Master Port 1 010Park on Master Port 2 011Park on Master Port 3 100Park on Master Port 4 101Park on Master Port 5 110Park on Master Port 6 111Park on Master Port 7

The SGPCR can only be accessed in supervisor mode with 32-bit accesses. Once the RO (Read Only) bit has been set in the SGPCR the SGPCR can only be read, attempts to write to it will have no effect on the SGPCR and result in an error response.

15.3.2.3 Master General Purpose Control Register

The Master General Purpose Control Register (MGPCR) presently controls only whether or not the master's undefined length burst accesses will be allowed to complete uninterrupted or whether they can be broken by requests from higher priority masters.

The AULB (Arbitrate on Undefined Length Bursts) bit field determines whether (and when) or not the XBAR will arbitrate away the slave port the master owns when the master is performing undefined length burst accesses.

If the user has configured the XBAR to have less than eight master ports only the registers associated with the remaining master ports will be present, all other registers will become reserved locations in memory.

MGPCRn **Master General Purpose Control Register n** **Addr $BASE + 0x800 + n*100 Wait State: 0 Access: S**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TYPE:	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Note:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
														AULB		
TYPE:	r	r	r	r	r	r	r	r	r	r	r	r	r	rw	rw	rw
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Note:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Note: for n = 0 to 7

Figure 136. Master General Purpose Control Register n

Table 133. Master General Purpose Control Register Descriptions

Name	Description	Setting
Bits 0–28	**Master General Purpose Control Register Reserved —** These bits are reserved for future expansion. They read as zero and should be written with zero for upward compatibility.	NA
AULB	**Arbitrate on Undefined Length Bursts —** These bits are used to select the arbitration policy during undefined length bursts by this master. These bits are initialized by hardware reset. The reset value is 000	000No arbitration will be allowed during an undefined length burst. 001Arbitration will be allowed at any time during an undefined length burst. 010Arbitration will be allowed after four beats of an undefined length burst. 011Arbitration will be allowed after eight beats of an undefined length burst. 100Arbitration will be allowed after 16 beats of an undefined length burst. 101Reserved 110Reserved 111Reserved

The MGPCR can only be accessed in supervisor mode with 32-bit accesses.

15.3.3 Coherency

Since the content of the registers has a real time effect on the operation of the XBAR it is important for the user to understand that any register modifications take effect as soon as the register is written. The values of the registers do not track with slave port related AHB accesses but instead track only with IP bus accesses.

The exception to this rule are the AULB bits in the MGPCR. These update of these bits is only recognized when the master on that master port runs an IDLE cycle, even though the IP bus cycle to write them will have long since terminated successfully. If the AULB bits in the MGPCR are written in between two burst accesses the new AULB encodings will not take effect until an IDLE cycle has been initiated by the master on that master port.

15.4 Function

This section describes in more detail the functionality of the XBAR.

15.4.1 Arbitration

The XBAR supports two arbitration schemes; a simple fixed-priority comparison algorithm, and a simple round-robin fairness algorithm. The arbitration scheme is independently programmable for each slave port.

15.4.1.1 Arbitration During Undefined Length Bursts

Arbitration points during an undefined length burst are defined by the current master's MGPCR AULB field setting. When a defined length is imposed on the burst via the AULB bits the undefined length burst will be treated as a single or series of single back to back fixed length burst accesses.

Example: A master runs an undefined length burst and the AULB bits in the MGPCR indicate arbitration will occur after the fourth beat of the burst. The master runs two sequential beats and then starts what will be an 12 beat undefined length burst access to a new address within the same slave port region as the previous access. The XBAR will not allow an arbitration point until the fourth overall access (second beat of the second burst). At that point all remaining accesses will be open for arbitration until the master loses control of the slave port.

Assume the master loses control of the slave port after the fifth beat of the second burst. Once the master regains control of the slave port no arbitration point will be available until after the master has run four more beats of its burst. After the fourth beat of the (now continued) burst (ninth beat of the second burst from the master's perspective) is taken all beats of the burst will once again be open for arbitration until the master loses control of the slave port.

Assume the master again loses control of the slave port on the fifth beat of the third (now continued) burst (10th beat of the second burst from the master's perspective). Once the master regains control of the slave port it will be allowed to complete its final two beats of its burst without facing arbitration.

Note that fixed length burst accesses are not affected by the AULB bits. All fixed length burst accesses lock out arbitration until the last beat of the fixed length burst.

15.4.1.2 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.

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.

15.4.1.3 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. 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.

15.4.2 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.

15.4.2.1 Context Switching

The XBAR has a hardware input per slave port (**sX_ampr_sel**) which is used to select which registers the master priority levels and general purpose control bits will be taken from. When **sX_ampr_sel** is 0 the MPR and SGPCR will be selected. This hardware input is useful for context switching so the user does not have to rewrite the MPR or SGPCR if a particular slave port would temporarily benefit from modifying the master priority levels or functions affected by the bits in the SGPCR.

15.4.2.2 Priority Elevation

The XBAR has a hardware input per master port (**mX_high_priority**) which is used to temporarily elevate the master's priority level on all slave ports. When the master's **mX_high_priority** input is asserted the master port will automatically have higher priority than all other master ports that do not have their **mX_high_priority** input asserted regardless of the priority levels programmed in the MPR and AMPR. If multiple master ports have their **mX_high_priority** input asserted they will have higher priority than all master ports which do not have their **mX_high_priority** inputs asserted. The MPR priority level (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 in the slave port in order for the **mX_high_priority** inputs to be received by the slave port.

15.4.3 Master Port Functionality

15.4.3.1 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.

15.4.3.2 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.

15.4.3.3 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.

15.4.3.4 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.

15.4.3.5 Master Port State Machine

15.4.3.5.1 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).

15.4.3.5.2 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.

15.4.4 Slave Port Functionality

15.4.4.1 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.

15.4.4.2 Slave Port Muxes

The XBAR instantiates many 8 to 1 muxes, one for each master-to-slave signal in fact. All the muxes are designed in an AND - OR fashion, so that if no master is selected the output of the muxes will be zero. (This is an important feature for low power park mode.)

The muxes also have an override signal which is used by the slave port to asynchronously force IDLE cycles onto the slave bus. When the state machine forces an IDLE cycle it zeros out **htrans** and **hmastlock**, making sure the slave bus sees a valid IDLE cycle being run by the XBAR.

The enable to the mux controlling **htrans** also contains an additional control signal from the state machine so that a NSEQ transaction can be forced. This is done any time the slave port switches masters to ensure that no IDLE-SEQ, BUSY-SEQ or NSEQ-SEQ transactions are seen on the slave port when they shouldn't be. If the state machine indicates to run both an IDLE and an NSEQ cycle, the IDLE directive will have priority.

Note: IDLE-SEQ is in fact an illegal access, but a possible scenario given the multi-master environment in the XBAR unless corrected by the XBAR.

15.4.4.3 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.

15.4.4.4 Slave port state machine

15.4.4.4.1 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.

15.4.4.4.2 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.

15.4.4.4.3 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 137* 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.

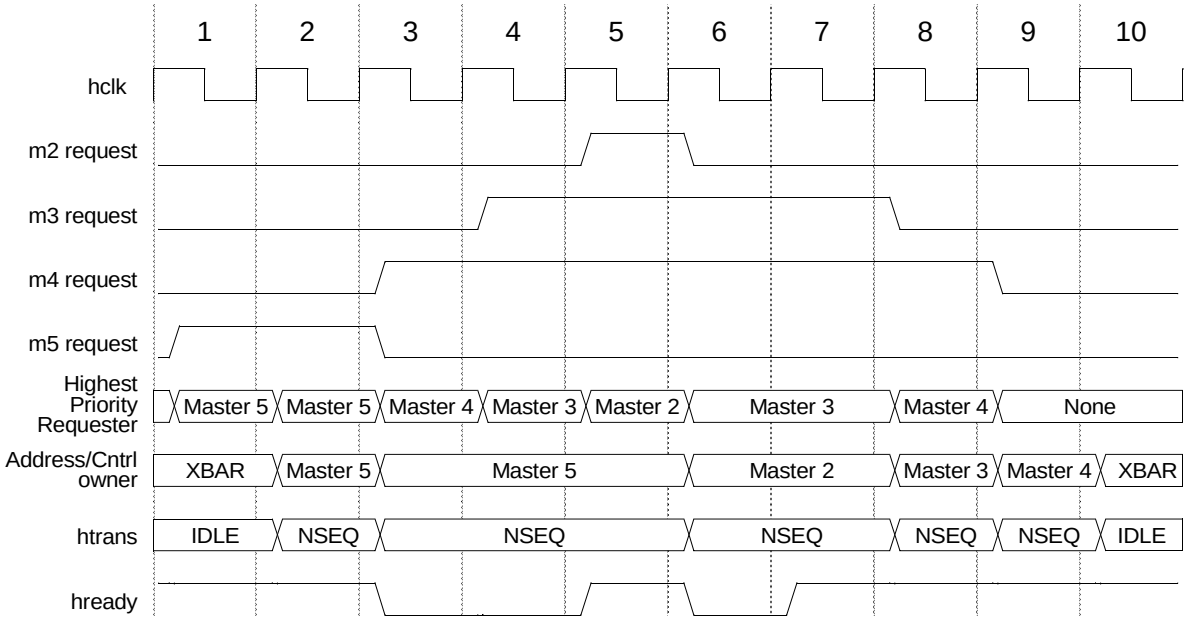


Figure 137. Low to high priority mastership change

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 138* illustrates the effect of a higher priority master giving up control of the bus.

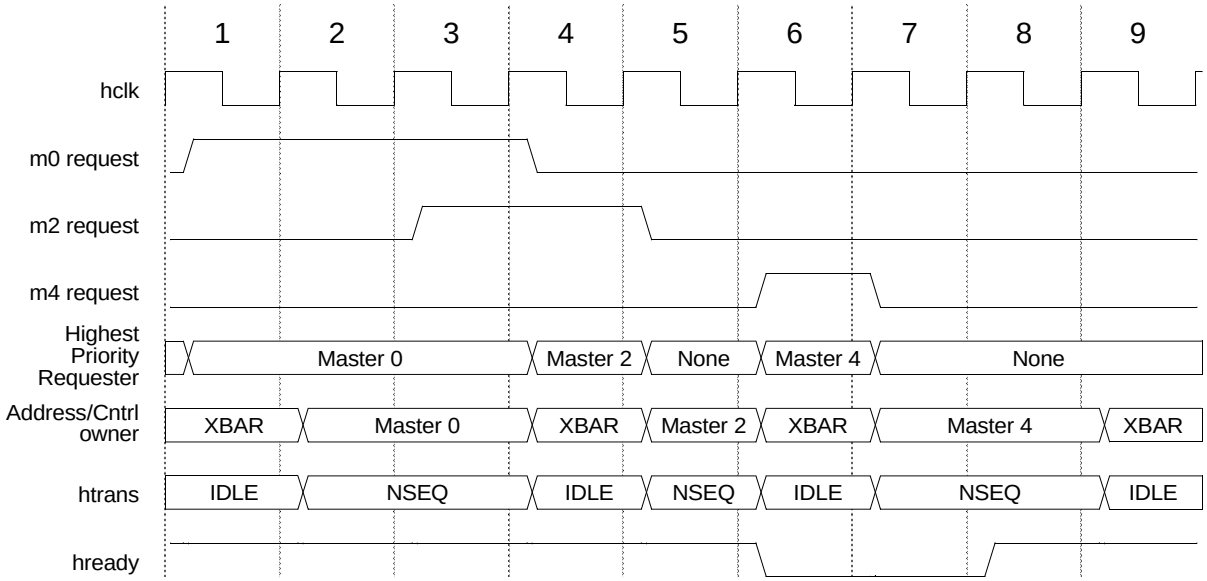


Figure 138. High to low priority mastership change

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 139* shows an example of round-robin mode of operation.

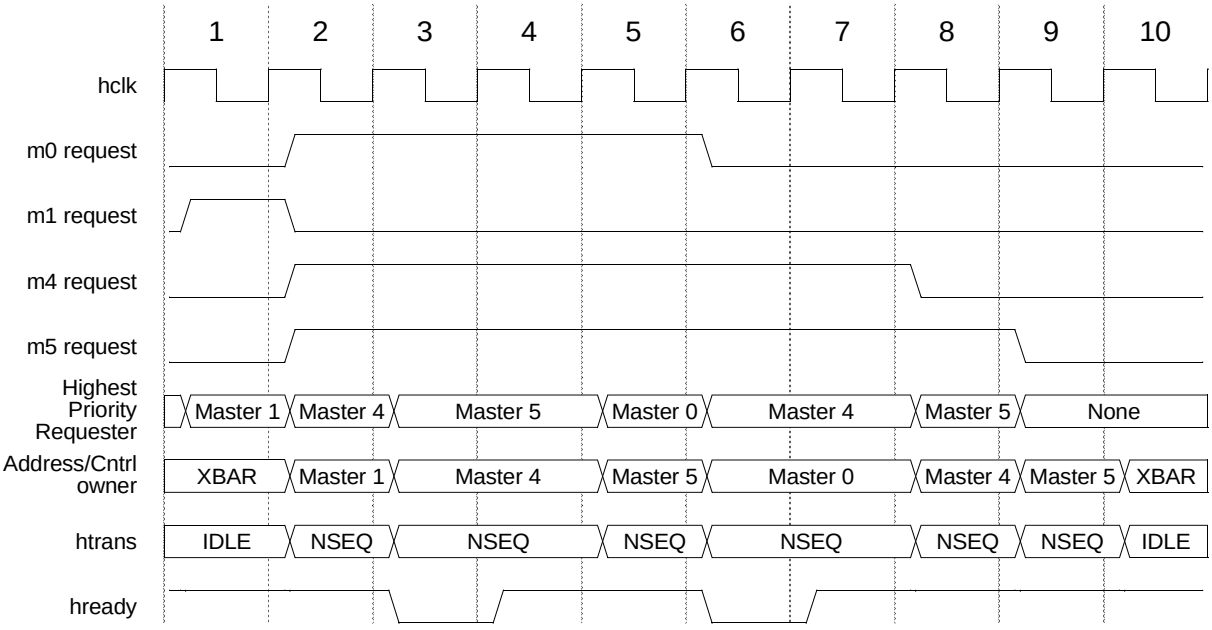


Figure 139. Round-robin mastership change

15.4.4.4.4 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 140 illustrates parking on a specific master.

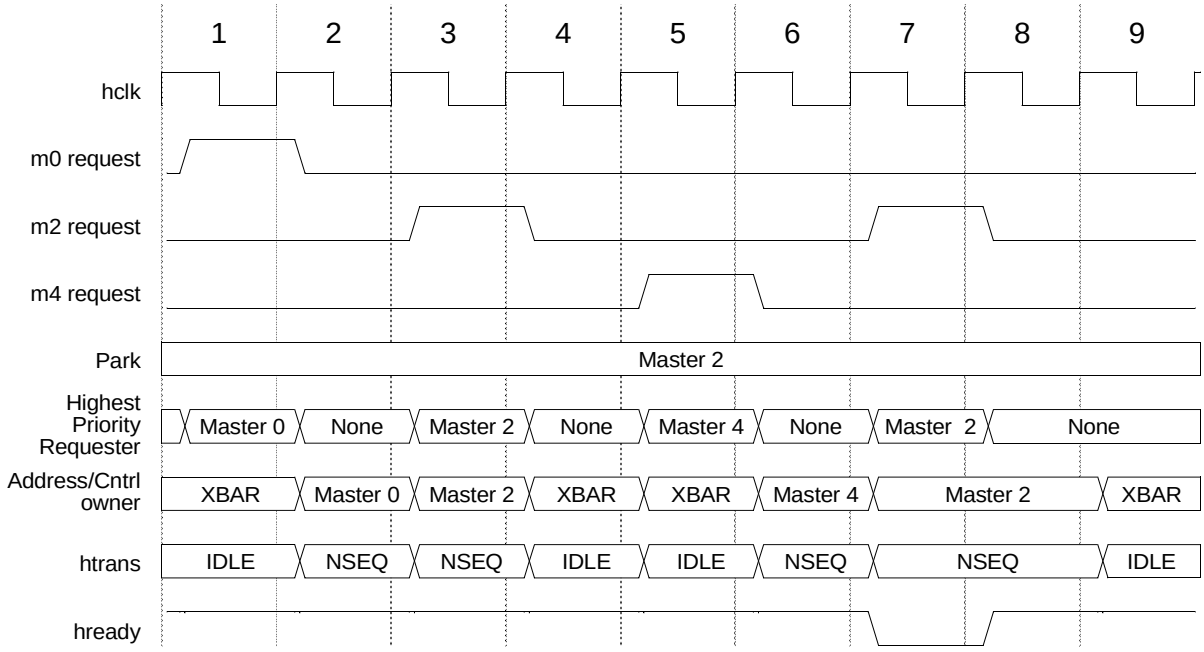


Figure 140. Parking on a specific master

Figure 141 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.

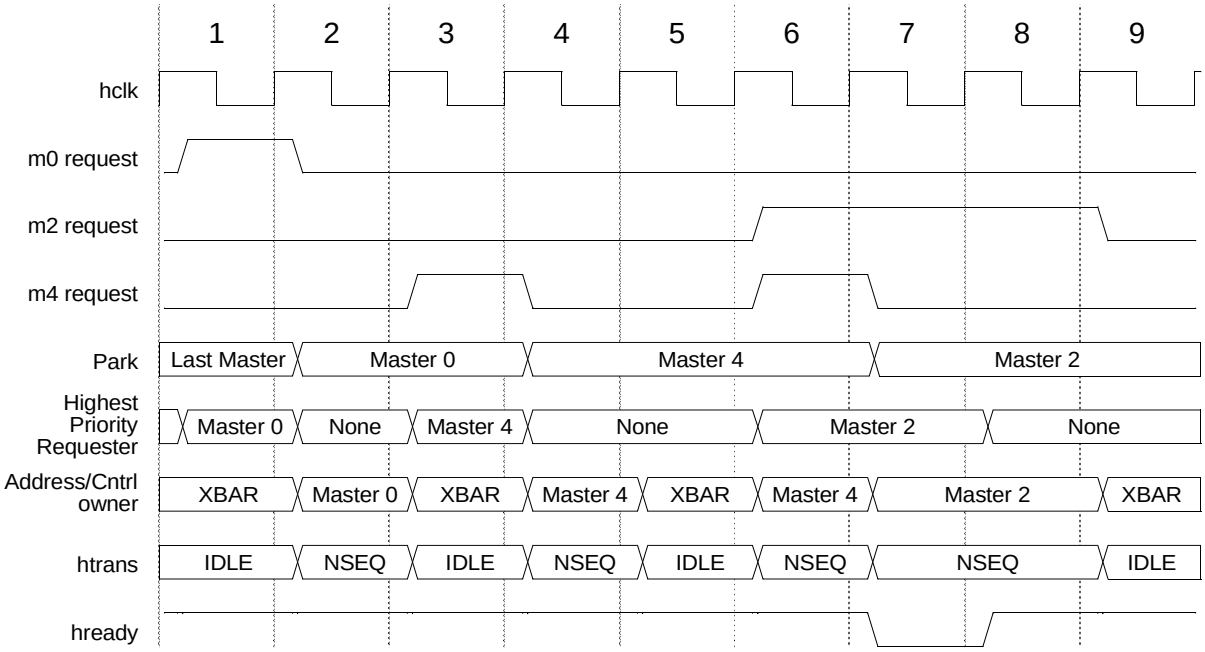


Figure 141. Parking on last master

15.4.4.4.5 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.

15.5 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.

15.6 Interface

This section provides information on the XBAR interface.

15.6.1 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.

15.6.2 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.

15.6.2.1 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.

15.6.2.2 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.

15.6.2.3 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.

15.6.2.4 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.

15.6.2.5 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.

15.6.3 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.

16 Cyclic Redundancy Checker (CRC) Unit

16.1 Introduction

The CRC computing unit is dedicated to the computation of CRC off-loading the CPU. Each context has a separate CRC computation engine in order to allow the concurrent computation of the CRC of multiple data streams. The CRC computation is performed at speed without wait states insertion. Bit-swap and bit-inversion operations can be applied on the final CRC signature. Each context can be configured with one of two (on cut1) or three (on cut2/3) hard-wired polynomials, normally used for most of the standard communication protocols. The data stream supports multiple data width (byte/half-word/word) formats.

16.1.1 Glossary

- CRC: cyclic redundancy check
- IPS: internal peripheral system
- CPU: central processing unit
- DMA: direct memory access
- CCITT: ITU-T (for Telecommunication Standardization Sector of the International Telecommunications Union)
- SW: software
- WS: wait state
- SPI: serial peripheral interface

16.2 Features

- 1 to 16 contexts (static parameter) for the concurrent CRC computation. For this application the number of context is CRC_CNTX_NUM = 3
- Separate CRC engine for each context
- Zero-wait states during the CRC computation (pipeline scheme)
- Up to 3 hard-wired polynomials:
  - CRC-8 (supported on cut2/3 only)
  - CRC-16-CCITT
  - CRC-32 ethernet
- Support for byte/half-word/word width of the input data stream
- IPS slave bus interface

16.3 Block diagram

The top level diagram of the CRC module is given in *Figure 142*. The number of context CRC_CNTX_NUM is equal to 3 for this application.

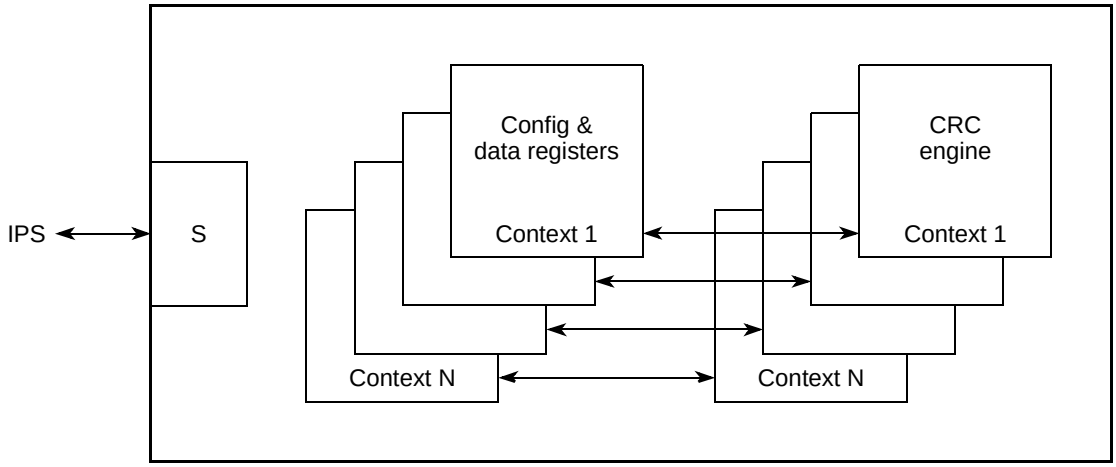


Figure 142. CRC top level diagram

16.4 Signal description

The CRC unit does not generate any external signals.

16.5 Register description

The CRC registers are listed in *Table 134*. The N index selects the specific context and is included in the range 1 to 3.

Table 134. CRC registers

Address offset	Register	Location
0x0 + (N–1)×0x10	CRC Configuration Register (CRC_CFG)	*on page 349*
0x4 + (N–1)×0x10	CRC Input Register (CRC_INP)	*on page 350*
0x8 + (N–1)×0x10	CRC Current Status Register (CRC_CSTAT)	*on page 351*
0xC + (N–1)×0x10	CRC Output Register (CRC_OUTP)	*on page 352*

16.5.1 CRC Configuration Register (CRC_CFG)

Figure 143. CRC Configuration Register (CRC_CFG) for cut1

Offset: 0x0 + (N–1)×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	0	0	0	0	0	0	0	0	0	0	0	0	0	POLYG	SWAP	INV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0/1(1)	0	0

1. 0 when N is even; 1 when N is odd

Figure 144. CRC Configuration Register (CRC_CFG) for cut2/3

Offset: 0x0 + (N–1)×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	0	0	0	0	0	0	0	0	0	0	0	0	POLYG		SWAP	INV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0/1(1)	0	0

1. 0 when N is even; 1 when N is odd

Table 135. CRC_CFG field descriptions

Field	Description
POLYG (for cut1)	*Polynomial selection* 0 CRC-CCITT polynomial 1 CRC-32 polynomial. This bit can be read and written by the software. This bit can be written only during the configuration phase.
POLYG (for cut2/3)	*Polynomial selection* 00 CRC-CCITT polynomial 01 CRC-32 polynomial 10 CRC-8 polynomial 11 Reserved This field can be read and written by the software. This field can be written only during the configuration phase.
SWAP	*SWAP selection* 0: No swap selection applied on the CRC_OUTP content 1: Swap selection (MSB -> LSB, LSB -> MSB) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the swap operation is applied on the 16 LSB bits. This bit can be read and written by the software. This bit can be written only during the configuration phase.
INV	*INV selection* 0: No inversion selection applied on the CRC_OUTP content 1: Inversion selection (bit x bit) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the inversion operation is applied on the 16 LSB bits. This bit can be read and written by the software. This bit can be written only during the configuration phase.

16.5.2 CRC Input Register (CRC_INP)

Figure 145. CRC Input Register (CRC_INP)

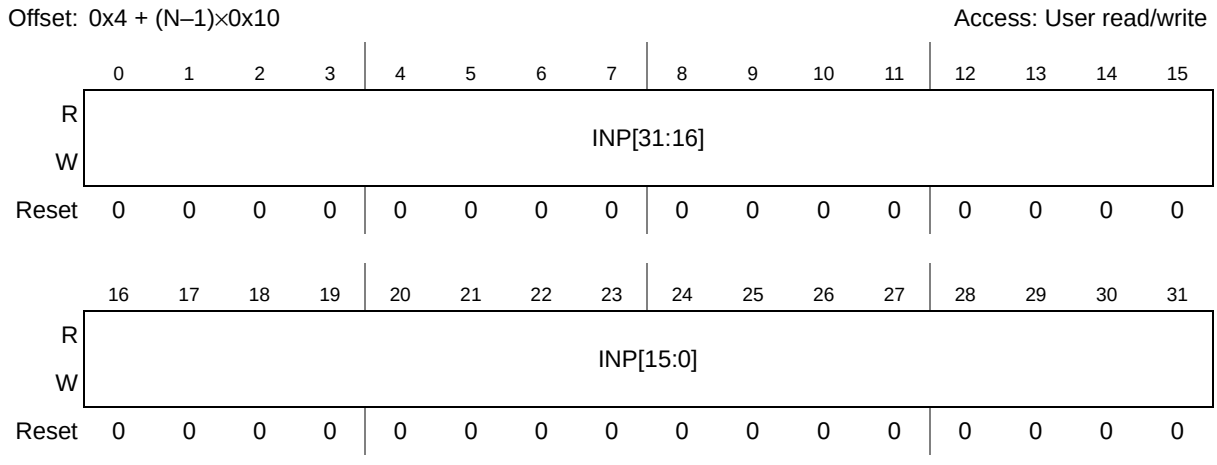


Table 136. CRC_INP field descriptions

Field	Description
INP	*Input data for the CRC computation* The INP register can be written at byte, half-word (high and low) or word in any sequence. In case of half-word write operation, the bytes must be contiguous. This register can be read and written by the software.

16.5.3 CRC Current Status Register (CRC_CSTAT)

Figure 146. CRC Current Status Register (CRC_CSTAT)

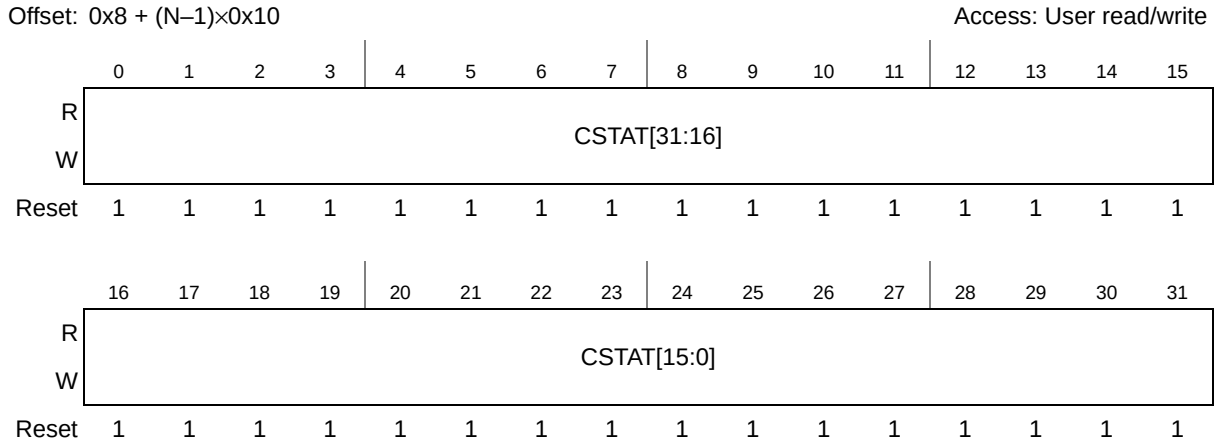


Table 137. CRC_CSTAT field descriptions

Field	Description
CSTAT	*Status of the CRC signature* The CSTAT register includes the current status of the CRC signature. No bit swap and inversion are applied to this register. In case of CRC-CCITT polynomial only the16 LSB bits are significative. The 16 MSB bits are tied at 0b during the computation. The CSTAT register can be written at byte, half-word or word. This register can be read and written by the software. This register can be written only during the configuration phase.

16.5.4 CRC Output Register (CRC_OUTP)

Figure 147. CRC Output Register (CRC_OUTP)

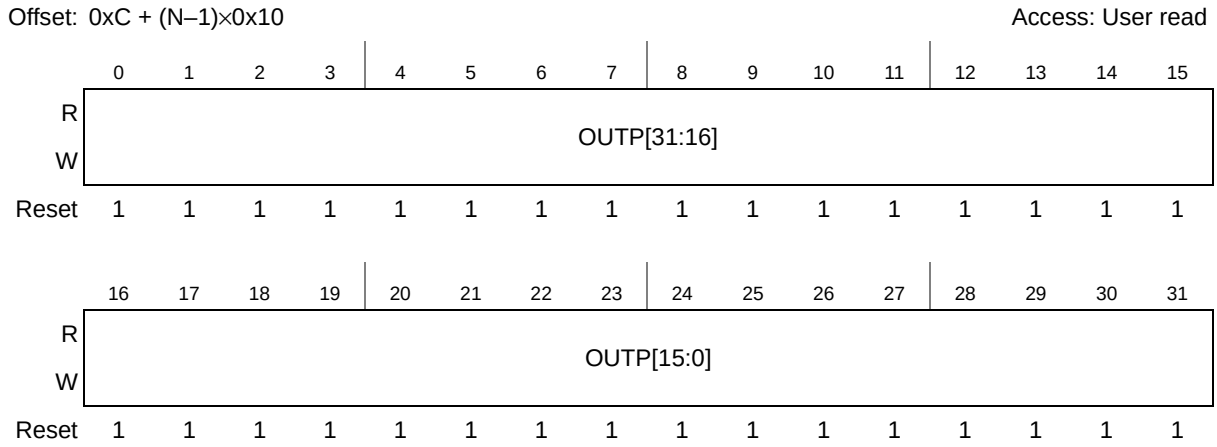


Table 138. CRC_OUTP field descriptions

Field	Description
OUTP	*Final CRC signature* The OUTP register includes the final signature corresponding to the CRC_CSTAT register value eventually swapped and inverted. In case of CRC-CCITT polynomial only the16 LSB bits are significative. The 16 MSB bits are tied at 0b during the computation. This register can be read by the software.

16.6 Functional description

The CRC module supports the CRC computation for each context. Each context has a own complete set of registers including the CRC engine. The data flow of each context can be interleaved. The data stream can be structured as a sequence of byte, half-words or words. The input data sequence is provided, eventually mixing the data formats (byte, half-word, word), writing to the input data register (CRC_INP).

The data stream is generally executed by N concurrent DMA data transfers (mem2mem) where N is less or equal to the number of contexts (3).

Three standard generator polynomials are given in *Equation 2* through *Equation 4* for the CRC computation of each context.

$$X^8 + X^4 + X^3 + X^2 + 1$$

Equation 2 CRC-8 (cut2/3 only)

$$X^{16} + X^{12} + X^5 + 1$$

Equation 3 CRC-CCITT (x25 protocol)

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

Equation 4 CRC-32 (ethernet protocol)

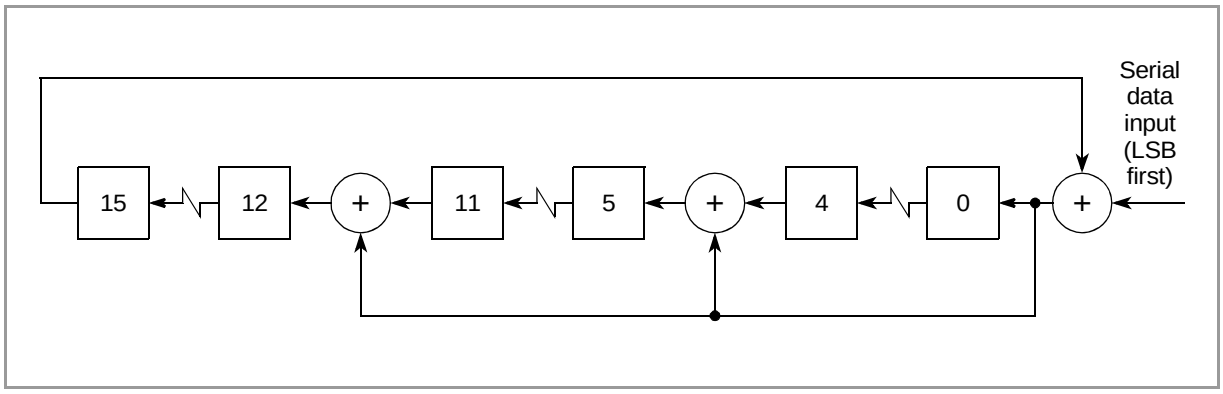


Figure 148. CRC-CCITT engine concept scheme

Note: CRC-32 (Ethernet) and CRC-16 (CCITT) do not calculate CRCs according to the industry standards. The CRC engine expects the CRC Input to be byte swapped for CRC- 32 and CRC Input to be bit reversed for CRC-16. The input data should be bit reversed and entered into the CRC Input Register (CRC_INP) for CRC-16 calculations and should be byte reversed for CRC-32 calculations to get the correct CRC signature.

For the given input data of 0xABCDEF98, the user software should enter 0x98EFCDAB in CRC_INP register to get the correct result for CRC-32. Also, the Inversion (INV) and Swap selection (SWAP) bits of the CRC Configuration Register (CRC_CFG) should be set to 1 for CRC-32 polynomial calculations along with the byte swaps.

For the given input data of 0xABDEF98, the user software must enter 0x19F7B3D5 into the CRC_INP register to get the correct result for CRC-16. The INV and SWAP bits are programmed to zero for CRC-16 operation.

The initial seed value of the CRC can be programmed initializing the CRC_CSTAT register. The concept scheme (serial data loading) of the CRC engine is given in *Figure 148* for the CRC-CCITT. The design implementation executes the CRC computation in a single clock cycle (parallel data loading). A pipeline scheme has been adopted to de-couple the IPS bus interface from the CRC engine in order to allow the computation of the CRC at speed (zero wait states).

In case of usage of the CRC signature for encapsulation in the data frame of a communication protocol (e.g. SPI, ..) a bit swap (MSB → LSB, LSB → MSB) and/or bit inversion of the final CRC signature can be applied (CRC_OUTP register) before to transmit the CRC.

The usage of the CRC module is summarized in the flowchart given in *Figure 149*.

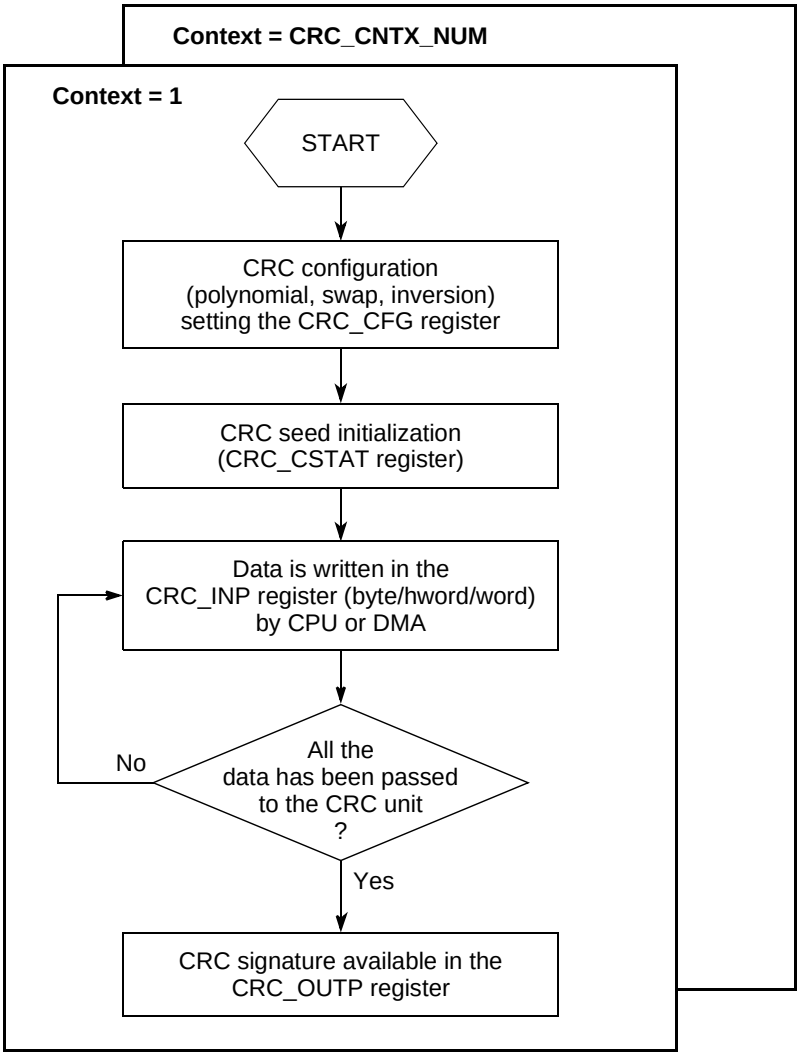


Figure 149. CRC computation flow

16.7 Use cases and limitations

The number of contexts is dependent on the application. The overall number of contexts for the CRC peripheral depends on the number of the peripherals that require concurrently the intervention of the CRC module. 2 main use cases shall be considered:

- calculation of the CRC of the configuration registers during the process safety time
- calculation of the CRC on the incoming/outcoming frames for the communication protocols (not protected with CRC by definition of the protocol itself) used as a safety-relevant peripheral.

The signature of the configuration registers is computed in a correct way only if these registers do not contain any status bit. Assuming that the DMA engine has N channels (greater or equal to the number of contexts) configurable for the following type of data transfer: mem2mem, periph2mem, mem2periph, the following sequence, as given in

Figure 150, shall be applied to manage the transmission data flow:

- DMA/CRC module configuration (context x, channel x) by CPU
- Payload transfer from the MEM to the CRC module (CRC_INP register) to calculate the CRC signature (phase1) by DMA (mem2mem data transfer, channel x)
- CRC signature copy from the CRC module (CRC_OUTP register) to the MEM (phase 2) by CPU
- Data block (payload + CRC) transfer from the MEM to the PERIPH module (e.g. SPI Tx fifo) (phase 3) by DMA (mem2periph data transfer, channel x)

The following sequence, as given in *Figure 150*, shall be applied to manage the reception data flow:

- DMA/CRC module configuration (context x, channel x) by CPU
- Data block (payload + CRC) transfer from the PERIPH (e.g. SPI Rx fifo) module to the MEM (phase 1) by DMA (periph2mem data transfer, channel x)
- Data block transfer (payload + CRC) transfer from the MEM to the CRC module (CRC_INP register) to calculate the CRC signature (phase 2) by DMA (mem2mem data transfer, channel x)
- CRC signature check from the CRC module (CRC_OUTP register) by CPU (phase 3)

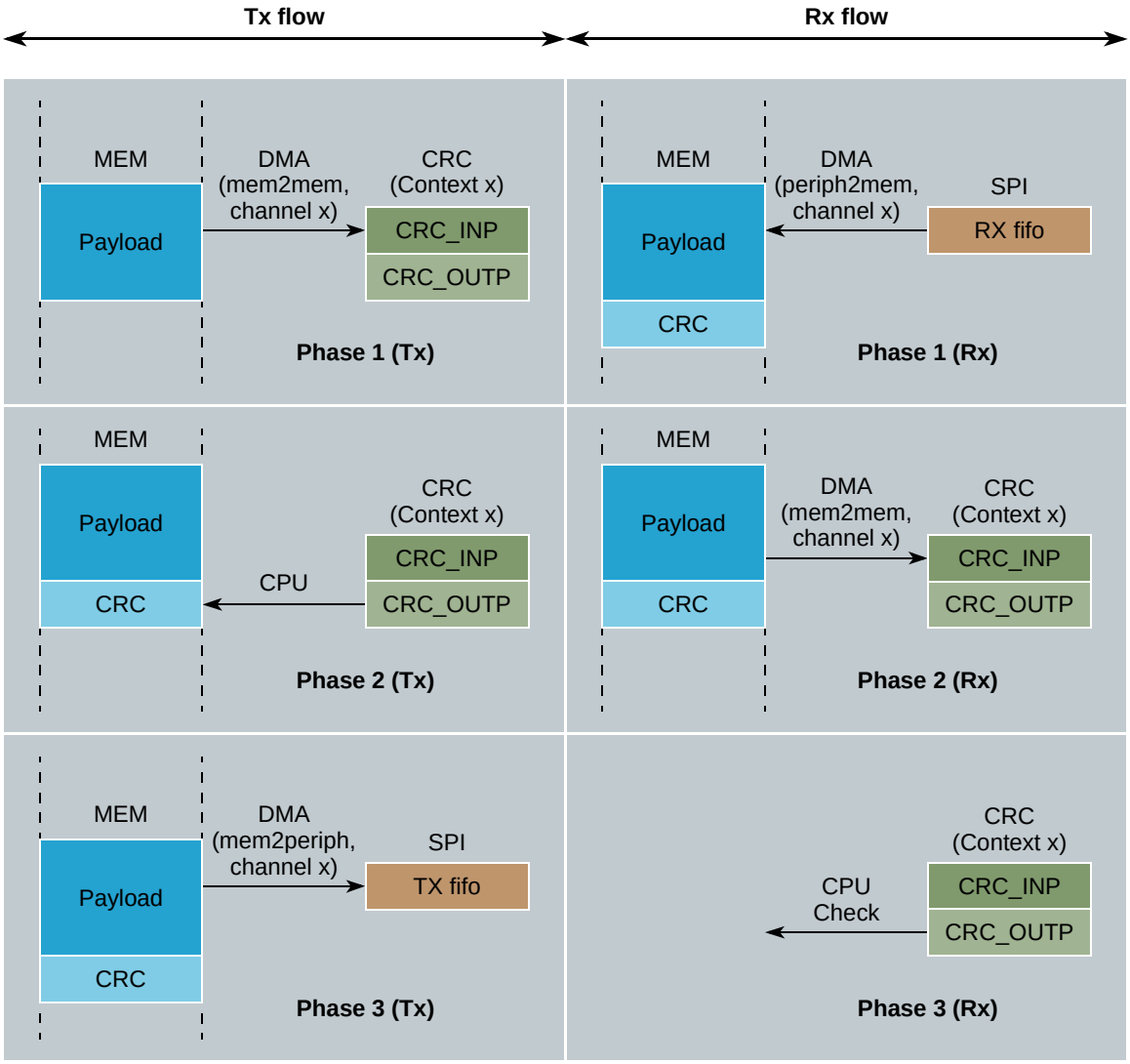


Figure 150. DMA-CRC (Tx flow, Rx flow)

17 Deserial Serial Peripheral Interface (DSPI)

17.1 Introduction

17.1.1 Overview

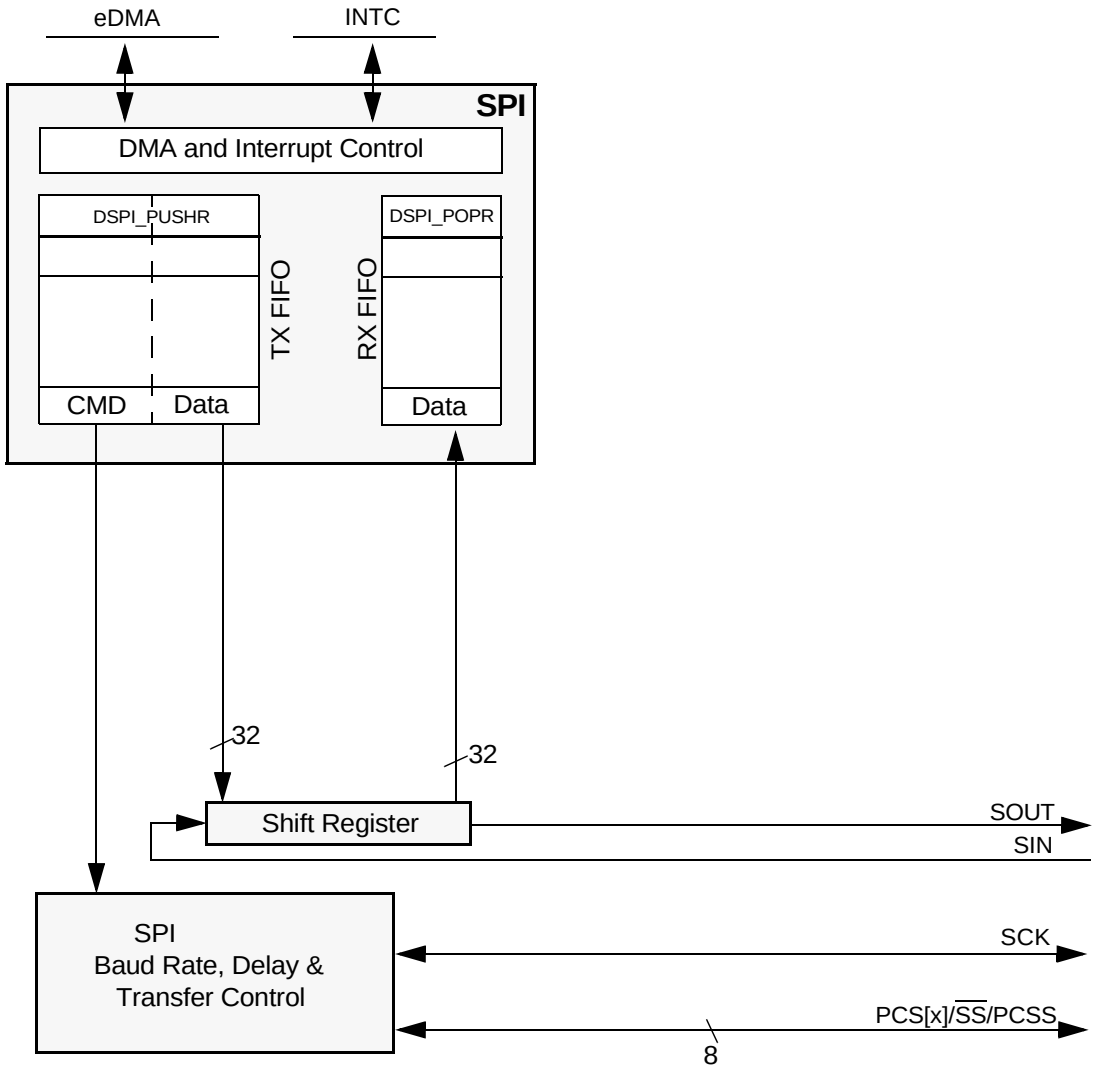
The Deserial Serial Peripheral Interface (DSPI) module provides a synchronous serial bus for communication between an SoC and an external peripheral device.

The SPC56ELxx device has three DSPI modules, referred to as DSPI_0, DSPI_1, and DSPI_2.

The DSPI modules interact with the CTU as described in *Section 14.4.1, Interaction with other peripherals.*

Figure 151 is a block diagram of the DSPI module.

Figure 151. DSPI block diagram



17.1.2 Features

The DSPI provides these features:

- Full-duplex, synchronous transfers
- Master or slave operation
  - Data streaming operation in the slave mode with continuous slave selection
- Buffered transmit operation using the TX FIFO with 5 entries
- Buffered receive operation using the RX FIFO with 5 entries
- Programmable transfer attributes on a per-frame basis:
  - 4 transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Various programmable delays
  - Programmable serial frame size of 4 to 16 bits, expandable by software control
  - Programmable master bit rates
- End-of-transmission interrupt flag
- Programmable transfer baud rate
- As many as 8 chip select lines available, depending on package and pin multiplexing
- 4 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 as many as 5 transfers on the transmit and receive side
- Queueing operation possible through use of the eDMA
- General-purpose I/O functionality on pins when not used for SPI

17.1.3 DSPI configurations

The DSPI module on this device operates only in the SPI configuration.

17.1.3.1 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 152* shows a system example with DMA, DSPI and external queues in system RAM.

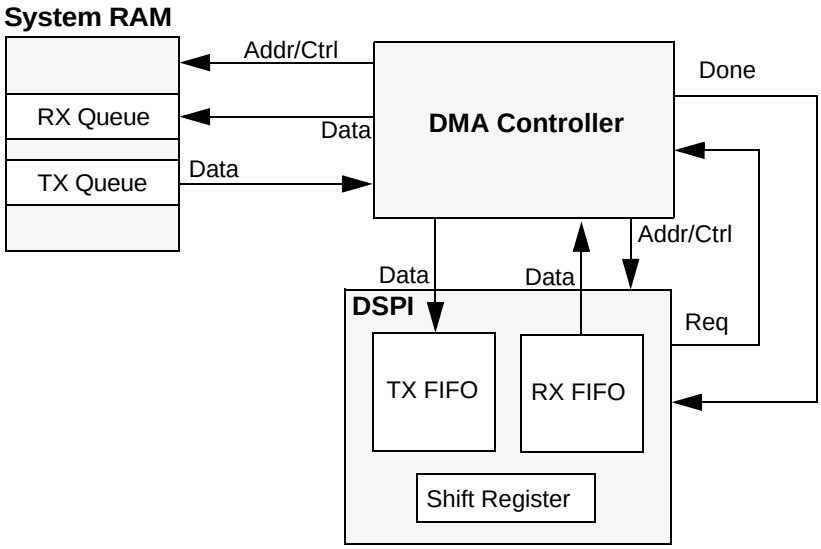


Figure 152. DSPI with queues and DMA

17.1.4 Modes of operation

The DSPI has five 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.

17.1.4.1 Master Mode

Master mode allows the DSPI to initiate and control serial communication. In this mode, the SCK signal and the PCS[*x*] signals are controlled by the DSPI and configured as outputs.

17.1.4.2 Slave Mode

The slave mode allows the DSPI to communicate with SPI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The SCK signal and the PCS[0]/$\overline{SS}$ signal are configured as inputs and driven by a SPI bus master.

Note: If the DSPI operates in slave mode and the external master has selected this module with the SS signal, stop mode can't be acknowledged until the module is deselected.

17.1.4.3 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.

17.1.4.4 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.

No register access is possible in this mode.

17.1.4.5 Debug Mode

The debug mode is used for system development and debugging. In this mode, you can read the last data from the FIFO without any changes. 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.

The distinguishing feature of Debug Mode is that if the Tx/Rx FIFOs have valid entries, the module will still not initiate any serial transfer and all registers can be accessed.

17.2 External signal description

Table 139 lists the DSPI signals. See *Section 4.4, Pin muxing*, for details of what signals are available on the DSPI modules on this chip.

Table 139. Signal properties

Name	I/O Type	Function	
		Master Mode	Slave Mode
PCS[0]/$\overline{SS}$	Output / Input	Peripheral Chip Select 0	Slave Select
PCS[1] - PCS[3]	Output	Peripheral Chip Select 1 - 3	Unused
PCS[4]	Output	Peripheral Chip Select 4	Unused
PCS[5]/$\overline{PCSS}$	Output	Peripheral Chip Select 5 / Peripheral Chip Select Strobe	Unused
PCS[6] - PCS[7]	Output	Peripheral Chip Select 6- 7	Unused
SIN	Input	Serial Data In	Serial Data In
SOUT	Output	Serial Data Out	Serial Data Out
SCK	Output / Input	Serial Clock (output)	Serial Clock (input)

17.2.1 PCS[0]/$\overline{SS}$ — Peripheral Chip Select/Slave Select

In master mode, the PCS[0] signal is a Peripheral Chip Select output that selects which slave device the current transmission is intended for.

In slave mode, the active low $\overline{SS}$ signal is a Slave Select input signal that allows a SPI master to select the DSPI as the target for transmission.

17.2.2 PCS[1] - PCS[3] — Peripheral Chip Selects 1–3

In master mode, PCS[1] - PCS[3] are Peripheral Chip Select output signals.

In slave mode these signals are not used.

17.2.3 PCS[4] — Peripheral Chip Select 4

PCS[4] is a Peripheral Chip Select output signal.

17.2.4 PCS[5]/$\overline{PCSS}$ — Peripheral Chip Select 5/Peripheral Chip Select Strobe

PCS[5] is a Peripheral Chip Select output signal.

When the DSPI is in master mode and the DSPI_MCR[PCSSE] bit is cleared, this signal selects which slave device the current transfer is intended for.

When the DSPI is in master mode and the DSPI_MCR[PCSSE] bit is set, the $\overline{PCSS}$ signal acts as a strobe to external peripheral chip select demultiplexer, which decodes the PCS[0] - PCS[4] and PCS[6] - PCS[7] signals, preventing glitches on the demultiplexer outputs.

This signal is not used in slave mode.

17.2.5 PCS[6] - PCS[7] — Peripheral Chip Selects 6 - 7

In master mode, PCS[6] - PCS[7] are Peripheral Chip Select output signals.

In slave mode these signals are not used.

17.2.6 SIN — Serial Input

SIN is a serial data input signal.

17.2.7 SOUT — Serial Output

SOUT is a serial data output signal.

17.2.8 SCK — Serial Clock

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.

17.3 Memory map and register definition

17.3.1 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 140 shows the DSPI memory map.

Table 140. DSPI memory map

Address offset	Register name	Location
0x0	DSPI Module Configuration Register (DSPI_MCR)	*on page 362*
0x4	DSPI Hardware Configuration Register (DSPI_HCR)(1) [cut2/3 only]	*on page 366*
0x8	DSPI Transfer Count Register (DSPI_TCR)	*on page 367*
0xC–0x18	DSPI Clock and Transfer Attributes Register 0 (DSPI_CTAR0) - DSPI Clock and Transfer Attributes Register 3(DSPI_CTAR3)	*on page 367*
0x2C	DSPI Status Register (DSPI_SR)	*on page 373*
0x30	DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)	*on page 375*
FIFO Registers		
0x34	DSPI Push TX FIFO Register (DSPI_PUSHR)	*on page 377*
0x38	DSPI Pop RX FIFO Register (DSPI_POPR)	*on page 380*
0x3C–0x4C	DSPI Transmit FIFO Register 0 (DSPI_TXFR0) - DSPI Transmit FIFO Register 4 (DSPI_TXFR4)	*on page 380*
0x7C - 0x8C	DSPI Receive FIFO Register 0 (DSPI_RXFR0) - DSPI Receive FIFO Register 4 (DSPI_RXFR4)	*on page 381*

1. The DSPI_HCR register provides parametization information about particular instance of the DSPI module.

17.3.2 Register descriptions

17.3.2.1 DSPI Module Configuration Register (DSPI_MCR)

The DSPI_MCR contains bits that 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 DSPIx_MCR are allowed to be changed, while the DSPI is in the Running state.

Figure 153. DSPI Module Configuration Register (DSPI_MCR) for cut1

Address: DSPI_BASE Access:

	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	PCSIS7	PCSIS6	PCSIS5	PCSIS4	PCSIS3	PCSIS2	PCSIS1	PCSIS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	MDIS	DIS_TXF	DIS_RXF	0	0	SMPL_PT		0	0	0	0	0	0	0	HALT
W					CLR_TXF	CLR_RXF										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 154. DSPI Module Configuration Register (DSPI_MCR) for cut2/3

Address: DSPI_BASE Access:

	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	PCSIS7	PCSIS6	PCSIS5	PCSIS4	PCSIS3	PCSIS2	PCSIS1	PCSIS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Table 141. 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	Continuous SCK Enable. The CONT_SCKE bit enables the Serial Communication Clock (SCK) to run continuously. See *Section 17.4.5, 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 Reserved 10 Reserved 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 17.4.4.4, Modified SPI Transfer Format (MTFE = 1, CPHA = 1)*, for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled
PCSSE	Peripheral Chip Select Strobe Enable. The PCSSE bit enables the PCS[5]/$\overline{\text{PCSS}}$ to operate as an PCS Strobe output signal. See *Section 17.4.3.5, Peripheral Chip Select Strobe Enable (PCSS)*, for more information. 0 PCS[5]/$\overline{\text{PCSS}}$ is used as the Peripheral Chip Select[5] signal 1 PCS[5]/$\overline{\text{PCSS}}$ is used as an active-low PCS 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 17.4.6.6, 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. The PCSIS bit determines the inactive state of the PCS*x* signal. 0 The inactive state of PCS*x* is low 1 The inactive state of PCS*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 17.4.7, Power Saving Features*, for more information. The reset value of the MDIS bit is parameterized, with a default reset value of ‘0’. 0 Enable DSPI clocks. 1 Allow external logic to disable DSPI clocks.

Table 141. DSPI_MCR field descriptions (continued)

Field	Description
DIS_TXF	Disable Transmit FIFO. When DIS_TXF is set, the TX FIFO acts as a single-entry (unit depth) FIFO. Therefore, serial operation is performed as if the FIFO has only one valid entry space for serial-word transfer. See *Section 17.4.2.3, FIFO Disable Operation*, for details. 0 TX FIFO is enabled 1 TX FIFO is disabled
DIS_RXF	Disable Receive FIFO. When DIS_RXF is set, the RX FIFO acts as a single-entry (unit depth) FIFO. Therefore, serial operation is performed as if the FIFO has only one valid entry space for serial-word transfer. See *Section 17.4.2.3, 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 175* 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 (on cut2/3 only)	Parity Error Stop. This 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 17.4.1, Start and Stop of DSPI Transfers*, for details on the operation of this bit. 0 Start transfers 1 Stop transfers

17.3.2.2 DSPI Hardware Configuration Register (DSPI_HCR) [cut2/3 only]

The DSPI_HCR provides particular implementation details about the DSPI module, i.e. number of Receive and Transmit FIFO entries, and the number of CTAR registers. It is a read-only register.

Figure 155. DSPI Hardware Configuration Register (DSPI_HCR) [cut2/3 only]

Address: DSPI_BASE + 0x4 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	PISR	0	0	0	CTAR			TXFR				RXFR			
W																
Reset	—(1)	—	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

1. The reset bits in the DSPI_HCR are set by configuration parameters in the chip.

Table 142. DSPI_HCR field descriptions

Field	Description
PISR	PISR, PISR0-3 and parallel inputs frame positions selection logic are implemented for the module. 0 - DSPI_PISR0-3 registers are not implemented. 1 - DSPI_PISR0-3 registers are implemented
CTAR	CTAR, Maximum implemented DSPI_CTAR register number.
TXFR	TXFR, Maximum implemented DSPI_TXFR register number.
RXFR	RXFR, Maximum implemented DSPI_RXFR register number.

17.3.2.3 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.

Figure 156. DSPI Transfer Count Register (DSPI_TCR)

Address: DSPI_BASE + 0x8 Access:

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

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

Table 143. DSPI_TCR Field Descriptions

Field	Description
0–15 SPI_TCNT[0:15]	SPI Transfer Counter. The SPI_TCNT field counts the number of SPI transfers the DSPI makes. The SPI_TCNT field increments every time the last bit of a SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The Transfer Counter ‘wraps around’ i.e. incrementing the counter past 65535 resets the counter to zero.
16–31	Reserved, should be cleared.

17.3.2.4 DSPI Clock and Transfer Attributes Registers 0–3 (DSPI_CTAR0–DSPI_CTAR3)

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_CTAR3 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.

Figure 157. DSPI Clock and Transfer Attributes Register 0–3 (DSPI_CTAR0–DSPI_CTAR3) in master mode

Address: DSPI_BASE + 0xC–DSPI_BASE + 0x18 Access:

	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 158. DSPI Clock and Transfer Attributes Register 0, 1 (DSPI_CTAR0 for cut1, DSPI_CTAR1) in slave mode

Address: DSPI_BASE + 0xC /0x10 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FMSZ					CPOL	CPHA	0	0	0	0	0	0	0	0	0
W																
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	0	0	0	0	0	0	0	0	0	0	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 159. DSPI Clock and Transfer Attributes Register 0 (DSPI_CTAR0) in slave mode for cut2/3

Address: DSPI_BASE + 0xC Access:

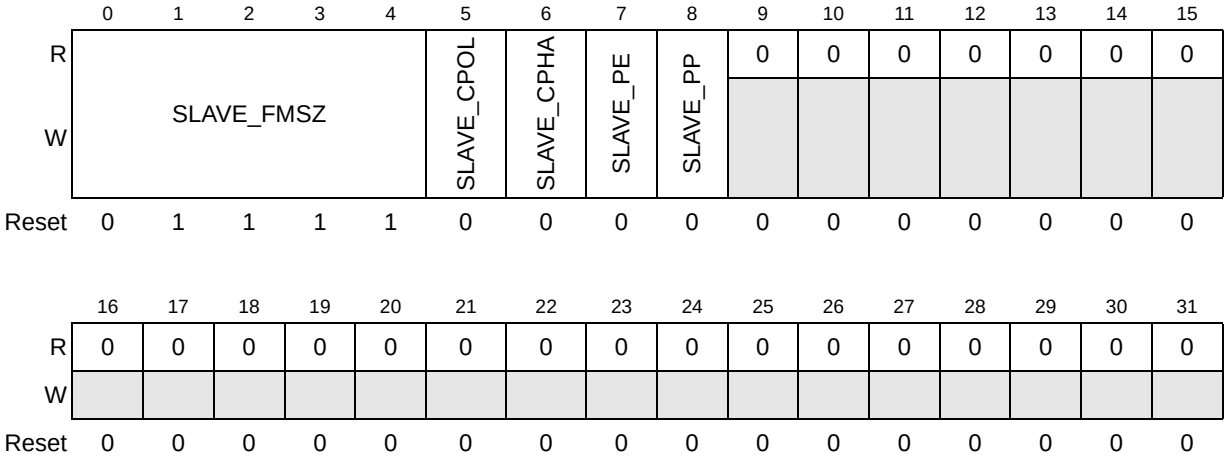


Table 144. DSPI_CTAR*n* field descriptions in master mode

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 145*. See the BR field description for details on how to calculate 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.
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 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

Table 144. DSPI_CTAR*n* field descriptions in master mode (continued)

Field	Descriptions
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. 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. 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
PDT	Delay after Transfer Prescaler. The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is only used in master mode. 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	PCS to SCK Delay Scaler. The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in master mode. The PCS to SCK Delay (t_{CSC}) is the delay between the assertion of PCS and the first edge of the SCK. *Table 146* list the scaler values.The PCS to SCK Delay is a multiple of the system clock period and it is computed according to the following equation: **Equation 5** $t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$ See *Section 17.4.3.2, PCS to SCK Delay (t_{CSC})*, for more details.

Table 144. DSPI_CTAR*n* field descriptions in master mode (continued)

Field	Descriptions
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 (t_{ASC}) is the delay between the last edge of SCK and the negation of PCS. *Table 146* 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 6** $t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$ See *Section 17.4.3.3, After SCK Delay (t_{ASC})*, for more details.
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 (t_{DT}) is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. *Table 146* 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 7** $t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$ See *Section 17.4.3.4, 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 147* lists the Baud Rate Scaler values.The baud rate is computed according to the following equation: **Equation 8** $\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$ See *Section 17.4.3.1, Baud rate generator*, for more details.

Table 145. 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 146. 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
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 147. 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 148. DSPI_CTAR0, DSPI_CTAR1 Field Descriptions in slave mode

Field	Descriptions
SLAVE_ FMSZ	Slave frame size. The number of bits transferred per frame is equal SLAVE_FMSZ field value plus 1. Minimum valid SLAVE_FMSZ field value is 3.
SLAVE_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
SLAVE_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

Table 148. DSPI_CTAR0, DSPI_CTAR1 Field Descriptions in slave mode

Field	Descriptions
SLAVE_PE (for cut2/3 only)	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.
SLAVE_PP (for cut2/3 only)	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.

17.3.2.5 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.

Figure 160. DSPI Status Register (DSPI_SR) for cut1

Address: DSPI_BASE + 0x2C Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	0	0	RFOF	0	RFDF	0
W	w1c	w1c		w1c	w1c		w1c						w1c		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 161. DSPI Status Register (DSPI_SR) for cut2/3

Address: DSPI_BASE + 0x2C Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	SPEF	0	RFOF	0	RFDF	0
W	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

Table 149. DSPI_SR field descriptions

Field	Description
TCF	Transfer Complete Flag. A value of TCF = 1 indicates that at least one serial-data-word (frame) from the FIFO has been transferred/received over the serial link. For example, if after 4 transfer words are written, then TCF would be set after the first word has been completely transferred. 0 Transfer not complete 1 Transfer complete *Note: It is recommended not to write this bit when the transfer is occurring, since the update from the serial-link side has a higher priority than the register access. In other words, register access is ignored if it occurs in the same clock cycle as the completion of any serial transfer.*
TXRXS	TX & RX Status. The TXRXS bit reflects the run status of the DSPI. See *Section 17.4.1, 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 (DSPI Push Tx FIFO) 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 *Note: EOQF does not function in slave mode.*
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

Table 149. DSPI_SR field descriptions (continued)

Field	Description
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
SPEF (on cut2/3 only)	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
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 17.4.6.4, 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 17.4.2.5, Receive First In First Out (RX FIFO) Buffering Mechanism*, for more details.

17.3.2.6 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.

You are allowed to write to this register in Module Disable mode.

Figure 162. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER) for cut1

Address: DSPI_BASE + 0x30 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF_RE	0	0	EOQF_RE	TFUF_RE	0	TFFF_RE	TFFF_DIRS	0	0	0	0	RFOF_RE	0	RFDF_RE	RFDF_DIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 163. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER) for cut2/3

Address: DSPI_BASE + 0x30 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF_RE	0	0	EOQF_RE	TFUF_RE	0	TFFF_RE	TFFF_DIRS	0	0	SPEF_RE	0	RFOF_RE	0	RFDF_RE	RFDF_DIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Table 150. DSPI_RSER field descriptions

Field	Description
TCF_RE	Transmission Complete Request Enable. The TCF_RE 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
EOQF_RE	DSPI Finished Request Enable. The EOQF_RE 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

Table 150. DSPI_RSER field descriptions (continued)

Field	Description
TFUF_RE	Transmit FIFO Underflow Request Enable. The TFUF_RE 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
TFFF_RE	Transmit FIFO Fill Request Enable. The TFFF_RE bit enables the TFFF flag in the DSPI_SR to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled 1 TFFF interrupt requests or DMA requests are enabled
TFFF_DIRS	Transmit FIFO Fill DMA or Interrupt Request Select. The TFFF_DIRS bit selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI_SR is set, and the TFFF_RE bit in the DSPI_RSER register is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated
SPEF_RE (for cut2/3 only)	SPI Parity Error Request Enable. The SPEF_RE bits enables SPEF flag in the DSPI_SR to generate an interrupt requests. 0 PEF interrupt requests are disabled 1 PEF interrupt requests are enabled
RFOF_RE	Receive FIFO Overflow Request Enable. The RFOF_RE 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
RFDF_RE	Receive FIFO Drain Request Enable. The RFDF_RE bit enables the RFDF flag in the DSPI_SR to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled 1 RFDF interrupt requests or DMA requests are enabled
RFDF_DIRS	Receive FIFO Drain DMA or Interrupt Request Select. The RFDF_DIRS bit selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI_SR is set, and the RFDF_RE bit in the DSPI_RSER register is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated

17.3.2.7 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 17.4.2.4, 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.

Figure 164. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode for cut1

Address: DSPI_BASE + 0x34 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CONT	CTAS			EOQ	CTCNT	0	0	PCS7	PCS6	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 165. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode for cut2/3

Address: DSPI_BASE + 0x34 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CONT	CTAS			EOQ	CTCNT	PUSHR_PE	PUSHR_PP	PCS7	PCS6	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Table 151. 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 PCS signals to remain asserted between transfers. See *Section 17.4.4.5, 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.

Table 151. DSPI_PUSHR field descriptions in master mode (continued)

Field	Descriptions
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 SPI_TCNT field in the DSPI_TCR register. The SPI_TCNT field is cleared before transmission of the current SPI frame begins. 0 Do not clear SPI_TCNT field in the DSPI_TCR 1 Clear SPI_TCNT field in the DSPI_TCR
PUSHR_PE (for cut2/3 only)	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.
PUSHR_PP (for cut2/3 only)	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 0–7. The PCS bits select which PCS signals will be asserted for the transfer. 0 Negate the PCS[x] signal 1 Assert the PCS[x] signal
TXDATA	Transmit Data. The TXDATA field holds SPI data to be transferred according to the associated SPI command.

Figure 166. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in slave mode

Address: DSPI_BASE + 0x34 Access:

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

Table 152. DSPI_PUSHR field descriptions in slave mode

Field	Descriptions
TXDATA	Transmit Data. The TXDATA field holds SPI data to be transferred.

17.3.2.8 DSPI POP RX FIFO Register (DSPI_POPR)

The DSPI_POPR provides means to read the RX FIFO. See *Section 17.4.2.5, 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.

Figure 167. DSPI POP RX FIFO Register (DSPI_POPR)

Address: DSPI_BASE + 0x38 Access:

	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

Table 153. DSPI_POPR field descriptions

Field	Description
0–31 RXDATA[0:31]	Received Data. The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the Pop Next Data Pointer.

17.3.2.9 DSPI Transmit FIFO Registers 0–4 (DSPI_TXFR0–DSPI_TXFR4)

These 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.

Figure 168. DSPI Transmit FIFO Register 0–4 (DSPI_TXFR0–DSPI_TXFR4)

Address: DSPI_BASE+0x3C–DSPI_BASE+0x4C Access:

	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

Table 154. DSPI_TXFR*n* Field Descriptions

Field	Description
0–15 TXCMD[0:15]/ 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 17.3.2.7, 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
16–31 TXDATA[16:31]	Transmit Data. The TXDATA field contains the SPI data to be shifted out.

17.3.2.10 DSPI Receive FIFO Registers 0–4 (DSPI_RXFR0–DSPI_RXFR4)

These 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.

Figure 169. DSPI Receive FIFO Registers 0–4 (DSPI_RXFR0–DSPI_RXFR4)

Address: DSPI_BASE + 0x7C–DSPI_BASE + 0x8C Access:

	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

Table 155. DSPI_RXFR*n* Field Descriptions

Field	Description
0–31 RXDATA[0:31]	Receive Data. The RXDATA field contains the received SPI data.

17.4 Functional description

The 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 and deserializing up to 32 Parallel Input/Output signals. All communications are done with SPI-like protocol.

The DSPI has one configuration:

- SPI Configuration in which the DSPI operates as a basic SPI or a queued SPI.

The DSPI_MCR[DCONF] field determines the DSPI Configuration. See *Table 141* for the DSPI configuration values.

The DSPI_CTAR0 - DSPI_CTAR3 registers hold clock and transfer attributes (see *Section 17.3.2.4, DSPI Clock and Transfer Attributes Registers 0–3 (DSPI_CTAR0–DSPI_CTAR3)*). The SPI configuration allows to select which CTAR to use on a frame by frame basis by setting a field in the SPI command. See *Section 17.3.2.4, DSPI Clock and Transfer Attributes Registers 0–3 (DSPI_CTAR0–DSPI_CTAR3)*, for information on the fields of the DSPI_CTAR registers.

Typical master to slave connections are shown in the *Figure 170*. 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.

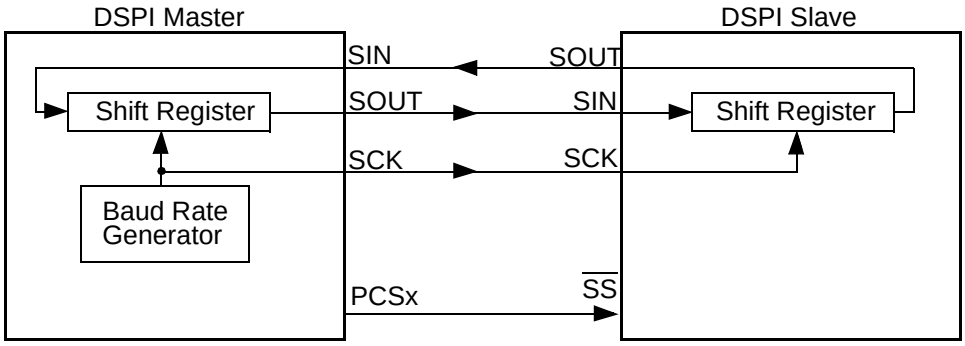


Figure 170. SPI Serial Protocol Overview

Generally more than one slave device can be connected to the DSPI master. Eight Peripheral Chip Select (PCS) 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 17.4.4, Transfer formats*. The transfer rate and delay settings are described in *Section 17.4.3, DSPI baud rate and clock delay generation*.

The shifting of data between FIFO and shift registers is more dependent upon the state of the internal FSM rather than the exact number of elapsed clock cycles. For example, TX-DATA is loaded into the shift register whenever the “next state” of FSM is PCS-to-SCK delay stage while the current state is IDLE, “After SCK delay,” “Delay after transfer,” or STALL.

17.4.1 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.

17.4.2 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 17.4.2.4, Transmit First In First Out (TX FIFO) Buffering Mechanism*, and *Section 17.4.2.5, Receive First In First Out (RX FIFO) Buffering Mechanism*. The interrupt and DMA request conditions are described in *Section 17.4.6, 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.

17.4.2.1 Master Mode

In SPI master mode the DSPI initiates the serial transfers by controlling the Serial Communications Clock (SCK) and the Peripheral Chip Select (PCS) 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 PCS signal to assert. The command field also contains various bits that help with queue management and transfer protocol. See *Section 17.3.2.7, 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.

17.4.2.2 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.

17.4.2.3 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.

17.4.2.4 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 1–5 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 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 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.

Because the PUSHR is a 32-bit register, any writes to PUSHR will transfer the all 32 bits of data from the write data bus to the register. Data byte strobes are ignored.

17.4.2.4.1 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 17.4.6.2, 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.

17.4.2.4.2 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 17.4.6.4, Transmit FIFO Underflow Interrupt Request*, for details.

The TFFF and TCF bits in the DSPI_SR are independent of each other. The TX FIFO is updated (and the TFFF field is updated) whenever the TX data is loaded into the shift register. The TCF bit is updated when all of the TX data is shifted out.

17.4.2.5 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 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 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.

17.4.2.5.1 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.

17.4.2.5.2 Draining the RX FIFO

Host CPU or a DMA can remove (pop) entries from the RX FIFO by reading the 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.

17.4.3 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 171* shows conceptually how the SCK signal is generated.

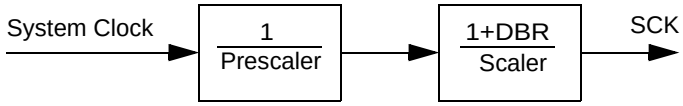


Figure 171. Communications clock prescalers and scalers

17.4.3.1 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_CTARs (see *Section 17.3.2.4, DSPI Clock and Transfer Attributes Registers 0–3 (DSPI_CTAR0–DSPI_CTAR3)*) select the frequency of SCK by the formula in the BR field description. *Table 156* shows an example of how to compute the baud rate.

Table 156. 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

17.4.3.2 PCS to SCK Delay (t_{CSC})

The PCS to SCK delay is the length of time from assertion of the PCS signal to the first SCK edge. See *Figure 173* for an illustration of the PCS to SCK delay. The PCSSCK and CSSCK fields in the DSPI_CTAR*x* registers select the PCS to SCK delay by the formula in the CSSCK field description (see *Section 17.3.2.4, DSPI Clock and Transfer Attributes Registers 0–3 (DSPI_CTAR0–DSPI_CTAR3)*). *Table 157* shows an example of how to compute the PCS to SCK delay.

Table 157. PCS to SCK Delay Computation Example

f_{sys}	PCSSCK	Prescaler	CSSCK	Scaler	PCS to SCK Delay
100 MHz	0b01	3	0b0100	32	0.96 µs

17.4.3.3 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 173* and *Figure 174* 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 (see *Section 17.3.2.4, DSPI Clock and Transfer Attributes Registers 0–3 (DSPI_CTAR0–DSPI_CTAR3)*). *Table 158* shows an example of how to compute the After SCK delay.

Table 158. After SCK Delay Computation Example

f_{sys}	PASC	Prescaler	ASC	Scaler	After SCK Delay
100 MHz	0b01	3	0b0100	32	0.96 µs

17.4.3.4 Delay after Transfer (t_{DT})

The Delay after Transfer is the minimum time between negation of the PCS signal for a frame and the assertion of the PCS signal for the next frame. See *Figure 173* 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 (see *Section 17.3.2.4, DSPI Clock and Transfer Attributes Registers 0–3 (DSPI_CTAR0–DSPI_CTAR3)*). *Table 159* shows an example of how to compute the Delay after Transfer.

Table 159. 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 7*. When in continuous clock mode the delay is fixed at 1 SCK period.

17.4.3.5 Peripheral Chip Select Strobe Enable ($\overline{PCSS}$)

The $\overline{PCSS}$ signal provides a delay to allow the PCS 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, $\overline{PCSS}$ provides a signal for an external demultiplexer to decode the PCS[0] - PCS[4] and PCS[6] -PCS[7] signals into as many as 128 glitch-free PCS signals. *Figure 172* shows the timing of the $\overline{PCSS}$ signal relative to PCS signals.

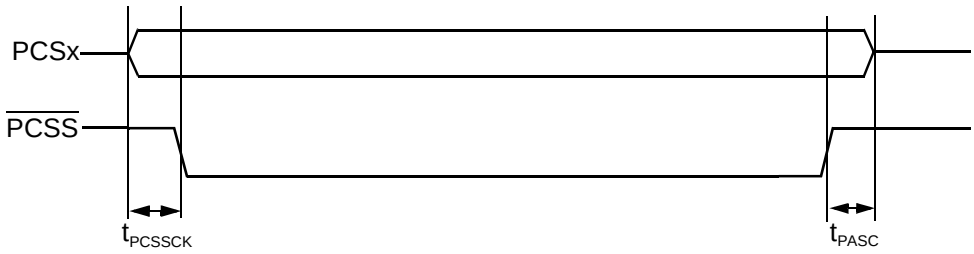


Figure 172. Peripheral Chip Select Strobe Timing

The delay between the assertion of the PCS signals and the assertion of $\overline{PCSS}$ is selected by the PCSSCK field in the DSPI_CTAR based on the following formula:

Equation 9 $$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between $\overline{PCSS}$ negation and PCS negation is selected by the PASC field in the DSPI_CTAR based on the following formula:

Equation 10 $$t_{PASC} = \frac{1}{f_{SYS}} \times PASC$$

Table 160 shows an example of how to compute the t_{pcssck} delay.

Table 160. Peripheral Chip Select Strobe Assert Computation Example

f_{sys}	PCSSCK	Prescaler	Delay before Transfer
100 MHz	0b11	7	70.0 ns

Table 161 shows an example of how to compute the t_{pasc} delay.

Table 161. Peripheral Chip Select Strobe Negate Computation Example

f_{sys}	PASC	Prescaler	Delay after Transfer
100 MHz	0b11	7	70.0 ns

The $\overline{\text{PCSS}}$ signal is not supported when Continuous Serial Communication SCK is enabled.

17.4.4 Transfer formats

The SPI serial communication is controlled by the Serial Communications Clock (SCK) signal and the PCS signals. The SCK signal provided by the master device synchronizes shifting and sampling of the data on the SIN and SOUT pins. The PCS 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 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.

The DSPI provides the option of keeping the PCS signals asserted between frames. See *Section 17.4.4.5, Continuous selection format*, for details.

17.4.4.1 Classic SPI Transfer Format (CPHA = 0)

The transfer format shown in *Figure 173* 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.

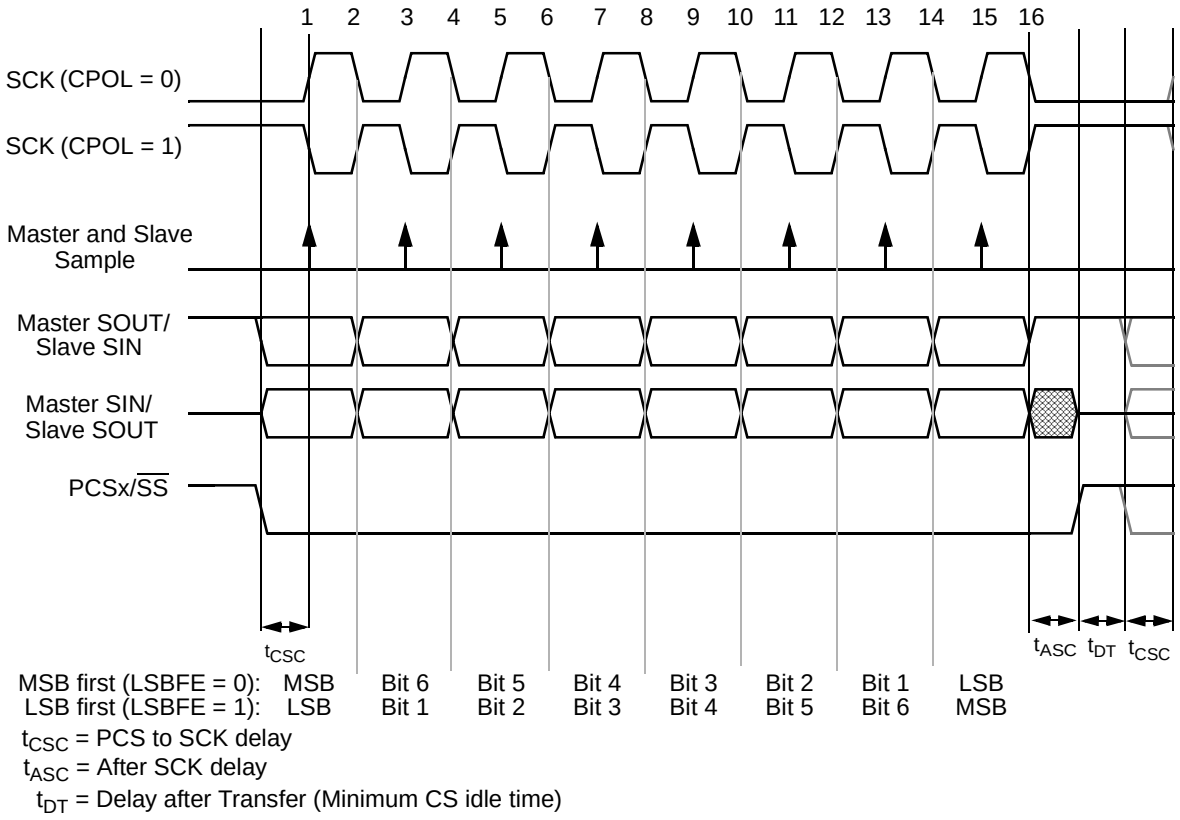
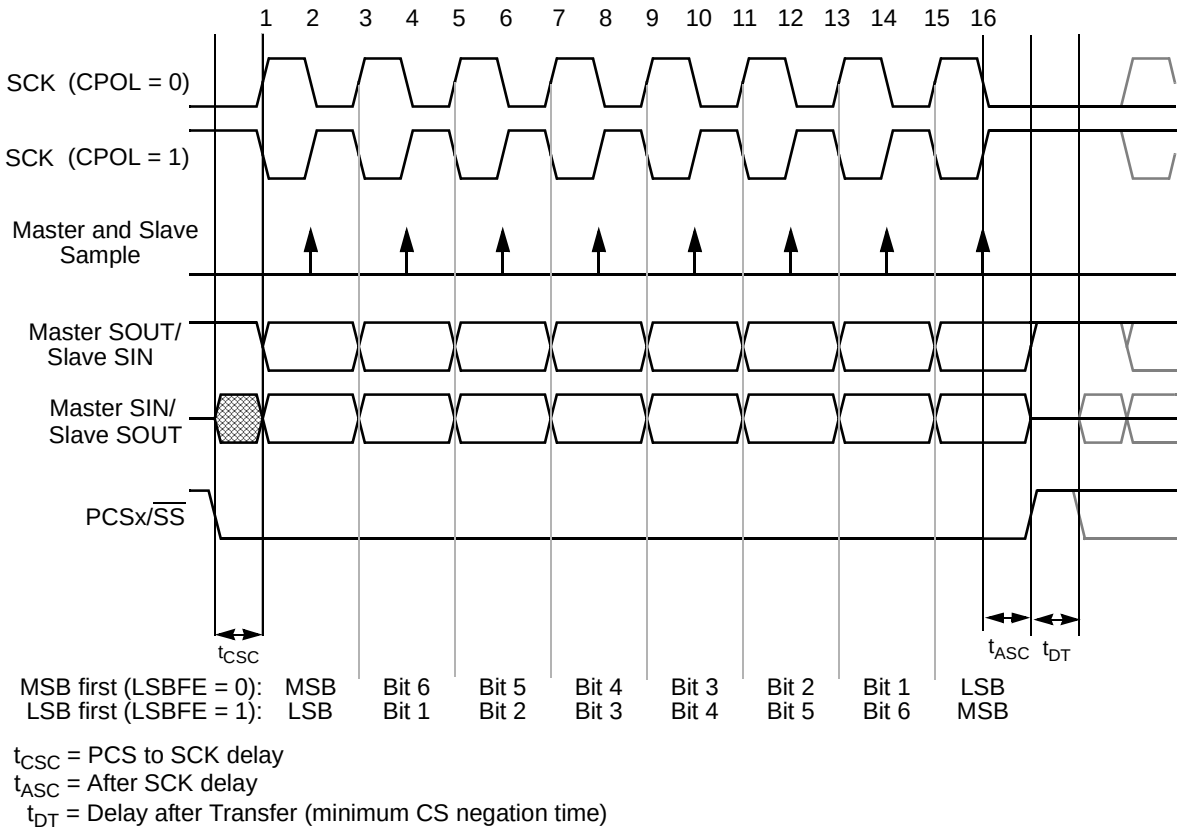


Figure 173. DSPI Transfer Timing Diagram (MTFE=0, CPHA=0, FMSZ=8)

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 PCS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

17.4.4.2 Classic SPI Transfer Format (CPHA = 1)

This transfer format shown in *Figure 174* 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} = PCS to SCK delay
t_{ASC} = After SCK delay
t_{DT} = Delay after Transfer (minimum CS negation time)

Figure 174. DSPI Transfer Timing Diagram (MTFE=0, CPHA=1, FMSZ=8)

The master initiates the transfer by asserting the PCS 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 PCS signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

17.4.4.3 Modified SPI Transfer Format (MTFE = 1, CPHA = 0)

In this Modified Transfer Format both the master and the slave sample later in the SCK period than in Classic SPI mode to allow 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 PCS signal. After the PCS 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 141* 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} - PCS to SCK assertion delay
- T_{acs} - After SCK PCS negation 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 175 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.

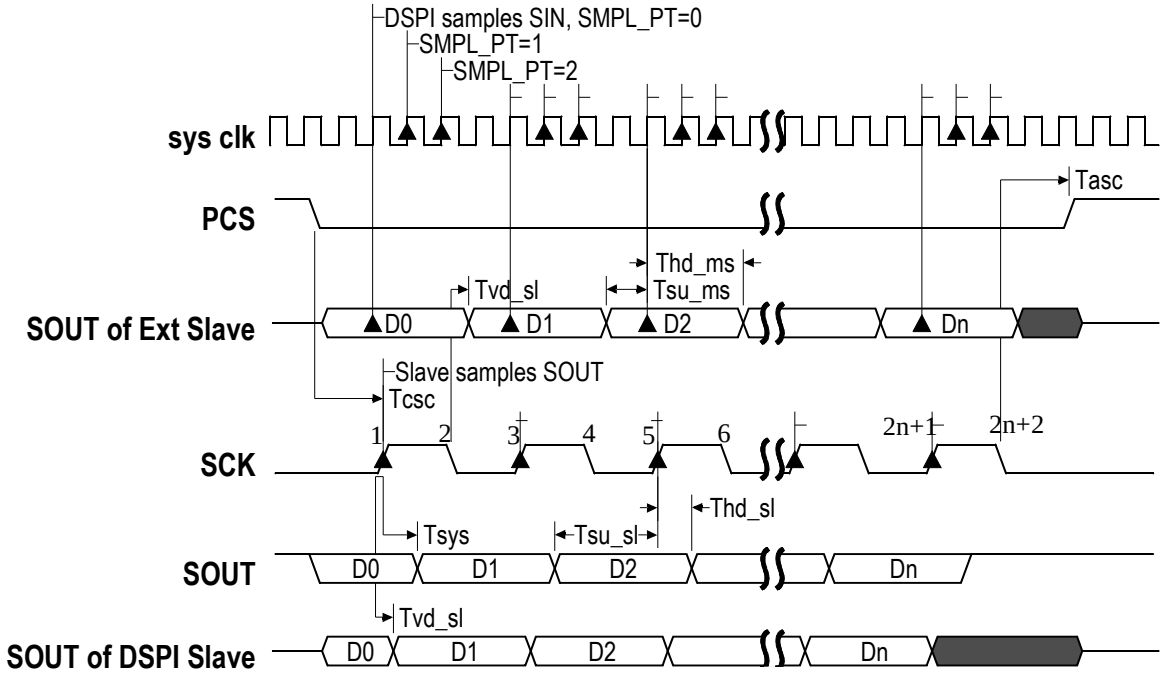


Figure 175. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/4$)

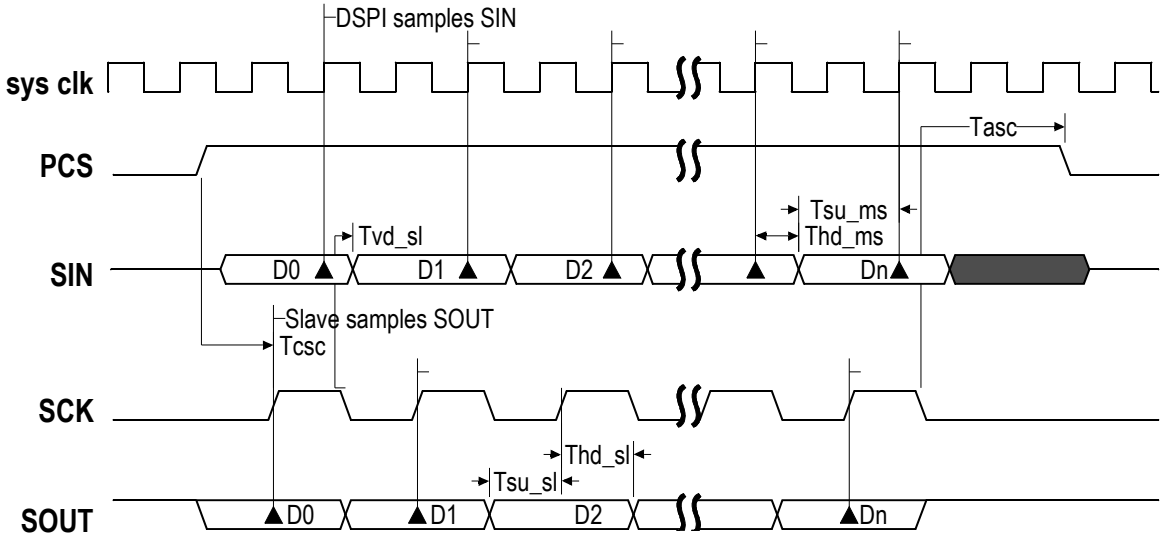


Figure 176. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/2$)

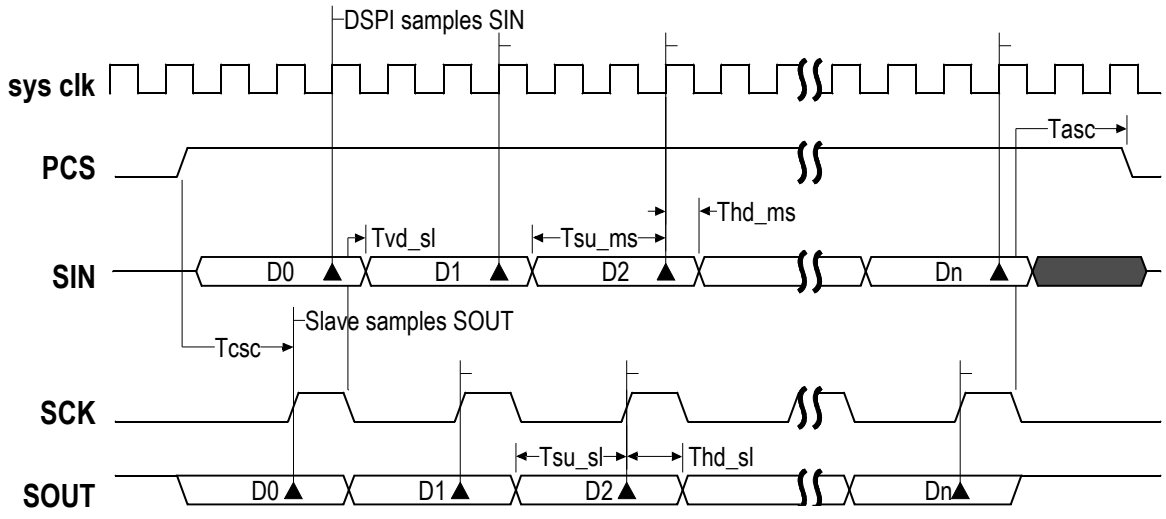


Figure 177. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/3$)

17.4.4.4 Modified SPI Transfer Format (MTFE = 1, CPHA = 1)

Figure 178 - (180) show 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 PCS signal to the slave device. After the PCS 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 PCS delay must be greater or equal to half of the SCK period.**

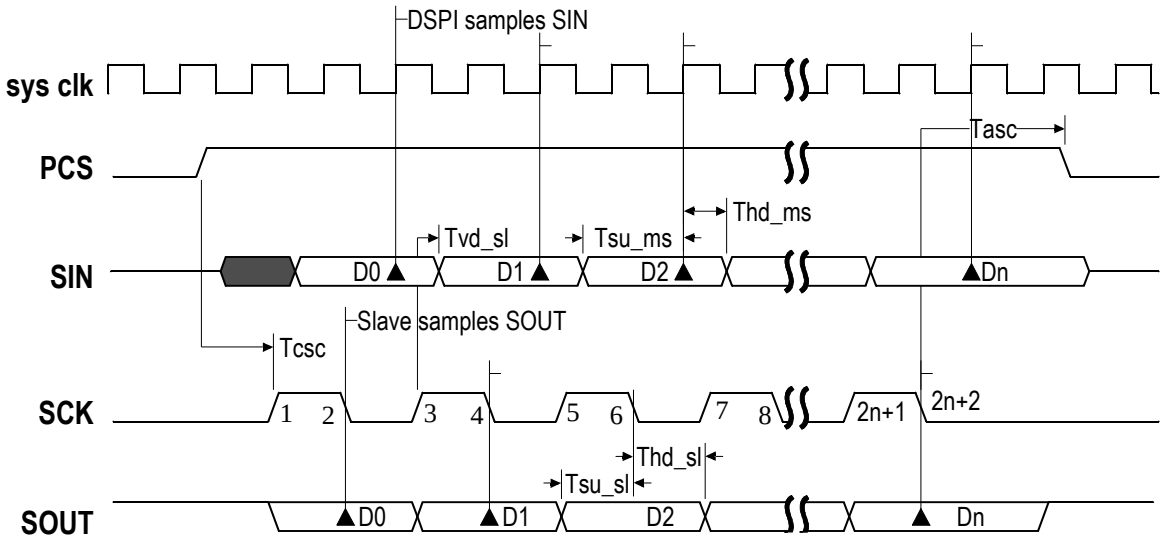


Figure 178. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/2$)

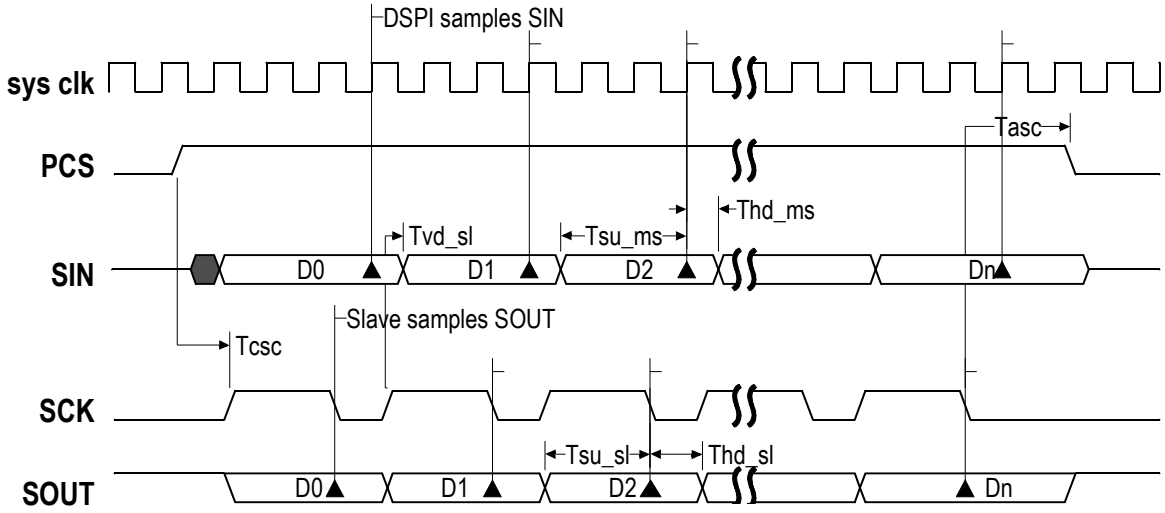


Figure 179. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/3$)

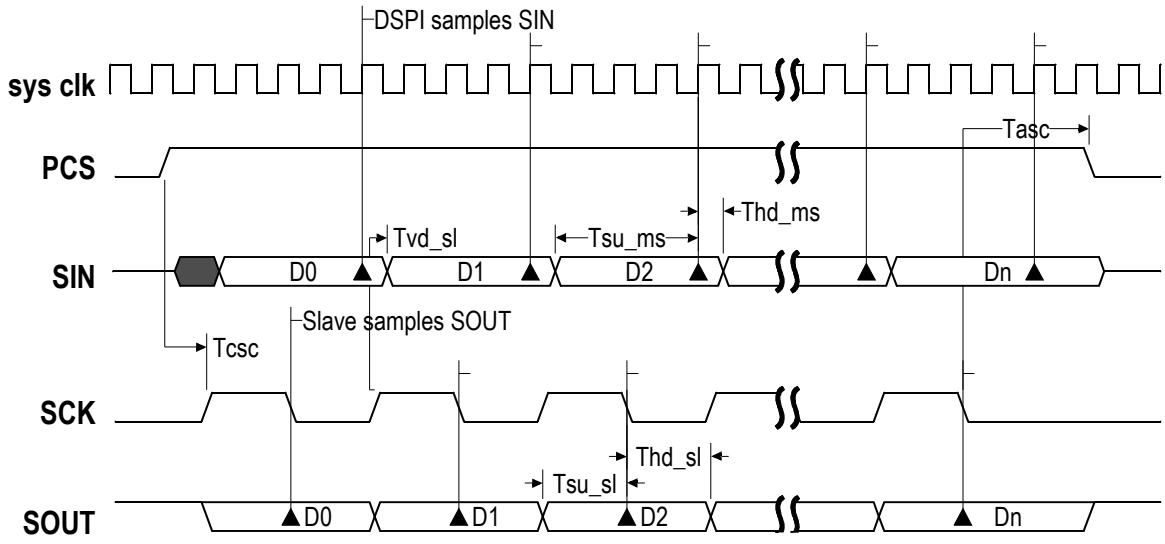


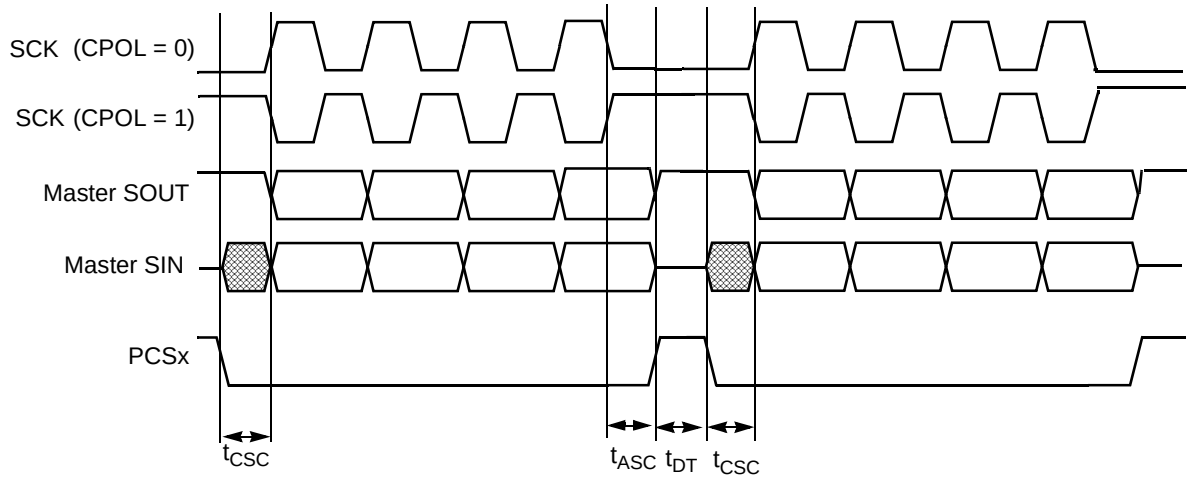
Figure 180. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/4$)

17.4.4.5 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.

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 181* shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 0.

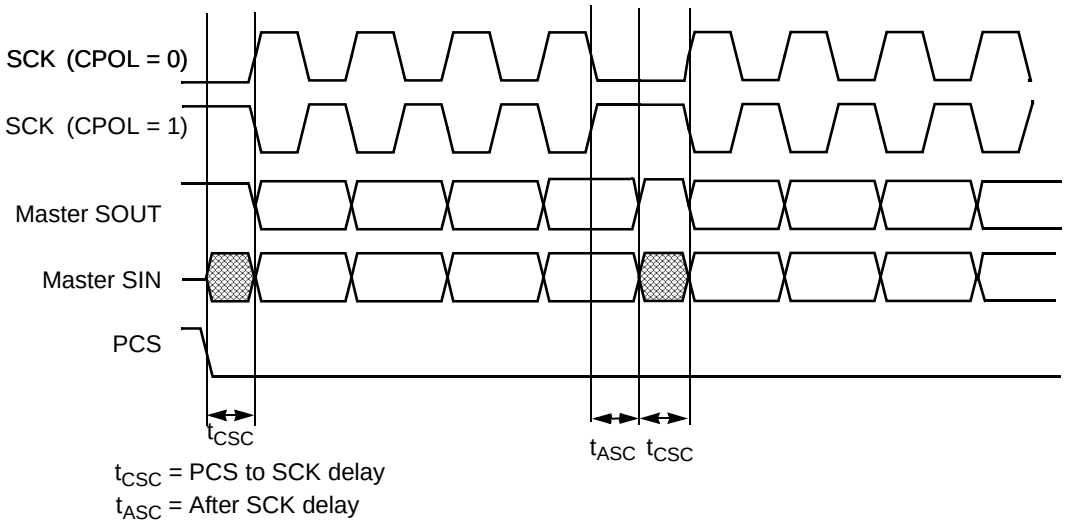
Figure 181. Example of Non-Continuous Format (CPHA=1, CONT=0)



t_{CSC} = PCS to SCK delay
t_{ASC} = After SCK delay
t_{DT} = Delay after Transfer (minimum CS negation time)

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 182* shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 1.

Figure 182. Example of Continuous Transfer (CPHA=1, CONT=1)



You must fill the TXFIFO with the number of entries that will be concatenated together under one PCS assertion for both master and slave before the TXFIFO becomes empty. For example: While transmitting in master mode, you should ensure that the last entry in the TXFIFO, after which TXFIFO becomes empty, must have the CONT bit in command frame as deasserted (i.e. CONT bit = 0).While operating in slave mode, it should be ensured that when the last-entry in the TXFIFO is completely transmited (i.e. the corresponding TCF flag is asserted and TXFIFO is empty) the slave should be de-selected for any further serial communication; else an underflow error occurs.

The TX FIFO must be cleared before initiating any SPI configuration transfer.

When the DSPI is in SPI configuration, CTAR0 must be used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame must be CTAR0.

When the DSPI is in DSI configuration, the CTAR specified by the DSICTAS field must be used at all times.

When the DSPI is in CSI configuration, the CTAR selected by the DSICTAS field must be used initially. At the start of a SPI frame transfer, the CTAR specified by the CTAS value (which is CTAR0) for the frame shall be used. At the start of a DSI frame transfer, the CTAR specified by the DSICTAS field must be used.

Note: When in Continuous SCK mode, for the SPI transfer CTAR0 must always be used, and the TX-FIFO must be clear using the MCR.CLR_TXF field before initiating transfer.

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.

17.4.5 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:

- CTAR0 is used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame is used.
- 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 PCS to SCK delay and the Delay after Transfer (t_{DT}) is fixed to one SCK cycle. *Figure 183* shows timing diagram for Continuous SCK format with Continuous Selection disabled.

Note: When in Continuous SCK mode, for the SPI transfer CTAR0 must always be used, and the TX-FIFO must be clear using the MCR.CLR_TXF field before initiating transfer.

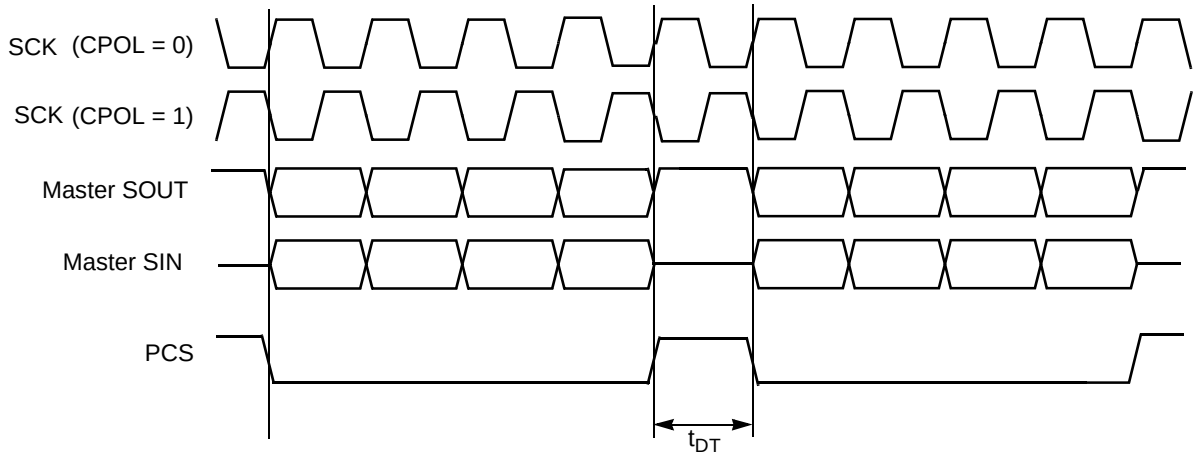


Figure 183. Continuous SCK Timing Diagram (CONT=0)

If the CONT bit in the TX FIFO entry is set, PCS remains asserted between the transfers. Under certain conditions, SCK can continue with PCS 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 17.4.1, Start and Stop of DSPI Transfers*).
- Continuous SCK with CONT bit set and entering Stop mode or Module Disable mode.

Figure 184 shows timing diagram for Continuous SCK format with Continuous Selection enabled.

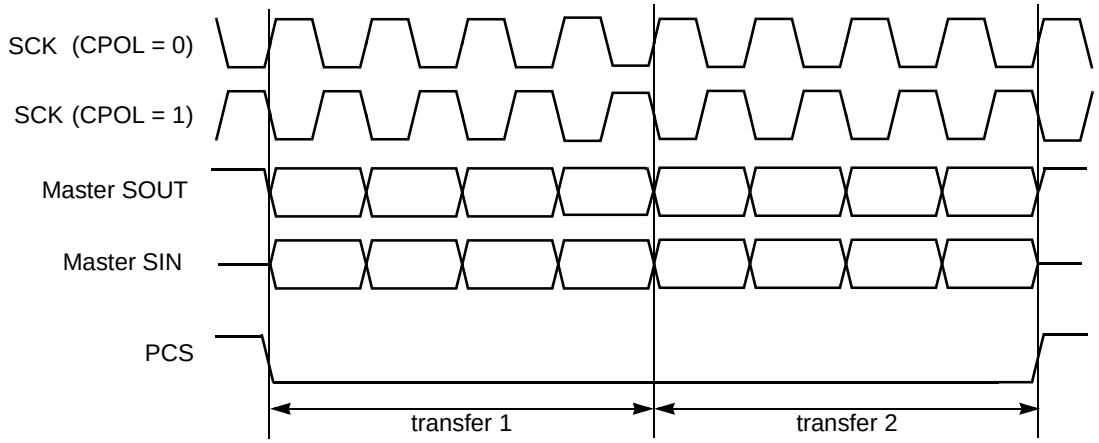


Figure 184. Continuous SCK Timing Diagram (CONT=1)

17.4.6 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 162* lists these conditions.

Table 162. 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	

Each condition has a flag bit in the DSPI Status Register (DSPI_SR) and an Request Enable bit in the 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 TFFF_DIRS and RFDF_DIRS 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.

17.4.6.1 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 EOQF_RE bit in the DSPI_RSER is set.

17.4.6.2 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 TFFF_RE bit in the DSPI_RSER is set. The TFFF_DIRS bit in the DSPI_RSER selects whether a DMA request or an interrupt request is generated.

17.4.6.3 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 TCF_RE bit is set in the DSPI_RSER.

17.4.6.4 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 TFUF_RE bit in the DSPI_RSER is set, an interrupt request is generated.

17.4.6.5 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 RFDF_RE bit in the DSPI_RSER is set. The RFDF_DIRS bit in the DSPI_RSER selects whether a DMA request or an interrupt request is generated.

17.4.6.6 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 RFOF_RE bit in the DSPI_RSER 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.

17.4.7 Power Saving Features

The DSPI supports two power-saving strategies:

- External Stop mode
- Module Disable mode - Clock gating of non-memory mapped logic

17.4.7.1 .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.

17.4.7.2 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.

17.5 Initialization/Application Information

17.5.1 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 SPI_TCNT field in the DSPI_TCR.
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

17.5.2 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].

17.5.3 Baud Rate Settings

Table 163 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 163. 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

17.5.4 Delay Settings

Table 164 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.

Table 164. 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

17.5.5 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 185* 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 17.4.2.4, Transmit First In First Out (TX FIFO) Buffering Mechanism*, and *Section 17.4.2.5, Receive First In First Out (RX FIFO) Buffering Mechanism*, for details on the FIFO operation.

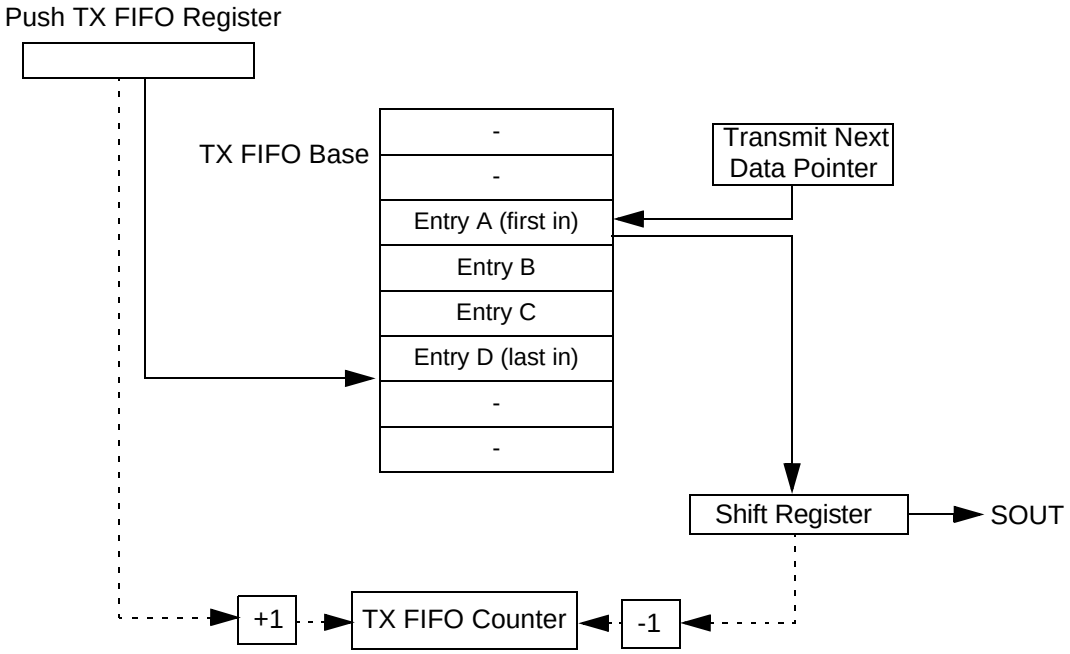


Figure 185. TX FIFO Pointers and Counter

17.5.5.1 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 11 $$\text{First-in Entry Address} = \text{TX FIFO Base} + (4 \times \text{TXNXTPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

Equation 12

$$\text{Last-in Entry address} = \text{TX FIFO Base} + 4 \times (\text{TXCTR} + \text{TXNXTPTR} - 1) \bmod (\text{TXFIFOdepth})$$

TX FIFO Base - Base address of TX FIFO

TXCTR - TX FIFO Counter

TXNXTPTR - Transmit Next Pointer

TX FIFO Depth - Transmit FIFO depth, implementation specific

17.5.5.2 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 13 $$\text{First-in Entry Address} = \text{RX FIFO Base} + (4 \times \text{POPNXTPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

Equation 14

$$\text{Last-in Entry address} = \text{RX FIFO Base} + 4 \times (\text{RXCTR} + \text{POPNXTPTR} - 1) \bmod (\text{RXFIFOdepth})$$

RX FIFO Base - Base address of RX FIFO

RXCTR - RX FIFO counter

POPNXTPTR - Pop Next Pointer

RX FIFO Depth - Receive FIFO depth, implementation specific

18 e200z4d Core Complex Overview

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 186*)
- A summary of the feature set for this core (see *Section 18.2, Features*)
  - A description of the execution units (see *Section 18.2.1, Execution Unit Features*)
  - A description of the memory management architecture (see *Section 18.2.3, Memory Management Unit Features*)
  - High-level details of the external core compex interface (see *Section 18.2.4, Exernal core complex interface features*)
  - High-level details of the Nexus 3+ features (see *Section 18.2.5, Nexus 3+ Features*)
- A summary of the programming model for this core (see *Section 18.3, Programming model*)
  - An overview of the register set (see *Section 18.3.1, Register set*)
  - An overview of the instruction set (see *Section 18.3.2, Instruction set)*
  - An overview of interrupts and exception handling (see *Section 18.3.3, Interrupts and Exception Handling*)
- A summary of instruction pipeline and flow (see *Section 18.4, Microarchitecture summary*)

18.1 Overview

The e200z4d processor family is a set of CPU cores that implement low-cost versions of Power Architecture technology. The e200z4d core is a dual-issue, 32-bit design with 64-bit general-purpose registers (GPRs). The e200z4d integrates a CPU core, a memory management unit (MMU), a 4-Kbyte instruction cache, and a Nexus Class 3+ real-time debug unit. Separate instruction and data AHB 2.v6 system interfaces are provided.

The e200z4d is compliant with the Power Architecture instruction set architecture (ISA). It does not support Power Architecture ISA floating-point instructions in hardware, but traps them so they can be emulated by software.

Instructions of the embedded floating-point category are provided to support real-time single-precision embedded numerics operations using the general-purpose registers.

Instructions of the signal processing extension (SPE) category are 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 category. 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 ISA embedded category instruction set, the core also implements the variable-length encoding category (VLE), which provides improved code density. See the *EREF* and supplementary VLE Programming Environments Manual (*VLEPEM)* for more information about the VLE extension.

The 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 cycle. 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.

Throughout the remainder of this document, the core is referred to as the “e200z4d” when speaking of e200z4d-specific implementations, the “e200z4xx” when speaking of a specific variety of e200z4 core, or “e200” when referring to the whole e200 family.

Figure 186 shows the block diagram for the device.

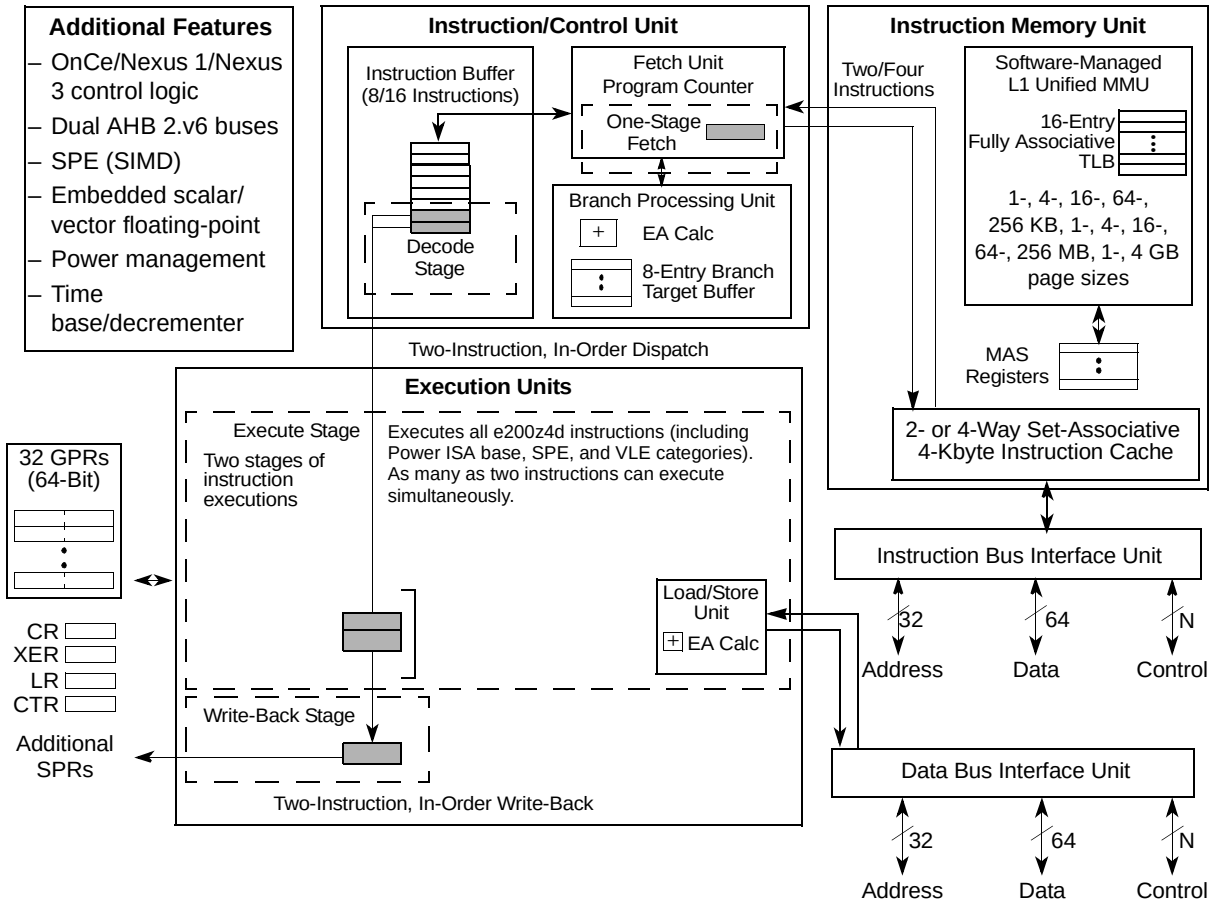


Figure 186. e200z4d block diagram

18.2 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 unit, version 1.1 supporting SIMD fixed-point operations using the 64-bit general-purpose register file
- Embedded floating-point (FPU) unit, version 2 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: doze, nap, sleep, wait
  - Dynamic power management of execution units, cache, and MMU

See the following sections for more details about specific units.

18.2.1 Execution Unit Features

The following subsections describes the execution units' main features.

18.2.1.1 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.

18.2.1.2 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 ≤ 14 clock cycles with minimized execution timing
- Pipelined 32 × 32 hardware multiplier array supports 32 × 32→32 multiply with 2 clock latency, 1 clock throughput

18.2.1.3 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
- Two-cycle load latency
- Big- and little-endian support
- Misaligned access support

18.2.2 L1 Cache Features

The L1 cache features the following:

- 4 KB, 2- or 4-way configurable set-associative instruction cache
- 64-bit data, 32-bit address bus plus attributes and control
- 32-byte line size
- Cache line locking
- Way allocation
- Tag and data parity or multi-bit EDC protection with correction/auto-invalidation capability
- Virtually indexed, physically tagged
- Pseudo round-robin replacement algorithm
- Line-fill buffer
- Hit under fill

18.2.3 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
- Software managed by **tlbre**, **tlbwe**, **tlbsx**, **tlbsync**, and **tlbivax** instructions
- Freescale EIS MMU architecture compliant
- Support for external control of entry matching for a subset of TID values to support non-intrusive runtime mapping modifications

18.2.4 Exernal 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
- Separate unidirectional 64-bit read and write data buses
- Support for HCLK running at a slower rate than CPU clock

18.2.5 Nexus 3+ Features

The Nexus 3+ module provides real-time development capabilities for e200z4d processors in compliance with the IEEE-ISTO 5001™-2003. 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.

18.3 Programming model

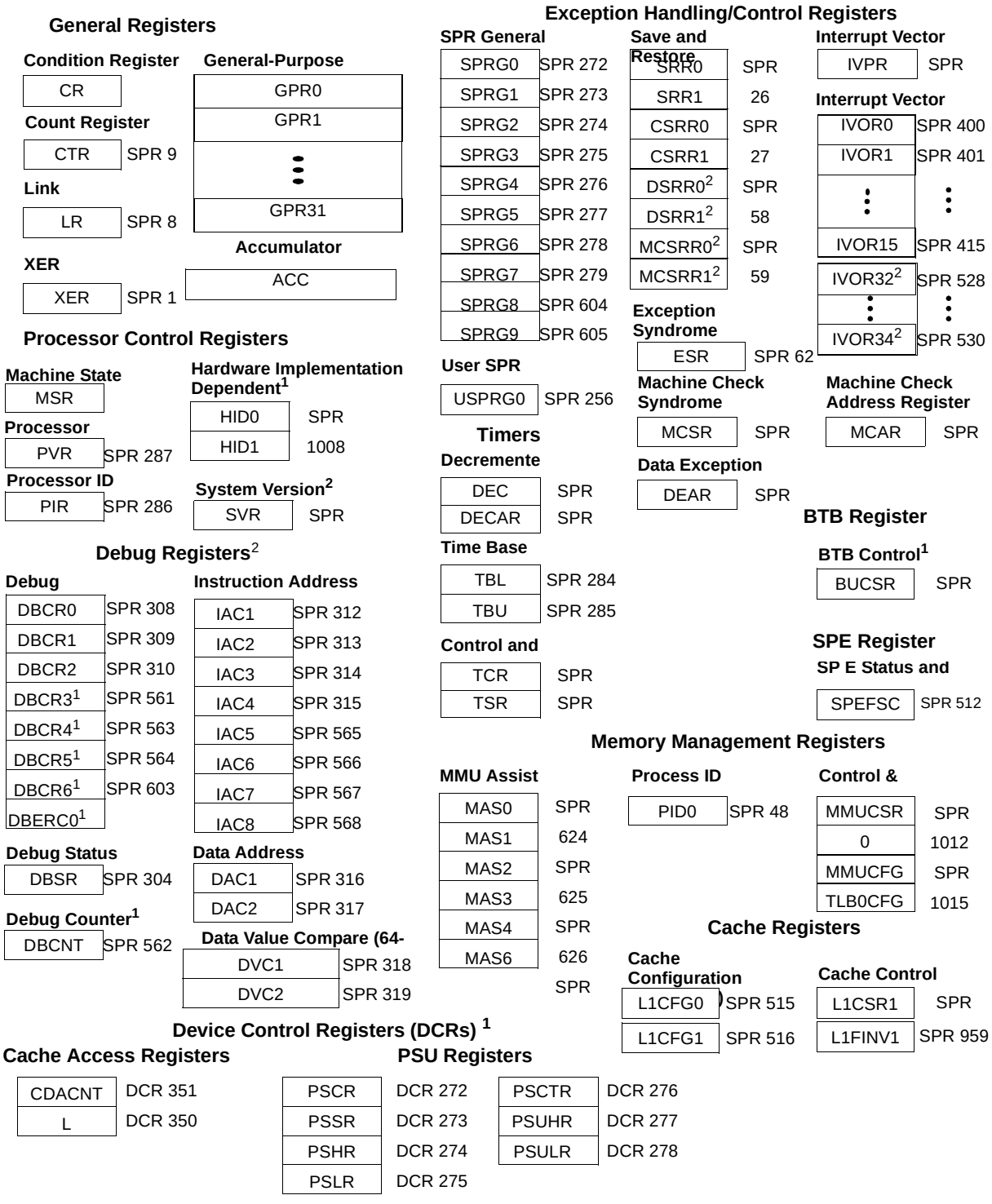
This section describes the register model, instruction model, and the interrupt model as they are defined by the Power ISA, Freescale EIS, and the e200z4d implementation.

18.3.1 Register set

Figure 187 and *Figure 188* 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 187 shows the registers that can be accessed by supervisor-level software. User-level software can access only those registers listed in *Figure 188*.

Figure 187. 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 188 shows the user-mode special-purpose registers.

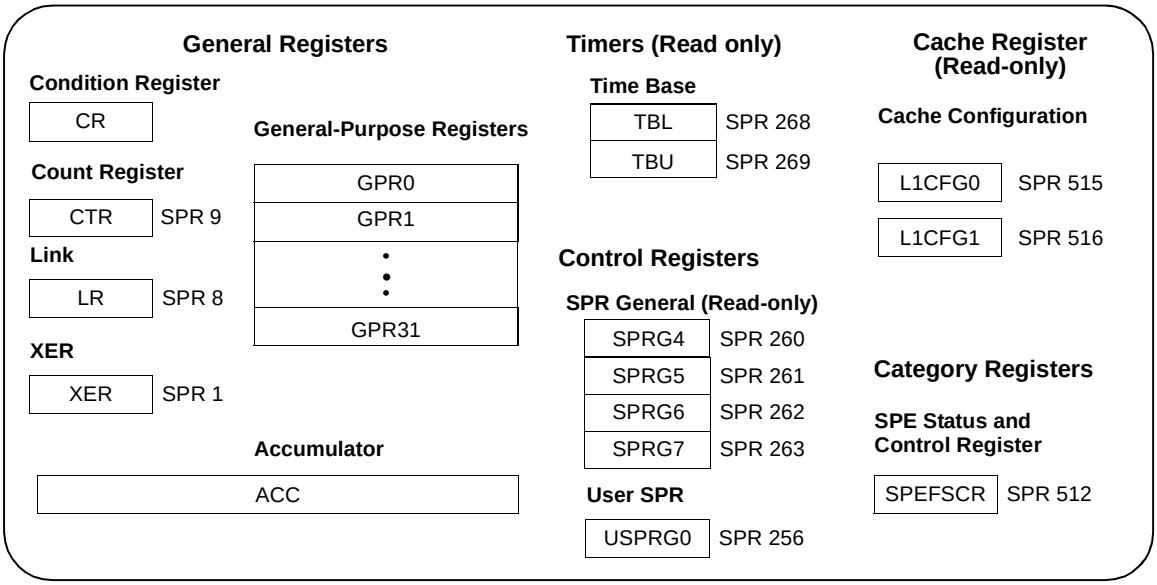


Figure 188. e200z4d User Mode Programmer's Model SPRs

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.

18.3.1.1 Processor Version Register (PVR)

The PVR contains the processor version identification for the CPU core.

Figure 189. Processor Version Register (PVR)

	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	1	0	0	0	0	0	TYPE						VERSION				0	0	0	0	MINREV				MAJREV				0	0	0	1
W																																
Reset	1	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 165. PVR field descriptions

Field	Description
TYPE	Processor type 010101 = e200z4d core
VERSION	Processor version 0100 e200z4d core on this device
MINREV	Minor revision number
MAJREV	Major revision number

18.3.1.2 Processor ID Register (PIR)

The processor ID for each of the two CPU cores is contained in its own Processor ID Register (PIR). The contents of the PIR are a reflection of hardware input signals to the core following reset. This register may be written by software to modify the default reset value. This register value can be used by the application software to determine the core actually running the software.

Figure 190. Processor ID Register (PIR)

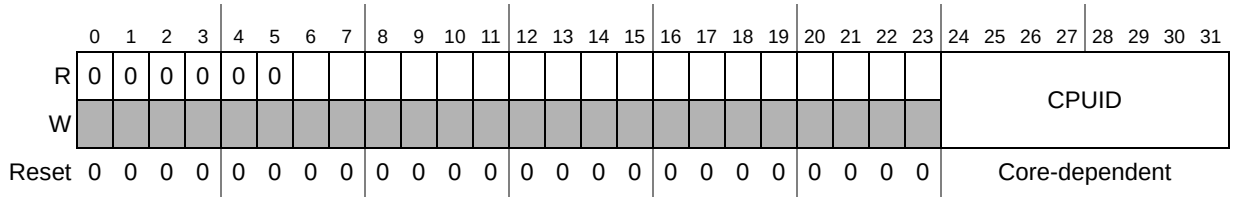


Table 166. PIR field descriptions

Field	Description
CPUID	Processor ID

18.3.1.3 System Version Register (SVR)

The SVR contains system version information for this device.

Figure 191. System Version Register (SVR)

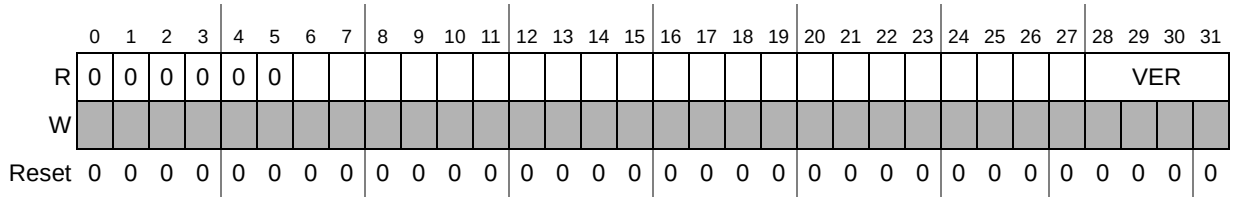


Table 167. SVR field descriptions

Field	Description
VER	Device version

18.3.2 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

18.3.3 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 168.*

Table 168. 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.

Table 168. Interrupt Registers (continued)

Register	Description
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.
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 169* 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] || 0b0 define the address of an interrupt-processing routine. See *Table 169* 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] || 0b0). 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 169 lists IVOR registers implemented on the e200z4d and the associated interrupts.

Table 169. 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

Table 169. Exceptions and conditions (continued)

IVOR*n*	Interrupt Type		IVOR*n*	Interrupt Type
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]*] || 0b0.
2. Autovectored external and critical input interrupts use this IVOR. Vectored interrupts supply an interrupt vector offset directly.

18.4 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 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.

18.5 Availability of detailed documentation

Detailed documentation of the e200z4d core is provided in a separate core reference manual (CRM). This CRM is available online at http://www.st.com.

19 eDMA Channel Mux (DMA_MUX)

19.1 Introduction

19.1.1 Overview

The DMA_MUX allows to route 27 DMA peripheral sources (called slots) to 16 DMA channels. This is illustrated in *Figure 192*.

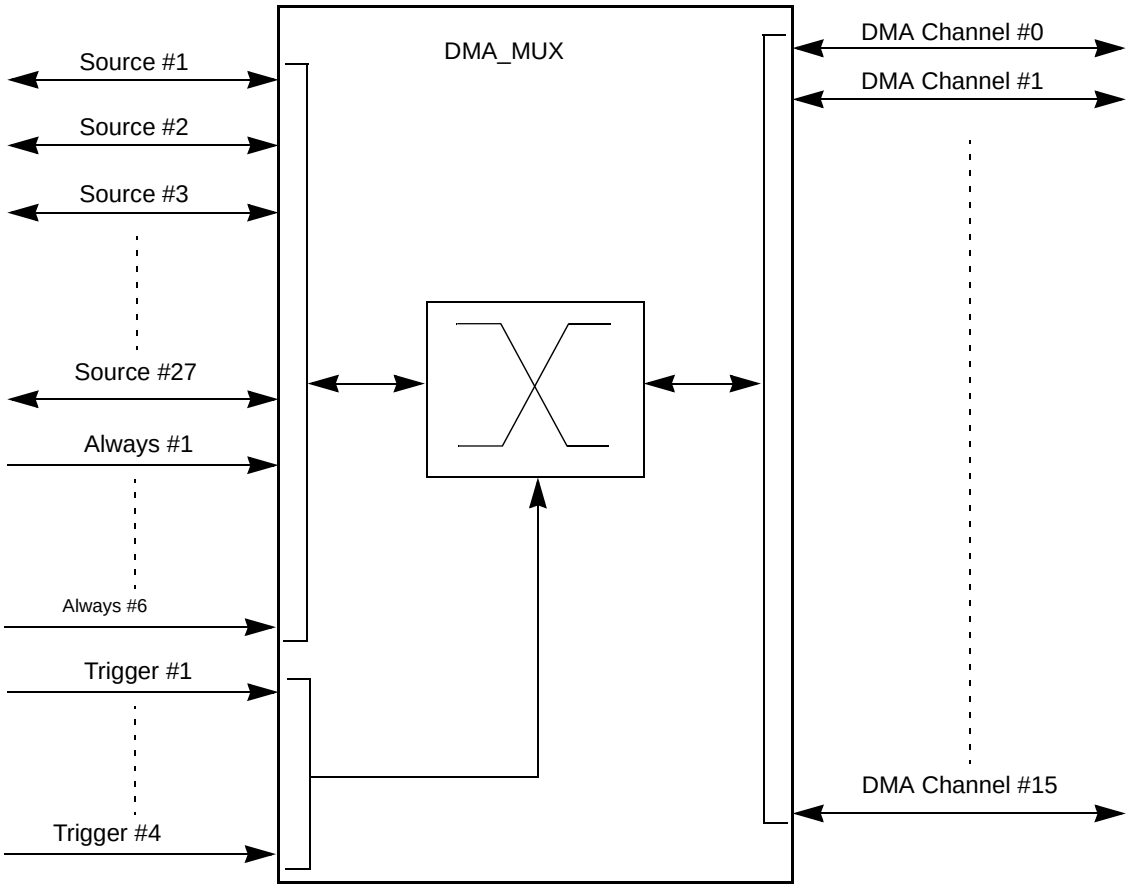


Figure 192. DMA_MUX block diagram

19.1.2 Features

The DMA_MUX provides these features:

- 27 peripheral slots (plus 6 always-on slots) can be routed to 16 channels
- 16 independently selectable DMA channels routers
  - the first 4 channels additionally provide a trigger functionality
- Each channel router can be assigned to one of 27 possible peripheral DMA slots or to one of the 6 always-on slots.

19.1.3 Modes of operation

The following operation modes are available:

- Disabled Mode

  In this mode, the DMA channel is disabled. Since disabling and enabling of DMA channels is done primarily via the DMA configuration registers, this mode is used mainly as the reset state for a DMA channel in the DMA_MUX. It may also be used to temporarily suspend a DMA channel while reconfiguration of the system takes place (e.g. changing the period of a DMA trigger).

- Normal Mode

  In this mode, a DMA source (such as DSPI transmit or DSPI receive for example) is routed directly to the specified DMA channel. The operation of the DMA_MUX in this mode is completely transparent to the system.

- Periodic Trigger Mode

  In this mode, a DMA source may only request a DMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. Configuration of the period is done in the registers of the Periodic Interrupt Timer (PIT). This mode is only available for channels 0–3.

19.2 External signal description

19.2.1 Overview

The DMA_MUX has no external pins.

19.3 Memory map and register definition

This section provides a detailed description of all memory-mapped registers in the DMA_MUX.

Table 170 shows the memory map for the DMA_MUX. All addresses are offsets; the absolute address may be computed by adding the specified offset to the base address of the DMA_MUX.

Table 170. DMA_MUX memory map

Address	Use	Access
Base + 0x00	Channel #0 Configuration (CHCONFIG0)	R/W
Base + 0x01	Channel #1 Configuration (CHCONFIG1)	R/W
..	..	..
Base + *#n-1*	Channel #*n* Configuration (CHCONFIG#*n-1*)(1)	R/W

1. In the table *n* refers to the *number of channels - 1*

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.

19.3.1 Register descriptions

The following memory-mapped registers are available in the DMA_MUX.

19.3.1.1 Channel Configuration Registers

Each of the DMA channels can be independently enabled/disabled and associated with one of the DMA slots (peripheral slots or always-on slots) in the system.

Figure 193. Channel Configuration Registers (CHCONFIG#*n*)

Address: Base + #*n* Access: User read/write

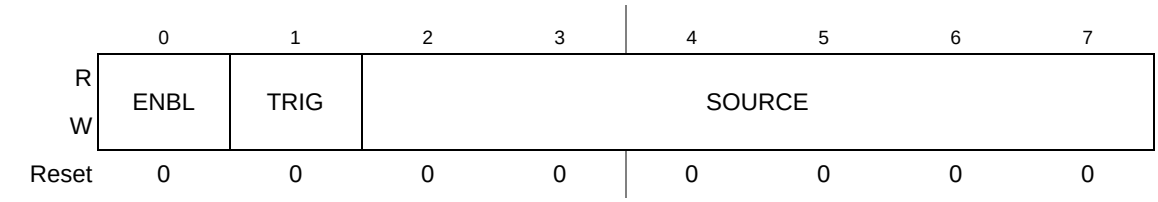


Table 171. CHCONFIG*xx* Field Descriptions

Field	Description
7 ENBL	DMA Channel Enable. ENBL enables the DMA Channel 0 DMA channel is disabled. This mode is primarily used during configuration of the DMA_MUX. The DMA has separate channel enables/disables, which should be used to disable or re-configure a DMA channel. 1 DMA channel is enabled
6 TRIG	DMA Channel Trigger Enable (for triggered channels only). TRIG enables the periodic trigger capability for the DMA Channel 0 Triggering is disabled. If triggering is disabled, and the ENBL bit is set, the DMA Channel will simply route the specified source to the DMA channel. 1 Triggering is enabled
5–0 SOURCE	DMA Channel Source (slot). SOURCE specifies which DMA source, if any, is routed to a particular DMA channel (see *Section 19.4, DMA_MUX request source slot mapping*).

Table 172. Channel and Trigger Enabling

ENBL	TRIG	Function	Mode
0	X	DMA Channel is disabled	Disabled Mode
1	0	DMA Channel is enabled with no triggering (transparent)	Normal Mode
1	1	DMA Channel is enabled with triggering	Periodic Trigger Mode

Note: Setting multiple CHCONFIG registers with the same Source value will result in unpredictable behavior.

Note: Before changing the trigger or source settings a DMA channel must be disabled via the CHCONFIG[#n].ENBL bit.

19.4 DMA_MUX request source slot mapping

Table 173 defines the mapping of the DMA_MUX source slots to the interrupt request sources on the device. *Table 173* is valid for both instantiations of the DMA_MUX module.

Table 173. DMA_MUX source slot mapping

DMA_MUX source slot #	Source module	Source resource
1	DSPI_0	DSPI_TFFF
2	DSPI_0	DSPI_RFDF
3	DSPI_1	DSPI_TFFF
4	DSPI_1	DSPI_RFDF
5	DSPI_2	DSPI_TFFF
6	DSPI_2	DSPI_RFDF
7	CTU	CTU
8	CTU	FIFO1
9	CTU	FIFO2
10	CTU	FIFO3
11	CTU	FIFO4
12	FlexPWM_0	comp_val
13	FlexPWM_0	capt
14	eTimer_0	Channel 0
15	eTimer_0	Channel 1
16	eTimer_1	Channel 0
17	eTimer_1	Channel 1
18	eTimer_2	Channel 0
19	eTimer_2	Channel 1
20	ADC_0	DMA
21	ADC_1	DMA
22	LINFlexD_0	Transmit
23	LINFlexD_0	Receive
24	LINFlexD_1	Transmit
25	LINFlexD_1	Receive
26	FlexPWM_1	comp_val
27	FlexPWM_1	capt
28	Always Requestor	-

Table 173. DMA_MUX source slot mapping (continued)

DMA_MUX source slot #	Source module	Source resource
29	Always Requestor	-
30	Always Requestor	-
31	Always Requestor	-
32	Always Requestor	-
33	Always Requestor	-

19.5 DMA_MUX trigger inputs

Table 174 defines the signal sources for the trigger support of the first 4 channels of the DMA_MUX. *Table 174* is valid for both instantiations of the DMA_MUX module.

Table 174. DMA_MUX trigger sources

Source Module	Source Signal	DMACHMUX Channel Trigger #
PIT	Trigger Channel 0	0
PIT	Trigger Channel 1	1
PIT	Trigger Channel 2	2
PIT	Trigger Channel 3	3

19.6 Functional description

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 19.7.2, Enabling and configuring sources,* is followed, the configuration of the DMA_MUX may be changed during the normal operation of the system.

Functionally, the DMA_MUX channels may be divided into two classes:

- Channels that implement the normal routing functionality plus periodic triggering capability
- Channels that implement only the normal routing functionality

19.6.1 DMA Channels with periodic triggering capability

Besides the normal routing functionality, the first 4 channels of the DMA_MUX provide a special periodic triggering capability that can be used to provide an automatic mechanism to transmit bytes, frames or packets at fixed intervals without the need for processor intervention. The trigger is generated by the Periodic Interrupt Timer (PIT); as such, the configuration of the periodic triggering interval is done via configuration registers in the PIT. Please refer to the Periodic Interrupt Timer Block Guide for more information on this topic.

Note: Because of the dynamic nature of the system (i.e. DMA channel priorities, bus arbitration, interrupt service routine lengths, etc.), the number of clock cycles between a trigger and the actual DMA transfer cannot be guaranteed.

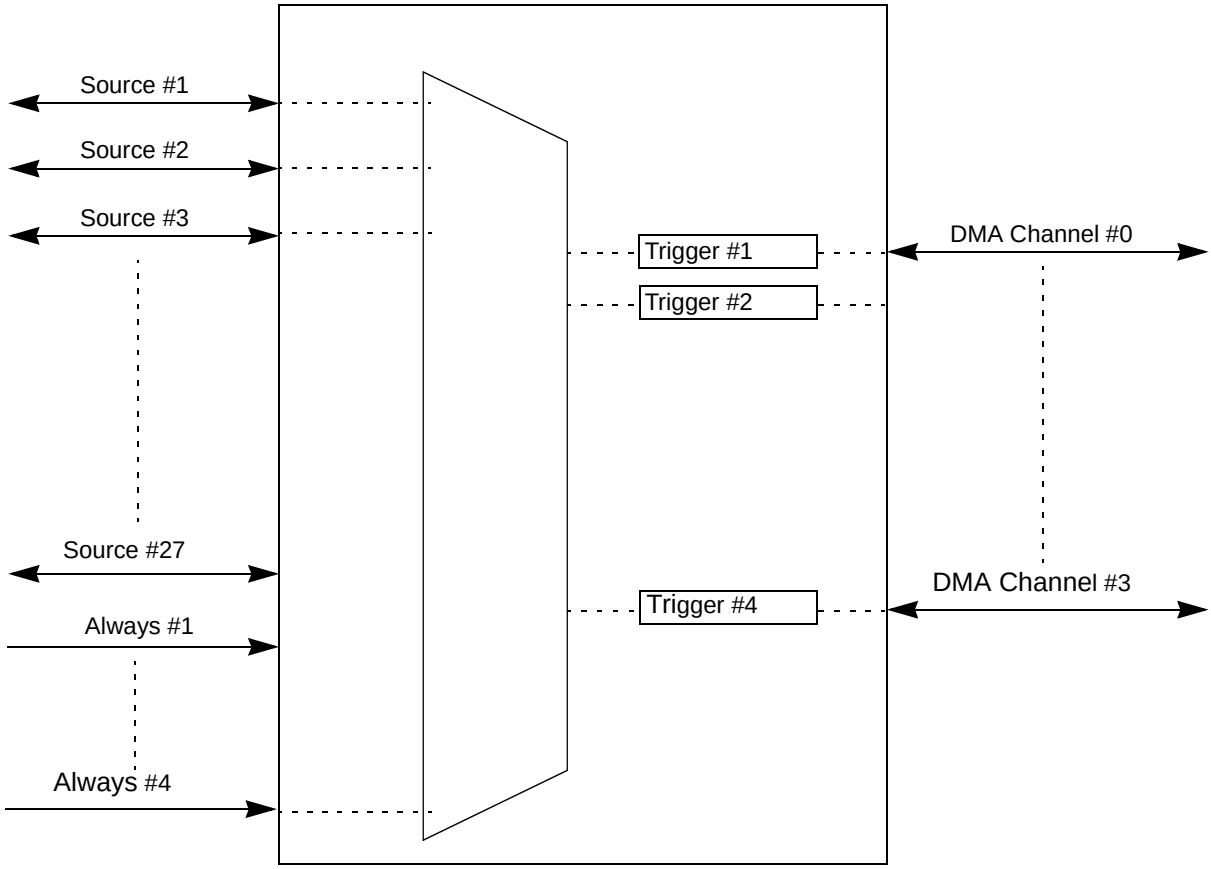


Figure 194. DMA_MUX triggered channels

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 195*.

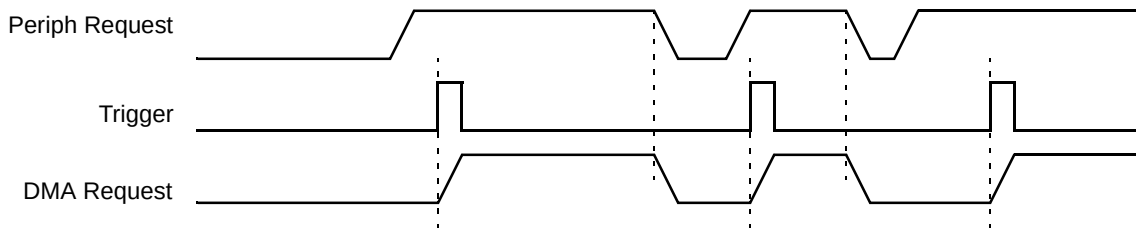


Figure 195. DMA_MUX Channel Triggering: Normal Operation

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 trigger will be ignored. This situation is illustrated in *Figure 196*.

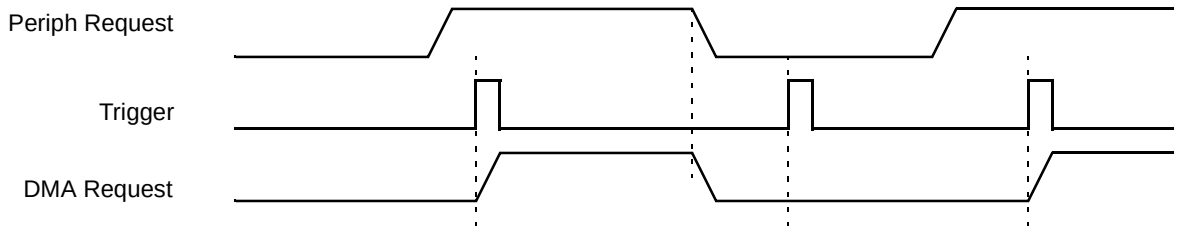


Figure 196. DMA_MUX Channel Triggering: Ignored Trigger

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 Periodic Interrupt Timer (PIT) Block Guide.

19.6.2 DMA channels with no triggering capability

The other channels of the DMA Mux provide the normal routing functionality as described in *Section 19.1.3, Modes of operation*.

19.6.3 "Always Enabled" DMA sources

In addition to the peripherals that can be used as DMA sources, there are 6 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_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_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_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_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.

19.7 Initialization/application Information

19.7.1 Reset

The reset state of each individual bit is shown within the Register Description section (*Section 19.3.1, Register descriptions*). In summary, after reset, all channels are disabled and must be explicitly enabled before use.

19.7.2 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 4 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 5 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 6 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 #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"
   :
   :
*CHCONFIG8 = 0x00;
*CHCONFIG8 = 0x87;
```

20 Enhanced Direct Memory Access (eDMA)

20.1 Introduction

The eDMA is a second-generation platform module capable of performing complex data transfers with minimal intervention from a host processor via 16 programmable channels. Intended for use as part of the Standard Product Platform (SPP), the hardware microarchitecture includes a eDMA engine which performs source and destination address calculations, and the actual data movement operations, along with a local memory containing the *transfer control descriptors* (TCD) for the channels. This SRAM-based implementation is used to minimize the overall module size.

DMA channel muxing is provided in *Chapter 19: eDMA Channel Mux (DMA_MUX)*.

Figure 197 is a block diagram of the eDMA module.

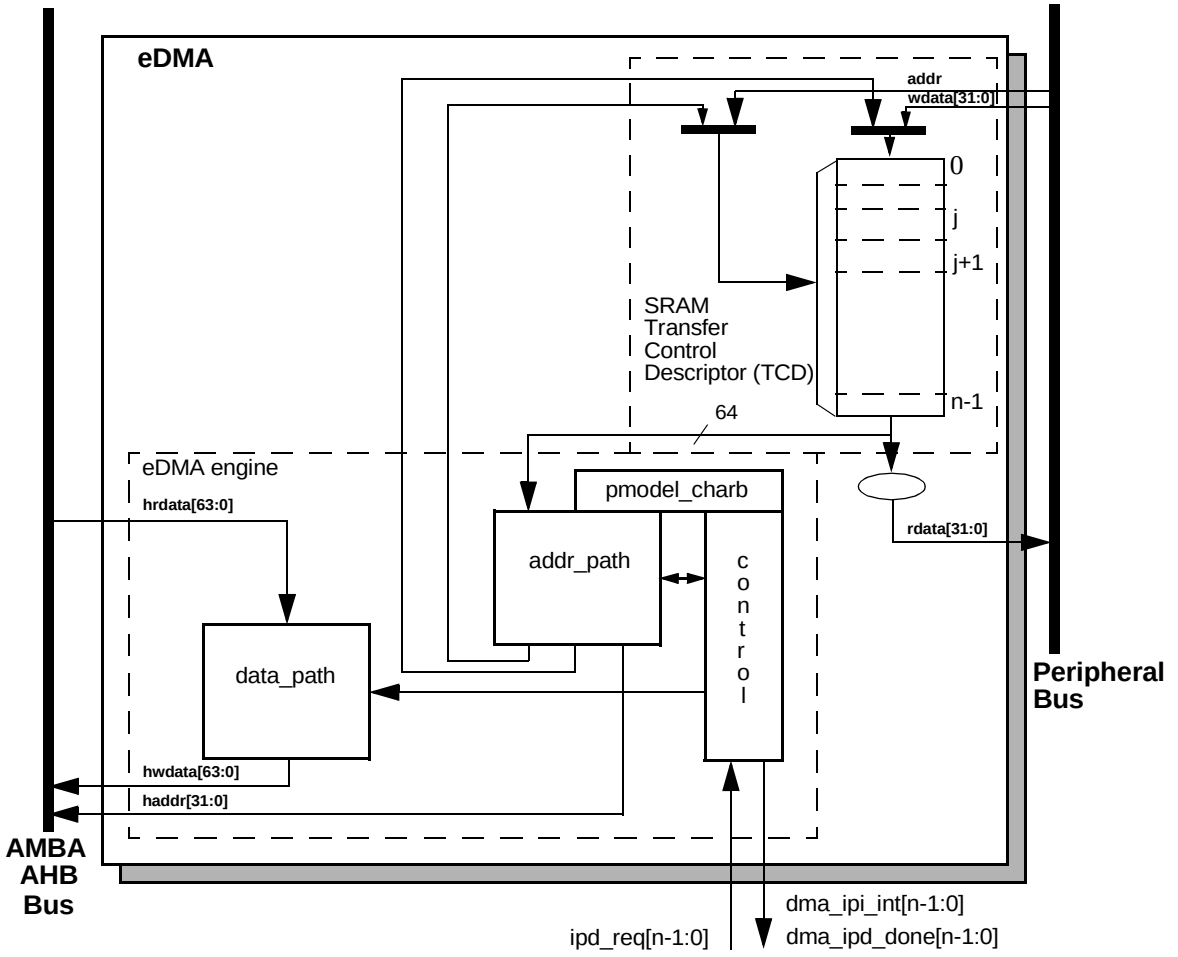


Figure 197. eDMA block diagram

20.1.1 Overview

The eDMA is a highly-programmable data transfer engine, which has been optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known, and is *not* defined within the data packet itself. The eDMA hardware supports:

- 16-channel implementation
- Connection to the AMBA-AHB crossbar switch (XBAR) for bus mastering the data movement
  - 64-bit AMBA-AHB datapath width
- Connection to the slave bus for programming the module
  - Parameterized support for 32- and 64-bit AMBA-AHB datapath widths
- 32-byte transfer control descriptor per channel stored in local memory
- 32 bytes of data registers, used as temporary storage to support burst transfers

Throughout this document, *n* is used to reference the channel number. Additionally, data sizes are defined as byte (8-bit), halfword (16-bit), word (32-bit) and doubleword (64-bit).

20.1.2 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

The basic operation of a channel is defined as:

1. The channel is initialized by software loading the transfer control descriptor into the eDMA's programming model, memory-mapped through the IPS space, and implemented as local memory.

2. The channel requests service; either explicitly by software, a peripheral request or a linkage from another channel.

Note: The major loop executes one iteration per service request.

3. The contents of the transfer control descriptor for the activated channel is read from the local memory and loaded into the eDMA engine's internal register file.

4. The eDMA engine executes the data transfer defined by the transfer control descriptor, reading from the source and writing to the destination. The number of iterations in the minor loop is automatically calculated by the eDMA engine. The number of iterations within the minor loop is a function of the number of bytes to transfer (nbytes), the source size (ssize) and the destination size (dsize). The completion of the minor loop is equal to one iteration of the major loop.

5. At the conclusion of the minor loop's execution, certain fields of the transfer control descriptor are written back to the local TCD memory.

The process (steps 2-5) is repeated until the outer major loop's iteration count is exhausted. At that time, additional processing steps are completed, e.g., the optional assertion of an interrupt request signaling the transfer's completion, final adjustments to the source and destination addresses, etc.

For more details, consult *Section 20.2.1, Register descriptions,* and *Section 20.3, Functional description.*

20.2 Memory map/register definition

The eDMA's programming model is partitioned into two sections, both mapped into the slave bus space: the first region defines a number of registers providing control functions, while the second region corresponds to the local transfer control descriptor memory.

Reading an *unimplemented* register bit or memory location will return the value of zero. Write the value of zero to *unimplemented* register bits. Any access to a *reserved* memory location will result in a bus error. *Reserved* memory locations are indicated in the memory map.

Many of the control registers have a bit width that matches the number of channels implemented in the module. The unused bits are not implemented: reads return zeroes, and writes are ignored.

The eDMA does not include any logic which provides access control. Rather, this function is supported using the standard access control logic provided by the PBRIDGE controller.

Table 175 is a 32-bit view of the eDMA's memory map.

Table 175. eDMA 32-bit memory map

Address offset	Register
0x0000	eDMA Control Register (DMACR)
0x0004	eDMA Error Status (DMAES)
0x0008	Reserved

Table 175. eDMA 32-bit memory map (continued)

Address offset	Register			
0x000C	eDMA Enable Request Low (DMAERQL, Channels 15–0)			
0x0010	Reserved			
0x0014	eDMA Enable Error Interrupt Low (DMAEEIL, Channels 15–0)			
0x0018	eDMA Set Enable Request (DMASERQ)	eDMA Clear Enable Request (DMACERQ)	eDMA Set Enable Error Interrupt (DMASEEI)	eDMA Clear Enable Error Interrupt (DMACEEI)
0x001C	eDMA Clear Interrupt Request (DMACINT)	eDMA Clear Error (DMACERR)	eDMA Set Start Bit (DMASSRT)	eDMA Clear Done Status Bit (DMACDNE)
0x0020	Reserved			
0x0024	eDMA Interrupt Request Low (DMAINTL, Channels 15–0)			
0x0028	Reserved			
0x002C	eDMA Error Low (DMAERRL, Channels 15–0)			
0x0030	Reserved			
0x0034	eDMA Hardware Request Status Low (DMAHRSL, Channels 15–0)			
0x0038–0x00FC	Reserved			
0x0100	eDMA Channel 0 Priority (DCHPRI0)	eDMA Channel 1 Priority (DCHPRI1)	eDMA Channel 2 Priority (DCHPRI2)	eDMA Channel 3 Priority (DCHPRI3)
0x0104	eDMA Channel 4 Priority (DCHPRI4)	eDMA Channel 5 Priority (DCHPRI5)	eDMA Channel 6 Priority (DCHPRI6)	eDMA Channel 7 Priority (DCHPRI7)
0x0108	eDMA Channel 8 Priority (DCHPRI8)	eDMA Channel 9 Priority (DCHPRI9)	eDMA Channel 10 Priority (DCHPRI10)	eDMA Channel 11 Priority (DCHPRI11)
0x010c	eDMA Channel 12 Priority (DCHPRI12)	eDMA Channel 13 Priority (DCHPRI13)	eDMA Channel 14 Priority (DCHPRI14)	eDMA Channel 15 Priority (DCHPRI15)
0x0110–0x0FFC	Reserved			
0x1000-0x11FC	TCD0–TCD15			
0x1200-0x17FC	Reserved			

20.2.1 Register descriptions

20.2.1.1 eDMA Control Register (DMACR)

The 32-bit DMACR defines the basic operating configuration of the eDMA.

The eDMA arbitrates channel service requests. This arbitration can be configured to use either a fixed-priority or a round-robin selection. 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 20.2.1.16, eDMA Channel n Priority (DCHPRIn), n = 0–15*). In round-robin arbitration mode, the channel priorities are ignored and the channels are cycled through without regard to priority.

Minor loop offsets are address offset values added to the final source address (saddr) or destination address (daddr) upon minor loop completion. When minor loop offsets are enabled, the minor loop offset (mloff) is added to the final source address (saddr), or the final destination address (daddr), or both prior to the addresses being written back into the TCD. If the major loop is complete, the minor loop offset is ignored and the major loop address offsets (slast and dlast_sga) are used to compute the next saddr and daddr values.

When minor loop mapping is enabled (DMACR[EMLM] = 1), TCDn word2 is redefined. A portion of TCDn word2 is used to specify multiple fields: an source enable bit (smloe) to specify the minor loop offset should be applied to the source address (saddr) upon minor loop completion, an destination enable bit (dmloe) to specify the minor loop offset should be applied to the destination address (daddr) upon minor loop completion, and the sign extended minor loop offset value (mloff). The same offset value (mloff) is used for both source and destination minor loop offsets. When either minor loop offset is enabled (smloe set or dmloe set), the nbytes field is reduced to 10 bits. When both minor loop offsets are disabled (smloe cleared and dmloe cleared), the nbytes field is a 30-bit vector.

When minor loop mapping is disabled (DMACR[EMLM] = 0), all 32 bits of TCDn word2 are assigned to the nbytes field. See *Section 20.2.1.17, Transfer Control Descriptor (TCD),* for more details.

See *Figure 198* and *Table 176* for the DMACR definition.

Register address: DMA_Offset + 0x0000

	0	1	2	3	4	5	6	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	0	0	0	0	EMLM	CLM	HALT	HOE	0	ERCA	EDBG	EBW
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 198. eDMA Control Register (DMACR)

Table 176. eDMA Control Register (DMACR) 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 cancel has been honored. In addition to cancelling the transfer, the ECX treats the cancel as an error condition; thus updating the DMAES register and generating an optional error interrupt (see *Section 20.2.1.2, eDMA Error Status (DMAES)*).
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 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 eDMA Operations	0Normal 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	0Normal 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 176. eDMA Control Register (DMACR) field descriptions

Name	Description	Value
ERCA	Enable Round Robin Channel Arbitration	0 Fixed priority arbitration is used for channel selection. 1 Round robin arbitration is used for channel selection.
EDBG	Enable Debug	0 The assertion of the ipg_debug input is ignored. 1 The assertion of the ipg_debug input causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution will resume when either the ipg_debug input is negated 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.

20.2.1.2 eDMA Error Status (DMAES)

The DMAES register provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively. In fixed arbitration mode, a configuration error is caused by any two channel priorities being equal. All channel priority levels must be unique when fixed arbitration mode is enabled. If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (dlast_sga) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD.citer.e_link bit does not equal the TCD.biter.e_link bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and assert an error interrupt request, if enabled. A scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion when properly enabled. 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 stopped and the appropriate bus error flag set. In this case, the state of the channel's transfer control descriptor is updated by the eDMA engine with the current source address, destination address and current iteration count at the point of the fault. When a system bus error occurs, the channel is terminated after the read or write transaction which is already pipelined after errant access, has completed. 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.

A transfer may be cancelled by software via the DMACR[CX] bit or hardware via the **dma_cancel_xfer** input signal. When a cancel transfer request is recognized, the eDMA engine stops processing the channel. The current read-write sequence is allowed to finish. If the cancel occurs on the last read-write sequence of a major or minor loop, the cancel request is discarded and the channel retires normally.

The error cancel transfer is the same as a cancel transfer except the DMAES register is updated with the cancelled channel number and error cancel bit is set. The TCD of a cancelled channel has the source address and destination address of the last transfer saved in the TCD. It is the responsibility of the user to initialize the TCD again should the channel need to be restarted because the aforementioned fields have been modified by the eDMA engine and no longer represent the original parameters. When a transfer is cancelled via the error cancel transfer mechanism (setting the DMACR[ECX]), the channel number is loaded into the ERRCHN field and the ECX and VLD bits are set are set in the DMAES register. In addition, an error interrupt may be generated if enabled. See *Section 20.2.1.14, eDMA Error Low (DMAERRL) Register,* for error interrupt details.

The occurrence of any type of error causes the eDMA engine to immediately stop, 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 DMAES register. 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.

See *Figure 199* and *Table 177* for the DMAES definition.

Register address: DMA_Offset + 0x0004

	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	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	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 199. eDMA Error Status (DMAES) Register

Table 177. eDMA Error Status (DMAES) field descriptions

Name	Description	Value
VLD	Logical OR of all DMAERRH and DMAERRL status bits.	0 No DMAERR bits are set. 1 At least one DMAERR bit is set indicating a valid error exists that has not been cleared.
ECX	Transfer cancelled	0 No cancelled transfers. 1 The last recorded entry was a cancelled transfer via the error cancel transfer input.

Table 177. eDMA Error Status (DMAES) field descriptions (continued)

Name	Description	Value
CPE	Channel Priority Error	0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities. All channel priorities 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.
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.

20.2.1.3 eDMA Enable Request Low (DMAERQL)

The DMAERQL register provides a bit map for the implemented channels to enable the request signal for each channel. The state of any given channel enable is directly affected by writes to this register; it is also affected by writes to the DMASERQ and DMACERQ

registers. The eDMA{S,C}ERQ registers are provided so that the request enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the DMAERQL 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 explicitly through software or a linked channel request.

See *Figure 200* and *Table 178* for the DMAERQL definition.

Register address: DMA_Offset + 0x000C

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

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

= Unimplemented

Figure 200. eDMA Enable Request Low (DMAERQL) Register

Table 178. DMAERQL field descriptions

Name	Description	Value
ERQn, n = 0,... 15	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 DMAERQ bit for that channel. If the TCD.d_req bit is set, then the corresponding DMAERQ bit is cleared, disabling the eDMA request; else if the d_req bit is cleared, the state of the DMAERQ bit is unaffected.

20.2.1.4 eDMA Enable Error Interrupt Low (DMAEEIL)

The DMAEEIL register provides a bit map for the implemented channels to enable the error interrupt signal for each channel. The state of any given channel's error interrupt enable is directly affected by writes to this register; it is also affected by writes to the DMASEEI and DMACEEI registers. The eDMA{S,C}EEI registers are provided so that the error interrupt enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the DMAEEIL 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 201* and *Table 179* for the DMAEEIL definition.

Register address: DMA_Offset + 0x0014

	0	1	2	3	4	5	6	7	8	9	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	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 201. eDMA Enable Error Interrupt Low (DMAEEIL) Register

Table 179. DMAEEIL field descriptions

Name	Description	Value
EEIn, n = 0,... 15	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.

20.2.1.5 eDMA Set Enable Request (DMASERQ)

The DMASERQ register provides a simple memory-mapped mechanism to set a given bit in the DMAERQL register to enable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the DMAERQL register to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing the entire contents of DMAERQL to be asserted. If the NOP bit is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

See *Figure 202* and *Table 180* for the DMASERQ definition.

Register address: DMA_Offset + 0x0018

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	SERQ						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 202. eDMA Set Enable Request (DMASERQ) Register

Table 180. DMASERQ field descriptions

Name	Description	Value
NOP	No Operation	0 Normal operation 1 No operation, ignore the other bits in the register
SERQ	Set Enable Request	See the field structure in *Table 181*

Table 181. DMASERQ[SERQ] field structure

Bit number	Description
0	"Set all" bit: 0 Affects only the channel specified in bit numbers 4–7 1 Affects all channels (bit numbers 4–7 are ignored)
1–2	Reserved
3–6	Set the corresponding bit in DMAERQL

20.2.1.6 eDMA Clear Enable Request (DMACERQ)

The DMACERQ register provides a simple memory-mapped mechanism to clear a given bit in the DMAERQL register to disable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the DMAERQL register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAERQL to be zeroed, disabling all eDMA request inputs. If the NOP bit is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

See *Figure 203* and *Table 182* for the DMACERQ definition.

Register address: DMA_Offset + 0x0019

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CERQ						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 203. eDMA Clear Enable Request (DMACERQ) Register

Table 182. DMACERQ field descriptions

Name	Description	Value
NOP	No Operation	0 Normal operation 1 No operation, ignore the other bits in the register
CERQ	Clear Enable Request	See the field structure in *Table 183*

Table 183. DMACERQ[CERQ] field structure

Bit number	Description
0	"Clear all" bit: 0 Affects only the channel specified in bit numbers 4–7 1 Affects all channels (bit numbers 4–7 are ignored)
1–2	Reserved
3–6	Clear the corresponding bit in DMAERQL

20.2.1.7 eDMA Set Enable Error Interrupt (DMASEEI)

The DMASEEI register provides a simple memory-mapped mechanism to set a given bit in the DMAEEIL register to enable the error interrupt for a given channel. The data value on a

register write causes the corresponding bit in the DMAEEIL register to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing the entire contents of DMAEEIL to be asserted. If the NOP bit is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

See *Figure 204* and *Table 184* for the DMASEEI definition.

Register address: DMA_Offset + 0x001A

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	SEEI						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 204. eDMA Set Enable Error Interrupt (DMASEEI) Register

Table 184. DMASEEI field descriptions

Name	Description	Value
NOP	No Operation	0 Normal operation 1 No operation, ignore the other bits in the register
SEEI	Set Enable Error Interrupt	See the field structure in *Table 185*

Table 185. DMASEEI[SEEI] field structure

Bit number	Description
0	“Set all” bit: 0 Affects only the channel specified in bit numbers 4–7 1 Affects all channels (bit numbers 4–7 are ignored)
1–2	Reserved
3–6	Set the corresponding bit in DMAEEIL

20.2.1.8 eDMA Clear Enable Error Interrupt (DMACEEI)

The DMACEEI register provides a simple memory-mapped mechanism to clear a given bit in the DMAEEIL register to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMAEEIL register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAEEIL to be zeroed, disabling all eDMA request inputs. If the NOP is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

See *Figure 205* and *Table 186* for the DMACEEI definition.

Register address: DMA_Offset + 0x001B

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CEEI						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 205. eDMA Clear Enable Error Interrupt (DMACEEI) Register

Table 186. DMACEEI field descriptions

Name	Description	Value
NOP	No Operation	0 Normal operation. 1 No operation, ignore the other bits in the register
CEEI	Clear Enable Error Interrupt	See the field structure in *Table 187*

Table 187. DMACEEI[CEEI] field structure

Bit number	Description
0	"Clear all" bit: 0 Affects only the channel specified in bit numbers 4–7 1 Affects all channels (bit numbers 4–7 are ignored)
1–2	Reserved
3–6	Clear the corresponding bit in DMAEEIL

20.2.1.9 eDMA Clear Interrupt Request (DMACINT) Register

The DMACINT register provides a simple memory-mapped mechanism to clear a given bit in the DMAINTL register to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the DMAINTL register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAINTL to be zeroed, disabling all eDMA interrupt requests. If the NOP bit is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

See *Figure 206* and *Table 188* for the DMACINT definition.

Register address: DMA_Offset + 0x001C

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CINT						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 206. eDMA Clear Interrupt Request (DMACINT) Register

Table 188. DMACINT field descriptions

Name	Description	Value
NOP	No Operation	0 Normal operation 1 No operation, ignore the other bits in the register
CINT	Clear Interrupt Request	See the field structure in *Table 189*

Table 189. DMACINT[CINT] field structure

Bit number	Description
0	"Clear all" bit: 0 Affects only the channel specified in bit numbers 4–7 1 Affects all channels (bit numbers 4–7 are ignored)
1–2	Reserved
3–6	Clear the corresponding bit in DMAINTL

20.2.1.10 eDMA Clear Error (DMACERR) Register

The DMACEER register provides a simple memory-mapped mechanism to clear a given bit in the DMAERRL register to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the DMAERRL register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAERRL to be zeroed, clearing all channel error indicators. If the NOP bit is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

See *Figure 207* and *Table 190* for the DMACERR definition.

Register address: DMA_Offset + 0x001D

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CERR						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 207. eDMA Clear Error (DMACERR) Register

Table 190. DMACERR field descriptions

Name	Description	Value
NOP	No Operation	0 Normal operation 1 No operation, ignore the other bits in the register
CERR	Clear Error Indicator	See the field structure in *Table 191*

Table 191. DMACERR[CERR] field structure

Bit number	Description
0	“Clear all” bit: 0 Affects only the channel specified in bit numbers 4–7 1 Affects all channels (bit numbers 4–7 are ignored)
1–2	Reserved
3–6	Clear the corresponding bit in DMAERRL

20.2.1.11 eDMA Set START Bit (DMASSRT) Register

The DMASSRT register provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding Transfer Control Descriptor to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing all START bits to be set. If the NOP bit is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

See *Table 209* for the TCD START bit definition.

Register address: DMA_Offset + 0x001E

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	SSRT						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 208. eDMA Set START Bit (DMASSRT) Register

Table 192. DMASSRT field descriptions

Name	Description	Value
NOP	No Operation	0 Normal operation 1 No operation, ignore the other bits in the register
SSRT	Set START Bit (Channel Service Request)	See the field structure in *Table 193*

Table 193. DMASSRT[SSRT] field structure

Bit number	Description
0	“Set all” bit: 0 Affects only the channel specified in bit numbers 4–7 1 Affects all channels (bit numbers 4–7 are ignored)
1–2	Reserved
3–6	Set the corresponding channel’s TCD.start

20.2.1.12 eDMA Clear DONE Status (DMACDNE) Register

The DMACDNE register provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit

in the corresponding Transfer Control Descriptor to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing all DONE bits to be cleared. If the NOP bit is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

See *Table 209* for the TCD DONE bit definition.

Register address: DMA_Offset + 0x001F

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CDNE						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 209. eDMA Clear DONE Status (DMACDNE) Register

Table 194. DMACDNE field descriptions

Name	Description	Value
NOP	No Operation	0 Normal operation 1 No operation, ignore the other bits in the register
CDNE	Clear DONE Status Bit	See the field structure in *Table 195*

Table 195. DMACDNE[CDNE] field structure

Bit number	Description
0	“Clear all” bit: 0 Affects only the channel specified in bit numbers 4–7 1 Affects all channels (bit numbers 4–7 are ignored)
1–2	Reserved
3–6	Clear the corresponding channel’s DONE bit

20.2.1.13 eDMA Interrupt Request Low (DMAINTL) Register

The DMAINTL register provides a bit map for the implemented channels signaling the presence of an interrupt request for each channel. The eDMA engine signals the occurrence of a programmed interrupt upon the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the platform’s interrupt controller. During the execution of the interrupt service routine associated with any given channel, it is software’s responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the DMACINT register 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 DMACINT register. On writes to the DMAINTL, a one in any bit position clears the corresponding channel’s interrupt request. A zero in any bit position has no affect on the corresponding channel’s current interrupt status. The DMACINT register is provided so the interrupt request for a *single* channel can easily be cleared without the need to perform a read-modify-write sequence to the DMAINTL register.

See *Figure 210* and *Table 196* for the DMAINTL definition.

Register address: DMA_Offset + 0x0024

	0	1	2	3	4	5	6	7	8	9	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	INT15	INT14	INT13	INT12	INT11	INT10	INT9	INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 210. eDMA Interrupt Request Low (DMAINTL) Register

Table 196. DMAINTL field descriptions

Name	Description	Value
INTn, n = 0,... 15	DMA Interrupt Request n	0 The interrupt request for channel n is cleared. 1 The interrupt request for channel n is active.

20.2.1.14 eDMA Error Low (DMAERRL) Register

The DMAERRL register provides a bit map for the implemented channels signaling the presence of an error for each channel. The eDMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the DMAEEIL register, then logically summed to form an error interrupt request which is then routed to the platform's 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 DMACERR register in the interrupt service routine is used for this purpose. Recall 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 DMAEEIL register. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the DMACERR register. On writes to the DMAERRL, a one in any bit position clears the corresponding channel's error status. A zero in any bit position has no affect on the corresponding channel's current error status. The DMACERR register is provided so the error indicator for a *single* channel can easily be cleared.

See *Figure 211* and *Table 197* for the DMAERRL definition.

Register address: DMA_Offset + 0x002C

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

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

= Unimplemented

Figure 211. eDMA Error Low (DMAERRL) Register

Table 197. DMAERRL field descriptions

Name	Description	Value
ERRn, n = 0,... 15	DMA Error n	0 An error in channel n has not occurred. 1 An error in channel n has occurred.

20.2.1.15 eDMA Hardware Request Status Low (DMAHRSL) Register

The DMAHRSL register provides a bit map for the implemented channels to show the current hardware request status for each channel. Hardware request status reflects the current state of the registered and qualified (via the DMAERQ field) ipd_req lines as seen by the eDMA's arbitration logic. This view into the hardware request signals may be used for debug purposes.

See *Figure 212* and *Figure 198* for the DMAHRSL definition.

Register address: DMA_Offset + 0x0034

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

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

= Unimplemented

Figure 212. eDMA Hardware Request Status Low (DMAHRSL) Register

Table 198. DMAHRSL field descriptions

Name	Description	Value
HRSn, n = 0,... 15	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 DMAERQn bit.*

20.2.1.16 eDMA Channel n Priority (DCHPRIn), n = 0–15

When the fixed-priority channel arbitration mode is enabled (DMACR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value, i.e., 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. Software must program the channel priorities with unique values, otherwise a configuration error will be reported. The range of the priority value is limited to the values of 0–15.

Channel preemption is enabled on a per channel basis by setting the ECP bit in the DCHPRIn register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all of its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel 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 only available 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 DCHPRIn 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 213* and *Table 199* for the DCHPRIn definition.

Register address: DMA_Offset + 0x100 + n

	0	1	2	3	4	5	6	7
R	ECP	DPA	*		CHPRI[0:3]			
W								
RESET:	0	0	*	*	*	*	*	*

= Unimplemented,
= defaults to channel number (n) after reset

Figure 213. eDMA Channel n Priority (DCHPRIn) Register

Table 199. eDMA Channel n Priority (DCHPRIn) 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.

20.2.1.17 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The TCD structure was previously discussed in detail in *Section 20.1.2, Features.* The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1, ... channel [n-1]. The definitions of the TCD are presented as eight 32-bit values. *Table 200* is a 32-bit view of the basic TCD structure.

Table 200. TCDn 32-bit memory structure

eDMA Offset	TCDn Field		
0x1000 + (32 x n) + 0x00	Source Address (saddr)		
0x1000 + (32 x n) + 0x04	Transfer Attributes (smod, ssize, dmod, dsize)	Signed Source Address Offset (soff)	
0x1000 + (32 x n) + 0x08	Signed Minor Loop Offset (smloe, dmloe, mloff)		Inner "Minor" Byte Count (nbytes)
0x1000 + (32 x n) + 0x0c	Last Source Address Adjustment (slast)		
0x1000 + (32 x n) + 0x10	Destination Address (daddr)		
0x1000 + (32 x n) + 0x14	Current "Major" Iteration Count (citer)	Signed Destination Address Offset (doff)	
0x1000 + (32 x n) + 0x18	Last Destination Address Adjustment/Scatter Gather Address (dlast_sga)		
0x1000 + (32 x n) + 0x1c	Beginning "Major" Iteration Count (biter)	Channel Control/Status (bwc, major.linkch, done, active, major.e_link, e_sg, d_req, int_half, int_maj, start)	

Figure 214 and *Table 201* define word 0 of the TCDn structure, the saddr field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x00

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	saddr[16:31]															
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

= Unimplemented

Figure 214. TCDn Word 0 (TCDn.saddr) Field

Table 201. TCDn Word 0 (TCDn.saddr) field description

Name	Description	Value
saddr[0:31]	Source address	Memory address pointing to the source data.

Figure 215 and *Table 202* define word 1 of the TCDn structure, the soff and transfer attribute fields.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x04

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	smod[0:4]					ssize[0:2]			dmod[0:4]					dsize[0:2]		
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	soff[16:31]															
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

= Unimplemented

Figure 215. TCDn Word 1 (TCDn.{soff,smod,ssize,dmod,dsize}) Fields

Table 202. TCDn Word 1 (TCDn.{smod,ssize,dmod,dsize,soff}) field descriptions

Name	Description	Value
smod[0:4]	Source address modulo	0Source address modulo feature is disabled. non-0 The value defines a specific address bit which is selected to be either the value after saddr + soff calculation is performed or the original register value. This feature provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 “size” bytes, the queue should be based at a 0-modulo-size address and the smod field set to the appropriate value to freeze the upper address bits. The bit select is defined as ((1 << smod[4:0]) - 1) where a resulting 1 in a bit location selects the next state address for the corresponding address bit location and a 0 selects the original register value for the corresponding address bit location. For this 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.
ssize[0:2]	Source data transfer size	000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 Reserved 101 32-byte burst (64-bit × 4) 110 Reserved 111 Reserved
dmod[0:4]	Destination address modulo	See the smod[5:0] definition.
dsize[0:2]	Destination data transfer size	See the ssize[2:0] definition.
soff[16:31]	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.

Figure 216 and *Table 203* define word 2 of the TCDn structure, the nbytes field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x08

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	nbytes[16:31]															
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

= Unimplemented

Figure 216. TCDn Word 2 (TCDn.nbytes) Field (DMACR[EMLM] = 0)

Table 203. TCDn Word 2 (TCDn.nbytes) field description

Name	Description	Value
nbytes[0:31]	Inner “minor” byte transfer count	Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. After the minor count is exhausted, the current values of the saddr and daddr are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. The nbytes value 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GB transfer.

When minor loop mapping (DMACR[EMLM] = 1) is enabled, TCD word2 is redefined as four fields: a source minor loop offset enable, a destination minor loop offset enable, a minor loop offset field and a nbytes field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x08

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	smloe	dmloe	mloff[0:13] or nbytes[0:13]													
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	mloff[14:19] or nbytes[14:19]						nbytes[20:29]									
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

= Unimplemented

Figure 217. TCDn Word 2 (TCDn.nbytes) Field (DMACR[EMLM] = 1)

Table 204. TCDn Word 2 (TCDn.nbytes) field descriptions

Name	Description	Value
smloe	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.
dmloe	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.
nbytes[0:19] or mloff[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.
nbytes[0:9]	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. This field is extended to 30 bits when both smloe and dmloe are cleared (disabled).

Figure 218 and *Table 205* define word 3 of the TCDn structure, the slast field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x0c

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	slast[16:31]															
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

= Unimplemented

Figure 218. TCDn Word 3 (TCDn.slast) Field

Table 205. TCDn Word 3 (TCDn.slast) field descriptions

Name	Description	Value
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.

Figure 219 and *Table 206* define word 4 of the TCDn structure, the daddr field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x10

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	daddr[16:31]															
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

= Unimplemented

Figure 219. TCDn Word 4 (TCDn.daddr) Field

Table 206. TCDn Word 4 (TCDn.daddr) field description

Name	Description	Value
daddr[0:31]	Destination address	Memory address pointing to the destination data.

Figure 220 and *Table 207* define word 5 of the TCDn structure, the citer and doff fields.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x14

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	citer. e_link	citer[0:5] or citer.linkch[0:5]						citer[6:14]								
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	doff[16:31]															
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

= Unimplemented

Figure 220. TCDn Word 5 (TCDn.{citer,doff}) Fields

Table 207. TCDn Word 5 (TCDn.{doff,citer}) Field Descriptions

Name	Description	Value
citer.e_link	Enable 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 citer.linkch[5:0]. 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. *This bit must be equal to the biter.e_link bit otherwise a configuration error will be reported.* 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.

citer[0:5]
or
citer.linkch[0:5]
	Current “major” iteration count
	or
	Link channel number
		if (TCD.citer.e_link = 0) then
		No channel-to-channel linking (or chaining) is performed after the inner "minor" loop is exhausted. TCD word 5, bits [30:25] are used to form a 15 bit citer field.
		else
		After the "minor" loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by citer.linkch[5:0] by setting that channel’s TCD.start bit.
		The value contained in citer.linkch[5:0] must not exceed the number of implemented channels.
		For this device, bits 0:1 of this field are reserved.

Table 207. TCDn Word 5 (TCDn.{doff,citer}) Field Descriptions (continued)

Name	Description	Value
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. Once the major iteration count is exhausted, the channel performs a number of operations (e.g., final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the citer field from the beginning iteration count (biter) field. When the citer field is initially loaded by software, it must be set to the same value as that contained in the biter field. If the channel is configured to execute a single service request, the initial values of biter and citer should be 0x0001.
doff[16:31]	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.

Figure 221 and *Table 208* define word 6 of the TCDn structure, the dlast_sga field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x18

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	dlast_sga[16:31]															
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

= Unimplemented

Figure 221. TCDn Word 6 (TCDn.dlast_sga) Field

Table 208. TCDn Word 6 (TCDn.dlast_sga) field description

Name	Description	Value

dlast_sga[31:00:31]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather)	if (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.
		
		else
		This address points to the beginning of a 0-modulo-32 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, else a configuration error is reported.

Figure 222 and *Table 209* define word 7 of the TCDn structure, the biter and control/status fields.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	biter.e_link	biter[0:5] or biter.linkch[0:5]							biter[6:14]							
RESET:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	bwc		major.linkch[0:5]						done	active	major.e_link	e_sg	d_req	int_half	int_maj	start
RESET:	-	-	-	-	-	-	-	-	0	0	-	-	-	-	-	0

= Unimplemented

Figure 222. TCDn Word 7 (TCDn.{biter,control/status}) Fields

Table 209. TCDn Word 7 (TCDn.{biter, control/status}) field descriptions

Name	Description	Value
biter.e_link	Enable channel-to-channel linking on minor loop complete	This is the initial value copied into the citer.e_link field when the major loop is completed. The citer.e_link field controls channel linking during channel execution. *This bit must be equal to the citer.e_link bit otherwise a configuration error will be reported.* 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.

biter[0:5]
or
biter.linkch[0:5]
	Beginning “major” iteration count
	or
	Beginning Link channel number
		This is the initial value copied into the citer field or citer.linkch field when the major loop is completed. The citer fields controls the iteration count and linking during channel execution.
		
		if (TCD.biter.e_link = 0) then
		No channel-to-channel linking (or chaining) is performed after the inner "minor" loop is exhausted. TCD word 5, bits [30:25] are used to form a 15 bit biter field.
		
		else
		After the "minor" loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by biter.linkch[5:0] by setting that channel’s TCD.start bit.
		
		The value contained in biter.linkch[5:0] must not exceed the number of implemented channels.
		
		For this device, bits 0:1 of this field are reserved.

biter[6:14]	Beginning “major” iteration count	This is the initial value copied into the citer field or citer.linkch field when the major loop is completed. The citer fields controls the iteration count and linking during channel execution. This 9- or 15-bit count represents the beginning major loop count for the channel. As the major iteration count is exhausted, the contents of the entire 16-bit biter entry is reloaded into the 16-bit citer entry. When the biter field is initially loaded by software, it must be set to the same value as that contained in the citer field. If the channel is configured to execute a single service request, the initial values of biter and citer should be 0x0001.

Table 209. TCDn Word 7 (TCDn.{biter, control/status}) field descriptions (continued)

Name	Description	Value
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, 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 platform's cross-bar arbitration switch. To minimize start-up latency, bandwidth control stalls are suppressed for the first two AHB bus cycles and after the last write of each minor loop. The dynamic priority elevation setting elevates the priority of the eDMA as seen by the cross-bar arbitration switch for the executing channel. Dynamic priority elevation is suppressed during the first two AHB bus cycles. 00 No eDMA engine stalls 01 Dynamic priority elevation 10 eDMA engine stalls for 4 cycles after each r/w 11 eDMA engine stalls for 8 cycles after each r/w

major.linkch[0:5]	Link channel number	if (TCD.major.e_link = 0) then
		No channel-to-channel linking (or chaining) is performed after the outer "major" loop counter is exhausted.
		
		else
		After the "major" loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by major.linkch[5:0] by setting that channel's TCD.start bit.
		
		The value contained in major.linkch[5:0] must not exceed the number of implemented channels.
		
		For this device, bits 0:1 of this field are reserved.

done	Channel done	This flag indicates the eDMA has completed the outer major loop. It is set by the eDMA engine as the citer count reaches zero; it is cleared by software, or the hardware when the channel is activated. This bit must be cleared in order to write the major.e_link or e_sg bits.
active	Channel active	This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the eDMA engine as the inner minor loop completes or if any error condition is detected.

Table 209. TCDn Word 7 (TCDn.{biter, control/status}) field descriptions (continued)

Name	Description	Value
major.e_link	Enable channel-to-channel linking on major loop complete	As the channel completes the outer major loop, this flag enables the linking to another channel, defined by major.linkch[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.start bit of the specified channel. *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.
e_sg	Enable scatter/gather processing	As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If enabled, the eDMA engine uses dlast_sga as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. *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.
d_req	Disable request	If this flag is set, the eDMA hardware automatically clears the corresponding DMAERQ bit when the current major iteration count reaches zero. 0 The channel's DMAERQ bit is not affected. 1 The channel's DMAERQ bit is cleared when the outer major loop is complete.
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 DMAINT register 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 schemes or other types of data movement where the processor needs an early indication of the transfer's progress. The halfway complete interrupt is disabled when biter values are less than two. 0The half-point interrupt is disabled. 1The half-point interrupt is enabled.

Table 209. TCDn Word 7 (TCDn.{biter, control/status}) field descriptions (continued)

Name	Description	Value
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 DMAINT register 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.
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.

20.3 Functional description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

20.3.1 eDMA microarchitecture

The eDMA module is partitioned into two major modules: the eDMA engine and the transfer control descriptor local memory. Additionally, the eDMA engine is further partitioned into four submodules, which are detailed below.

- eDMA engine
  - *addr_path*: This module implements registered versions of two channel transfer control descriptors: channel "x" and channel “y”, and is responsible for all the master bus address calculations. All the implemented channels provide the exact same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. Once 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 DCHPRIn[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.

    When any other channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other addr_path.channel_{x,y}. Once the inner minor loop completes execution, the addr_path hardware writes the new values for the TCDn.{saddr, daddr, citer} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCDn.citer field, and a possible fetch of the next TCDn 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 AMBA-AHB read data bus is the primary input, and the AHB write data bus is the primary output.
The addr_ and data_path modules directly support the 2-stage pipelined AMBA-AHB bus. The addr_path module represents the 1st stage of the bus pipeline (the address phase), while the data_path module implements the 2nd stage of the pipeline (the data phase).

  - *pmodel_charb*: This module implements the first section of eDMA's programming model as well as the channel arbitration logic. The programming model registers are connected to the IPS bus (not shown). The ipd_req[n] inputs and dma_ipi_int[n] outputs are also connected to this module (via the control logic).
  - *control*: This module provides all the control functions for the eDMA engine. For data transfers where the source and destination sizes are equal, the eDMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner 'minor loop' byte count has been moved. For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. As an example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- transfer_control_descriptor local memory
  - *memory controller*: This logic implements the required dual-ported controller, handling accesses from both the eDMA engine as well as references from the IPS bus. As noted earlier, in the event of simultaneous accesses, the eDMA engine is given priority and the IPS 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

20.3.2 eDMA basic data flow

The basic flow of a data transfer can be partitioned into three segments. As shown in *Figure 223*, the first segment involves the channel service request. In the diagram, this example uses the assertion of the ipd_req[n] signal to request service for channel n. Channel service request via software and the TCDn.start bit follows the same basic flow as an ipd_req. The ipd_req[n] input signal is registered internally and then routed to through the eDMA engine, first through the control module, then into the programming model/channel arbitration (pmodel_charb) module. In the next cycle, the channel arbitration is performed, either using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path (addr_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.addr_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 dma_engine.addr_path.channel_{x,y} registers.

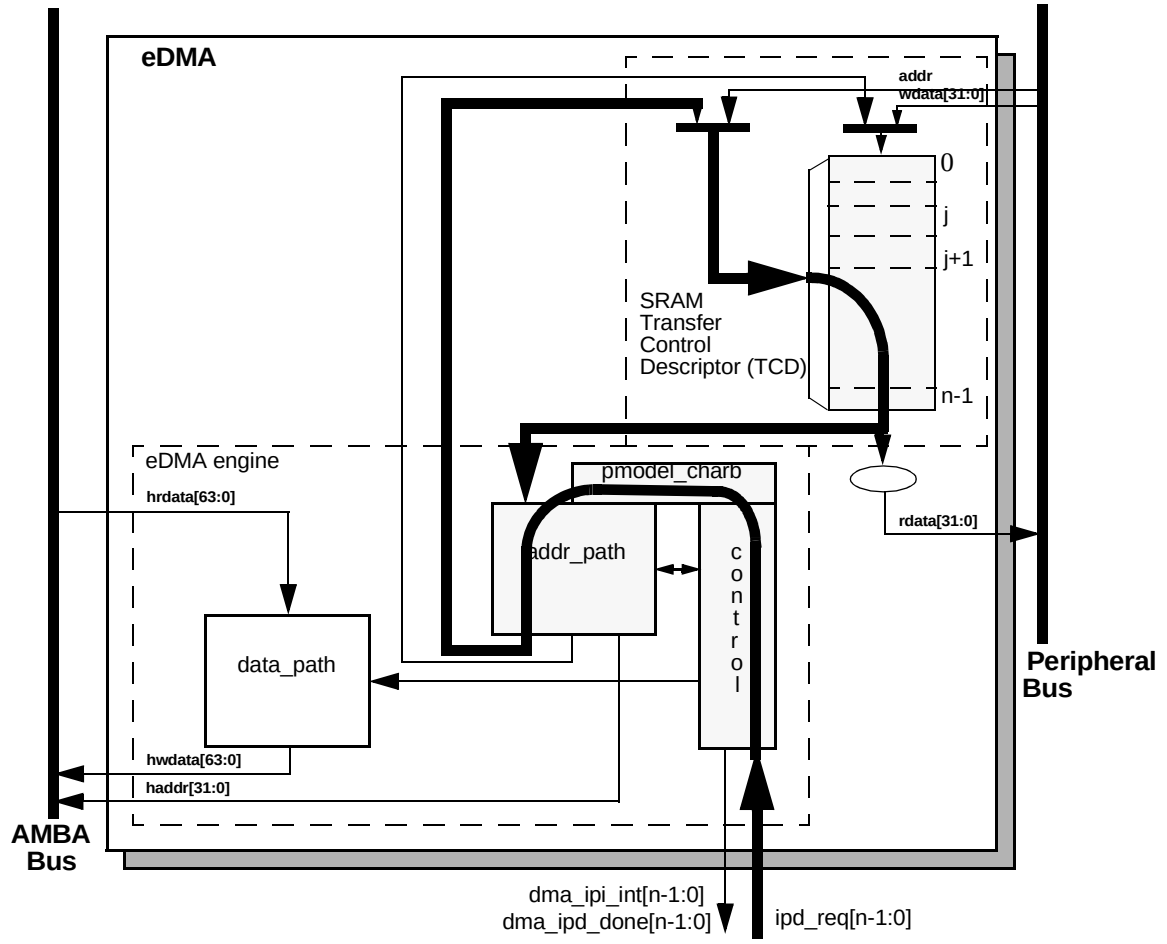


Figure 223. eDMA operation, part 1

In the second part of the basic data flow as shown in *Figure 224*, the modules associated with the data transfer (addr_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 AMBA-AHB bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The dma_ipd_done[n] signal is asserted at the end of the minor byte count transfer.

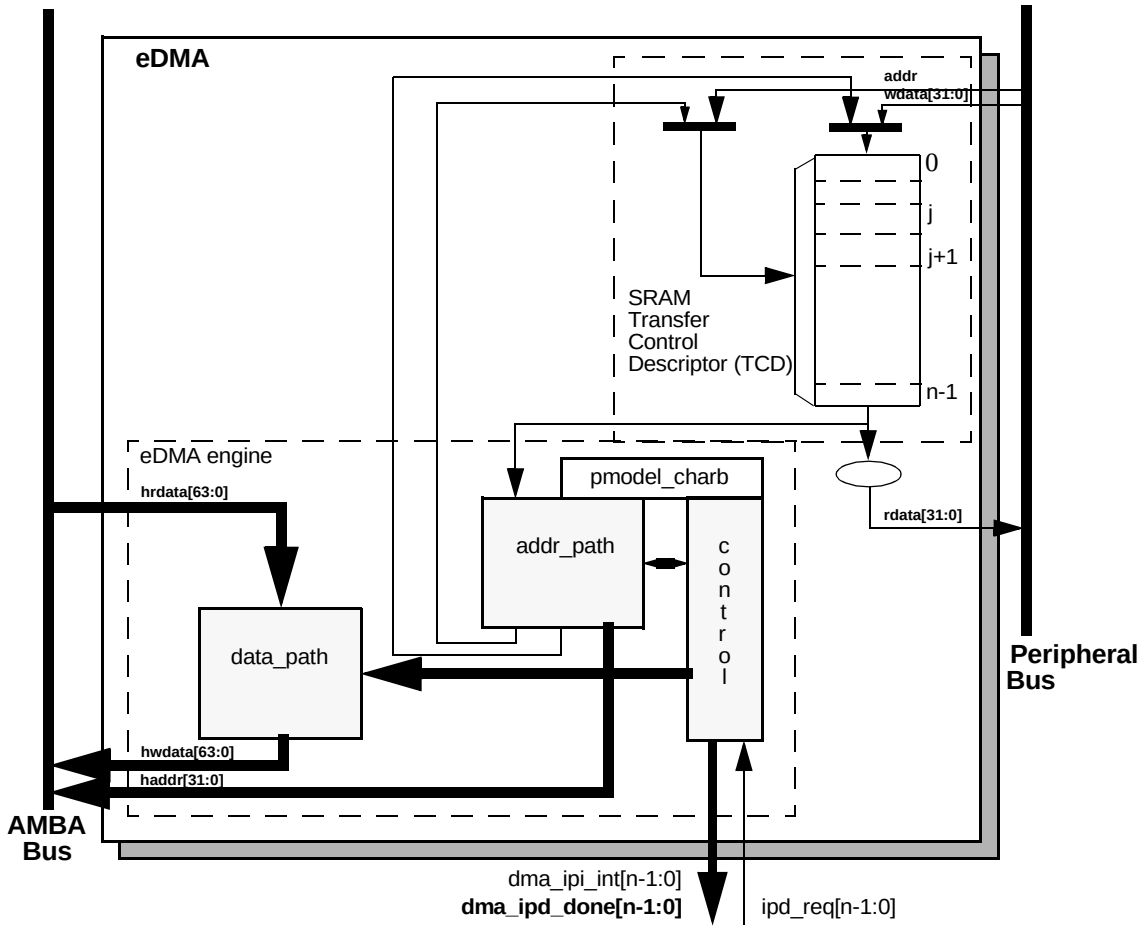


Figure 224. eDMA operation, part 2

Once the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the addr_path logic performs the required updates to certain fields in the channel's TCD, e.g., saddr, daddr, citer. If the outer major iteration count is exhausted, then there are additional operations which are performed. These include the final address adjustments and reloading of the biter field into the citer. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in *Figure 225*.

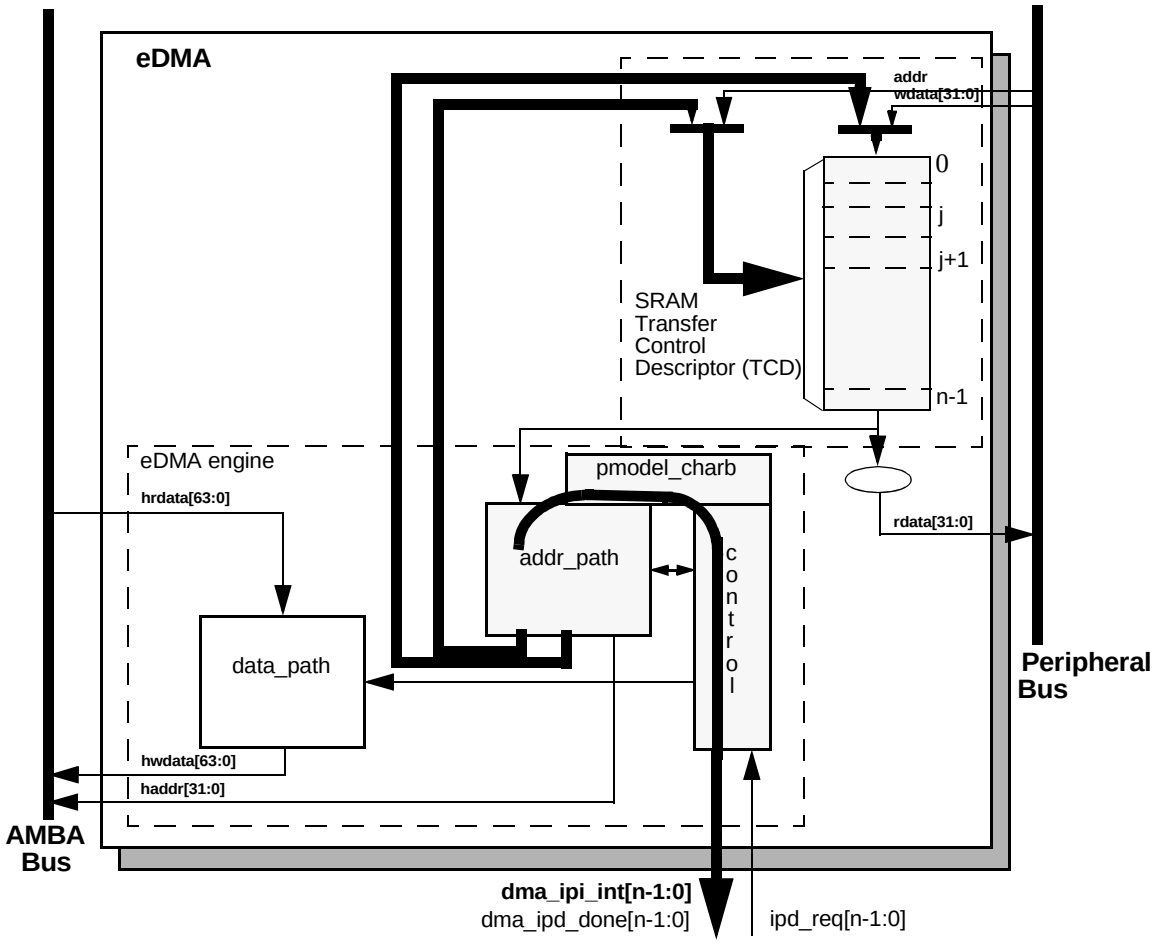


Figure 225. eDMA operation, part 3

20.3.3 eDMA performance

This section addresses the performance of the eDMA module, focusing on two separate metrics. In the traditional data movement context, performance is best expressed as the peak data transfer rates achieved using the eDMA. In most implementations, this transfer rate is limited by the speed of the source and destination address spaces. In a second context where device-paced movement of single data values to/from peripherals is dominant, a measure of the requests 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 210*. The following assumptions apply to *Table 210* and *Table 211*:

- 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 210 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 210. 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 211*. This metric represents millions of requests per second.

Table 211. 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.

20.4 Initialization/application information

20.4.1 eDMA initialization

A typical initialization of the eDMA is:

1. Write the DMACR register if a configuration other than the default is desired.
2. Write the channel priority levels into the DCHPRIn registers if a configuration other than the default is desired.
3. Enable error interrupts in the DMAEEI registers if so desired.
4. Write the 32 byte TCD for each channel that may request service.
5. Enable any hardware service requests via the DMAERQ register.
6. Request channel service by either software (setting the TCD.start bit) or by hardware (slave device asserting its ipd_req signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The eDMA engine will read the entire TCD for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the AHB bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.saddr) to the destination (as defined by the destination address, TCD.daddr) continue until the specified number of bytes (TCD.nbytes) have been transferred. When the transfer is complete, the eDMA engine's local TCD.saddr, TCD.daddr, and TCD.citer are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed, i.e. interrupts, major loop channel linking, and scatter/gather operations, if enabled.

20.4.2 eDMA programming errors

The eDMA performs various tests on the Transfer Control Descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of two errors; Channel Priority Error, GPE and CPE in the DMAES register respectively.

For all error types other than Channel Priority Errors, the channel number causing the error is recorded in the DMAES register. If the error source is not removed before the next activation of the problem channel, the error will be detected and recorded again.

In general, if priority levels are not unique, the highest channel priority that has an active request will be selected, but the lowest numbered channel with that priority will be selected by arbitration and executed by the eDMA engine. The hardware service request handshake signals, error interrupts and error reporting will be associated with the selected channel.

20.4.3 eDMA arbitration mode considerations

20.4.3.1 Fixed channel arbitration

In this mode, the channel service request from the highest priority channel will be selected to execute. If the eDMA is programmed so the channels use "fixed" priorities.

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.

20.4.3.2 Round-robin channel arbitration

Channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned.

This scenario could cause the same bandwidth consumption problem as indicated in *Section 20.4.3.1, Fixed channel arbitration,* but all the channels will be serviced.

20.4.4 eDMA transfer

20.4.4.1 Single request

To perform a simply transfer of ‘n’ bytes of data with one activation, set the major loop to one (TCD.citer = TCD.biter = 1). The data transfer will begin after the channel service request is acknowledged and the channel is selected to execute. Once 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 (TCD.word7 should be written last after all other fields have been initialized)

All other TCD fields = 0

This generates the following sequence of events:

1. IPS 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, DMAINT[n] = 1
8. The channel retires

The eDMA goes idle or services next channel.

20.4.4.2 Multiple requests

The next example is the same as the previous example, with the exception of transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The DMAis programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests is enabled in the DMAERQ register, channel service requests are initiated by the slave device.

TCD.citer = TCD.biter = 2

TCD.slast = -32

TCD.dlast_sga = -32

This would generate the following sequence of events:

1. First hardware (ipd_req) 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 next channel.

9. Second hardware (ipd_req) 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, DMAINT[n] = 1
16. The channel retires -> *major loop complete*

The eDMA goes idle or services the next channel.

20.4.5 TCD status

20.4.5.1 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 shown 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 one. 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
   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. ipd_req 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
   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.

20.4.5.2 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.

20.4.5.3 Preemption status

Preemption is only available when *fixed* arbitration is selected for 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 channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel priorities are treated as equal (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.

The worst case latency when switching to a preempt channel is the summation of:

- arbitration latency (2 cycles)
- bandwidth control stalls (if enabled)
- the time to execute two read/write sequences (including AHB bus holds; a system dependency driven by the slave devices or the crossbar)

20.4.6 Channel linking

Channel linking (or chaining) is a mechanism where one channel sets the TCD.start bit of another channel (or itself) thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.citer.e_link field are used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the major 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: The TCD.citer.e_link bit and the TCD.biter.e_link bit must equal or a configuration error will be reported. The citer and biter vector widths must be equal to calculate the major loop, half-way done interrupt point.

20.4.7 Dynamic programming

This section provides recommended methods to change the programming model during channel execution.

20.4.7.1 Dynamic priority changing

The following two options are recommended for dynamically changing ***channel*** priority levels:

1. Switch to round-robin channel arbitration mode, change the channel priorities, then switch back to fixed arbitration mode.
2. Disable all the channels, then change the channel priorities, then enable the appropriate channels.

20.4.7.2 Dynamic channel linking and dynamic scatter/gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCD.major.e_link or TCD.e_sg bits during channel execution. These bits are read from the TCD local memory at the ***end*** of channel execution thus allowing the user to enable either feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the TCD.major.e_link bit at the same time the eDMA engine is retiring the channel. The TCD.major.e_link would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The following coherency model is recommended when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the TCD.major.e_link bit.
2. Read back the TCD.major.e_link bit.
3. Test the TCD.major.e_link request status:
   a) If the bit is set, the dynamic link attempt was successful.
   b) If the bit is cleared, the attempted dynamic link did not succeed, the channel was already retiring.

This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the TCD.major.e_link and TCD.e_sg bits to zero on any writes to a channel's TCD.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 or TCD.e_sg bits. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

21 Enhanced Motor Control Timer (eTimer)

21.1 Introduction

This device contains up to three[i] eTimer modules, referred to as eTimer_0, eTimer_1, and eTimer_2.

All eTimer modules have 6 channels. eTimer_0 also has a watchdog timer function.

Each 16 bit counter/timer channel contains a prescaler, a counter, a load register, a hold register, two queued capture registers, two compare registers, two compare preload registers, and four control registers.

Note: This document uses the terms "Timer" and "Counter" interchangeably because the counter/timers may perform either or both tasks.

The Load register provides the initialization value to the counter when the counter's terminal value has been reached. For true modulo counting the counter can also be initialized by the CMPLD1 or CMPLD2 registers.

The Hold register captures the counter's value when other counters are being read. This feature supports the reading of cascaded counters coherently.

The Capture registers enable an external signal to take a "snap shot" of the counter's current value.

The COMP1 and COMP2 registers provide the values to which the counter is compared. If a match occurs, the OFLAG signal can be set, cleared, or toggled. At match time, an interrupt is generated if enabled, and the new compare value is loaded into the COMP1 or COMP2 registers from CMPLD1 and CMPLD2 if enabled.

The Prescaler provides different time bases useful for clocking the counter/timer.

The Counter provides the ability to count internal or external events.

Within the eTimer module (set of 6 timer/counter channels) the input pins are shareable.

The eTimer block diagram is shown in *Figure 226*.

i. On the 144 LQFP package, eTimer_2 is available internally only without any external I/O access.

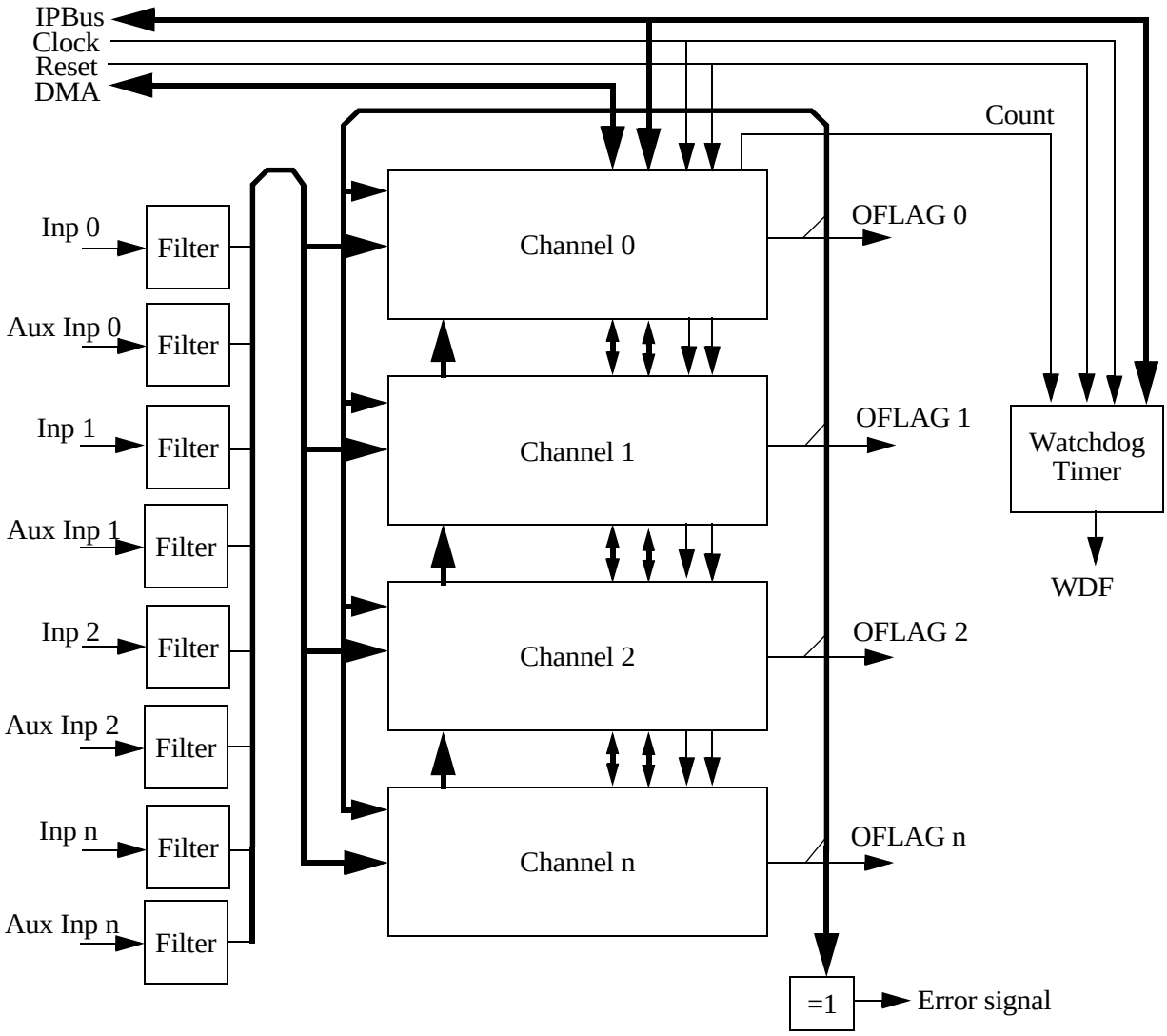


Figure 226. eTimer block diagram

Each of the timer/counter channels within the eTimer are shown in *Figure 227*.

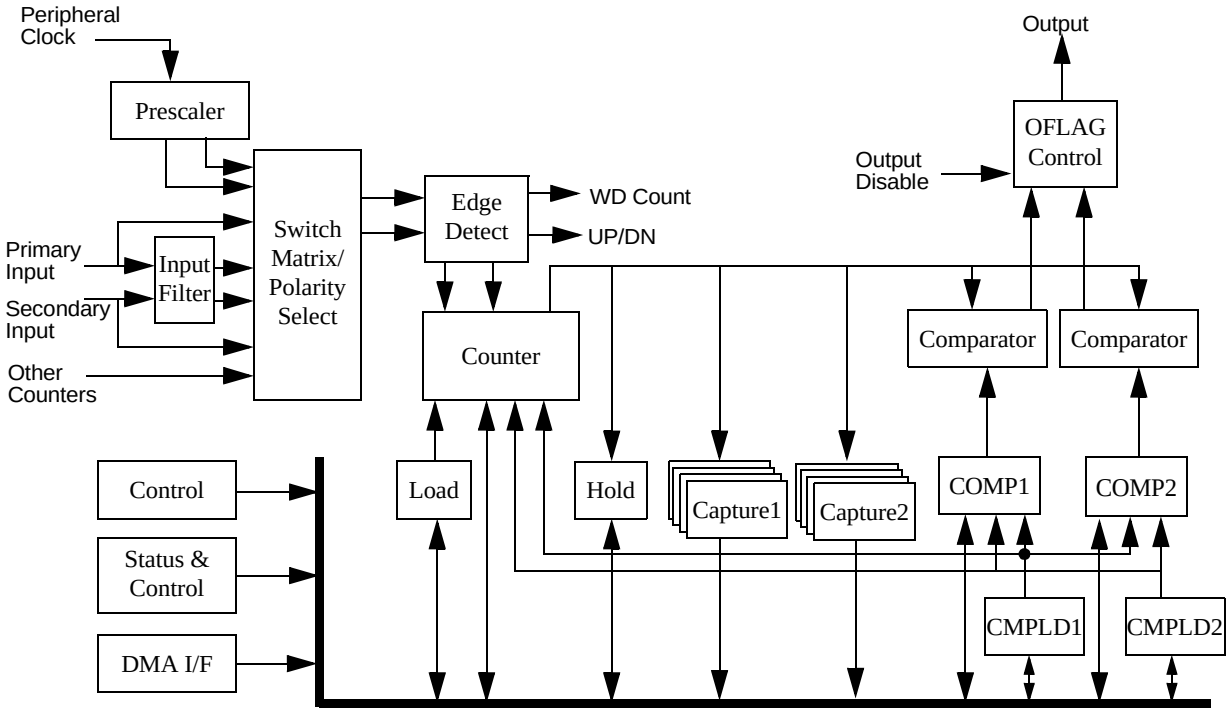


Figure 227. eTimer channel block diagram

21.2 Features

The eTimer module design includes these distinctive features:

- 6 16-bit counters/timers
- Count up/down
- Counters are cascadable
- Enhanced programmable up/down modulo counting
- Max count rate equals peripheral clock/2 for external clocks
- Max count rate equals peripheral clock for internal clocks
- Count once or repeatedly
- Counters are preloadable
- Compare registers are preloadable
- Counters can share available input pins
- Separate prescaler for each counter
- Each counter has capture and compare capability
- Continuous and single shot capture for enhanced speed measurement
- DMA support of capture registers and compare registers
- 24-bit watchdog capability to detect stalled quadrature counting

- OFLAG comparison for safety critical applications
- Programmable operation during debug mode and stop mode
- Programmable input filter
- Counting start can be synchronized across counters

21.3 External signal descriptions

The eTimer module has 6 external signals that can be used as either inputs or outputs. There are also 3 auxiliary inputs. The eTimer also interfaces to the Peripheral Bus.

21.3.1 TIO[5:0] - Timer Input/Outputs

These pins can be independently configured to be either timer input sources or output flags.

21.3.2 TAI[2:0] - Timer Auxiliary Inputs

These pins act as alternate input choices for the timer channels.

The module-to-module input signals are described in *Table 212*. Additional information on module-to-module interaction is shown in *Figure 98*.

Table 212. eTimer module-to-module input signals

Source (output ports)		Destination (input ports)	
Module name	**Port name**	**Module name**	**Port name**
CTU	ETIMER0_TRG	eTimer_0	AUX_0
eTimer_1	T2	eTimer_0	AUX_1
DSPI_1	SCK	eTimer_0	AUX_2
CTU	ETIMER1_TRG	eTimer_1	AUX_0
eTimer_0	T2	eTimer_1	AUX_1
FlexRAY	FR_CA_TX	eTimer_1	AUX_2
CTU	ETIMER2_TRG	eTimer_2	AUX_0
CTU	ETIMER3_TRG	eTimer_2	AUX_1

The eTimer module interacts with the CTU as described in *Section 14.4.1, Interaction with other peripherals*.

21.4 Memory map and register definition

21.4.1 Module memory map

Table 213. eTimer memory map

Address	Reg Name	Description
Timer channel registers (repeated for each channel as CHNL goes from 0 to 5)		
eTimer_BASE + ($20 * CHNL) + $0	COMP1	Compare Register 1
eTimer_BASE + ($20 * CHNL) + $2	COMP2	Compare Register 2
eTimer_BASE + ($20 * CHNL) + $4	CAPT1	Capture Register 1
eTimer_BASE + ($20 * CHNL) + $6	CAPT2	Capture Register 2
eTimer_BASE + ($20 * CHNL) + $8	LOAD	Load Register
eTimer_BASE + ($20 * CHNL) + $A	HOLD	Hold Register
eTimer_BASE + ($20 * CHNL) + $C	CNTR	Counter Register
eTimer_BASE + ($20 * CHNL) + $E	CTRL1	Control Register 1
eTimer_BASE + ($20 * CHNL) + $10	CTRL2	Control Register 2
eTimer_BASE + ($20 * CHNL) + $12	CTRL3	Control Register 3
eTimer_BASE + ($20 * CHNL) + $14	STS	Status Register
eTimer_BASE + ($20 * CHNL) + $16	INTDMA	Interrupt and DMA Enable Register
eTimer_BASE + ($20 * CHNL) + $18	CMPLD1	Comparator Load Register 1
eTimer_BASE + ($20 * CHNL) + $1A	CMPLD2	Comparator Load Register 2
eTimer_BASE + ($20 * CHNL) + $1C	CCCTRL	Compare and Capture Control Register
eTimer_BASE + ($20 * CHNL) + $1E	FILT	Input Filter Register
Watchdog timer registers		
eTimer_BASE + $100	WDTOL(1)	Watchdog Time-out Low Register
eTimer_BASE + $102	WDTOH(1)	Watchdog Time-out High Register
Configuration Registers		
eTimer_BASE + $10C	ENBL	Channel Enable Register
eTimer_BASE + $110	DREQ0	DMA Request 0 Select Register
eTimer_BASE + $112	DREQ1	DMA Request 1 Select Register

1. Exists only on eTimer_0

21.4.2 Register descriptions

The address of a register is the sum of a base address and an address offset. The base address is defined at the chip level and the address offset is defined at the module level. There are a set of registers for each timer channel, a set for the watchdog timer, a set for the fault inputs, and a set of configuration registers.

21.4.3 Timer Channel Registers

These registers are repeated for each timer channel. The base address of channel 0 is the same as the base address of the eTimer module as a whole. The base address of channel 1 is $20. This is the base address of the eTimer module plus an offset based on the number of bytes of registers in a timer channel. The base address of each subsequent timer channel is equal to the base address of the previous channel plus this same offset of $20.

21.4.3.1 Compare Register 1 (COMP1)

eTimer_CHN-L_BASE + $0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	COMP1[15:0]															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 228. Compare Register 1 (COMP1)

This read/write register stores the value used for comparison with the counter value. This register is not byte accessible. More explanation on the use of COMP1 can be found in *Section 21.5.2.13, Usage of Compare Registers.*

21.4.3.2 Compare Register 2 (COMP2)

eTimer_CHN-L_BASE + $2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	COMP2[15:0]															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 229. Compare Register 2 (COMP2)

This read/write register stores the value used for comparison with the counter value. This register is not byte accessible. More explanation on the use of COMP2 can be found in *Section 21.5.2.13, Usage of Compare Registers.*

21.4.3.3 Capture Register 1 (CAPT1)

eTimer_CHN-L_BASE + $4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	CAPT1[15:0]															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 230. Capture Register 1 (CAPT1)

This read only register stores the value captured from the counter. Exactly when a capture occurs is defined by the CPT1MODE bits. This is actually a 2-deep FIFO and not a single register. This register is not byte accessible.

21.4.3.4 Capture Register 2 (CAPT2)

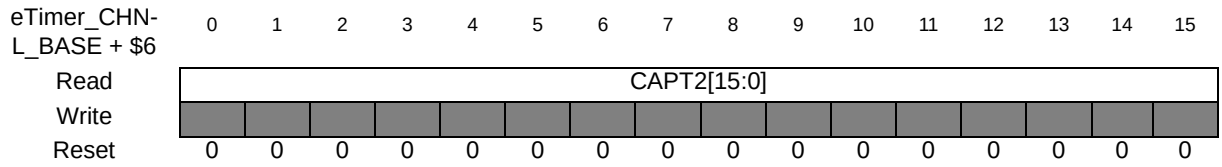


Figure 231. Capture Register 2 (CAPT2)

This read only register stores the value captured from the counter. Exactly when a capture occurs is defined by the CPT2MODE bits. This is actually a 2-deep FIFO and not a single register. This register is not byte accessible.

21.4.3.5 Load Register (LOAD)

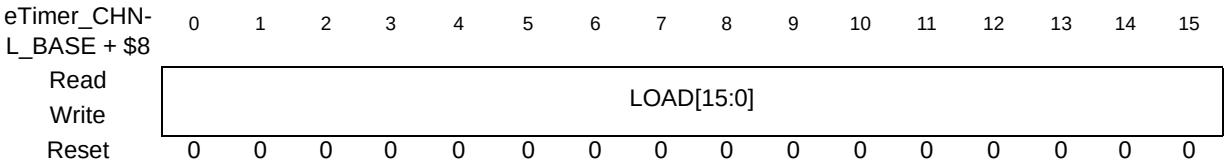


Figure 232. Load Register (LOAD)

This read/write register stores the value used to initialize the counter. This register is not byte accessible.

21.4.3.6 Hold Register (HOLD)

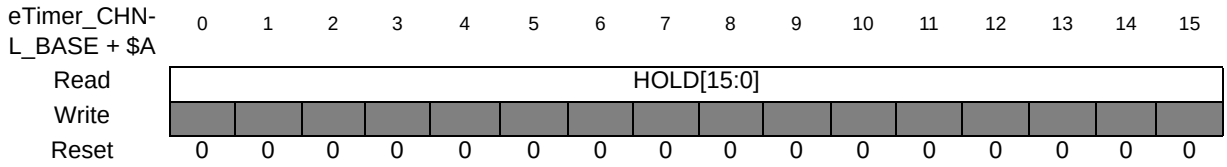


Figure 233. Hold Register (HOLD)

This read only register stores the counter's value whenever any of the other counters within a module are read. This is used to support coherent reading of cascaded counters.

In addition, this read only register stores the counter's value if any of the Compare and Capture Control Registers (CCCTRL) within a module are read.

21.4.3.7 Counter Register (CNTR)

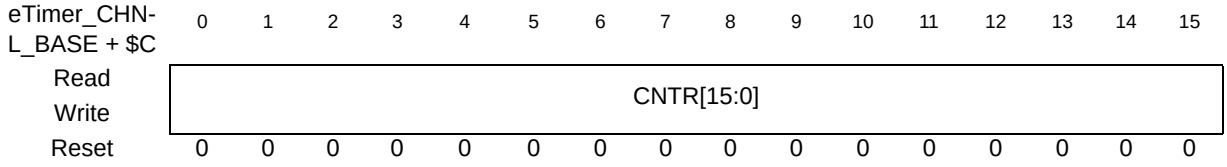


Figure 234. Counter (CNTR)

This read/write register is the counter for this channel of the timer module. This register is not byte accessible.

21.4.3.8 Control Register 1 (CTRL1)

eTimer_CHN-L_BASE + $E	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Write	CNTMODE[2:0]			PRISRC[4:0]					ONCE	LENGTH	DIR	SECSRC[4:0]				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 235. Control Register 1 (CTRL1)

CNTMODE - Count Mode

These bits control the basic counting and behavior of the counter.

Table 214. Count Mode Values

Value	Meaning
000	No Operation
001	Count rising edges of primary source(1)
010	Count rising and falling edges of primary source(2)
011	Count rising edges of primary source while secondary input high active
100	Quadrature count mode, uses primary and secondary sources
101	Count primary source rising edges, secondary source specifies direction (1 = minus)(3)
110	Edge of secondary source triggers primary count till compare
111	Cascaded counter mode, up/down(4)

1. Rising edges counted only when PIPS = 0. Falling edges counted when PIPS = 1. If primary count source is IP bus clock, only rising edges are counted regardless of PIPS value.
2. IP Bus clock divide by 1 can not be used as a primary count source in edge count mode.
3. Rising edges counted only when PIPS = 0. Falling edges counted when PIPS = 1.
4. Primary count source must be set to one of the counter outputs.

PRISRC - Primary Count Source

These bits select the primary count source.

Table 215. Primary Count Source Values

Value	Meaning
00000	Counter #0 input pin
00001	Counter #1 input pin
00010	Counter #2 input pin
00011	Counter #3 input pin
00100	Counter #4 input pin

Table 215. Primary Count Source Values (continued)

Value	Meaning
00101	Counter #5 input pin
00110	Reserved
00111	Reserved
01000	Auxiliary input #0 pin
01001	Auxiliary input #1 pin
01010	Auxiliary input #2 pin
01011	Reserved
01100	Reserved
01101	Reserved
01110	Reserved
01111	Reserved
10000	Counter #0 output
10001	Counter #1 output
10010	Counter #2 output
10011	Counter #3 output
10100	Counter #4 output
10101	Counter #5 output
10110	Reserved
10111	Reserved
11000	IP Bus clock divide by 1 prescaler
11001	IP Bus clock divide by 2 prescaler
11010	IP Bus clock divide by 4 prescaler
11011	IP Bus clock divide by 8 prescaler
11100	IP Bus clock divide by 16 prescaler
11101	IP Bus clock divide by 32 prescaler
11110	IP Bus clock divide by 64 prescaler
11111	IP Bus clock divide by 128 prescaler

Note: A timer selecting its own output as its primary count source is not a legal choice. The result is no counting.

ONCE - Count Once

This bit selects continuous or one shot counting mode.

1 = Count until compare and then stop. When output mode $4 is used, the counter re-initializes after reaching the COMP1 value and continues to count to the COMP2 value then stops.
0 = Count repeatedly.

LENGTH - Count Length

This bit determines whether the counter counts to the compare value and then re-initializes itself to the value specified in the LOAD, CMPLD1, or CMPLD2 registers, or the counter continues counting past the compare value, to the binary roll over.

1 = Count until compare, then reinitialize.
The value that the counter is reinitialized with depends on the settings of CLC1 and CLC2. If neither of these indicates the counter is to be loaded from one of the CMPLD registers, then the LOAD register is used to reinitialize the counter upon matching either COMP register. If one of CLC1 or CLC2 indicates that the counter is to be loaded from one of the CMPLD registers, then the counter will reinitialize to the value in the appropriate CMPLD register upon a match with the appropriate COMP register. If both of the CLC1 and CLC2 fields indicate that the counter is to be loaded from the CMPLD registers, then CMPLD1 will have priority if both compares happen at the same value.
When output mode $4 is used, alternating values of COMP1 and COMP2 are used to generate successful comparisons. For example, the counter counts until COMP1 value is reached, re-initializes, then counts until COMP2 value is reached, re-initializes, then counts until COMP1 value is reached, etc.
0 = Continue counting to roll over.

DIR - Count Direction

This bit selects either the normal count direction *up*, or the reverse direction, *down*.

1 = Count down
0 = Count up

SECSRC - Secondary Count Source

These bits identify the source to be used as a count command or timer command. The selected input can trigger the timer to capture the current value of the CNTR register. The selected input can also be used to specify the count direction. The polarity of the signal can be inverted by the SIPS bit of the CTRL2 register.

Table 216. Secondary Count Source Values

Value	Meaning
00000	Counter #0 input pin
00001	Counter #1 input pin
00010	Counter #2 input pin
00011	Counter #3 input pin
00100	Counter #4 input pin
00101	Counter #5 input pin
00110	Reserved
00111	Reserved
01000	Auxiliary input #0 pin
01001	Auxiliary input #1 pin
01010	Auxiliary input #2 pin
01011	Reserved

Table 216. Secondary Count Source Values (continued)

Value	Meaning
01100	Reserved
01101	Reserved
01110	Reserved
01111	Reserved
10000	Counter #0 output
10001	Counter #1 output
10010	Counter #2 output
10011	Counter #3 output
10100	Counter #4 output
10101	Counter #5 output
10110–11111	Reserved

21.4.3.9 Control Register 2 (CTRL2)

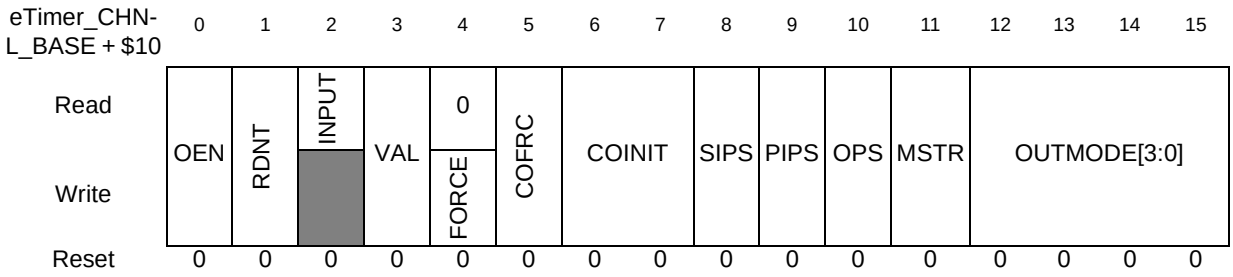


Figure 236. Control Register 2 (CTRL2)

OEN - Output Enable.

This bit determines the direction of the external pin.

1 = OFLAG output signal will be driven on the external pin. Other timer channels using this external pin as their input will see the driven value. The polarity of the signal will be determined by the OPS bit.
0 = The external pin is configured as an input.

RDNT - Redundant Channel Enable.

This bit enables redundant channel checking between adjacent channels (0 and 1, 2 and 3, 4 and 5). When this bit is clear, the RCF bit in this channel cannot be set. When this bit is set, the RCF bit will be set by a miscompare between the OFLAG of this channel and the OFLAG of its redundant adjacent channel which will cause the output of this channel to go inactive (logic 0 prior to consideration of the OPS bit).

1 = Enable redundant channel checking.
0 = Disable redundant channel checking.

INPUT - External input signal

This read only bit reflects the current state of the signal selected via SECSRC after application of the SIPS bit and filtering.

VAL - Forced OFLAG Value

This bit determines the value of the OFLAG output signal when a software triggered FORCE command occurs.

FORCE- Force the OFLAG output

This write only bit forces the current value of the VAL bit to be written to the OFLAG output. This bit always reads as a zero. The VAL and FORCE bits can be written simultaneously in a single write operation. Write to the FORCE bit only if OUTMODE is 0000 (software controlled). Setting this bit while the OUTMODE is a different value may yield unpredictable results.

COFRC - Co-channel OFLAG Force

This bit enables the compare from another channel within the module to force the state of this counter's OFLAG output signal.

1 = Other channels may force the OFLAG of this channel.
0 = Other channels cannot force the OFLAG of this channel.

COINIT - Co-channel Initialization

These bits enable another channel within the module to force the re-initialization of this channel when the other channel has an active compare event.

Table 217. Values for Co-channel Initilization

Value	Meaning
00	Other channels cnnot force re-initialization of this channel.
01	Other channels may force a re-initialization of this channel's counter using the LOAD reg.
10	Other channels may force a re-initialization of this channel's counter with the CMPLD2 reg when this channel is counting down or the CMPLD1 reg when this channel is counting up.
11	Reserved.

SIPS - Secondary Source Input Polarity Select

This bit inverts the polarity of the signal selected by the SECSRC bits.

1 = Inverted polarity.
0 = True polarity.

PIPS - Primary Source Input Polarity Select

This bit inverts the polarity of the signal selected by the PRISRC bits. This only applies if the signal selected by PRISRC is not the prescaled IP Bus clock.

1 = Inverted polarity.
0 = True polarity.

OPS - Output Polarity Select.

This bit inverts the OFLAG output signal polarity.

1 = Inverted polarity.
0 = True polarity.

MSTR - Master Mode

This bit enables the compare function's output to be broadcasted to the other channels in the module. The compare signal then can be used to reinitialize the other counters and/or force their OFLAG signal outputs.

1 = Enable broadcast of compare events from this channel.
0 = Disable broadcast of compare events from this channel.

OUTMODE - Output Mode

These bits determine the mode of operation for the OFLAG output signal.

Table 218. OUTMODE Values

Value	Meaning
0000	Software controlled
0001	Clear OFLAG output on successful compare (COMP1 or COMP2)
0010	Set OFLAG output on successful compare (COMP1 or COMP2)
0011	Toggle OFLAG output on successful compare (COMP1 or COMP2)
0100	Toggle OFLAG output using alternating compare registers
0101	Set on compare with COMP1, cleared on secondary source input edge
0110	Set on compare with COMP2, cleared on secondary source input edge
0111	Set on compare, cleared on counter roll-over
1000	Set on successful compare on COMP1, clear on successful compare on COMP2
1001	Asserted while counter is active, cleared when counter is stopped.
1010	Asserted when counting up, cleared when counting down.
1011	Reserved
1100	Reserved
1101	Reserved
1110	Reserved
1111	Enable gated clock output while counter is active

21.4.3.10 Control Register 3 (CTRL3)

Figure 237. Control Register 3 (CTRL3)

eTimer_CHN-L_BASE + $12	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	STPEN	ROC		0	1	1	1	1	0	C2FCNT [1:0]		0	C1FCNT [1:0]		DBGEN [1:0]	
Write																
Reset	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

STPEN - Stop Actions Enable

This bit allows the tri-stating of the timer output during stop mode.

1 = Output enable is disabled during stop mode.
0 = Output enable is unaffected by stop mode.

ROC - Reload on Capture

These bits enable the capture function to cause the counter to be reloaded from the LOAD register.

Table 219. Values for Reload on Capture

Value	Meaning
00	Do not reload the counter on a capture event.
01	Reload the counter on a capture 1 event.
10	Reload the counter on a capture 2 event.
11	Reload the counter on both a capture 1 event and a capture 2 event.

C2FCNT - CAPT2 FIFO Word Count

This field reflects the number of words in the CAPT2 FIFO.

C1FCNT - CAPT1 FIFO Word Count

This field reflects the number of words in the CAPT1 FIFO.

DBGEN - Debug Actions Enable

These bits allow the counter channel to perform certain actions in response to the chip entering debug mode.

Table 220. Values for DBGEN

Value	Meaning
00	Continue with normal operation during debug mode. (default)
01	Halt channel counter during debug mode.
10	Force OFLAG to logic 0 (prior to consideration of the OPS bit) during debug mode.
11	Both halt counter and force OFLAG to 0 during debug mode.

21.4.3.11 Status Register (STS)

eTimer_CHN-L_BASE + $14	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	0	0	WDF	RCF	ICF2	ICF1	IEHF	IELF	TOF	TCF2	TCF1	TCF
Write							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 238. Status Register (STS)

WDF - Watchdog Time-out Flag

This bit is set when the watchdog times out by counting down to zero. The watchdog must be enabled for time-out to occur and channel 0 must be in quadrature decode count mode (CNTMODE = 100). This bit is cleared by writing a 1 to this bit position after either writing a non-zero value to WDTOL and/or WDTOH or exiting quadrature decode counting mode. This bit is only in channel 0.

RCF - Redundant Channel Flag

This bit is set when there is a miscompare between this channel's OFLAG value and the OFLAG value of the corresponding redundant channel. Corresponding channels are grouped together in the following pairs: 0 and 1, 2 and 3, or 4 and 5. This bit can only be set if the RDNT bit is set. This bit is cleared by writing a 1 to this bit position. This bit is only in even channels (0, 2, and 4).

ICF2 - Input Capture 2 Flag

This bit is set when an input capture event (as defined by CPT2MODE) occurs while the counter is enabled and the word count of the CAPT2 FIFO exceeds the value of the CFWM field. This bit is cleared by writing a one to this bit position if ICF2DE is clear (no DMA) or it is cleared automatically by the DMA access if ICF2DE is set (DMA).

ICF1 - Input Capture 1 Flag

This bit is set when an input capture event (as defined by CPT1MODE) occurs while the counter is enabled and the word count of the CAPT1 FIFO exceeds the value of the CFWM field. This bit is cleared by writing a one to this bit position if ICF1DE is clear (no DMA) or it is cleared automatically by the DMA access if ICF1DE is set (DMA).

IEHF - Input Edge High Flag

This bit is set when a positive input transition occurs (on an input selected by SECSRC) while the counter is enabled. This bit is cleared by writing a one to this bit position.

IELF - Input Edge Low Flag

This bit is set when a negative input transition occurs (on an input selected by SECSRC) while the counter is enabled. This bit is cleared by writing a one to this bit position.

TOF - Timer Overflow Flag

This bit is set when the counter rolls over its maximum value $FFFF or $0000 (depending on count direction). This bit is cleared by writing a one to this bit location.

TCF2 - Timer Compare 2 Flag

This bit is set when a successful compare occurs with COMP2. This bit is cleared by writing a one to this bit location.

TCF1 - Timer Compare 1 Flag

This bit is set when a successful compare occurs with COMP1. This bit is cleared by writing a one to this bit location.

TCF - Timer Compare Flag

This bit is set when a successful compare occurs. This bit is cleared by writing a one to this bit location.

21.4.3.12 Interrupt and DMA Enable Register (INTDMA)

eTimer_CHN-L_BASE + $16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	ICF2DE	ICF1DE	CMPLD2DE	CMPLD1DE	0	0	WDFIE	RCFIE	ICF2IE	ICF1IE	IEHFIE	IELFIE	TOFIE	TCF2IE	TCF1IE	TCFIE
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 239. Interrupt and DMA Enable Register (INTDMA)

ICF2DE - Input Capture 2 Flag DMA Enable

Setting this bit enables DMA read requests for CAPT2 when the ICF2 bit is set. Do not set both this bit and the ICF2IE bit.

ICF1DE - Input Capture 1 Flag DMA Enable

Setting this bit enables DMA read requests for CAPT1 when the ICF1 bit is set. Do not set both this bit and the ICF1IE bit.

CMPLD2DE - Comparator Load Register 2 Flag DMA Enable

Setting this bit enables DMA write requests to the CMPLD2 register whenever data is transferred out of the CMPLD2 reg into either the CNTR, COMP1, or COMP2 registers.

CMPLD1DE - Comparator Load Register 1 Flag DMA Enable

Setting this bit enables DMA write requests to the CMPLD1 register whenever data is transferred out of the CMPLD1 reg into either the CNTR, COMP1, or COMP2 registers.

WDFIE - Watchdog Flag Interrupt Enable

Setting this bit enables interrupts when the WDF bit is set. This bit is only in channel 0.

RCFIE - Redundant Channel Flag Interrupt Enable

Setting this bit enables interrupts when the RCF bit is set. This bit is only in even channels (0, 2, and 4).

ICF2IE - Input Capture 2 Flag Interrupt Enable

Setting this bit enables interrupts when the ICF2 bit is set. Do not set both this bit and the ICF2DE bit.

ICF1IE - Input Capture 1 Flag Interrupt Enable

Setting this bit enables interrupts when the ICF1 bit is set. Do not set both this bit and the ICF1DE bit.

IEHFIE - Input Edge High Flag Interrupt Enable

Setting this bit enables interrupts when the IEHF bit is set.

IELFIE - Input Edge Low Flag Interrupt Enable

Setting this bit enables interrupts when the IELF bit is set.

TOFIE - Timer Overflow Flag Interrupt Enable

Setting this bit enables interrupts when the TOF bit is set.

TCF2IE - Timer Compare 2 Flag Interrupt Enable

Setting this bit enables interrupts when the TCF2 bit is set.

TCF1IE - Timer Compare 1 Flag Interrupt Enable

Setting this bit enables interrupts when the TCF1 bit is set.

TCFIE - Timer Compare Flag Interrupt Enable

Setting this bit enables interrupts when the TCF bit is set.

21.4.3.13 Comparator Load Register 1 (CMPLD1)

eTimer_CHN-L_BASE + $18	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Write	CMPLD1[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 240. Comparator Load 1 (CMPLD1)

This read/write register is the preload value for the COMP1 register. This register can also be used to load into the CNTR register. This register is not byte accessible. More information on the use of this register can be found in *Section 21.5.2.14, Usage of Compare Load Registers.*

21.4.3.14 Comparator Load Register 2 (CMPLD2)

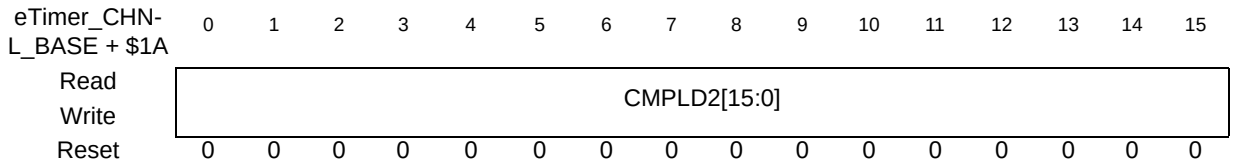


Figure 241. Comparator Load 2 (CMPLD2)

This read/write register is the preload value for the COMP2 register. This register can also be used to load into the CNTR register. This register is not byte accessible. More information on the use of this register can be found in *Section 21.5.2.14, Usage of Compare Load Registers.*

21.4.3.15 Compare and Capture Control Register (CCCTRL)

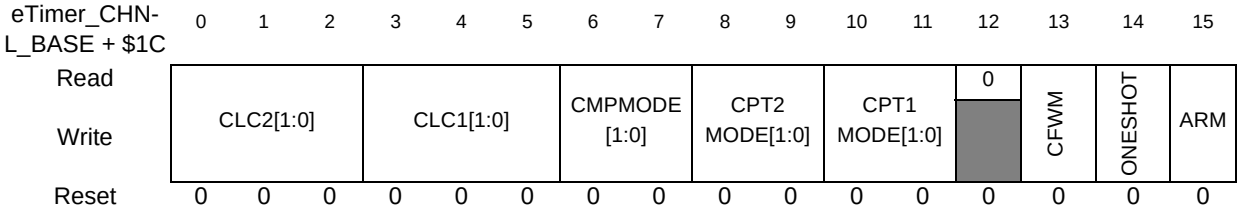


Figure 242. Compare and Capture Control Register (CCCTRL)

CLC2 - Compare Load Control 2

These bits control when COMP2 is preloaded. It also controls the loading of CNTR.

Table 221. Values for Compare Load Control 2

Value	Meaning
000	Never preload
001	Reserved
010	Load COMP2 with CMPLD1 upon successful compare with the value in COMP1.
011	Load COMP2 with CMPLD1 upon successful compare with the value in COMP2.
100	Load COMP2 with CMPLD2 upon successful compare with the value in COMP1.
101	Load COMP2 with CMPLD2 upon successful compare with the value in COMP2.
110	Load CNTR with CMPLD2 upon successful compare with the value in COMP1.
111	Load CNTR with CMPLD2 upon successful compare with the value in COMP2.

CLC1 - Compare Load Control 1

These bits control when COMP1 is preloaded. It also controls the loading of CNTR.

Table 222. Values for Compare Load Control 1

Value	Meaning
000	Never preload
001	Reserved
010	Load COMP1 with CMPLD1 upon successful compare with the value in COMP1.
011	Load COMP1 with CMPLD1 upon successful compare with the value in COMP2.
100	Load COMP1 with CMPLD2 upon successful compare with the value in COMP1.
101	Load COMP1 with CMPLD2 upon successful compare with the value in COMP2.
110	Load CNTR with CMPLD1 upon successful compare with the value in COMP1.
111	Load CNTR with CMPLD1 upon successful compare with the value in COMP2.

CMPMODE - Compare Mode

These bits control when the COMP1 and COMP2 registers are used in regards to the counting direction.

Table 223. Values for Compare Mode

Value	Meaning
00	COMP1 register is used when the counter is counting up. COMP2 register is used when the counter is counting up.
01	COMP1 register is used when the counter is counting down. COMP2 register is used when the counter is counting up.
10	COMP1 register is used when the counter is counting up. COMP2 register is used when the counter is counting down.
11	COMP1 register is used when the counter is counting down. COMP2 register is used when the counter is counting down.

CPT2MODE - Capture 2 Mode Control

These bits control the operation of the CAPT2 register as well as the operation of the ICF2 flag by defining which input edges cause a capture event. The input source is the secondary count source.

Table 224. Values for Capture 2 Mode Control

Value	Meaning
00	Disabled
01	Capture falling edges
10	Capture rising edges.
11	Capture any edge.

CPT1MODE - Capture 1 Mode Control

These bits control the operation of the CAPT1 register as well as the operation of the ICF1 flag by defining which input edges cause a capture event. The input source is the secondary count source.

Table 225. Values for Capture 1 Mode Control

Value	Meaning
00	Disabled
01	Capture falling edges
10	Capture rising edges.
11	Capture any edge.

CFWM - Capture FIFO Water Mark

This field represents the water mark level for the CAPT1 and CAPT2 FIFOs. The capture flags, ICF1 and ICF2, won't be set until the word count of the corresponding FIFO is greater than this water mark level.

ONESHOT - One Shot Capture Mode

This bit selects between free running and one shot mode for the input capture circuitry.

1 = One shot mode is selected.

If both capture circuits are enabled, then capture circuit 1 is armed first after the ARM bit is set. Once a capture occurs, capture circuit 1 is disarmed and capture circuit 2 is armed. After capture circuit 2 performs a capture, it is disarmed and the ARM bit is cleared. No further captures will be performed until the ARM bit is set again.

If only one of the capture circuits is enabled, then a single capture will occur on the enabled capture circuit and the ARM bit is then cleared.

0 = Free running mode is selected

If both capture circuits are enabled, then capture circuit 1 is armed first after the ARM bit is set. Once a capture occurs, capture circuit 1 is disarmed and capture circuit 2 is armed. After capture circuit 2 performs a capture, it is disarmed and capture circuit 1 is re-armed. The process continues indefinitely.

If only one of the capture circuits is enabled, then captures continue indefinitely on the enabled capture circuit.

ARM - Arm Capture

Setting this bit high starts the input capture process. This bit can be cleared at any time to disable input capture operation. This bit is self cleared when in one shot mode and the enabled capture circuit(s) has had a capture event(s).

1 = Input capture operation as specified by the CPT1MODE and CPT2MODE bits is enabled.

0 = Input capture operation is disabled.

21.4.3.16 Input Filter Register (FILT)

The FILT register programs the values for filtering of the corresponding inputs irrespective of the fact that any of the eTimer channel can use the input as a count source. The FILT register is also used to control the filtering of the corresponding auxiliary input.

eTimer_CHN-L_BASE + $1E	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	0	FILT_CNT[2:0]			FILT_PER[7:0]							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 243. Input Filter Register (FILT)

FILT_CNT - Input Filter Sample Count

These bits represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0 represents 3 samples. A value of 7 represents 10 samples. The value of FILT_CNT affects the input latency as described in *Section 21.4.3.16.1, Input Filter Considerations.*

FILT_PER - Input Filter Sample Period

These bits represent the sampling period (in IPBus clock cycles) of the eTimer input signal. Each input is sampled multiple times at the rate specified by FILT_PER. If FILT_PER is $00 (default), then the input filter is bypassed. The value of FILT_PER affects the input latency as described in *Section 21.4.3.16.1, Input Filter Considerations.*

21.4.3.16.1 Input Filter Considerations

The FILT_PER value should be set such that the sampling period is larger the period of the expected noise. This way a noise spike will only corrupt one sample. The FILT_CNT value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The probability of an incorrect transition is defined as the probability of an incorrect sample raised to the FILT_CNT + 3 power.

The values of FILT_PER and FILT_CNT must also be traded off against the desire for minimal latency in recognizing input transitions. Turning on the input filter (setting FILT_PER to a non-zero value) introduces a latency of: (((FILT_CNT + 3) x FILT_PER) + 2) IPBus clock periods.

21.4.4 Watchdog Timer Registers

The base address of the Watchdog Timer registers is equal to the base address of the eTimer plus an offset of $100.

21.4.4.1 Watchdog Time-out Registers (WDTOL and WDTOH)

These registers are available only on eTimer_0.

eTimer_BASE + $100	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	WDTOL															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 244. Watchdog Time-out Low Word Register (WDTOL)

eTimer_BASE + $102	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	WDTOH															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 245. Watchdog Time-out High Word Register (WDTOH)

WDTO - Watchdog Time-out

These registers are combined to form the 32 bit time-out count for the Timer watchdog function. This time-out count is used to monitor for inactivity on the inputs when channel 0 is in the quadrature decode count mode. The watchdog function is enabled whenever WDTO contains a non-zero value (although actual counting only occurs if channel 0 is in quadrature decode counting mode). The watchdog time-out down counter is loaded whenever WDTOH is written. These registers are not byte accessible. See *Section 21.5.3.5, Watchdog Timer,* for more information on the use of the watchdog timer.

21.4.5 Configuration Registers

The base address of the configuration registers is equal to the base address of the eTimer plus an offset of $10C.

21.4.5.1 Channel Enable Register (ENBL)

eTimer_BASE + $10C	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	0	0	0	0	0	0	ENBL[5:0]					
Write																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Figure 246. Channel Enable Register (ENBL)

ENBL - Timer Channel Enable

These bits enable the prescaler (if it is being used) and counter in each channel (ENBL[x] controls channel number x). Multiple ENBL bits can be set at the same time to synchronize the start of separate channels. If an ENBL bit is set, then the corresponding channel will start counting as soon as the CNTMODE field has a value other than 000. When an ENBL bit is clear, the corresponding channel maintains its current value.

1 = Timer channel is enabled. (default)
0 = Timer channel is disabled.

21.4.5.2 DMA Request Select Registers (DREQ0, DREQ1)

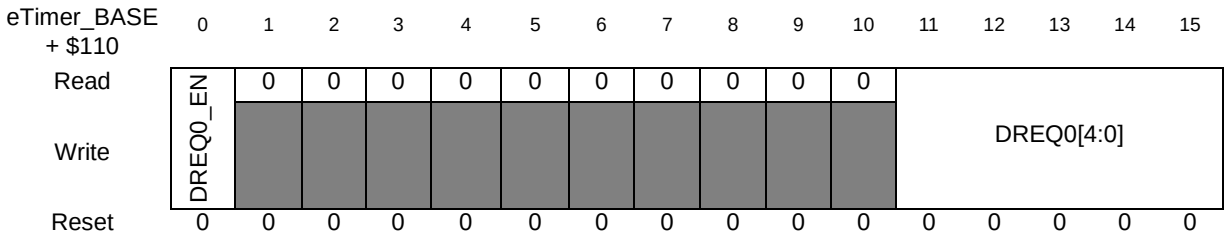


Figure 247. DMA Request 0 Select Register (DREQ0)

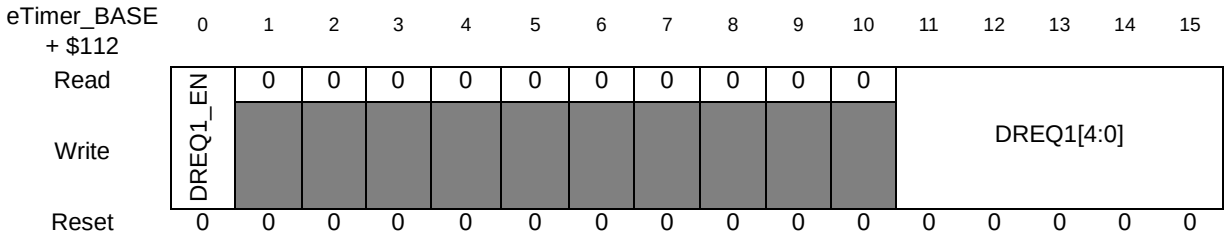


Figure 248. DMA Request 1 Select Register (DREQ1)

DREQn_EN- DMA Request

Each of these fields enables DMA request outputs. Program the DREQ fields prior to setting the corresponding enable bit. Clearing this enable bit will remove the request but won't clear the flag that is causing the request.

1 = DMA request enabled.

0 = DMA request disabled.

DREQn - DMA Request Select

Each of these two fields is used to select the source of one of the eTimer's DMA requests. Enable a DMA request in the channel specific INTDMA register and then use these registers to mux that request onto the module level DMA request outputs.

Table 226. Values for DREQn

Value	Selected DMA Request
00000	Channel 0 CAPT1 DMA read request
00001	Channel 0 CAPT2 DMA read request
00010	Channel 0 CMPLD1 DMA write request
00011	Channel 0 CMPLD2 DMA write request
00100	Channel 1 CAPT1 DMA read request
00101	Channel 1 CAPT2 DMA read request
00110	Channel 1 CMPLD1 DMA write request
00111	Channel 1 CMPLD2 DMA write request
01000	Channel 2 CAPT1 DMA read request

Table 226. Values for DREQn (continued)

Value	Selected DMA Request
01001	Channel 2 CAPT2 DMA read request
01010	Channel 2 CMPLD1 DMA write request
01011	Channel 2 CMPLD2 DMA write request
01100	Channel 3 CAPT1 DMA read request
01101	Channel 3 CAPT2 DMA read request
01110	Channel 3 CMPLD1 DMA write request
01111	Channel 3 CMPLD2 DMA write request
10000	Channel 4 CAPT1 DMA read request
10001	Channel 4 CAPT2 DMA read request
10010	Channel 4 CMPLD1 DMA write request
10011	Channel 4 CMPLD2 DMA write request
10100	Channel 5 CAPT1 DMA read request
10101	Channel 5 CAPT2 DMA read request
10110	Channel 5 CMPLD1 DMA write request
10111	Channel 5 CMPLD2 DMA write request
11000–11111	Reserved

21.5 Functional description

21.5.1 General

Each channel has two basic modes of operation: it can count internal or external events, or it can count an internal clock source while an external input signal is asserted, thus timing the width of the external input signal.

- The counter can count the rising, falling, or both edges of the selected input pin.
- The counter can decode and count quadrature encoded input signals.
- The counter can count up and down using dual inputs in a “count with direction” format.
- The counter’s terminal count value (modulo) is programmable.
  - The value that is loaded into the counter after reaching its terminal count is programmable.
- The counter can count repeatedly, or it can stop after completing one count cycle.
- The counter can be programmed to count to a programmed value and then immediately reinitialize, or it can count through the compare value until the count “rolls over” to zero.

The external inputs to each counter/timer are shareable among each of the six channels within the module. The external inputs can be used as:

- Count commands
- Timer commands
- They can trigger the current counter value to be “captured”
- They can be used to generate interrupt requests

The polarity of the external inputs is selectable.

The primary output of each channel is the output signal OFLAG. The OFLAG output signal can be:

- Set, cleared, or toggled when the counter reaches the programmed value.
- The OFLAG output signal may be output to an external pin instead of having that pin serve as a timer input.
- The OFLAG output signal enables each counter to generate square waves, PWM, or pulse stream outputs.
- The polarity of the OFLAG output signal is programmable.
- The response of the OFLAG output to a fault input is programmable.

Any channel can be assigned as a “Master”. A master’s compare signal can be broadcasted to the other channels within the module. The other channels can be configured to reinitialize their counters and/or force their OFLAG output signals to predetermined values when a Master channel’s compare event occurs.

21.5.2 Counting Modes

The selected external signals are sampled at the eTimer’s base clock rate and then run through a transition detector. The maximum count rate is one-half of the eTimer’s base clock rate when using an external signal. Internal clock sources can be used to clock the counters at the eTimer’s base clock rate.

If a counter is programmed to count to a specific value and then stop, the CNTMODE field in the CTRL1 register is cleared when the count terminates.

21.5.2.1 STOP Mode

If the CNTMODE field is set to ‘000’, the counter is inert. No counting will occur. Stop mode will also disable the interrupts caused by input transitions on a selected input pin.

21.5.2.2 COUNT Mode

If the CNTMODE field is set to ‘001’, the counter will count the rising edges of the selected clock source. This mode is useful for generating periodic interrupts for timing purposes, or counting external events such as “widgets” on a conveyor belt passing a sensor. If the selected input is inverted by setting the PIPS bit, then the negative edge of the selected external input signal is counted.

See *Section 21.5.2.9, CASCADE-COUNT Mode,* through *Section 21.5.2.12, VARIABLE-FREQUENCY PWM Mode,* for additional capabilities of this operating mode.

21.5.2.3 EDGE-COUNT Mode

If the CNTMODE field is set to ‘010’, the counter will count both edges of the selected external clock source. This mode is useful for counting the changes in the external environment such as a simple encoder wheel.

21.5.2.4 GATED-COUNT Mode

If the CNTMODE field is set to ‘011’, the counter will count while the selected secondary input signal is high. This mode is used to time the duration of external events. If the selected input is inverted by setting the PIPS bit, then the counter will count while the selected secondary input is low.

Note: In the following counting modes an incorrect counting of 1 tick is possible:

- GATED-COUNT mode, the CNTMODE field is '011' (count rising edges of primary source while secondary input is high);

-SIGNED-COUNT mode, the CNTMODE field is '101' (count primary source rising edges, secondary source specifies direction (up/down)).

Delays in the edge detection circuitry lead to behavior where the rising edge on the primary source is compared to the secondary source value one clock later in time. This means that if there is a rising edge on the primary source followed immediately by the secondary source going high, the eTimer logic could see this as a rising primary edge while the secondary is high even though the secondary input was low at the time of the rising primary edge. This behavior can occur when the transition on the secondary edge occurs within 1 IPBus clock cycle of the transition on the primary input. The counter will also increment if the primary source is already high when the secondary source goes high.

To prevent this problem the source selected as the secondary input to the eTimer channel needs to have an additional clock cycle of delay added to it. This can be done by using the input filters. For the primary source set FILT_PER==1 and FILT_CNT==0. For the secondary source set FILT_PER==1 and FILT_CNT==1. This will introduce a 5 clock cycle latency on the primary source and a 6 cycle latency on the secondary source which will properly align the two signals for count modes '011' and '101'.

To prevent this problem ensure that the primary source is low before the secondary source goes high to avoid a false count.

21.5.2.5 QUADRATURE-COUNT Mode

If the CNTMODE field is set to ‘100’, the counter will decode the primary and secondary external inputs as quadrature encoded signals. Quadrature signals are usually generated by rotary or linear sensors used to monitor movement of motor shafts or mechanical equipment. The quadrature signals are square waves that are 90 degrees out of phase. The decoding of quadrature signal provides both count and direction information.

Figure 249 shows a timing diagram illustrating the basic operation of a quadrature incremental position encoder.

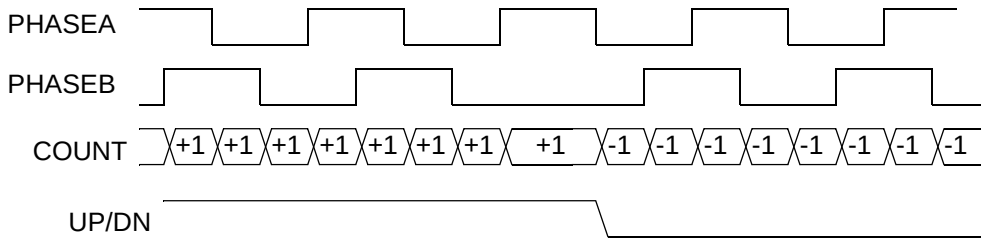


Figure 249. Quadrature Incremental Position Encoder

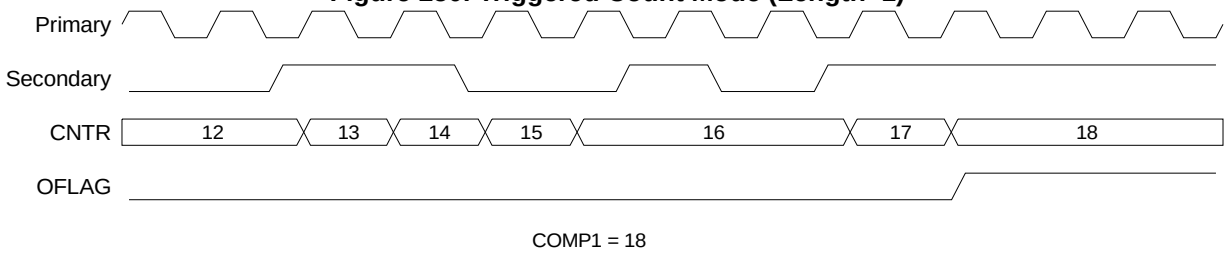
21.5.2.6 SIGNED-COUNT Mode

If the CNTMODE field is set to ‘101’, the counter counts the primary clock source while the selected secondary source provides the selected count direction (up/down).

21.5.2.7 TRIGGERED-COUNT Mode

If the CNTMODE field is set to ‘110’, the counter will begin counting the primary clock source after a positive transition (negative if SIPS = 1) of the secondary input occurs. The counting will continue until a compare event occurs or another positive input transition is detected. Subsequent secondary positive input transitions will continue to restart and stop the counting until a compare event occurs.

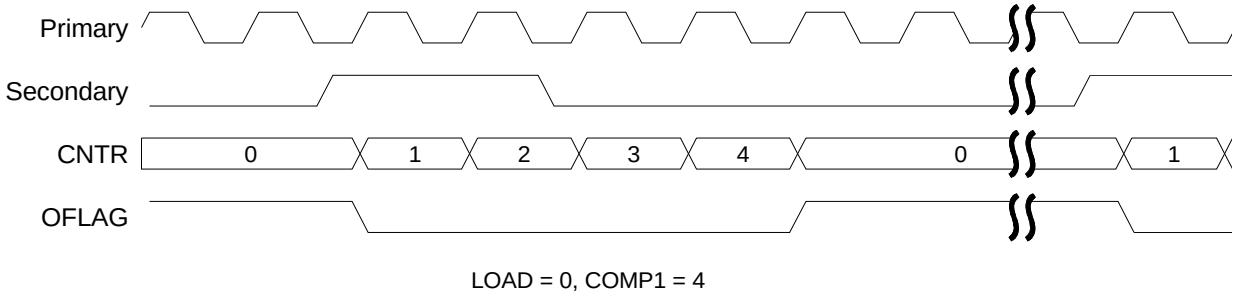
Figure 250. Triggered Count Mode (Length=1)



21.5.2.8 ONE-SHOT Mode

If the CNTMODE field is set to ‘110’, and the counter is set to reinitialize at a compare event (LENGTH =1), and the OFLAG OUTMODE is set to ‘0101’ (cleared on init, set on compare), the counter works in a “One-Shot Mode”. An external events causes the counter to count, when terminal count is reached, the output is asserted. This “delayed” output can be used to provide timing delays.

Figure 251. One-Shot Mode (Length=1)



21.5.2.9 CASCADE-COUNT Mode

If the CNTMODE field is set to ‘111’, the counter’s input is connected to the output of another selected counter. The counter will count up and down as compare events occur in the selected source counter. This “Cascade” or “Daisy-Chained” mode enables multiple counters to be cascaded to yield longer counter lengths. When operating in cascade mode, a special high speed signal path is used between modules rather than the OFLAG output signal. If the selected source counter is counting up and it experiences a compare event, the counter will be incremented. If the selected source counter is counting down and it experiences a compare event, the counter will be decremented.

Up to two counters may be cascaded to create a 32 bit wide synchronous counter.

Whenever any counter is read within a counter module, all of the counters’ values within the module are captured in their respective HOLD registers. This action supports the reading of a cascaded counter chain. First read any counter of a cascaded counter chain, then read the HOLD registers of the other counters in the chain. The cascaded counter mode is synchronous.

Note: It is possible to connect counters together by using the other (non-cascade) counter modes and selecting the outputs of other counters as a clock source. In this case, the counters are operating in a “ripple” mode, where higher order counters will transition a clock later than a purely synchronous design.

Note: A channel can be cascaded with any other channel, but you can’t cascade more than 2 channels together. You can create separate cascades of pairs of channels. For example, you can cascade channels 0 and 1 and separately cascade channels 6 and 5. You can’t cascade channels 0, 1, and 5.

21.5.2.10 PULSE-OUTPUT Mode

If the counter is setup for CNTMODE = 001, and the OFLAG OUTMODE is set to ‘1111’ (gated clock output), and the ONCE bit is set, then the counter will output a pulse stream of pulses that has the same frequency of the selected clock source, and the number of output pulses is equal to the compare value minus the init value. This mode is useful for driving step motor systems.

Note: This does not work if the PRISRC is set to 11000 (IP_bus/1).

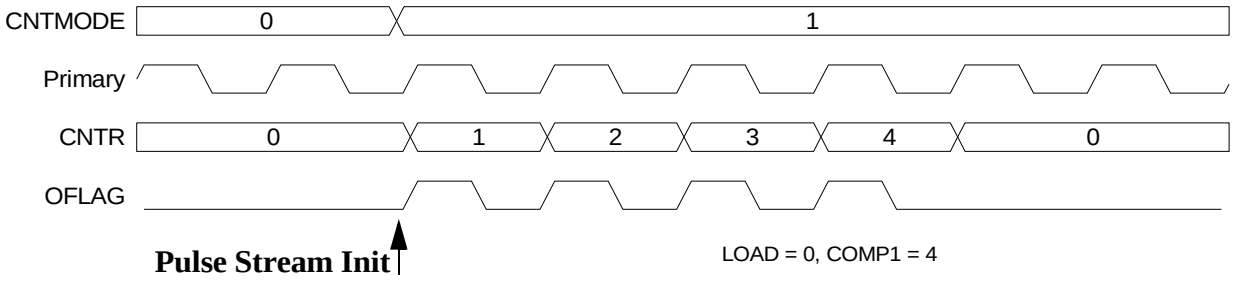


Figure 252. Pulse Output Mode

21.5.2.11 FIXED-FREQUENCY PWM Mode

If the counter is setup for CNTMODE = 001, count through roll-over (LENGTH = 0), continuous count (ONCE = 0) and the OFLAG OUTMODE is ‘0111’ (set on compare, cleared on counter roll-over) then the counter output yields a Pulse Width Modulated (PWM)

signal with a frequency equal to the count clock frequency divided by 65,536 and a pulse width duty cycle equal to the compare value divided by 65,536. This mode of operation is often used to drive PWM amplifiers used to power motors and inverters.

21.5.2.12 VARIABLE-FREQUENCY PWM Mode

If the counter is setup for CNTMODE = 001, count till compare (LENGTH = 1), continuous count (ONCE = 0) and the OFLAG OUTMMODE is '0100' (toggle OFLAG and alternate compare registers) then the counter output yields a Pulse Width Modulated (PWM) signal whose frequency and pulse width is determined by the values programmed into the COMP1 and COMP2 registers, and the input clock frequency. This method of PWM generation has the advantage of allowing almost any desired PWM frequency and/or constant on or off periods. This mode of operation is often used to drive PWM amplifiers used to power motors and inverters. The CMPLD1 and CMPLD2 registers are especially useful for this mode, as they allow the programmer time to calculate values for the next PWM cycle while the PWM current cycle is underway.

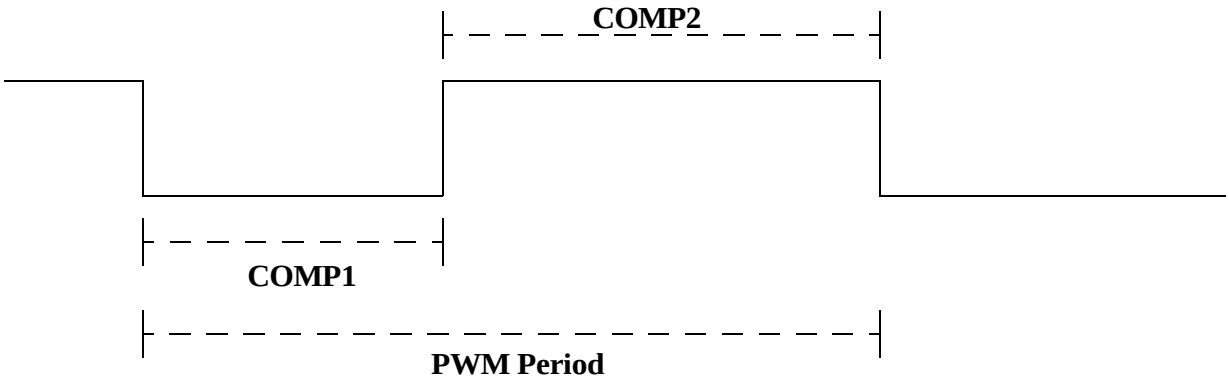
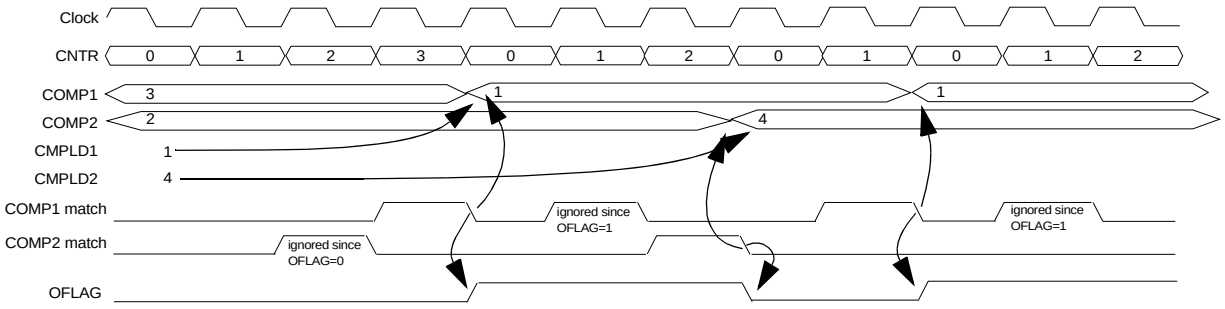


Figure 253. Variable PWM Waveform



CLC1=010, load COMP1 when CNTR=COMP1
CLC2=101, load COMP2 when CNTR=COMP2

Figure 254. Variable Frequency PWM Mode Timing

21.5.2.13 Usage of Compare Registers

The dual compare registers (COMP1 and COMP2) provide a bidirectional modulo count capability.

The COMP1 register should be set to the desired maximum count value or $FFFF to indicate the maximum unsigned value prior to roll-over, and the COMP2 register should be set to the minimum count value or $0000 to indicate the minimum unsigned value prior to roll-under.

If the output mode is set to 0100, the OFLAG will toggle while using alternating compare registers. In this variable frequency PWM mode, the COMP2 value defines the desired pulse width of the on time, and the COMP1 register defines the off time. COMP1 is used when OFLAG=0 and COMP2 is used when OFLAG=1. OFLAG can be forced to a value using CTRL2[FORCE] and CTRL2[VAL].

Use caution when changing COMP1 and COMP2 while the counter is active. If the counter has already passed the new value, it will count to $FFFF or $0000, roll over, then begin counting toward the new value. The check is: CNTR = COMPx, *not* CNTR > COMP1 or CNTR < COMP2.

The use of the CMPLD1 and CMPLD2 registers to preload compare values will help to minimize this problem.

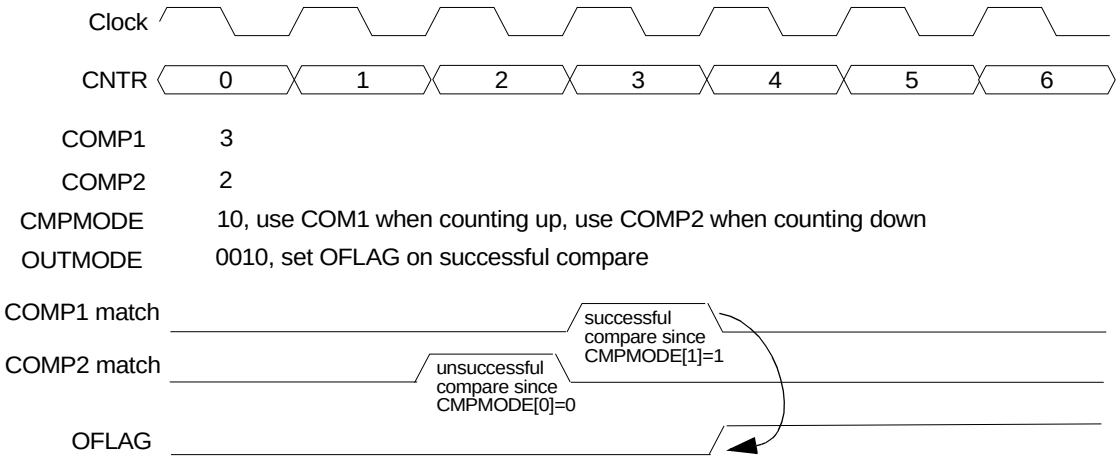


Figure 255. Compare Register and OFLAG Timing

21.5.2.14 Usage of Compare Load Registers

The CMPLD1, CMPLD2 and CCCTRL registers offer a high degree of flexibility for loading compare registers with user-defined values on different compare events. To ensure correct functionality while using these registers we strongly suggest using the following method described in this section.

The purpose of the compare load feature is to allow quicker updating of the compare registers. A compare register can be updated using interrupts. However, because of the latency between an interrupt event occurring and the service of that interrupt, there is the possibility that the counter may have already counted past the new compare value by the time the compare register is updated by the interrupt service routine. The counter would then continue counting until it rolled over and reached the new compare value.

To address this, the compare registers are updated in hardware in the same way the counter register is re-initialized to the value stored in the LOAD register. The compare load

feature allows the user to calculate new compare values and store them in to the comparator load registers. When a compare event occurs, the new compare values in the comparator load registers are written to the compare registers eliminating the use of software to do this.

The compare load feature is intended to be used in variable frequency PWM mode. The COMP1 register determines the pulse width for the logic low part of OFLAG and COMP2 determines the pulse width for the logic high part of OFLAG. The period of the waveform is determined by the COMP1 and COMP2 values and the frequency of the primary clock source. See *Figure 253*.

21.5.2.15 MODULO COUNTING Mode

To create a modulo counter using COMP1 and COMP2 as the counter boundaries (instead of $0000 and $FFFF), set the registers in the following manner. Set CNTMODE to either 100 (quadrature count mode) or 101 (count with direction mode). Use count until compare, then reinitialize (LENGTH = 1) and continuous count (ONCE = 0). Set COMP1 and CMPLD1 to the upper boundary value. Set COMP2 and CMPLD2 to the lower boundary value. Set CMPMODE = 10 (COMP1 is used when counting up and COMP2 is used when counting down). Set CLC2 = 110 (load CNTR with value of CMPLD2 on COMP1 compare) and CLC1 = 111 (load CNTR with value of CMPLD1 on COMP2 compare).

21.5.3 Other Features

21.5.3.1 Redundant OFLAG Checking

This mode allows the user to bundle two timer functions generating any pattern to compare their resulting OFLAG behaviors (output signal).

The redundant mode is used to support online checks for functional safety reasons. Whenever a mismatch between the two adjacent channels occurs, it is reported via an interrupt to the core and the two outputs are put into their inactive states. An error is flagged via the RCF flag.

This feature can be tested by forcing a transition on one of the OFLAGs using the VAL and FORCE bits of the channel.

21.5.3.2 Loopback Checking

This mode is always available in that one channel can generate an OFLAG while another channel uses the first channels OFLAG as its input to be measured and verified to be as expected.

21.5.3.3 Input Capture Mode

Input capture is used to measure pulse width (by capturing the counter value on two successive input edges) or waveform period (by capturing the counter value on two consecutive rising edges or two consecutive falling edges). The Capture Registers store a copy of the counter's value when an input edge (positive, negative, or both) is detected. The type of edge to be captured by each circuit is determined by the CPT1MODE and CPT2MODE bits whose functionality is listed in *Table 225*. Also, controlling the operation of the capture circuits is the arming logic which allows captures to be performed in a free running (continuous) or one shot fashion. In free running mode, the capture sequences will be performed indefinitely. If both capture circuits are enabled, they will work together in a ping-pong style where a capture event from one circuit leads to the arming of the other and

vice versa. In one shot mode, only one capture sequence will be performed. If both capture circuits are enabled, capture circuit 0 is first armed and when a capture event occurs, capture circuit 1 is armed. Once the second capture occurs, further captures are disabled until another capture sequence is initiated. Both capture circuits are also capable of generating an interrupt to the CPU.

21.5.3.4 Master/Slave Mode

Any timer channel can be assigned as a Master (MSTR = 1). A Master's compare signal can be broadcasted to the other channels within the module. The other counters can be configured to reinitialize their counters (COINIT = 1) and/or force their OFLAG output signals (COFRC = 1) to predetermined values when a Master counter compare event occurs.

21.5.3.5 Watchdog Timer

The watchdog timer is used to monitor for a stalled count when channel 0 is in quadrature count mode. When the watchdog is enabled, it loads the time-out value into a down counter. The down counter counts as long as channel 0 is in quadrature decode count mode. If this down counter reaches zero, an interrupt is asserted. The down counter is reloaded to the time-out value each time the counter value from channel 0 changes. If the channel 0 count value is toggling between two values (indicating a possibly stalled encoder), then the down counter is not reloaded.

The watchdog function of the eTimer counts using the peripheral bus clock which is the same clock as used by the eTimer counter channels. The watchdog function looks for counting activity in counter #0.

21.6 Interrupts

Each of the channels within the eTimer can generate an interrupt from several sources. The watchdog and the fault logic also generate interrupts. The interrupt service routine (ISR) must check the related interrupt enables and interrupt flags to determine the actual cause of the interrupt.

Table 227. Interrupt summary

Core interrupt	Interrupt flag	Interrupt enable	Name	Description
TC0IR–TC5IR(1)	TCF	TCFIE	Compare interrupt	Compare of counter and related compare register
	TCF1	TCF1IE	Compare 1 interrupt	Compare of the counter and COMP1 register
	TCF2	TCF2IE	Compare 2 interrupt	Compare of the counter and COMP2 register
	TOF	TOFIE	Overflow interrupt	Generated on counter roll-over or roll-under
	IELF	IELFIE	Input Low Edge interrupt	Falling edge of the secondary input signal
	IEHF	IEHFIE	Input High Edge interrupt	Rising edge of the secondary input signal
	ICF1	ICF1IE	Input Capture 1 interrupt	Input capture event for CAPT1
	ICF2	ICF2IE	Input Capture 2 interrupt	Input capture event for CAPT2

Table 227. Interrupt summary (continued)

Core interrupt	Interrupt flag	Interrupt enable	Name	Description
WTIF(2)	WDF	WDFIE	Watchdog time-out interrupt	Watchdog has timed out
RCF	RCF	RCFIE	Redundant Channel Fault interrupt	Miscompare with redundant channel

1. All flags are ORed together to generate the interrupt TC*n*IR.
2. Only for eTimer_0.

21.7 DMA

Each channel can request a DMA read access for each of the capture registers and a DMA write request for each of the compare preload registers. The DREQ registers select amongst these 24 DMA request sources to generate the 2 top level DMA request outputs.

When DMA is used to read the eTimer_CAPTn registers and the DMA has completed its programmed number of transfers, then the extra input capture events will set the eTimer_STS[ICFn] bits and also set the internal DMA request signal. While the ICFn bits can be cleared by writing a 1 to their bit positions, the DMA request can only be cleared by the internal DMA done signal. This means that when a new DMA transfer is programmed, the eTimer will request a DMA read with possibly unwanted data. In cases where extra eTimer input capture events might occur, the following procedure can be used to prevent unwanted DMA read requests: Upon completion of the DMA transfer, clear the eTimer_INTDMA[ICFnDE] bits.

Table 228. DMA Summary

DMA Request	DMA Enable	Name	Description
Channels 0-5	ICF1DE	CAPT1 read request	CAPT1 contains a value
	ICF2DE	CAPT2 read request	CAPT2 contains a value
	CMPLD1DE	CMPLD1 write request	CMPLD1 needs an update
	CMPLD2DE	CMPLD2 write request	CMPLD2 needs an update

22 Error Correction Status Module (ECSM)

22.1 Introduction

The Error Correction Status Module (ECSM) provides miscellaneous control functions for the device Standard Product Platform (SPP) including program-visible information about the platform configuration and revision levels, a reset status register, and wakeup control for exiting sleep modes, and optional features such as information on memory errors reported by error-correcting codes.

22.2 Overview

The Error Correction Status Module is mapped into the IPS space and supports a number of miscellaneous control functions for the platform device.

22.3 Features

The ECSM includes these features:

- Program-visible information on the platform device configuration and revision
- Reset status register (MRSR)
- Registers for capturing information on platform memory errors if error-correcting codes (ECC) are implemented
- Registers to specify the generation of single- and double-bit memory data inversions for test purposes if error-correcting codes are implemented

22.4 Memory map and register description

This section details the programming model for the Error Correction Status Module. This is an on-platform 128-byte space mapped to the region serviced by an IPS bus controller. Some of the control registers have a 64-bit width. These 64-bit registers are implemented as two 32-bit registers, and include an “H” and “L” suffixes, indicating the “high” and “low” portions of the control function.

The Error Correction Status Module does not include any logic which provides access control. Rather, this function is supported using the standard access control logic provided by the IPS controller.

22.4.1 Memory map

Table 229 is a 32-bit view of the ECSM’s memory map. The addresses presented here are the offsets relative to the controller base address 0xFFF4_0000. In DPM, ECSM_1 is located at base address 0x8FF4_0000.

Table 229. ECSM memory map

ECSM offset	Register	
0x0000	Processor Core Type (PCT)	Chip-defined Platform Revision (PLREV)

Table 229. ECSM memory map (continued)

ECSM offset	Register			
0x0004	Platform Crossbar Master Configuration (PLAMC)	Platform Crossbar Slave Configuration (PLASC)		
0x0008	IPS On-Platform Module Configuration (IOPMC)			
0x000C	Reserved			Misc Reset Status (MRSR)
0x0010–0x0023	Reserved			
0x0024	Miscellaneous User-Defined Control Register (MUDCR)			
0x0028	Reserved			
0x002C - 0x003C	Reserved			
0x0040	Reserved			ECC Configuration (ECR)
0x0044	Reserved			ECC Status (ESR)
0x0048	Reserved		ECC Error Generation (EEGR)	
0x004C	Reserved			
0x0050	Platform Flash Memory ECC Address (PFEAR)			
0x0054	Reserved		Platform Flash Memory ECC Master (PFEMR)	Platform Flash Memory ECC Attributes (PFEAT)
0x0058	Platform Flash Memory ECC Data High (PFEDRH)			
0x005C	Platform Flash Memory ECC Data Low (PFEDRL)			
0x0060	Platform RAM ECC Address (PREAR)			
0x0064	Reserved	Platform RAM ECC Syndrome (PRESR)	Platform RAM ECC Master (PREMR)	Platform RAM ECC Attributes (PREAT)
0x0068	Platform RAM ECC Data High (PREDRH)			
0x006C	Platform RAM ECC Data Low (PREDRL)			
0x0070 - 0x007C	Reserved			

22.4.2 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.

22.4.2.1 Processor Core Type (PCT) register

The PCT is a 16-bit read-only register specifying the architecture of the processor core in the device. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

See *Figure 256* and *Table 230* for the Processor Core Type definition.

Figure 256. Processor Core Type (PCT) Register

Register address: ECSM Base + 0x0000

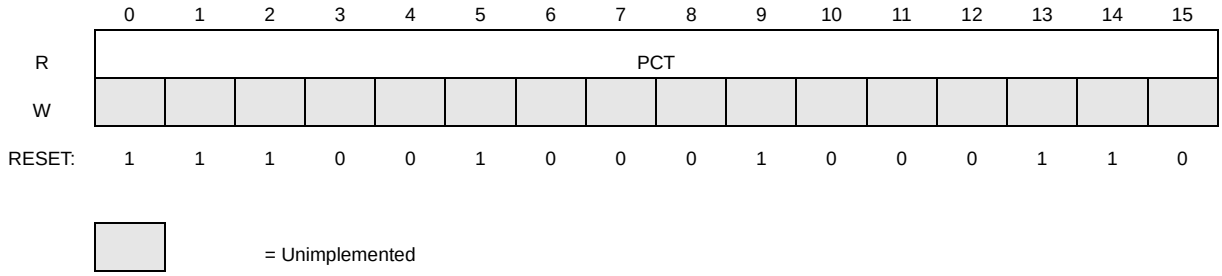


Table 230. Processor Core Type (PCT) Field Descriptions

Name	Description
PCT	**Processor Core Type** 0xE446: e200z4d Power Architecture core

22.4.2.2 Chip-Defined Platform Revision (REV) register

The REV is a 16-bit read-only register specifying a revision number. The state of this register is defined by an input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

See *Figure 257* and *Table 231* for the REV definition.

Figure 257. Chip-Defined Platform Revision (REV) Register

Register address: ECSM Base + 0x0002

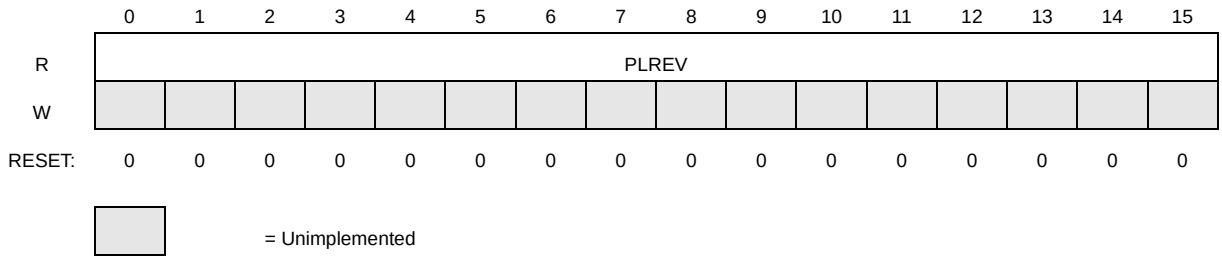


Table 231. REV field descriptions

Field	Description
PLREV	**Revision** This field is specified by an input signal to define a software-visible revision number.

22.4.2.3 Platform Crossbar Master Configuration (PLAMC)

The PLAMC is a 16-bit read-only register identifying the presence/absence of bus master connections to the platform's XBAR. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

See *Figure 258* and *Table 232* for the PLAMC definition.

Figure 258. Platform Crossbar Master Configuration (PLAMC)

Register address: ECSM Base + 0x0004

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

= Unimplemented

Table 232. PLAMC field descriptions

Field	Description
AMC[n]	**XBAR master configuration** 0 Bus master to XBAR input port n is absent 1 Bus master to XBAR input port n is present

22.4.2.4 Platform Crossbar Slave Configuration (PLASC)

The PLASC is a 16-bit read-only register identifying the presence/absence of bus slave connections to the platform's XBAR, plus a 1-bit flag defining the internal platform datapath width. 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.

See *Figure 259* and *Table 233* for the PLASC definition.

Figure 259. Platform Crossbar Slave Configuration (PLASC)

Register address: ECSM Base + 0x0006

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DP64	0	0	0	0	0	0	0	ASC							
W																
RESET:	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1

= Unimplemented

Table 233. PLASC field descriptions

Field	Description
DP64	**64-bit datapath** 0 Platform datapath width is 32 bits 1 Platform datapath width is 64 bits
ASC[n]	**XBAR slave configuration** 0 Bus slave to XBAR input port n is absent 1 Bus slave to XBAR input port n is present

22.4.2.5 IPS On-Platform Module Configuration (IOPMC) register

The IOPMC is a 32-bit read-only register identifying the presence or absence of the 32 low-order IPS peripheral modules connected to the primary slave bus controller. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

See *Figure 260* and *Table 234* for the IOPMC definition.

Figure 260. IPS On-Platform Module Configuration (IOPMC) Register

Register address: ECSM Base + 0x0008

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PMC[31:16]															
W																
RESET:	1	1	0	0	1	0	0	0	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	PMC[15:0]															
W																
RESET:	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Table 234. IOPMC field descriptions

Field	Description
PMC	**IPS Module Configuration** PMC[n] = 0 if an IPS module connection to decoded slot “n” is absent PMC[n] = 1 if an IPS module connection to decoded slot “n” is present

22.4.2.6 Miscellaneous Reset Status Register (MRSR)

The MRSR contains a bit for each of the reset sources to the device. An asserted bit indicates the last type of reset that occurred. Only one bit is set at any time in the MRSR,

reflecting the cause of the most recent reset as signalled by device reset input signals. The MRSR can only be read from the IPS programming model. Any attempted write is ignored.

See *Figure 261* and *Table 235* for the Miscellaneous Reset Status Register definition.

Figure 261. Miscellaneous Reset Status (MRSR) Register

Register address: ECSM Base + 0x000F

	0	1	2	3	4	5	6	7
R	POR	OFPLR	0	0	0	0	0	0
W								
RESET:	*	*	0	0	0	0	0	0

= Unimplemented

Table 235. MRSR field descriptions

Field	Description
POR	**Power-On Reset** 1 = Last recorded event was caused by a power-on reset (based on a device input signal)
OFPLR	**Off-Platform Reset** 1 = Last recorded event was a reset caused by an off-platform reset.

22.4.2.7 Miscellaneous User-Defined Control Register (MUDCR)

The MUDCR provides a program-visible register for user-defined control functions. It typically is used as configuration control for miscellaneous SoC-level modules. The contents of this register is simply output from ECSM to other modules where the user-defined control functions are implemented. See *Figure 262* and *Table 236* for the Miscellaneous User-Defined Control Register definition.

Figure 262. Miscellaneous User-Defined Control (MUDCR) Register

Register address: ECSM Base + 0x0024

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

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

= Unimplemented

Table 236. MUDCR field descriptions

Field	Description
MUDCR	**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 greater than 64 MHz. 0The platform RAM controller operates as a 0-wait state controller 1 The platform RAM controller operates as a 1-wait state controller *Note: The bit should not be changed while there is an ongoing transfer on the SRAM.*

Note: In DPM, the chip has an additional ECSM (ECSM_1) that also contains a MUDCR. Applications changing this configuration should take this fact into account.

22.4.2.8 Platform ECC registers

There are a number of program-visible registers for the sole purpose of reporting and logging of memory failures. These registers include the following:

- ECC Configuration Register (ECR)
- ECC Status Register (ESR)
- ECC Error Generation Register (EEGR)
- Platform Flash Memory ECC Address Register (PFEAR)
- Platform Flash Memory ECC Master Number Register (PFEMR)
- Platform Flash Memory ECC Attributes Register (PFEAT)
- Platform Flash Memory ECC Data Registers (PFEDRL and PFEDRH)
- 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 Registers (PREDRL and PREDRH)

The details on the ECC registers are provided in the subsequent sections.

The 32-bit ECC organization essentially provides two completely independent error checking mechanisms for the total 64-bit PRAM width. The ECC logic provides a 1-of-3 error response vector for each 32 bits of memory: no error, single-bit correctable error, multi-bit non-correctable error. *Table 237* defines the association between the reported ECC result and the PRAM bank chip selects.

Table 237. AHB response and ECC reporting for even and odd ECC

PRAM valid		ECC		Reported ECC	PRAM bus response		AHB HRESP
Even	Odd	Even	Odd		Even	Odd	
0	0	x	x	No access, No_error	xxxx	xxxx	okay
1	0	none	x	No_error	data	xxxx	okay
1	0	single	x	Even_single	corrected	xxxx	okay
1	0	multi	x	Even_multi	non-corrected	xxxx	err

Table 237. AHB response and ECC reporting for even and odd ECC (continued)

PRAM valid		ECC		Reported ECC	PRAM bus response		AHB HRESP
Even	Odd	Even	Odd		Even	Odd	
0	1	x	none	No_error	xxxx	data	okay
0	1	x	single	Odd_single	xxxx	corrected	okay
0	1	x	multi	Odd_multi	xxxx	non-corrected	err
1	1	none	none	No_error	data	data	okay
1	1	single	none	Even_single	corrected	data	okay
1	1	multi	none	Even_multi	non-corrected	data	err
1	1	none	single	Odd_single	data	corrected	okay
1	1	single	single	Even_single	corrected	corrected	okay
1	1	multi	single	Even_multi	non-corrected	corrected	err
1	1	none	multi	Odd_multi	data	non-corrected	err
1	1	single	multi	Odd_multi	corrected	non-corrected	err
1	1	multi	multi	Even_multi	non-corrected	non-corrected	err

As shown in *Table 237*, accesses of only a single memory bank report the ECC from that bank directly. For accesses involving both banks, the "most severe" ECC response is reported with the even bank taking priority if the responses are equivalent. This approach also provides improved correction capabilities compared to the 64-bit ECC implementation.

22.4.2.9 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.

See *Figure 263* and *Table 238* for the ECC Configuration Register definition.

Figure 263. ECC Configuration (ECR) Register

Register address: ECSM Base + 0x0043

	0	1	2	3	4	5	6	7
R	0	0	EPR1BR	EPF1BR	0	0	EPRNCR	EPFNCR
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Table 238. ECR field descriptions

Field	Description
EPR1BR	**Enable Platform RAM 1-bit Reporting** 0 = Reporting of single-bit platform RAM corrections is disabled. 1 = Reporting of single-bit platform RAM corrections is enabled. This bit can only be set if the SoC-configurable input enable signal is asserted. The occurrence of a single-bit platform RAM correction generates a non-critical fault as signalled by the assertion of ESR[R1BC]. The address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT, PREDRL, and PREDRH registers.
EPF1BR	**Enable Platform Flash Memory 1-bit Reporting** 0 = Reporting of single-bit platform flash memory corrections is disabled. 1 = Reporting of single-bit platform flash memory corrections is enabled. This bit can only be set if the SoC-configurable input enable signal is asserted. The occurrence of a single-bit platform flash memory correction generates a non-critical fault as signalled by the assertion of ESR[F1BC]. The address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT, PFEDRL, and PFEDRH registers.
EPRNCR	**Enable Platform RAM Non-Correctable Reporting** 0 = Reporting of non-correctable platform RAM errors is disabled. 1 = Reporting of non-correctable platform RAM errors is enabled. The occurrence of a non-correctable multi-bit platform RAM error generates a critical fault as signalled by the assertion of ESR[RNCE]. The faulting address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT, PREDRL, and PREDRH registers.
EPFNCR	**Enable Platform Flash Memory Non-Correctable Reporting** 0 = Reporting of non-correctable platform flash memory errors is disabled. 1 = Reporting of non-correctable platform flash memory errors is enabled. The occurrence of a non-correctable multi-bit platform flash memory error generates a critical fault as signalled by the assertion of ESR[FNCE]. The faulting address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT, PFEDRL, and PFEDRH registers.

22.4.2.10 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]// platform flash memory , 1-bit correction
ECSM_ECCRNCR_IRQ
    =  ECR[ERNCR] &  ESR[RNCE]// ram, noncorrectable error
ECSM_ECCFNCR_IRQ
    =  ECR[EFNCR] &  ESR[FNCE]// platform flash memory , noncorrectable
error
ECSM_ECC2BIT_IRQ
    =  ECSM_ECCRNCR_IRQ// ram, noncorrectable error
    |  ECSM_ECCFNCR_IRQ// platform flash memory , noncorrectable error
ECSM_ECC_IRQ
    =  ECSM_ECC1BIT_IRQ // 1-bit correction
    |  ECSM_ECC2BIT_IRQ// noncorrectable error
```

where the combination of a correctly-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 and ensure a correct handling in case of an ECC error, the following sequence in the ECSM error interrupt service routine is suggested:

1. 1. Read the ESR and save its content.
2. Check what error(s) are flagged.
3. Depending on the originator (RAM or flash memory) read and save all the address and attribute reporting registers to this originator:
   – Platform RAM error: PREAR, PRESR, PREMR, PREAT, PREDRL, PREDRH
   – Platform Flash error: PFEAR, PFEMR, PFEAT, PFEDRL, PFEDRH
4. 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.
5. When the values are identical, write a 1 to the asserted ESR flag to negate the interrupt request.

It is not recommended to read any of the following registers outside the ECSM error interrupt service routine (that is, without a previous ECC error event):

- PFEAR, PFEMR, PFEAT, PFEDRL, PFEDRH
- PREAR, PRESR, PREMR, PREAT, PREDRL, PREDRH

If no flash memory ECC event is defined to be handled for this module, accesses to the registers PFEAR, PFEMR, PFEAT, PFEDRL, and PFEDRH will terminate with an error.

If no RAM ECC event is defined to be handled for this module, accesses to the registers PREAR, PRESR, PREMR, PREAT, PREDRL, and PREDRH will terminate with an error.

Note: On SPC56ELxx, assertion of ECSM_ESR[PRNCE] or ECSM_ESR[PFNCE] causes MC_RGM functional reset.

Refer to Chapter 42: Reset Generation Module (MC_RGM) for more information about "flash, ECC or lockstep error" functional reset source (FL_ECC_RCC).

Therefore, ECSM_ECCRNCR_IRQ or ECSM_ECCFNCR_IRQ interrupts are not applicable on SPC56ELxx. The ECSM_ECC2BIT_IRQ interrupt request is connected to the INTC module, Interrupt source # 35, but the SPC56ELxx reset by the MC_RGM doesn't allow the ECSM error interrupt service routine to completely execute.

Notice the PFEAR, PFEMR, PFEAT, PFEDRH, PFEDRL, PREAR, PRESR, PREMR, PREAT, PREDRH and PREDRL registers are maintained through the MCU functional reset caused by the ECSM non-correctable ECC error detection.

See *Figure 264* and *Table 239* for the ECC Status Register definition.

Figure 264. ECC Status (ESR) Register

Register address: ECSM Base + 0x0047

	0	1	2	3	4	5	6	7
R	0	0	PR1BC	PF1BC	0	0	PRNCE	PFNCE
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Table 239. ESR field descriptions

Name	Description
PR1BC	**Platform RAM 1-bit Correction** 0 = No reportable single-bit platform RAM correction has been detected. 1 = A reportable single-bit platform RAM correction has been detected. This bit can only be set if ECR[EPR1BR] is asserted. The occurrence of a properly-enabled single-bit platform RAM correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT, PREDRL, and PREDRH registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.
PF1BC	**Platform Flash Memory 1-bit Correction** 0 = No reportable single-bit platform flash memory correction has been detected. 1 = A reportable single-bit platform flash memory correction has been detected. This bit can only be set if ECR[EPF1BR] is asserted. The occurrence of a properly-enabled single-bit platform flash memory correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT, PFEDRL, and PFEDRH registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.

Table 239. ESR field descriptions (continued)

Name	Description
PRNCE	**Platform RAM Non-Correctable Error** 0 = No reportable non-correctable platform RAM error has been detected. 1 = A reportable non-correctable platform RAM error has been detected. The occurrence of a properly-enabled non-correctable platform RAM error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT, PREDRL, and PREDRH registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.
PFNCE	**Platform Flash Memory Non-Correctable Error** 0 = No reportable non-correctable platform flash memory error has been detected. 1 = A reportable non-correctable platform flash memory error has been detected. The occurrence of a properly-enabled non-correctable platform flash memory error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT, PFEDRL, and PFEDRH registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.

In the event that multiple status flags are signaled simultaneously, ECSM records the event with the PR1BC as highest priority, then PF1BC, then PRNCE, and finally PFNCE.

22.4.2.11 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 platform RAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for “injecting” errors into the memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

Platform flash memory includes an ECC logic check to verify the integrity of its ECC logic (see *Section 24.1.6.12, ECC logic check*).

For platform RAM, the intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit non-correctable errors that are terminated with an error response.

The enabling of these error generation modes requires the same SoC-configurable input enable signal (as that used to enable single-bit correction reporting) be asserted.

See *Figure 265* and *Table 240* for the ECC Configuration Register definition.

Figure 265. ECC Error Generation (EEGR) Register

Register address: ECSM Base + 0x004A

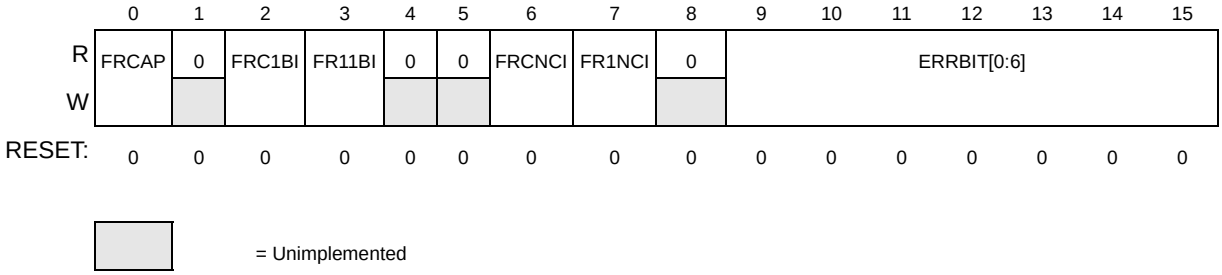


Table 240. ECC Error Generation (EEGR) Field Descriptions

Name	Description
FRCAP	Force Platform RAM Error Injection Access Protection 0 = All masters are able to generate platform RAM ECC errors via the EEGR register. 1 = Only the master defined with as having hmaster=0 (usually the core) can generate platform RAM ECC errors via the EEGR register. The assertion of this bit ensures that platform RAM data inversions can only occur from the master module with the master ID of 0. Since this is usually the core, this protects the platform RAM from errant or multiple simultaneous attempted data inversions from other master modules and, in the case of a multi-core system, ensures that only one core can issue a platform RAM data inversion. The reset value of the bit is 0 and as a result, platform RAM data inversions can be requested from any master module. It is the responsibility of the software to ensure the proper setting of this bit.
FRC1BI	**Force Platform RAM Continuous 1-bit Data Inversions** 0 = No platform RAM continuous 1-bit data inversions are generated. 1 = 1-bit data inversions in the platform RAM are continuously generated. The assertion of this bit forces the platform RAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT[0:6], continuously on every write operation. The normal ECC generation takes place in the platform RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the platform RAM. After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to correctly 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.
FR11BI	**Force Platform RAM One 1-bit Data Inversion** 0 = No platform RAM single 1-bit data inversion is generated. 1 = One 1-bit data inversion in the platform RAM is generated. The assertion of this bit forces the platform RAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT[0:6], on the first write operation after this bit is set. The normal ECC generation takes place in the platform RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the platform RAM. After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. This bit can only be set if the same SoC configurable input enable signal (as that used to enable single-bit correction reporting) is asserted.

Table 240. ECC Error Generation (EEGR) Field Descriptions (continued)

Name	Description
FRCNCI	**Force Platform RAM Continuous Non-Correctable Data Inversions** 0 = No platform RAM continuous 2-bit data inversions are generated. 1 = 2-bit data inversions in the platform RAM are continuously generated. The assertion of this bit forces the platform RAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT[0:6] 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 platform RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the platform RAM.
FR1NCI	**Force platform RAM One Non-Correctable Data Inversions** 0 = No platform RAM single 2-bit data inversions are generated. 1 = One 2-bit data inversion in the platform RAM is generated. The assertion of this bit forces the platform RAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT[0:6] and the overall odd parity bit, on the first write operation after this bit is set. The normal ECC generation takes place in the platform RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the platform RAM. After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic.

Table 240. ECC Error Generation (EEGR) Field Descriptions (continued)

Name	Description
ERRBIT	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 platform RAM width. For example, consider a 64-bit platform 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 platform RAM 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 between 78 and 98, no bit position is inverted. To accommodate address bus inversions, the ERRBIT values start at 99 as defined: if ERRBIT = 99, then ADDR[3] is inverted if ERRBIT = 100, then ADDR[4] is inverted ... if ERRBIT = 114, then ADDR[18] is inverted if ERRBIT = 115, then ADDR[19] is inverted For ERRBIT values greater than 115, the address bus inversion has no effect as only the lower 20 bits are used by the platform RAM controller.

Inversions of the address bus must be defined as non-correctable for the inversion to work and to resolve properly as a non-correctable error. One-bit data inversions of the address bus are ignored.

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 four 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 unpredictable operations.

22.4.2.12 Platform Flash Memory ECC Address Register (PFEAR)

The PFEAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the platform flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the platform flash memory causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, PFEDRL, and PFEDRH 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. If no flash memory ECC event is defined to be handled for this module, accesses to this register will terminate with an error.

See *Figure 266* and *Table 241* for the PFEAR definition.

Figure 266. Platform Flash Memory ECC Address (PFEAR) Register

Register address: ECSM Base + 0x0050

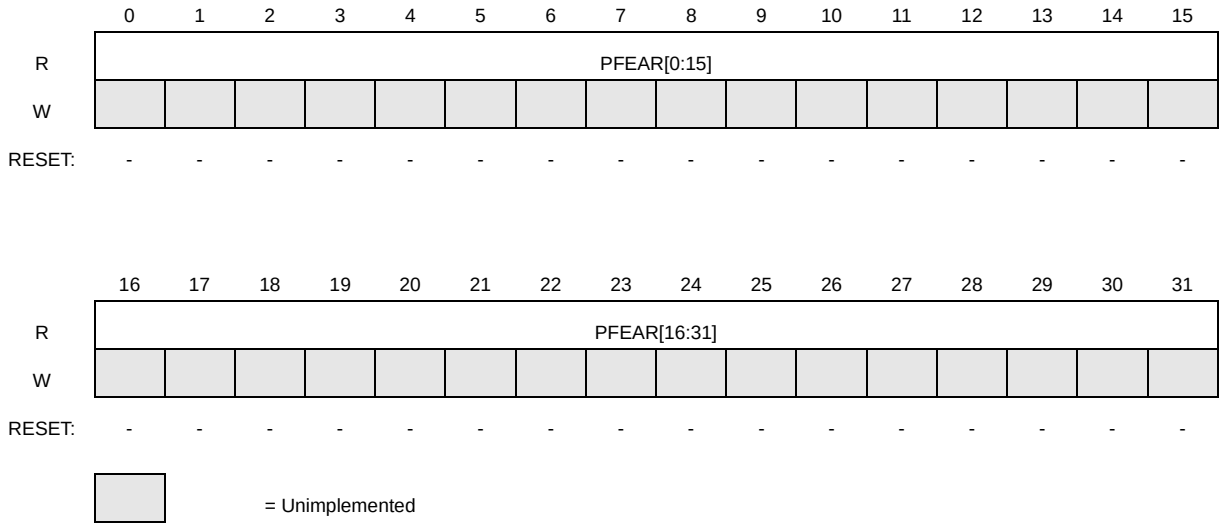


Table 241. PFEAR field descriptions

Field	Description
PFEAR	**Platform Flash Memory ECC Address** This 32-bit register contains the faulting access address of the last, properly-enabled platform flash memory ECC event.

22.4.2.13 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 platform flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the platform flash memory causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, PFEDRL, and PFEDRH registers, and the appropriate flag (PF1BC or PFNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. If no flash memory ECC event is defined to be handled for this module, accesses to this register will terminate with an error.

See *Figure 267* and *Table 242* for the PFEMR definition.

Figure 267. Platform Flash Memory ECC Master Number (PFEMR) Register

Register address: ECSM Base + 0x0056

	0	1	2	3	4	5	6	7
R	0	0	0	0	PFEMR			
W								
RESET:	0	0	0	0	-	-	-	-

= Unimplemented

Table 242. PFEMR field descriptions

Field	Description
PFEMR	**Platform Flash Memory ECC Master Number** This 4-bit field contains the XBAR bus master number of the faulting access of the last, properly-enabled platform flash memory ECC event.

22.4.2.14 Platform Flash Memory ECC Attributes (PFEAT) register

The PFEAT is an 8-bit register for capturing the XBAR bus master attributes of the last, properly-enabled ECC event in the platform flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the platform flash memory causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, PFEDRL, and PFEDRH 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. If no flash memory ECC event is defined to be handled for this module, accesses to this register will terminate with an error.

See *Figure 268* and *Table 243* for the PFEAT definition.

Figure 268. Platform Flash Memory ECC Attributes (PFEAT) Register

Register address: ECSM Base + 0x0057

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
RESET:	-	-	-	-	-	-	-	-

= Unimplemented

Table 243. PFEAT field descriptions

Field	Description
WRITE	**AMBA-AHB HWRITE** 0 = AMBA-AHB read access 1 = AMBA-AHB write access
SIZE	**AMBA-AHB HSIZE[0:2]** 0b000 = 8-bit AMBA-AHB access 0b001 = 16-bit AMBA-AHB access 0b010 = 32-bit AMBA-AHB access 0b1xx = Reserved
PROTECTION	**AMBA-AHB HPROT[0:3]** 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

22.4.2.15 Platform Flash Memory ECC Data Registers (PFEDRL and PFEDRH)

These two 32-bit registers contain a 64-bit field, PFEDR, for capturing the data associated with the last, properly-enabled ECC event in the platform flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the platform flash memory causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, PFEDRH, and PFEDRL 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.

These registers can only be read from the IPS programming model; any attempted write is ignored. If no flash memory ECC event is defined to be handled for this module, accesses to these registers will terminate with an error.

Figure 269. Platform Flash Memory ECC Data High Register (PFEDRH)

Register address: ECSM Base +0x0058

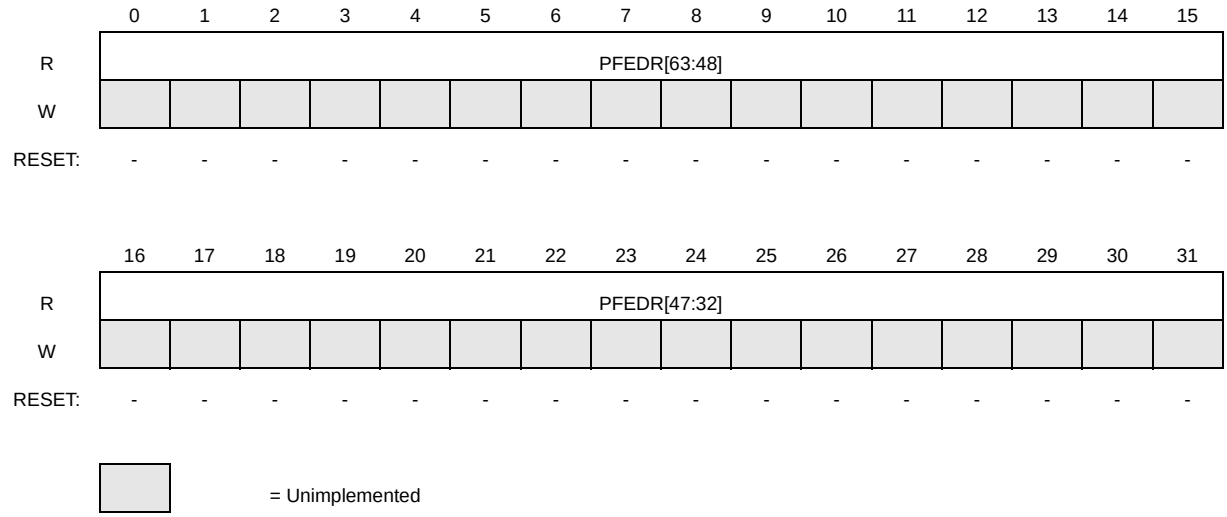


Figure 270. Platform Flash Memory ECC Data Low Register (PFEDRL)

Register address: ECSM Base +0x005C

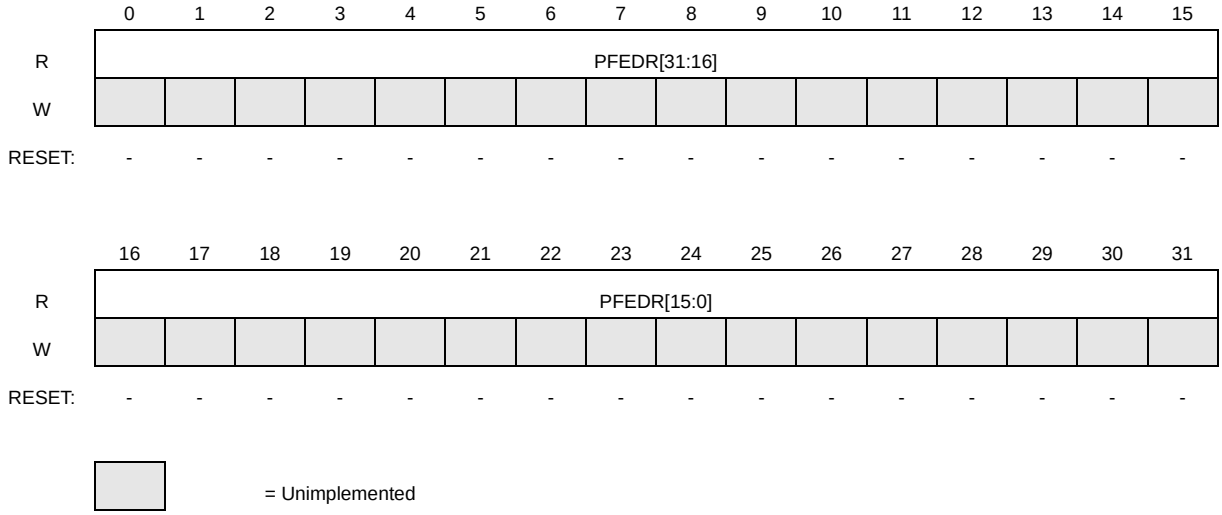


Table 244. PFEDRL and PFEDRH field descriptions

Field	Description
PFEDR	**Platform Flash Memory ECC Data** This 64-bit field contains the data associated with the faulting access of the last, properly-enabled platform flash memory ECC event. The field contains the data value taken directly from the data bus.

22.4.2.16 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 platform RAM. Depending on the state of the ECC Configuration Register, an ECC event in the platform RAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, PREDRL, and PREDRH 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. If no RAM ECC event is defined to be handled for this module, accesses to this register will terminate with an error.

See *Figure 271* and *Table 245* for the PREAR definition.

Figure 271. Platform RAM ECC Address (PREAR) Register

Register address: ECSM Base + 0x0060

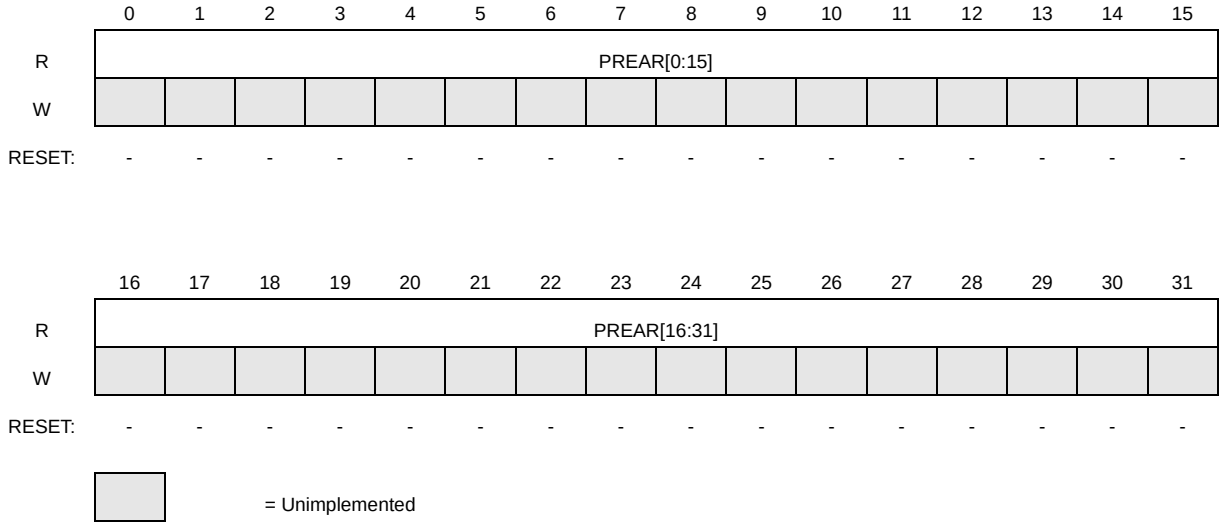


Table 245. PREAR field descriptions

Field	Description
PREAR	**Platform RAM ECC Address** This 32-bit field contains the faulting access address of the last, properly-enabled platform RAM ECC event.

22.4.2.17 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 platform RAM. Depending on the state of the ECC Configuration Register, an ECC event in the platform RAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, PREDRL, and PREDRH 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. If no RAM ECC event is defined to be handled for this module, accesses to this register will terminate with an error.

See *Figure 272* and *Table 246* for the PRESR definition.

Figure 272. Platform RAM ECC Syndrome (PRESR) Register

Register address: ECSM Base + 0x0065

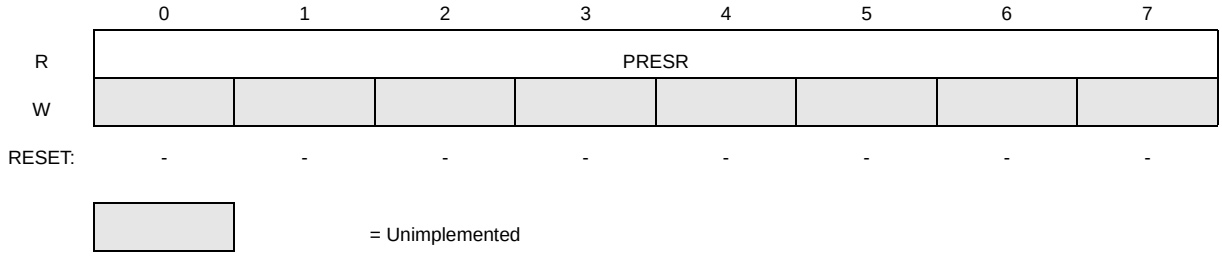


Table 246. PRESR field descriptions

Field	Description
PRESR	**Platform RAM ECC Syndrome** 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 code words, 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 246* associates the upper 7 bits of the syndrome with the data bit in error.

Note: Table 246 associates the 8 bits of the syndrome value with the data or ECC bit in error. This table follows the bit vectoring notation where the LSB=0.

Table 247. Platform RAM syndrome mapping for single-bit correctable errors

PRESR	Data Bit in Error
0x01	ECC ODD[0]
0x02	ECC ODD[1]
0x04	ECC ODD[2]
0x07	DATA ODD BANK[31]
0x08	ECC ODD[3]
0x10	ECC ODD[4]
0x20	ECC ODD[5]
0x40	ECC ODD[6]
0x43	DATA ODD BANK[0]
0x45	DATA ODD BANK[1]
0x46	DATA ODD BANK[2]
0x49	DATA ODD BANK[3]
0x4a	DATA ODD BANK[4]
0x4c	DATA ODD BANK[5]
0x4f	DATA ODD BANK[21]

Table 247. Platform RAM syndrome mapping for single-bit correctable errors

PRESR	Data Bit in Error
0x51	DATA ODD BANK[6]
0x52	DATA ODD BANK[7]
0x54	DATA ODD BANK[8]
0x57	DATA ODD BANK[22]
0x58	DATA ODD BANK[9]
0x5b	DATA ODD BANK[23]
0x5d	DATA ODD BANK[24]
0x5e	DATA ODD BANK[25]
0x61	DATA ODD BANK[10]
0x62	DATA ODD BANK[11]
0x64	DATA ODD BANK[12]
0x67	DATA ODD BANK[26]
0x68	DATA ODD BANK[13]
0x6b	DATA ODD BANK[27]
0x6d	DATA ODD BANK[28]
0x6e	DATA ODD BANK[29]
0x70	DATA ODD BANK[14]
0x73	DATA ODD BANK[15]
0x75	DATA ODD BANK[16]
0x76	DATA ODD BANK[17]
0x79	DATA ODD BANK[18]
0x7a	DATA ODD BANK[19]
0x7c	DATA ODD BANK[20]
0x7f	DATA ODD BANK[30]
0x81	ECC EVEN[0]
0x82	ECC EVEN[1]
0x84	ECC EVEN[2]
0x87	DATA EVEN BANK[31]
0x88	ECC EVEN[3]
0x90	ECC EVEN[4]
0xa0	ECC EVEN[5]
0xc0	ECC EVEN[6]
0xc3	DATA EVEN BANK[0]
0xc5	DATA EVEN BANK[1]
0xc6	DATA EVEN BANK[2]

Table 247. Platform RAM syndrome mapping for single-bit correctable errors

PRESR	Data Bit in Error
0xc9	DATA EVEN BANK[3]
0xca	DATA EVEN BANK[4]
0xcc	DATA EVEN BANK[5]
0xcf	DATA EVEN BANK[21]
0xd1	DATA EVEN BANK[6]
0xd2	DATA EVEN BANK[7]
0xd4	DATA EVEN BANK[8]
0xd7	DATA EVEN BANK[22]
0xd8	DATA EVEN BANK[9]
0xdb	DATA EVEN BANK[23]
0xdd	DATA EVEN BANK[24]
0xde	DATA EVEN BANK[25]
0xe1	DATA EVEN BANK[10]
0xe2	DATA EVEN BANK[11]
0xe4	DATA EVEN BANK[12]
0xe7	DATA EVEN BANK[26]
0xe8	DATA EVEN BANK[13]
0xeb	DATA EVEN BANK[27]
0xed	DATA EVEN BANK[28]
0xee	DATA EVEN BANK[29]
0xf0	DATA EVEN BANK[14]
0xf3	DATA EVEN BANK[15]
0xf5	DATA EVEN BANK[16]
0xf6	DATA EVEN BANK[17]
0xf9	DATA EVEN BANK[18]
0xfa	DATA EVEN BANK[19]
0xfc	DATA EVEN BANK[20]
0xff	DATA EVEN BANK[30]

22.4.2.18 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 platform RAM. Depending on the state of the ECC Configuration Register, an ECC event in the platform RAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, PREDRL, and PREDRH 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. If no RAM ECC event is defined to be handled for this module, accesses to this register will terminate with an error.

See *Figure 273* and *Table 248* for the PREMR definition.

Figure 273. Platform RAM ECC Master Number (PREMR) Register

Register address: ECSM Base + 0x0066

	0	1	2	3	4	5	6	7
R	0	0	0	0	PREMR			
W								
RESET:	0	0	0	0	-	-	-	-

= Unimplemented

Table 248. PREMR field descriptions

Field	Description
PREMR	**Platform RAM ECC Master Number** This 4-bit field contains the XBAR bus master number of the faulting access of the last, correctly-enabled RAM ECC event.

22.4.2.19 Platform RAM ECC Attributes (PREAT) register

The PREAT is an 8-bit register for capturing the XBAR bus master attributes of the last, properly-enabled ECC event in the platform RAM. Depending on the state of the ECC Configuration Register, an ECC event in the platform RAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, PREDRL, and PREDRH 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. If no RAM ECC event is defined to be handled for this module, accesses to this register will terminate with an error.

See *Figure 274* and *Table 249* for the PREAT definition.

Figure 274. Platform RAM ECC Attributes (PREAT) Register

Register address: ECSM Base + 0x0067

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
RESET:	-	-	-	-	-	-	-	-

= Unimplemented

Table 249. PREAT field descriptions

Field	Description
WRITE	**AMBA-AHB HWRITE** 0 = AMBA-AHB read access 1 = AMBA-AHB write access
SIZE	**AMBA-AHB HSIZE[0:2]** 0b000 = 8-bit AMBA-AHB access 0b001 = 16-bit AMBA-AHB access 0b010 = 32-bit AMBA-AHB access 0b1xx = Reserved
PROTECTION	**AMBA-AHB HPROT[0:3]** 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

22.4.2.20 Platform RAM ECC Data Registers (PREDRL and PREDRH)

These two 32-bit registers contain a 64-bit field, PREDR, for capturing the data associated with the last, properly-enabled ECC event in the platform RAM. Depending on the state of the ECC Configuration Register, an ECC event in the platform RAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, PREDRL, and PREDRH 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.

These registers can only be read from the IPS programming model; any attempted write is ignored. If no RAM ECC event is defined to be handled for this module, accesses to these registers will terminate with an error.

Figure 275. Platform RAM ECC Data High Register (PREDRH)

Register address: ECSM Base +0x0068

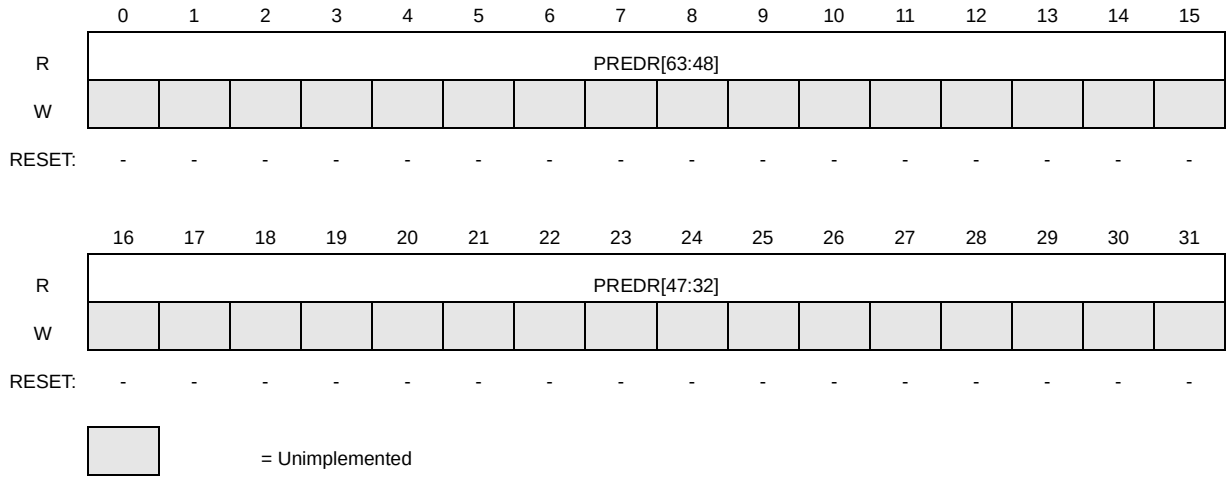


Figure 276. Platform RAM ECC Data Low Register (PREDRL)

Register address: ECSM Base +0x006C

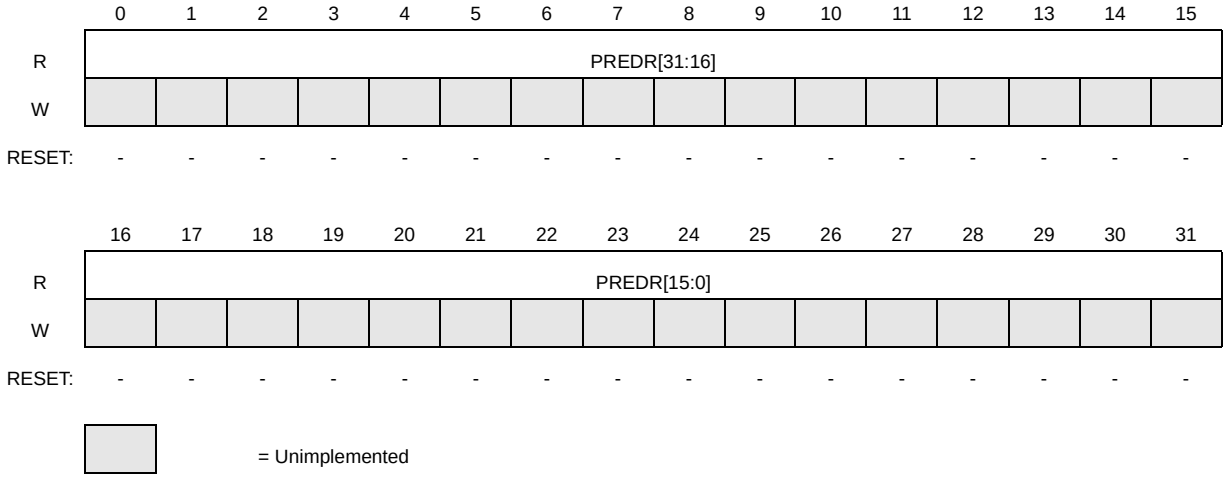


Table 250. PREDRL and PREDRH field descriptions

Field	Description
PREDR	**Platform RAM ECC Data** This 64-bit field contains the data associated with the faulting access of the last, properly-enabled platform RAM ECC event. The field contains the data value taken directly from the data bus.

23 Fault Collection and Control Unit (FCCU)

23.1 Introduction

The Fault Collection and Control Unit (FCCU) offers a programmable redundant hardware channel to collect errors and to lead the device in a controlled way to a safe state when a failure is present in the device. No CPU intervention is requested for collection and control operation.

The FCCU offers a systematic approach to manage fault detection and control.

The main functions supported by the module are:

- Redundant collection of hardware checker (for example, RCCU) results
- Redundant collection of error information from critical modules on the chip (such as the flash memory or the STCU)
- Collection of test results
- Configurable and graded fault control via user software control
- Internal reactions:
  - No reset reaction
  - IRQ
  - Short ‘functional’ reset
  - Long ‘functional’ reset
  - ***SAFE*** mode request
- External reaction (failure is reported to the outside world via output pins)
- Watchdog timer for the re-configuration phase
- Configuration lock
- NVM configuration loading
- Self checking capabilities:
  - FCCU internal control logic redundancy
  - Additional parity check for the associated configuration registers

The FCCU circuitry is checked at start-up by the self-checking procedure. The FCCU is operative with a default configuration (without CPU intervention) immediately after the completion of the self-checking procedure.

Two classes of faults — critical and non-critical — are identified based on the criticality and the related reactions. Internal (short or long ‘functional’ reset, interrupt request) and external (FCCU_F signaling) reactions are statically defined or programmable based on the fault criticality. The default configuration can be modified only in a specific FCCU state for application/test/debugging purposes.

23.1.1 Glossary and acronyms

Table 251. Acronyms

Term	Description
RCC	Redundancy control checkers
RCCU	Redundancy control checker unit

Table 251. Acronyms (continued)

Term	Description
IPS	Internal peripheral system
NMI	Non-maskable interrupts
IRQ	Interrupt request
STCU	Self testing control unit
FSM	Finite state machine
CF	Critical fault
NCF	Non-critical fault
WS	Wait state
SW	Software
NVM	Nonvolatile memory
CPU	Central processing unit

23.2 Main features

- Management of:
  - 32 critical faults
  - 32 non-critical faults
- Hardware or software fault recovery management
- Fault detection collection
- Fault injection (fake faults)
- External reaction via fault-state dedicated system signals (FCCU_F pins)
- Internal (chip) reactions (alarm state): interrupt request
- Internal (chip) reactions (fault state):
  - Long ‘functional’ reset
  - Short ‘functional’ reset
  - Non-maskable interrupt (NMI)
  - ***SAFE*** mode request
- Bi-Stable, Dual-Rail and Time Switching output protocols using FCCU_F pins
- Internal (to the FCCU) watchdog timer for the re-configuration phase
- Configuration lock
- NVM configuration loading
- Self-checking capabilities:
  - FSM redundancy
  - Parity check for the configuration registers

23.3 Block diagram

The top-level diagram of the FCCU is shown in *Figure 277*.

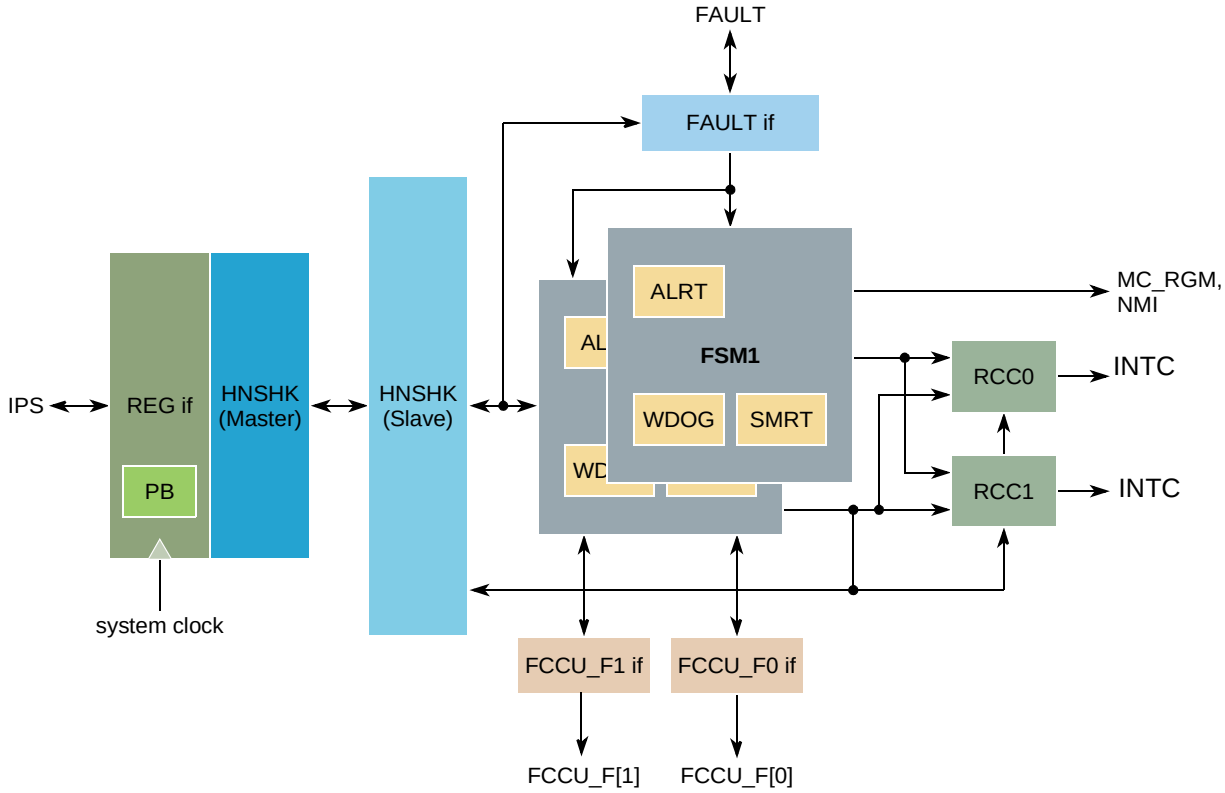


Figure 277. FCCU top-level diagram

As shown in *Figure 277*, the FCCU includes the following sub-modules:

- REG if: it includes the register file, the register interface, the IRQ interface and the parity block (PB) for the configuration registers.
- HNSHK blocks (master and slave blocks): it includes the FSMs to support the handshake between the REG if and the FSM unit due to the usage of 2 asynchronous clocks (system clock and IRCOSC clock).
- Finite State Machine (FSM) units implement the main functions of the FCCU. These units also include also the watchdog timer (WDOG), the ***SAFE*** mode request timer (SMRT), and the alarm timer (ALRT).
- FAULT if: implements the interface for the fault conditioning and management
- FCCU_Fx units implement the output stage to manage the FCCU_F pins.
- RCCx units implement the redundancy control checker to monitor the FSM unit state and its configuration.

Note: The FCCU Redundancy Control Checkers (RCC) should not be confused with the Redundancy Control Checker Units (RCCU) instantiated on the SPC56ELxx to detect critical lockstep failure).

23.4 Signal description

The FCCU generates two external signals, FCCU_F[0] and FCCU_F[1]. These are described in *Section 23.7.10, FCCU_F interface*.

23.5 Register interface

The register interface is a slave bus used for configuration purposes via the CPU.

The following bus operations are supported:

- Word (32-bit) data write/read operations to any registers
- Low and high half-words (16-bit, data[0:15] or data[16:31]) data write/read operations to any registers
- Byte (8 bits, data[0:7] or data[8:15] or data[16:23] or data[24:31]) data write/read operations to any registers

Any other operation (free byte enables, misaligned word or half-word access or other operations) are not supported.

The FCCU generates a transfer error in the following cases:

- Any write/read access executed outside the address space of the peripheral
- Any write/read operation different from byte/halfword/word (free byte enables, misaligned access, or other operations) on each register
- Any write access to the configuration registers not executed while the FCCU is not in CONFIG state

The registers of the FCCU are accessible (read/write) in each access mode: user, supervisor, or test.

23.6 Memory map and register description

The FCCU registers are listed in *Table 252*. The contents of the configuration registers (labeled as “W in CONFIG state only” in the Access column) can be locked by the OP16 operation as defined in the FCCU_CTRL register.

Table 252. FCCU memory map

Address offset	Register	Access(1)	Location
0x00	*FCCU Control Register (FCCU_CTRL)*	R/W always	*on page 541*
0x04	*FCCU CTRL Key Register (FCCU_CTRLK)*	W always	*on page 544*
0x08	*FCCU Configuration Register (FCCU_CFG)*	R always; W in CONFIG state only	*on page 545*
0x0C	*FCCU CF Configuration Register 0 (FCCU_CF_CFG0)*	R always; W in CONFIG state only	*on page 546*
0x10	*FCCU CF Configuration Register 1 (FCCU_CF_CFG1)*		
0x14	*FCCU CF Configuration Register 2 (FCCU_CF_CFG2)*		
0x18	*FCCU CF Configuration Register 3 (FCCU_CF_CFG3)*		

Table 252. FCCU memory map (continued)

Address offset	Register	Access[1]	Location
0x1C	FCCU NCF Configuration Register 0 (FCCU_NCF_CFG0)	R always; W in CONFIG state only	on page 548
0x20	FCCU NCF Configuration Register 1 (FCCU_NCF_CFG1)		
0x24	FCCU NCF Configuration Register 2 (FCCU_NCF_CFG2)		
0x28	FCCU NCF Configuration Register 3 (FCCU_NCF_CFG3)		
0x2C	FCCU CFS Configuration Register 0 (FCCU_CFS_CFG0)	R always; W in CONFIG state only	on page 549
0x30	FCCU CFS Configuration Register 1 (FCCU_CFS_CFG1)		
0x34	FCCU CFS Configuration Register 2 (FCCU_CFS_CFG2)		
0x38	FCCU CFS Configuration Register 3 (FCCU_CFS_CFG3)		
0x3C	FCCU CFS Configuration Register 4 (FCCU_CFS_CFG4)		
0x40	FCCU CFS Configuration Register 5 (FCCU_CFS_CFG5)		
0x44	FCCU CFS Configuration Register 6 (FCCU_CFS_CFG6)		
0x48	FCCU CFS Configuration Register 7 (FCCU_CFS_CFG7)		
0x4C	FCCU NCFS Configuration Register 0 (FCCU_NCFS_CFG0)	R always; W in CONFIG state only	on page 550
0x50	FCCU NCFS Configuration Register 1 (FCCU_NCFS_CFG1)		
0x54	FCCU NCFS Configuration Register 2 (FCCU_NCFS_CFG2)		
0x58	FCCU NCFS Configuration Register 3 (FCCU_NCFS_CFG3)		
0x5C	FCCU NCFS Configuration Register 4 (FCCU_NCFS_CFG4)		
0x60	FCCU NCFS Configuration Register 5 (FCCU_NCFS_CFG5)		
0x64	FCCU NCFS Configuration Register 6 (FCCU_NCFS_CFG6)		
0x68	FCCU NCFS Configuration Register 7 (FCCU_NCFS_CFG7)		
0x6C	FCCU CF Status Register 0 (FCCU_CF_S0)	R/W always	on page 550
0x70	FCCU CF Status Register 1 (FCCU_CF_S1)		
0x74	FCCU CF Status Register 2 (FCCU_CF_S2)		
0x78	FCCU CF Status Register 3 (FCCU_CF_S3)		
0x7C	FCCU CF Key Register (FCCU_CFK)	W always	on page 552
0x80	FCCU NCF Status Register 0 (FCCU_NCF_S0)	R/W always	on page 553
0x84	FCCU NCF Status Register 1 (FCCU_NCF_S1)		
0x88	FCCU NCF Status Register 2 (FCCU_NCF_S2)		
0x8C	FCCU NCF Status Register 3 (FCCU_NCF_S3)		
0x90	FCCU NCF Key Register (FCCU_NCFK)	W always	on page 554
0x94	FCCU NCF Enable Register 0 (FCCU_NCF_E0)	R always; W in CONFIG state only	on page 555
0x98	FCCU NCF Enable Register 1 (FCCU_NCF_E1)		
0x9C	FCCU NCF Enable Register 2 (FCCU_NCF_E2)		
0xA0	FCCU NCF Enable Register 3 (FCCU_NCF_E3)		

Table 252. FCCU memory map (continued)

Address offset	Register	Access(1)	Location
0xA4	*FCCU NCF Time-out Enable Register 0 (FCCU_NCF_TOE0)*	R always; W in CONFIG state only	*on page 556*
0xA8	*FCCU NCF Time-out Enable Register 1 (FCCU_NCF_TOE1)*		
0xAC	*FCCU NCF Time-out Enable Register 2 (FCCU_NCF_TOE2)*		
0xB0	*FCCU NCF Time-out Enable Register 3 (FCCU_NCF_TOE3)*		
0xB4	*FCCU NCF Time-out Register (FCCU_NCF_TO)*	R always; W in CONFIG state only	*on page 557*
0xB8	*FCCU CFG Timeout Register (FCCU_CFG_TO)*	R always; W in CONFIG state only	*on page 557*
0xBC	FCCO IO Control Register (FCCU_EINOUT)	Read/Write	*on page 558*
0xC0	*FCCU Status Register (FCCU_STAT)*	R always	*on page 559*
0xD4	*FCCU SC Freeze Status Register (FCCU_SCFS)*	R always; W in CONFIG state only	*on page 560*
0xD8	*FCCU CF Fake Register (FCCU_CFF)*	W always	*on page 562*
0xDC	*FCCU NCF Fake Register (FCCU_NCFF)*	W always	*on page 563*
0xE0	*FCCU IRQ Status Register (FCCU_IRQ_STAT)*	R/W always	*on page 563*
0xE4	*FCCU IRQ Enable Register (FCCU_IRQ_EN)*	R/W always	*on page 564*
0xE8	*FCCU XTMR Register (FCCU_XTMR)*	R always	*on page 565*
0xEC	*FCCU MCS Register (FCCU_MCS)*	R always	*on page 566*

1. In this column, R/W = read/write, R = read-only, and W = write-only.

23.6.1 FCCU Control Register (FCCU_CTRL)

The FCCU_CTRL register allows execution of the following operations:

- Move the FCCU state from the NORMAL state into the CONFIG state
- Move the FCCU state from the CONFIG state into the NORMAL state
- Read or clear the CF status register
- Read or clear the NCF status register
- Read the FCCU FSM status register
- Read or clear the FCCU freeze registers
- Lock the FCCU configuration registers
- Read the ALARM timer
- Read the SMRT timer
- Read the Watchdog timer
- Load the NVM configuration (only for test purposes)

Some critical operations (OP1, OP2, OP16, and OP31) require a key as defined in the FCCU_CTRLK register.

Figure 278. FCCU Control Register (FCCU_CTRL)

Offset: 0x00 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	NVML	OPS		0	OPR				
W																
Reset	0	0	0	0	0	0	0	0/1(1)	0/1(1)	0/1(1)	0	0	0	0	0	0

1. 1 when NVM interface is used, 0 otherwise.

Table 253. FCCU_CTRL field descriptions

Field	Description
NVML	NVM configuration loaded 0 No NVM configuration loaded. 1 NVM configuration loaded. At the end of the reset PHASE3, NVML = 1 and OPS = 11 if the NVM interface of the FCCU is correctly driven by the SSCM interface. This bit can be read and cleared (via OP15 operation) by the software.
OPS	Operation status 00 Idle. 01 In progress. 10 Aborted. 11 Successful. This bit can be read and cleared (via OP15 operation) by the software.

Table 253. FCCU_CTRL field descriptions (continued)

Field	Description
OPR	Operation run 00000 No operation [OP0]. 00001 Set the FCCU into the CONFIG state [OP1]. 00010 Set the FCCU into the NORMAL state [OP2]. 00011 Read the FCCU state (refer to the FCCU_STAT register) [OP3]. 00100 Read the FCCU frozen status flags [OP4]. 00101 Read the FCCU frozen status flags [OP5]. 00110 Read the FCCU frozen status flags [OP6]. 00111 Read the FCCU frozen status flags [OP7]. 01000 Read the FCCU frozen status flags [OP8]. 01001 Read the CF status register (refer to the FCCU_CFS register) [OP9]. 01010 Read the NCF status register (refer to the FCCU_NCFS register) [OP10]. 01011 CF status clear operation in progress (refer to the FCCU_CFS register) [OP11]. 01100 NCF status clear operation in progress (refer to the FCCU_NCFS register) [OP12]. 01101 Clear the freeze status registers (refer to the freeze registers) [OP13]. 01110 CONFIG to NORMAL FCCU state (configuration timeout) in progress [OP14]. 01111 Clear the operation status (OPS = Idle, NVML = 0) [OP15]. 10000 Lock the FCCU configuration [OP16]. 10001 Read the ALARM timer (refer to the FCCU_XTMR register) [OP17]. 10010 Read the SMRT timer (refer to the FCCU_XTMR register) [OP18]. 10011 Read the CFG timer (refer to the FCCU_XTMR register) [OP19]. 10100–11111 Reserved [OP20–OP30]. 11111 Run the NVM loading operation (only for test purposes) [OP31]. The software must not modify the OPR field until the completion of the operation (any write operation will be ignored until then). After the operation has been completed, the OPS field is set and the OPR field is automatically cleared (OPR = 000). Your software must not program the following opcodes: – OP11 and OP12 (these opcodes are automatically selected when the FCCU_CFS or FCCU_NCFS registers are cleared by a write-clear operation into the related register) – OP14 (This opcode is automatically selected when the timeout occurs [FCCU_CFG_TO] during the configuration procedure. In this case, the FCCU state is automatically forced in NORMAL mode setting the default configuration. In this phase any write operation to the FCCU configuration registers is inhibited.) – OP20–OP30 (these are reserved; if you attempt to use them, they will return an ABORT response without any side effect) The ABORT response occurs in the following cases: – wrong access (missing or wrong key) to the FCCU_CFS register (clear operation OP11) – wrong access (missing or wrong key) to the FCCU_NCFS register (clear operation OP12) – wrong access (missing or wrong key) to the FCCU_CTRL register (OP1, OP2, OP16 operation) – OP1 (CONFIG command) execution when FCCU state ≠ NORMAL or configuration locked – OP20–OP30 (RESERVED operations) execution The OP31 opcode executes the NVM configuration loading via the software. It should be used only for test/debug purposes.

23.6.2 FCCU CTRL Key Register (FCCU_CTRLK)

The FCCU_CTRLK register implements the key access for the operations OP1, OP2, OP16, OP31 according to the following sequence:

1. Write the key into the FCCU_CTRLK register.
2. Write the FCCU_CTRL register (operations OP1, OP2, OP16, or OP31).

Note: There should not be any other operation while performing the above steps in sequence.

The FCCU_CTRLK register is not readable. A read operation always returns 0x0000_0000. The key must be written by a word (32-bit) data write operation.

Figure 279. FCCU CTRL Key Register (FCCU_CTRLK)

Offset: 0x04 Access: User write-only

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

Table 254. FCCU_CTRLK field descriptions

Field	Description
CTRLK	Control register key: **CTRLK value** / **Function** 0x9137_56AF – Key for operation OP1 0x825A_132B – Key for operation OP2 0x7ACB_32F0 – Key for operation OP16 0x29AF_8752 – Key for operation OP31

23.6.3 FCCU Configuration Register (FCCU_CFG)

The FCCU_CFG register is accessible in write mode only in the CONFIG state. It defines the global configuration for the FCCU.

Figure 280. FCCU Configuration Register (FCCU_CFG)

Note: Reset values of the FCC Configuration Register (FCCU_CFG) depend on the contents of the user option bits.

Offset: 0x008 Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	RCCE1	RCCE0	SMRT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	FCCU_CFG.CM	FCCU_CFG.SM	FCCU_CFG.PS	FCCU_CFG.FOM			FCCU_CFG.FOP					
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

1. Writable only in the CONFIG state

Table 255. FCCU_CFG field descriptions

Field	Description
RCCE1	RCC1 enable 0 RCC1 disabled. 1 RCC1 enabled. *Note: In case a single checker (RCC1 or RCC0 unit) is enabled, both the checkers assert a fault condition (interrupt).*
RCCE0	RCC0 enable 0 RCC0 disabled. 1 RCC0 enabled. *Note: In case a single checker (RCC1 or RCC0 unit) is enabled, both the checkers assert a fault condition (interrupt).*
SMRT	***SAFE** Mode Request Timer* 0000 ***SAFE*** mode request delay = 1 period (IRCOSC clock). 0001 ***SAFE*** mode request delay = 2 periods (IRCOSC clock). 0010 ***SAFE*** mode request delay = 4 periods (IRCOSC clock). ... 1111 ***SAFE*** mode request delay = 32768 periods (IRCOSC clock).

Table 255. FCCU_CFG field descriptions (continued)

Field	Description
FCCU_CFG.CM	Configuration mode 0 Configuration labelling: a specific FCCU_F configuration is assigned in CONFIG state. 1 Configuration transparency: the FCCU_F protocol is the same in CONFIG and NORMAL states.
FCCU_CFG.SM	Switching mode 0 FCCU_F protocol (dual-rail, time-switching) slow switching mode. 1 FCCU_F protocol (dual-rail, time-switching) fast switching mode. SM has no effect on the bi-stable protocol.
FCCU_CFG.PS	Polarity selection 0 FCCU_F[1] active high, FCCU_F[0] active high. 1 FCCU_F[1] active low, FCCU_F[0] active low.
FCCU_CFG.FOM	Fault Output Mode selection 000 Dual-Rail (default state; FCCU_F[1:0]= outputs). 001 Time Switching (FCCU_F[1:0]= outputs). 010 Bi-Stable (FCCU_F[1:0]= outputs). 011 Reserved. 100 Reserved. 101 Test0 (FCCU_F[0] = input, FCCU_F[1]= output) 110 Test1 (FCCU_F[0] = output, FCCU_F[1]= output) 111 Test2 (FCCU_F[0] = output, FCCU_F[1]= input) *Note: In Testn mode, a simple double-stage resynchronization stage is used to resynchronize the FCCU_F input/outputs on the system/IRCOSC clock.*
FCCU_CFG.FOP	Fault Output Prescaler FOP defines the prescaler setting used to generate the FCCU_F protocol frequency. 00 0000 Input clock frequency (IRCOSC clock) is divided by 2 00 0001 Input clock frequency (IRCOSC clock) is divided by 4 00 0010 Input clock frequency (IRCOSC clock) is divided by 6 00 0011 Input clock frequency (IRCOSC clock) is divided by 8 00 0100 Input clock frequency (IRCOSC clock) is divided by 10 00 0101 Input clock frequency (IRCOSC clock) is divided by 12 00 0110 Input clock frequency (IRCOSC clock) is divided by 14 The following equation gives the FCCU_F frequency: $F_{FCCU_F} = F_{IRCOSC} / (1024 \times (FOP + 1) \times 2)$

23.6.4 FCCU CF Configuration Register (FCCU_CF_CFG0..3)

The FCCU_CF_CFGx register is accessible in write mode only in the CONFIG state. It contains the configuration of each critical fault in terms of fault recovery management.

The configuration depends on the type of signaling following a fault event. Hardware recoverable faults should be configured only if a previous latching stage captures and hold

the physical fault otherwise the fault can be lost. All the other faults should be configured as SW fault.

Figure 281. FCCU CF Configuration Register (FCCU_CF_CFG0..3)

Offset: 0x00C–0x018 (4 registers) Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CFC 31	CFC 30	CFC 29	CFC 28	CFC 27	CFC 26	CFC 25	CFC 24	CFC 23	CFC 22	CFC 21	CFC 20	CFC 19	CFC 18	CFC 17	CFC 16
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CFC 15	CFC 14	CFC 13	CFC 12	CFC 11	CFC 10	CFC 9	CFC 8	CFC 7	CFC 6	CFC 5	CFC 4	CFC 3	CFC 2	CFC 1	CFC 0
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

1. Writable only in the CONFIG state
2. 1 for FCCU_CF_CFG0; 0 otherwise

Table 256. FCCU_CF_CFG0..3 field descriptions

Field	Description
CFCx	Critical fault configuration 0 Hardware recoverable fault. 1 Software recoverable fault. The critical fault configuration defines the fault recovery mode. Hardware-recoverable faults are self recovered (status flag clearing) if the root cause has been removed. SW recoverable faults are recovered (status flag clearing) by software clearing the related status flag.

23.6.5 FCCU NCF Configuration Register (FCCU_NCF_CFG0..3)

The FCCU_NCF_CFGx register is accessible in write mode only in the CONFIG state. It contains the configuration of each non-critical fault in terms of fault recovery management. The configuration depends on the type of signaling of a fault event. HW recoverable faults should be configured only if a previous latching stage captures and hold the physical fault otherwise the fault can be lost. All the other faults should be configured as SW fault.

Figure 282. FCCU NCF Configuration Register (FCCU_NCF_CFG0..3)

Offset: 0x01C–0x028 (4 registers)

Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFC31	NCFC30	NCFC29	NCFC28	NCFC27	NCFC26	NCFC25	NCFC24	NCFC23	NCFC22	NCFC21	NCFC20	NCFC19	NCFC18	NCFC17	NCFC16
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFC15	NCFC14	NCFC13	NCFC12	NCFC11	NCFC10	NCFC9	NCFC8	NCFC7	NCFC6	NCFC5	NCFC4	NCFC3	NCFC2	NCFC1	NCFC0
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

1. Writable only in the CONFIG state
2. 1 for FCCU_NCF_CFG0; 0 otherwise

Table 257. FCCU_NCF_CFG0..3 field descriptions

Field	Description
NCFCx	*Non-critical fault configuration* 0: HW recoverable fault 1: SW recoverable fault The non-critical fault configuration defines the fault recovery mode. HW recoverable faults are self recovered (status flag clearing and related) if the root cause has been removed. SW recoverable faults are recovered (status flag clearing) by SW clearing of the related status flag. This register can be read and written by the software.

23.6.6 FCCU CFS Configuration Register (FCCU_CFS_CFG0..7)

The FCCU_CF_FS_CFGx register is accessible in write mode only in the CONFIG state. It contains the configuration of each critical fault in terms of fault reaction (short or long 'functional' reset) when it is the root cause for the FAULT state transition.

Figure 283. FCCU CFS Configuration Register 0 (FCCU_CFS_CFG0)

Offset: 0x02C — Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CFSC15		CFSC14		CFSC13		CFSC12		CFSC11		CFSC10		CFSC9		CFSC8	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CFSC7		CFSC6		CFSC5		CFSC4		CFSC3		CFSC2		CFSC1		CFSC0	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

1. Writable only in the CONFIG state

Figure 284. FCCU CFS Configuration Register 1 (FCCU_CFS_CFG1)

Offset: 0x030 — Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFSC31		CFSC30		CFSC29		CFSC28		CFSC27		CFSC26		CFSC25		CFSC24	

Table 258. FCCU_CFS_CFG0..7 field descriptions

Field	Description
CFSCx	*Critical fault state configuration* 00: No reset reaction 01: Short functional reset (FAULT state reaction) 10: Long functional reset (FAULT state reaction) 11: No reset reaction This register can be read and written by the software.

23.6.7 FCCU NCFS Configuration Register (FCCU_NCFS_CFG0..7)

The FCCU_NCF_FS_CFGx register is accessible in write mode only in the CONFIG state. It contains the configuration of each non-critical fault in terms of fault reaction (short or long 'functional' reset) when it is the root cause for the FAULT state transition.

Figure 285. FCCU NCFS Configuration Register (FCCU_NCFS_CFG0..7)

Offset: 0x04C–0x068 (8 registers) Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFSC15		NCFSC14		NCFSC13		NCFSC12		NCFSC11		NCFSC10		NCFSC9		NCFSC8	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFSC7		NCFSC6		NCFSC5		NCFSC4		NCFSC3		NCFSC2		NCFSC1		NCFSC0	
Reset	0/1(2)	0	0/1(2)	0	0/1(2)	0	0/1(2)	0	0/1(2)	0	0/1(2)	0	0/1(2)	0	0/1(2)	0

1. Writable only in the CONFIG state
2. 1 for FCCU_NCFS_CFG0; 0 otherwise

Table 259. FCCU_NCFS_CFG0..7 field descriptions

Field	Description
NCFSCx	*Non-critical fault state configuration* 00: No reset reaction 01: Short 'functional' reset (FAULT state reaction) 10: Long 'functional' reset (FAULT state reaction) 11: No reset reaction This register can be read and written by the software.

23.6.8 FCCU CF Status Register (FCCU_CF_S0..3)

The FCCU_CFSx register contains the latched fault indication collected from the critical fault sources. Faults are latched even if the FCCU is in the CONFIG state and independently from the enabling or reactions programmed for the CF.

No reactions are executed until the FCCU moves in the NORMAL state. FCCU reacts and moves from the NORMAL or ALARM state into the FAULT state if a critical fault is triggered.

The status bits of the FCCU_CFSx register, configured as SW recoverable faults, can be cleared by the following locked sequence:

- to write the proper key into the FCCU_CFK register
- to clear the status (flag) bit CFSx → the opcode OP11 is automatically set into the FCCU_CTRL.OPR field
- to wait for the completion of the operation (FCCU_CTRL.OPS field)
- to read the FCCU_CFSx register in order to verify the effective deletion and in case of failure to repeat the sequence

As result of the above sequence, in addition the FAULT interface provides support to clear the external FAULT root.

The FCCU moves from the FAULT state into the NORMAL state if the source fault which caused the transition into the FAULT state has been removed (HW recoverable fault) or cleared via SW (SW recoverable fault). Concurrently the FAULT interface provides support to clear the FAULT root. In case of nested faults that are not all recovered, the FCCU remains into the FAULT state or moves into the ALARM state.

The SW application executes the FCCU_CFSx read operation by the following sequence:

- to set the OP9 operation into the FCCU_CTRL.OPR field
- to wait for the completion of the operation (FCCU_CTRL.OPS field)
- to read the FCCU_CFSx register

The following errors are ignored:

- to write a wrong key into the FCCU_CFK register
- to attempt to clear a HW recoverable error

Figure 286. FCCU CF Status Register (FCCU_CFS0..3)

Offset: 0x06C–0x078 (4 registers) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS31	CFS30	CFS29	CFS28	CFS27	CFS26	CFS25	CFS24	CFS23	CFS22	CFS21	CFS20	CFS19	CFS18	CFS17	CFS16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFS15	CFS14	CFS13	CFS12	CFS11	CFS10	CFS9	CFS8	CFS7	CFS6	CFS5	CFS4	CFS3	CFS2	CFS1	CFS0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Register FCCU_CFS0 shows status of CF[0:31].

Register FCCU_CFS1 shows status of CF[37] (bit26). All other register bits of FCCU_CFS1 are unused.

Register FCCU_CFS2 and Register FCCU_CFS3 are implemented but are not used. They do not show any status.

Table 260. FCCU_CFS0..3 field descriptions

Field	Description
CFSx	*Critical fault status* 0: No critical fault latched 1: Critical fault latched The status bits related to the critical fault configured as HW recoverable faults are read-only and the flag is self cleared when the fault source is removed. The status bits related to the critical fault configured as SW recoverable faults are write-clear and the SW application can recover from a faulty condition.

23.6.9 FCCU CF Key Register (FCCU_CFK)

The FCCU_CFK register implements the key access to clear the status flags of the FCCU_CFSx register.

The status bits of the FCCU_CFSx register, configured as SW recoverable faults, can be cleared by the following locked sequence:

- Write the CFCK key into the FCCU_CFK register
- Clear the status (flag) bit CFSx

Note: The key must be written for each FCCU_CFSx clear operation.

The FCCU_CFK register is not readable, a 0x00000000 value is always returned in case of read operation.

Figure 287. FCCU CF Key Register (FCCU_CFK)

Offset: 0x07C Access: User write

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

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

Table 261. FCCU_CFK field descriptions

Field	Description
CFK	*Critical fault key* = 0x618B7A50

23.6.10 FCCU NCF Status Register (FCCU_NCF_S0..3)

The FCCU_NCFSx register contains the latched fault indication collected from the non-critical fault sources. Faults are latched also in the CONFIG state and independently from the enabling or reactions programmed for the NCF.
No reactions are executed until the FCCU moves in the NORMAL state.
FCCU reacts and moves from the NORMAL state into the ALARM state only if the respective enable bit for a fault is set in the FCCU_NCFEx register and the respective enable bit for the time-out is set in the FCCU_TOEx register.
FCCU reacts and moves from the NORMAL or ALARM state into the FAULT state if the respective enable bit for a fault is set in the FCCU_NCFEx register and the respective enable bit for the time-out is disabled in the FCCU_TOEx register.
FCCU reacts and moves from the ALARM state into the FAULT state if the time-out (FCCU_TO register) is elapsed before to recovery the fault.

The time-out is stopped only when the FCCU returns in the NORMAL state.

The status bits of the FCCU_NCFSx register, configured as SW recoverable faults, can be cleared by the following locked sequence:

- Write the proper key into the FCCU_NCFK register
- Clear the status (flag) bit NCFSx → the opcode OP12 is automatically set into the FCCU_CTRL.OPR field
- Wait for the completion of the operation (FCCU_CTRL.OPS field)
- Read the FCCU_NCFSx register in order to verify the effective deletion and in case of failure to repeat the sequence

Note: There should not be any other operation while performing the above steps in sequence.

As result of the above sequence, in addition the FAULT interface provides support to clear the external FAULT root.

The FCCU moves from the FAULT or ALARM state into the NORMAL state if all the source faults which caused the transition into the FAULT state has been removed (HW recoverable fault) or cleared via SW (SW recoverable fault). In case of nested faults that are not all recovered, the FCCU will remain in the FAULT or ALARM state.

The SW application executes the FCCU_NCFSx read operation by the following sequence:

- to set the OP10 operation into the FCCU_CTRL.OPR field
- to wait for the completion of the operation (FCCU_CTRL.OPS field)
- to read the FCCU_NCFSx register

In case of re-configuration of the FCCU (CONFIG state), before to return in NORMAL state the pending status bits into the FCCU_NCFSx must be cleared in order to avoid a false transition in ALARM/FAULT state.

The following errors are ignored:

- to write a wrong key into the FCCU_NCFK register
- to attempt to clear a HW recoverable error

Figure 288. FCCU NCF Status Register (FCCU_NCFS0..3)

Offset: 0x080–0x08C (4 registers) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCFS31	NCFS30	NCFS29	NCFS28	NCFS27	NCFS26	NCFS25	NCFS24	NCFS23	NCFS22	NCFS21	NCFS20	NCFS19	NCFS18	NCFS17	NCFS16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NCFS15	NCFS14	NCFS13	NCFS12	NCFS11	NCFS10	NCFS9	NCFS8	NCFS7	NCFS6	NCFS5	NCFS4	NCFS3	NCFS2	NCFS1	NCFS0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 262. FCCU_NCFS0..3 field descriptions

Field	Description
NCFSx	*Non-critical fault status* 0: No “non-critical” fault latched 1: “non-critical fault” latched The status bits related to the non-critical fault configured as HW recoverable faults are read-only and the flag is self cleared when the fault source is removed. The status bits related to the critical fault configured as SW recoverable faults are write-clear and the SW application can recover from a faulty condition.

23.6.11 FCCU NCF Key Register (FCCU_NCFK)

The FCCU_NCFK register implements the key access to clear the status flags of the FCCU_NCFSx register.
The status bits of the FCCU_NCFSx register, configured as SW recoverable faults, can be cleared by the following locked sequence:

- to write the key into the FCCU_NCFK register
- to clear the status (flag) bit NCFSx

Note: The key must be written for each FCCU_NCFSx clear operation.

There should not be any other operation while performing the above steps in sequence.

The FCCU_NCFK register is not readable, a 0x00000000 value is always returned in case of read operation.

Figure 289. FCCU NCF Key Register (FCCU_NCFK)

Offset: 0x090 Access: User write

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

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

Table 263. FCCU_NCFK field descriptions

Field	Description
NCFK	*Critical fault key* = 0xAB3498FE

23.6.12 FCCU NCF Enable Register (FCCU_NCF_E0..3)

The FCCU_NCFEx register enables the critical fault sources to allow a transition from the NORMAL into the FAULT or ALARM state. In case of fault masking, the respective status bit into the FCCU_NCFSx register is anyway set (for debugging purposes), only the reaction is masked.

The FCCU_NCFEs register is accessible in write mode only in the CONFIG state.

Figure 290. FCCU NCF Enable Register (FCCU_NCFE0..3)

Offset: 0x094–0x0A0 (4 registers) Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCFE31	NCFE30	NCFE29	NCFE28	NCFE27	NCFE26	NCFE25	NCFE24	NCFE23	NCFE22	NCFE21	NCFE20	NCFE19	NCFE18	NCFE17	NCFE16
W																
Reset	0	0	0	0	0	0	0	0/1(2)	0/1(2)	0/1(2)	0	0/1(2)	0	0	0/1(2)	0/1(2)

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NCFE15	NCFE14	NCFE13	NCFE12	NCFE11	NCFE10	NCFE9	NCFE8	NCFE7	NCFE6	NCFE5	NCFE4	NCFE3	NCFE2	NCFE1	NCFE0
W																
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0	0	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

1. Writable only in the CONFIG state
2. 1 for FCCU_NCFE0; 0 otherwise

Table 264. FCCU_NCFE0..3 field descriptions

Field	Description
NCFEx	*Non-critical fault enable* 0: No actions following the respective critical fault assertion 1: FCCU moves to ALARM or FAULT state

23.6.13 FCCU NCF Time-out Enable Register (FCCU_NCF_TOE0..3)

The FCCU_NCFTOEx register enables a transition from the NORMAL state into the ALARM state if the respective non-critical fault is enabled (NCFEx and NCFTOEx are set). In case the respective time-out is disabled (NCFTOEx is cleared) and the non-critical fault is enabled (NCFEx is set) the FCCU moves into the FAULT state if the related non-critical fault is asserted. The timer (preset with the time-out value defined by FCCU_TO register) is started when the FCCU moves into the ALARM state. If the fault is not recovered within the time-out the FCCU moves from the ALARM state to the FAULT state.

The FCCU_NCFTOEx register is accessible in write mode, only in the CONFIG state.

Figure 291. FCCU NCF Time-out Enable Register (FCCU_NCF_TOE0..3)

Offset: 0x0A4–0x0B0 (4 registers) Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFTOE31	NCFTOE30	NCFTOE29	NCFTOE28	NCFTOE27	NCFTOE26	NCFTOE25	NCFTOE24	NCFTOE23	NCFTOE22	NCFTOE21	NCFTOE20	NCFTOE19	NCFTOE18	NCFTOE17	NCFTOE16
Reset	0	0	0	0	0	0	0	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0	0/1(2)	0/1(2)

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFTOE15	NCFTOE14	NCFTOE13	NCFTOE12	NCFTOE11	NCFTOE10	NCFTOE9	NCFTOE8	NCFTOE7	NCFTOE6	NCFTOE5	NCFTOE4	NCFTOE3	NCFTOE2	NCFTOE1	NCFTOE0
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

1. Writable only in the CONFIG state
2. 1 for FCCU_NCF_TOE0; 0 otherwise

Table 265. FCCU_NCF_TOE0..3 field descriptions

Field	Description
NCFTOEx	*Non-critical fault time-out enable* 0: FCCU moves into the FAULT state if the respective fault is enabled (NCFEx is set) 1: FCCU moves into the ALARM state if the respective fault is enabled (NCFEx is set)

23.6.14 FCCU NCF Time-out Register (FCCU_NCF_TO)

The FCCU_NCF_TO register defines the preset value of the timer for the recovery of the non-critical faults (enabled). Once FCCU enters in ALARM state, following the assertion of a non-critical fault enabled (NCFEx and NCFTOEx are set), the timer starts the count down. If the fault is not recovered within the time-out the FCCU moves from the ALARM state to the FAULT state.

The FCCU_NCF_TO register is accessible in write mode, only in the CONFIG state.

Figure 292. FCCU NCF Time-out Register (FCCU_NCF_TO)

Offset: 0x0B4 Access: User read/write(1)

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

1. Writable only in the CONFIG state

Table 266. FCCU_NCF_TO field descriptions

Field	Description
TO	*Non-critical fault timeout* $Timeout = (TO) \times T_{RC16MHz}$

23.6.15 FCCU CFG Timeout Register (FCCU_CFG_TO)

The FCCU_CFG_TO register defines the preset value of the watchdog timer for the recovery from the CONFIG state. Once FCCU enters in CONFIG state, following a SW request (OP1 opcode), the watchdog timer is initialized and starts the countdown if the reset is not asserted.
If the configuration is not completed within the time-out, the FCCU moves automatically from the CONFIG state to the NORMAL state and the default values for the configuration register is restored. The watchdog time-out is clocked with the IRCOSC clock (16 MHz). The default time-out value is 4,096 ms.

The FCCU_CFG_TO register is accessible in write mode, in any state excluded the CONFIG state as follows the execution of the OP1 opcode (NORMAL to CONFIG state) and until the completion of the OP2 opcode (CONFIG to NORMAL state).
In case of watchdog time-out the FCCU_CFG_TO register is not accessible until the OP14 operation (CONFIG to NORMAL) has been completed.

Figure 293. FCCU CFG Timeout Register (FCCU_CFG_TO)

Offset: 0x0B8 Access: User read/write

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

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

Table 267. FCCU_CFG_TO field descriptions

Field	Description
TO	*Configuration time-out* $Timeout = T_{RC16MHz} \times 2^{(TO+10)}$ 000: Time-out = 64 µs .... 111: Time-out = 8,192 ms

23.6.16 FCCU IO Control Register (FCCU_EINOUT)

The FCCU_EINOUT register allows the following operations typically in NORMAL state:

- to control the fccu_eout[1] output level when the FCCU is configured in “Test1” or “Test0” fault output mode (FCCU_CFG.FOM)
- to control the fccu_eout[0] output level when the FCCU is configured in “Test1” or “Test2” fault output mode (FCCU_CFG.FOM)
- to observe the fccu_ein[1:0] inputs level

Table 268. Bi-stable encoding

Mode = FCCU_CFG.FOM	FCCU_EOUT[0]	FCCU_EOUT[1]
Test1	output	output
Test2	output	input
Test0	input	output

Note: Due to the resynchronization stage of the EOUT interface there is a latency of a few safe clock cycles following a write/read operation of the FCCU_EINOUT register.

Figure 294. FCCU IO Control Register (FCCU_EINOUT)

Offset: 0x0BC 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	EIN1	EIN0	0	0	EOUT1	EOUT0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 269. FCCU_EINOUT field descriptions

Field	Description
EIN1	*Error Input 1:* 0: when fccu_ein[1] = 0 1: when fccu_ein[1] = 1 The EIN1 captures the fccu_ein[1] input signal. This bit can be read by software.
EIN0	*Error Input 0:* 0: when fccu_ein[0] = 0 1: when fccu_ein[0] = 1 The EIN0 captures the fccu_ein[0] input signal. This bit can be read by software.
EOUT1	*Error Out 1 (relevant only if FCCU_CFG.FOM = Test1 or Test 0; fccu_eout[1] configured in output mode):* 0: force fccu_eout[1] = 0 1: force fccu_eout[1] = 1 The EOUT1 sets or clears the respective fccu_eout[1] output signal if FCCU_CFG.FOM = 110 or 101; otherwise, the value is a don't care. This bit can be read and written by software.
EOUT0	*Error Out 0 (relevant only if FCCU_CFG.FOM = Test1 or Test 2; fccu_eout[0] configured in output mode):* 0: force fccu_eout[0] = 0 1: force fccu_eout[0] = 1 The EOUT0 sets or clears the respective fccu_eout[0] output signal if FCCU_CFG.FOM = 110 or 101; otherwise, the value is a don't care. This bit can be read and written by software.

23.6.17 FCCU Status Register (FCCU_STAT)

The FCCU_STAT register includes the FCCU status for debugging/test purposes. The FCCU finite state machine operates by the IRCOSC clock asynchronous with the system clock. The FCCU status read operation requires a safe mechanism operated by a HW/SW

synchronization sequence. The SW application executes a FCCU status read operation by the following sequence:

- to set the OP3 operation into the FCCU_CTRL.OPR field
- to wait for the completion of the operation (FCCU_CTRL.OPS field)
- to read the FCCU status (FCCU_STAT register)

Figure 295. FCCU Status Register (FCCU_STAT)

Offset: 0x0C0 Access: User read

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

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

Table 270. FCCU_STAT field descriptions

Field	Description
STATUS	*FCCU Status* 000: NORMAL state. 001: CONFIG state 010: ALARM state 011: FAULT state Other: UNKNOWN state This bit can be read by the software.

23.6.18 FCCU SC Freeze Status Register (FCCU_SCFS)

The FCCU)SCFS register contains the status of the RCCx checkers.

The software application executes the FCCU_SCFS read operation by the following sequence:

- Set the OP8 operation into the FCCU_CTRL.OPR field
- Wait for the completion of the operation (FCCU_CTRL.OPS field)
- Read the FCCU_SCFS register

The software application executes the FCCU_SCFS clear operation by the following sequence:

- Set the OP13 operation in the FCCU_CTRL .OPR field
- Wait for the completion of the operation (FCCU_CTRL.OPS field)
- Note: All the freeze registers are cleared by this operation
- Note: The RCCx checkers must be disabled (FCCU_CFG.RCCEx = 00b) during a clear operation; otherwise, the RCCSx flags are asserted.

The SW application clears the NOK from the FCCU SC state by this procedure:

- Enter FCCU CONFIG state
- Read FCCU_CFG register
- Clear the FCCU_CFG FCCU_RCCE0 and FCCU_RCCE1 fields to disable the RCCS
- Read the FCCU_SCFS register
- Verify the FCCU_SCFS FCCU_RCCS0 or FCCU_RCCS1 fields are set
- Clear the FCCU freeze status registers
- Set the FCCU_CFG FCCU_RCCE0 and FCCU_RCCE1 fields to enable the RCCS
- Exit the FCCU CONFIG state

Figure 296. FCCU SC Freeze Status Register (FCCU_SCFS)

Offset: 0x0D4 Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	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	RCCS1	RCCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. Writable only in the CONFIG state

Table 271. FCCU_SC field descriptions

Field	Description
RCCS1	*RCC1 Status* 0: No miscompares 1: A miscompare has been detected. This bit is read or cleared by software.

Table 271. FCCU_SC field descriptions (continued)

Field	Description
RCCS0	*RCC0 Status* 0: No miscompares 1: A miscompare has been detected. This bit is read or cleared by software.

23.6.19 FCCU CF Fake Register (FCCU_CFF)

The FCCU_CFF register contains a unique code to set a “critical fault” in mutually exclusive mode by the external FAULT interface. It allows the SW emulation of the critical faults, by the injection of the fault directly in the FAULT root, in order to verify the entire path and reaction. The fault injection mechanism is optional. The reaction following a fake critical fault cannot be masked. The FCCU_CFF is a write-only register with a set of codes corresponding to each critical fault injection.

Figure 297. FCCU CF Fake Register (FCCU_CFF)

Offset: 0x0D8 Access: User write

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

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

Table 272. FCCU_CFF field descriptions

Field	Description
FCFC	*Fake critical fault code* 00h: Fake critical fault injection at critical fault source 0 01h: Fake critical fault injection at critical fault source 1 02h: Fake critical fault injection at critical fault source 2 .. 1Fh: Fake critical fault injection at critical fault source 31 others: No fault injection These bits are always read as ‘0’ by software.

23.6.20 FCCU NCF Fake Register (FCCU_NCFF)

The FCCU_NCFF register contains a unique code to set a “non-critical fault” in mutually exclusive mode by the external FAULT interface. It allows the SW emulation of the non-critical faults, by the injection of the fault directly in the FAULT root, in order to verify the entire path and reaction. The fault injection mechanism is optional. The reaction following a fake non-critical fault can be masked. The FCCU_NCFF is a write-only register with a set of codes corresponding to each non-critical fault injection.

Figure 298. FCCU NCF Fake Register (FCCU_NCFF)

Offset: 0x0DC Access: User write

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

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

Table 273. FCCU_NCFF field descriptions

Field	Description
FNCFC	*Fake non-critical fault code* 00h: Fake non-critical fault injection at non-critical fault source 0 01h: Fake non-critical fault injection at non-critical fault source 1 02h: Fake non-critical fault injection at non-critical fault source 2 .. 1Fh: Fake non-critical fault injection at non-critical fault source 31 others: No fault injection These bits are always read as ‘0’ by software.

23.6.21 FCCU IRQ Status Register (FCCU_IRQ_STAT)

The FCCU_IRQ_STAT register defines the FCCU interrupt status register related to the following events:

- Configuration time-out error
- Alarm interrupt
- NMI interrupt

The external interrupt is asserted if any interrupt status bit of the FCCU_IRQ_STAT is set and, in case of CFG_TO_STAT, the respective enable bit of the FCCU_IRQ_EN register is also set.

The NMI and ALARM interrupts are asserted and cleared according to the FCCU state. The status bits of the FCCU_IRQ_STAT trace the status of the related interrupt lines.

Figure 299. FCCU IRQ Status Register (FCCU_IRQ_STAT)

Offset: 0x0E0 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	NMI_STAT	ALRM_STAT	CFG_TO_STAT
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 274. FCCU_IRQ_STAT field descriptions

Field	Description
CFG_TO_STAT	*Configuration Time-out Status* 0: No configuration time-out error 1: Configuration time-out error This bit can be read and cleared by the software.
ALRM_STAT	*Alarm Interrupt Status* 0: Alarm interrupt is OFF 1: Alarm interrupt is ON This bit can be only read by the software.
NMI_STAT	*NMI Interrupt Status* 0: NMI interrupt is OFF 1: NMI interrupt is ON This bit can be only read by the software.

23.6.22 FCCU IRQ Enable Register (FCCU_IRQ_EN)

The FCCU_IRQ_EN register defines the FCCU interrupt enable register related to the following events:

- Configuration time-out error

The external interrupt is asserted if any interrupt status bit of the FCCU_IRQ_STAT is set and the respective enable bit of the FCCU_IRQ_EN register is also set.

Figure 300. FCCU IRQ Enable Register (FCCU_IRQ_EN)

Offset: 0x0E4 Access: User read/write

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

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

Table 275. FCCU_IRQ_EN field descriptions

Field	Description
CFG_TO_IEN	*Configuration Time-out Interrupt Enable* 0: Configuration time-out interrupt disabled 1: Configuration time-out interrupt enabled This bit can be read and written by the software.

23.6.23 FCCU XTMR Register (FCCU_XTMR)

The FCCU_XTMR register contains the read values of the Alarm, Watchdog or Safe Mode Request Timer. These timers are clocked on the IRCOSC clock.

The SW application executes the timer read operation by the following sequence:

- to set the OP17 or OP18 or OP19 operation into the FCCU_CTRL.OPR field
- to wait for the completion of the operation (FCCU_CTRL.OPS field)
- to read the FCCU_XTMR register

Table 276. Timer state/value

TIMER	CONFIG state	NORMAL state	ALARM state	FAULT state
ALARM	00000000h	Initial value	Running	Idle/End of count
SMRT	00000001h	Initial value	—	Running/End of count
CFG	Running	0001FFFFh	0001FFFFh	0001FFFFh

Figure 301. FCCU XTMR Register (FCCU_XTMR)

Offset: 0x0E8 Access: User read

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

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

Table 277. FCCU_XTMR field descriptions

Field	Description
XTMR	*Alarm/Watchdog/Safe request timer* The current timer value is measured in IRCOSC clock cycles. These bits can be read by the software.

23.6.24 FCCU MCS Register (FCCU_MCS)

The FCCU_MCS register contains a queue of the last 4 chip modes. MCS0 is the latest one, while MCS3 is the oldest one. In addition a qualifier indicates if the FCCU is in the FAULT state when the chip mode has been captured. The chip mode is synchronous to the system clock and provided by a different module while the FCCU state is synchronous to the IRCOSC clock, therefore some uncertainty must be considered regarding the FAULT state indication.

Figure 302. FCCU MCS Register (FCCU_MCS)

Offset: 0x0EC Access: User read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VL3	FS3	0	0	MCS3				VL2	FS2	0	0	MCS2			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VL1	FS1	0	0	MCS1				VL0	FS0	0	0	MCS0			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 278. FCCU_MCS field descriptions

Field	Description
VLx	*Valid* It indicates that the correspondent MCSx and FSx fields are valid. 0: MCSx, FSx fields are not significative 1: MCSx, FSx fields are significative These bits can be read by the software.
FSx	*Fault status* It indicates that the correspondent MCSx field has been captured when the FCCU is in FAULT state. 0: MCSx field captured in any state different from the FAULT state 1: MCSx field captured in FAULT state These bits can be read by the software.
MCSx	*Chip mode* The MCSx is the chip mode. MCS0 = latest state MCS3 = oldest state On any chip mode change the previous chip modes are shifted (MCS3 = MCS2, MCS2= MCS1, MCS1= MCS0) and the latest one is captured in MCS0. These bits can be read by the software.

23.7 Functional description

23.7.1 Definitions

In general, the following definitions are applicable for the fault management:

- HW recoverable fault: the fault indication is a level sensitive signal that is asserted as long as the fault cause has not been removed. Typically the fault signal is latched in a external module at the FCCU. The FCCU state transitions are consequently executed on the state changes of the input fault signal. No SW intervention in the FCCU is required to recover the fault condition.
- SW recoverable fault: the fault indication is a signal asserted without a defined time duration. The fault signal is captured in the FCCU. The fault recovery is executed following a SW recovery procedure (status/flag register clearing).

The following type of reset are applicable (see *Chapter 42: Reset Generation Module (MC_RGM)*):

- ‘Destructive’ reset: any type of reset related to a power failure condition that implies a complete system reinitialization
- Long ‘functional’ reset: it implies the flash memory and digital circuitry (most of it with some exceptions FCCU, STCU) initialization
- short functional reset: it implies the digital circuitry (most of it with some exceptions FCCU, STCU) initialization.

23.7.2 FSM description

The FCCU functionality is depicted by the FSM diagram given in *Figure 303*.

Basically four states are identified with the following meaning:

- CONFIG: the configuration state is used only to modify the default configuration of the FCCU. A sub-set of the FCCU registers, dedicated to define the FCCU configuration (global configuration, reactions to fault, time-out, non-critical fault masking) can be accessed in write mode only in the CONFIG state.
  The CONFIG state is accessible only from NORMAL state and if the configuration is not locked. The configuration lock can be disabled only by a global reset of the FCCU. The configuration shall always be locked when running a safety critical application.
  The CONFIG to NORMAL state transition can be executed by SW or automatically following a time-out condition of the watchdog.
  The incoming faults, occurring during the configuration phase (CONFIG state) are latched in order to process them when the FCCU is moved into the NORMAL state, according to the selected configuration.
- NORMAL: the FCCU operating state when no faults are occurring. It is also the default state on the reset exit. The FSM will leave the NORMAL state following one of these events:
  - Critical faults → the FCCU moves to the FAULT state
  - Unmasked non-critical faults with the time-out disabled → the FCCU moves to the FAULT state
  - Unmasked non-critical faults with the time-out enabled → the FCCU moves to the ALARM state
  - Masked non-critical faults → the FCCU stays in NORMAL state
- ALARM: the FCCU moves into the ALARM state when an unmasked non-critical fault occurs and the time-out is enabled. The transition to the ALARM state goes along with an interrupt. By definition, this fault may be recovered within a programmable time-out period, before it generates a transition to the FAULT state. The time-out is reinitialized if the FCCU state moves to the NORMAL state. The time-out is stopped if the FCCU state moves to the FAULT state due to a critical-fault occurring when the FCCU is in ALARM state. The time-out restarts following the recovery from the FAULT state.
- FAULT: the FCCU moves into the FAULT state when one of the following condition occurs:
  - Critical fault
  - Time-out related to a non-critical fault when the FCCU is in the ALARM state
  - Unmasked non-critical faults with the time-out disabled
- The transition from NORMAL/ALARM state goes along with the generation of:
  - NMI interrupt
  - FCCU_F signalling
  - ***SAFE*** mode request after a certain amount of time
  - SW option: Soft reaction (Short ‘functional’ reset)
  - SW option: Hard reaction (Long ‘functional’ reset)

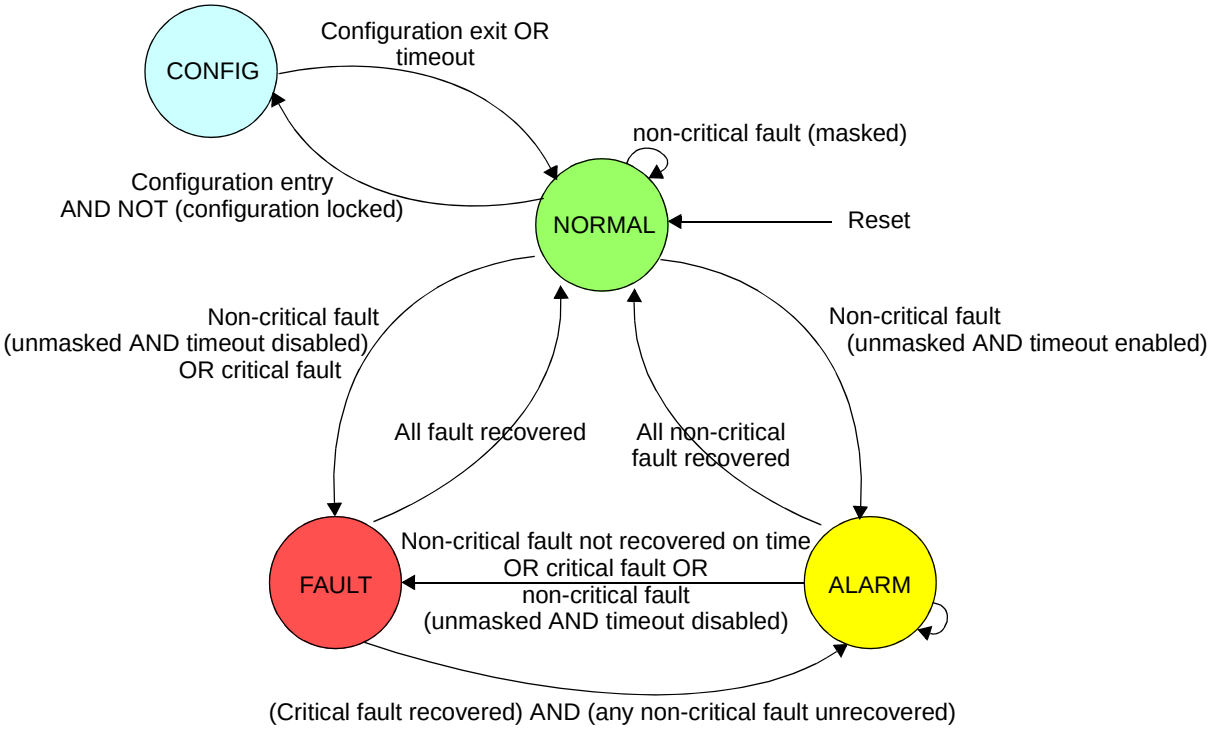


Figure 303. FCCU state diagram

23.7.3 Self-checking capabilities

The FCCU includes some features to support self-checking capabalities of the main FSM in running mode. The FSM unit is duplicated and its state (internal state and relevant outputs) is checked by 2 RCCx units (cycle accurate). The following checks (per IRCOSC clock cycle) are provided by the RCCUx units:

- all the FSM outputs and its internal state for the redundant FSM instances are checked to detect runtime faults on the FSM outputs and its internal state
- parity bits (computed at byte level) on the configuration registers are checked to detect run time faults on the configuration inputs of the FSM
- parity bits (computed at byte level) on the interface used to clear the FCCU_CFSx and FCCU_NCFSx status registers
- FCCU_F protocol state in dual-rail, time-switching and bi-stable mode
- common mode signals (handshake interface signals activated only in CONFIG state) congruence with the FSM state

In case of failure, each RCCx unit provides an interrupt request.

The RCCx state is frozen in a status register.

To guarantee the self checking capabilities through the FAULT interface, each critical fault source must be duplicated on a couple of CF inputs.

Two separate IRCOSC clock inputs are routed to the redundant sub-modules (FSMx, RCCx, FCCU_Fx, FAULT-if). An external (to the FCCU) monitor of the IRCOSC clocks and an external reaction ('functional' reset to the MC_RGM) is required to cover potential fault on the IRCOSC clocks.

Self checking capabilities cover the HW reaction of the FCCU due to the assertion of an external fault. The register interface and the handshake modules are not covered by self-checking capabilities. Any operation executed by SW (configuration, fault recovery) must be cross-verified by redundancy checks via SW (registers read following a write/clear operation, internal FCCU state, and so on). In order for the SW application to clear the NOK from the FCCU SC state it must follow the procedure provided with the description of the FCCU SCFS register.

A typical sequence with self checking failure and related reaction is given in *Figure 304*.

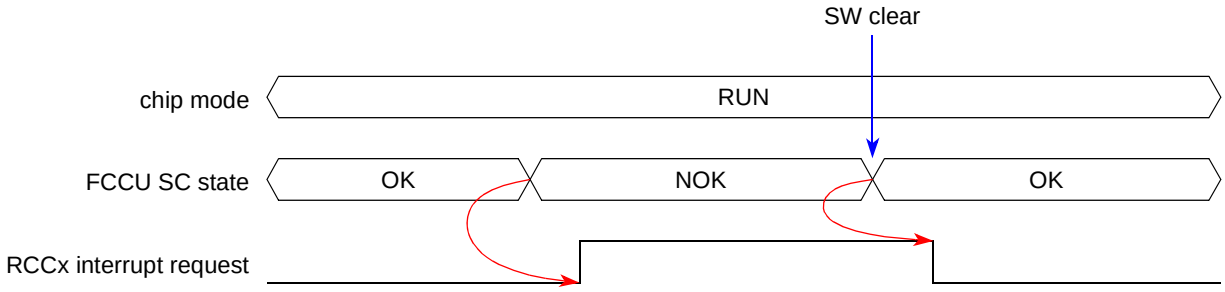


Figure 304. Self-checking reaction

23.7.4 Reset interface

The FCCU has two input resets and two output resets related to the FAULT state.

Table 279. Reset sources

Reset source	Support
External reset	enabled
POR	enabled
STCU	enabled
'Functional' reset	disabled
'Destructive' reset	enabled

23.7.5 Fault priority scheme and nesting

The FAULT state has a higher priority than the ALARM state in case of concurrent fault events (critical and non-critical) that occur in the NORMAL state. In case of concurrent critical faults, the fault reaction corresponds to the worst case (that is, a long 'functional' reset is asserted in case it has been programmed).

The ALARM to FAULT state transition occurs if a critical fault or a non-critical fault (unmasked and with time-out disabled) is asserted in the ALARM state.

Any critical fault (programmed to react with a hard or soft reaction) that occurs when the FCCU is already in the FAULT state causes an immediate hard or soft reaction (long or short 'functional' reset).

The ALARM to NORMAL state transition occurs only if all the non-critical faults (including the faults that have been collected after the entry in the ALARM state) have been cleared (SW or HW recovery) otherwise the FCCU will remain in the ALARM state.

The FAULT to NORMAL state transition occurs only if all the critical and non-critical faults (including the faults that have been collected after the entry in the FAULT/ALARM state) have been cleared (SW or HW recovery) otherwise the FCCU will remain in the FAULT state (if any critical fault is still pending) or will return in the ALARM state (if any non-critical fault is still pending and the time-out is not elapsed).

In general, no fault nesting is supported except for the non-critical versus critical faults that causes a ALARM to FAULT state transition. In this case the NCT timer is stopped until the FAULT state is recovered.

23.7.6 Fault recovery

The following timing diagrams describe the main use cases of the FCCU in terms of fault events and related recovery.

A typical sequence related to a critical FAULT management, given in *Figure 305* or *Figure 306*, is following described:

- Critical fault assertion
- FCCU state transition (automatic): NORMAL → FAULT
  - Short 'functional' reset
  - NMI assertion
  - ***SAFE*** mode request (delayed)
- Chip mode transition: ***RUN*** → ***RESET*** → ***SAFE***
- NMI interrupt management
  - FAULT recovery (by SW): FCCU state transition FAULT → NORMAL
  - Chip mode transition: ***SAFE*** → ***RUN***

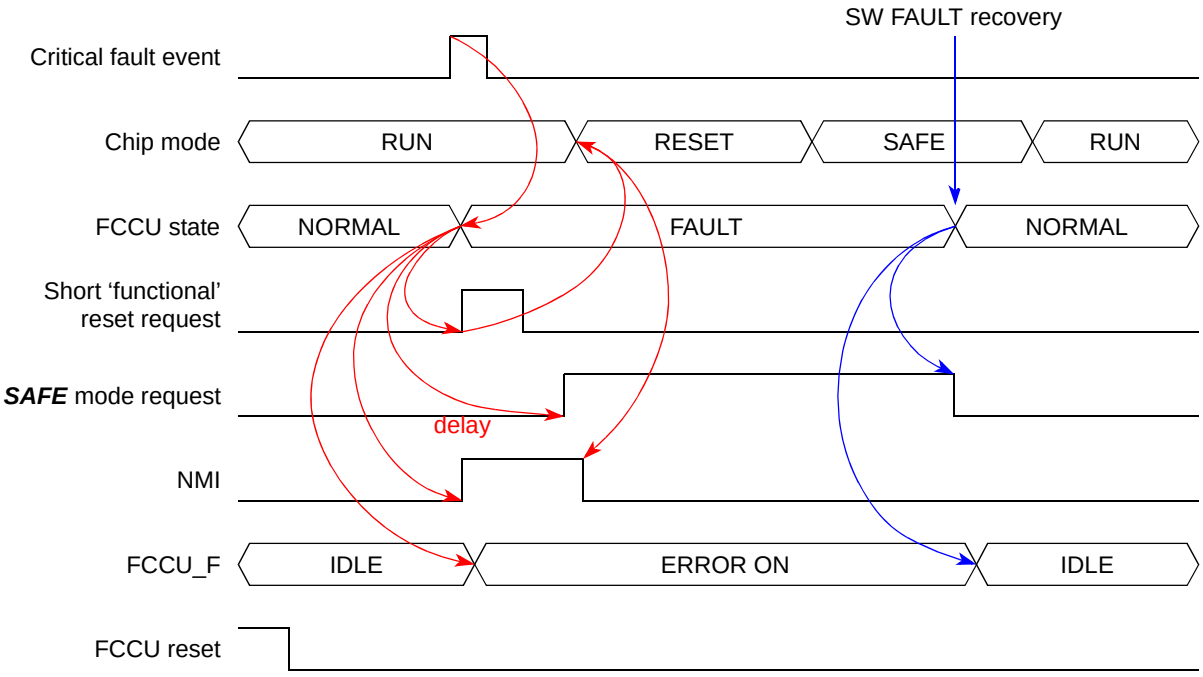


Figure 305. Critical FAULT recovery (a)

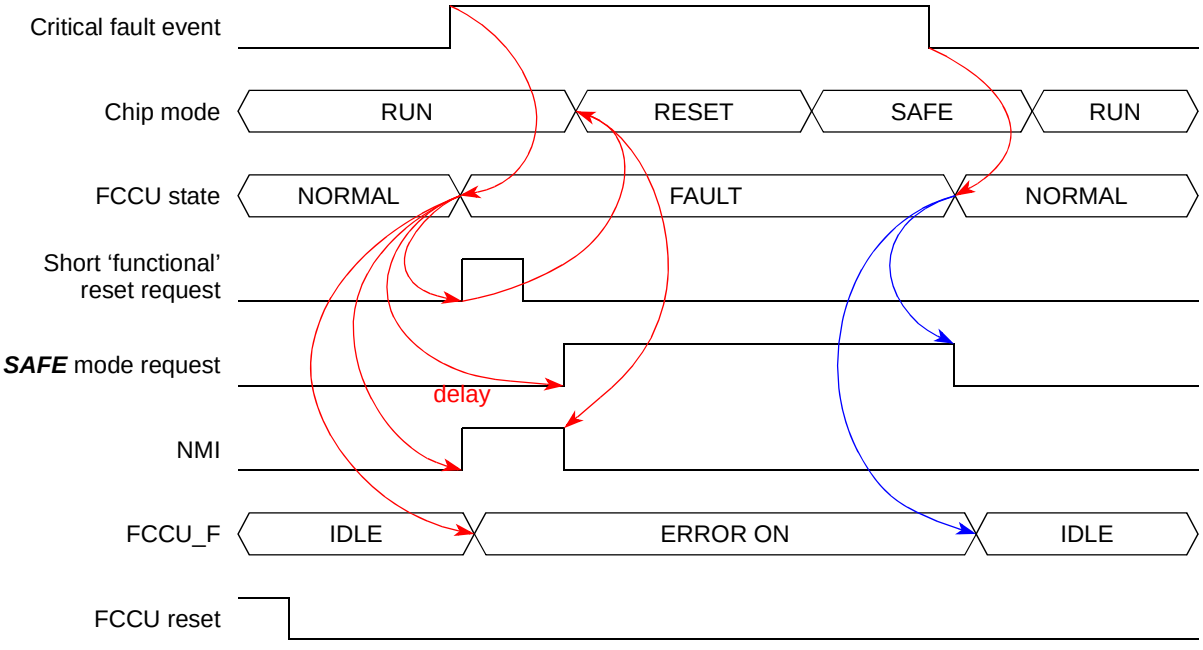


Figure 306. Critical FAULT recovery (b)

A typical sequence related to a non-critical FAULT management (ALARM state), given in *Figure 307* and *Figure 308*, is following described:

- Non-critical fault assertion
- FCCU state transition (automatic): NORMAL → ALARM
  - alarm interrupt request
  - time-out running
- Chip mode: RUN
- Alarm interrupt management
  - FAULT recovery (by SW): FCCU state transition ALARM → NORMAL

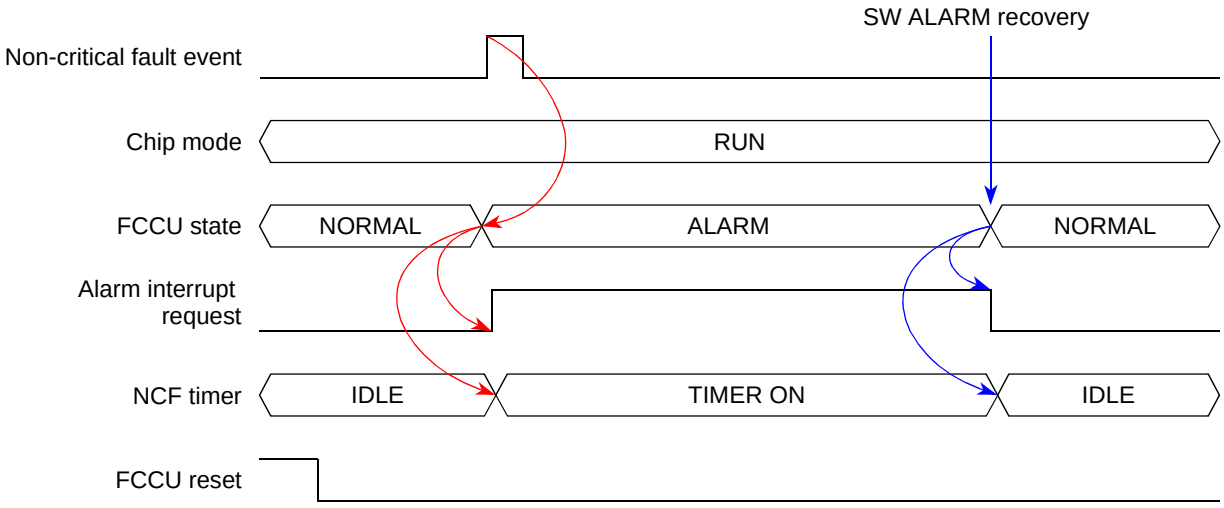


Figure 307. Non-critical FAULT (ALARM state) recovery (a)

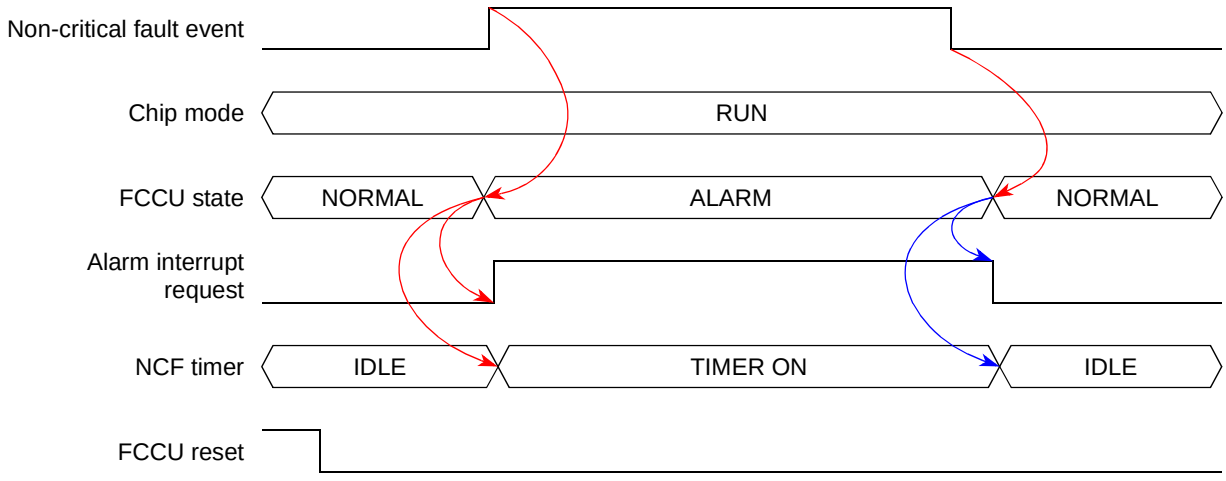


Figure 308. Non-critical FAULT (ALARM state) recovery (b)

A typical sequence related to a non-critical FAULT management (ALARM → FAULT state), given in *Figure 309*, is following described:

- Non-critical fault assertion
- FCCU state transition (automatic): NORMAL → ALARM
  - Alarm interrupt request
  - Time-out running
- FCCU state transition (following the time-out trigger): ALARM → FAULT
  - NMI assertion
  - ***SAFE*** mode request (delayed)
- Chip mode transition: ***RUN*** → ***SAFE***
- NMI interrupt management
  - FAULT recovery (by SW): FCCU state transition FAULT → NORMAL
  - System state transition: ***SAFE*** → ***RUN***

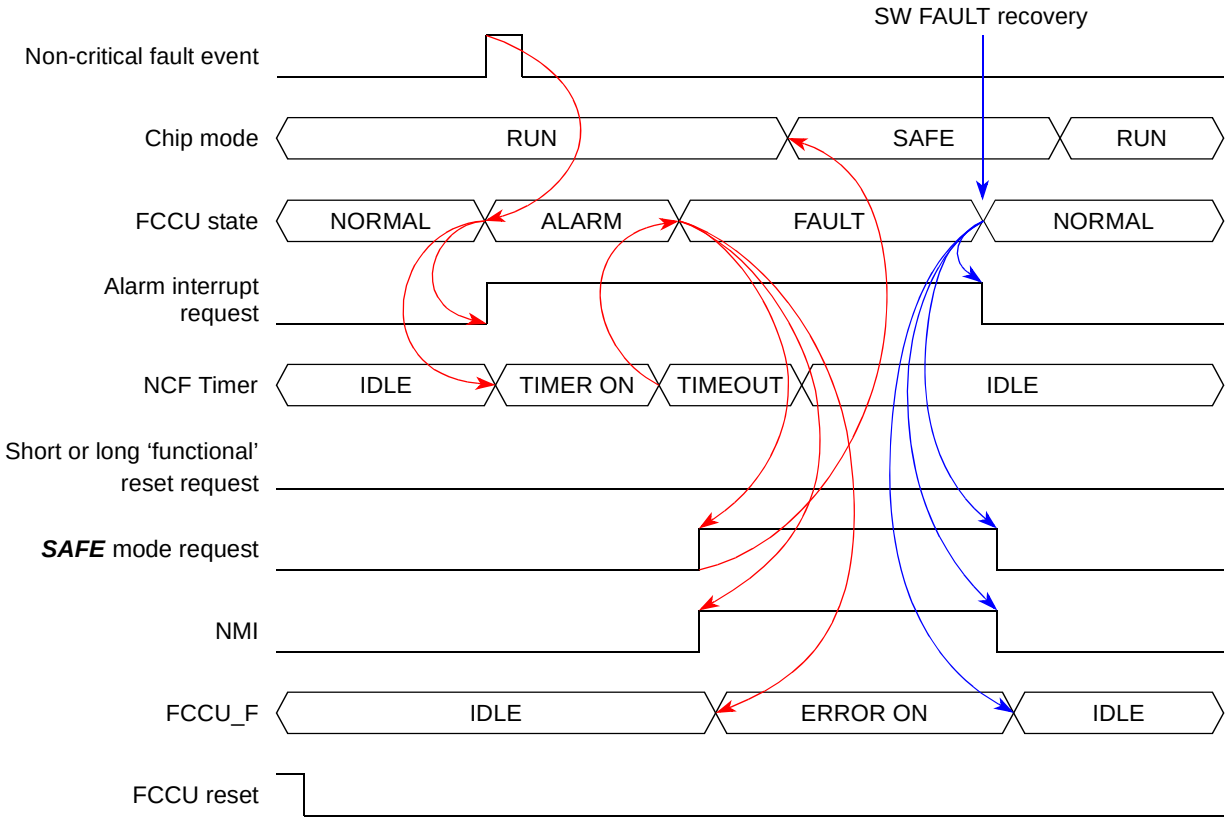


Figure 309. Non-critical FAULT (ALARM -> FAULT state) recovery

A typical sequence related to a critical FAULT (with nesting) management, given in *Figure 310*, is following described:

- Critical fault assertion (no short or long 'functional' reset)
- FCCU state transition (automatic): NORMAL → FAULT
  - NMI assertion
  - ***SAFE*** mode request (delayed)
- Chip mode state transition: ***RUN*** → ***SAFE***
- Critical fault assertion
  - Short 'functional' reset
- Chip mode transition: ***SAFE*** → ***RESET***
- NMI interrupt management
  - FAULT recovery (by SW): FCCU state transition FAULT → NORMAL
  - System state transition: SAFE → RUN

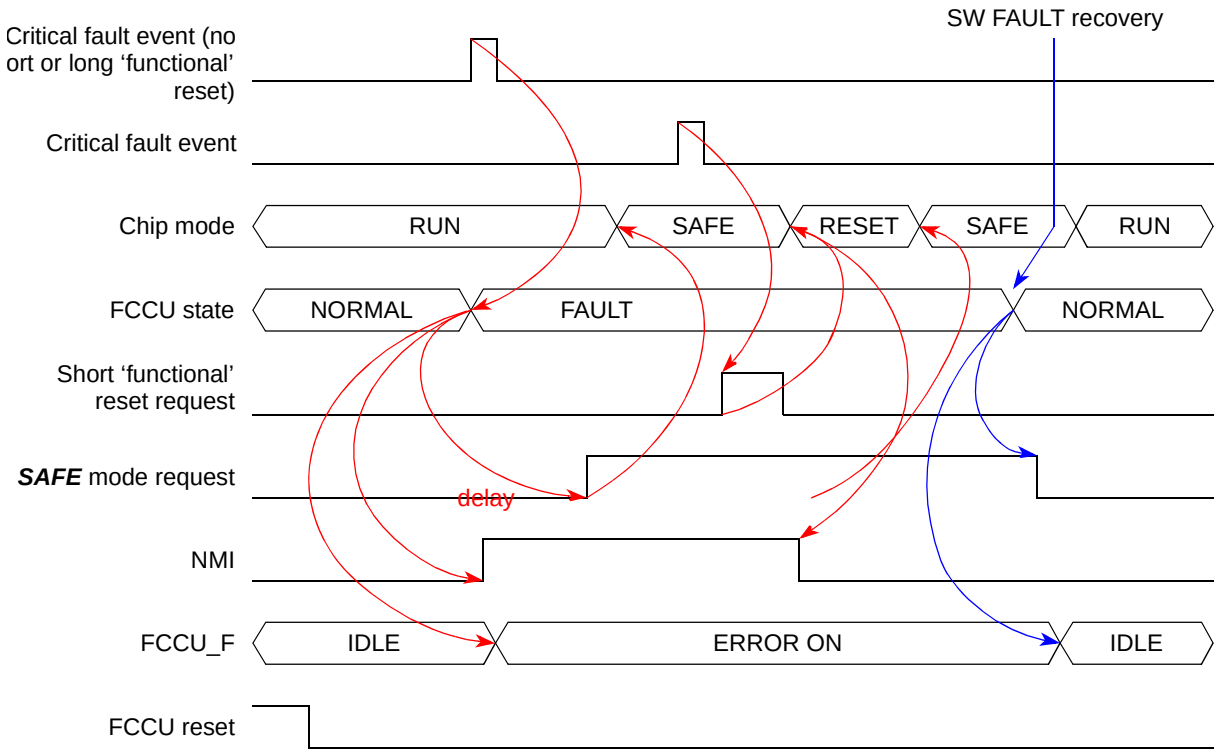


Figure 310. Critical FAULT (nesting) recovery

A typical sequence related to a critical FAULT (with non-critical fault nesting) management (ALARM → FAULT → ALARM state), given in *Figure 311*, where the faults are recovered sequentially, is following described:

- Non-critical fault assertion
- FCCU state transition (automatic): NORMAL → ALARM
  - Alarm interrupt request
  - Time-out running
- Critical fault assertion
- FCCU state transition (automatic): ALARM → FAULT
  - NMI assertion
  - ***SAFE*** mode request
- Chip mode transition: ***RUN*** → ***SAFE***
- NMI interrupt management
  - FAULT (CF) recovery (by SW): FCCU state transition FAULT → ALARM, because only the critical fault has been recovered
  - Chip mode transition: SAFE → RUN
  - Time-out is still running
- Alarm interrupt management
  - FAULT (NCF) recovery (by SW): FCCU state transition ALARM → NORMAL

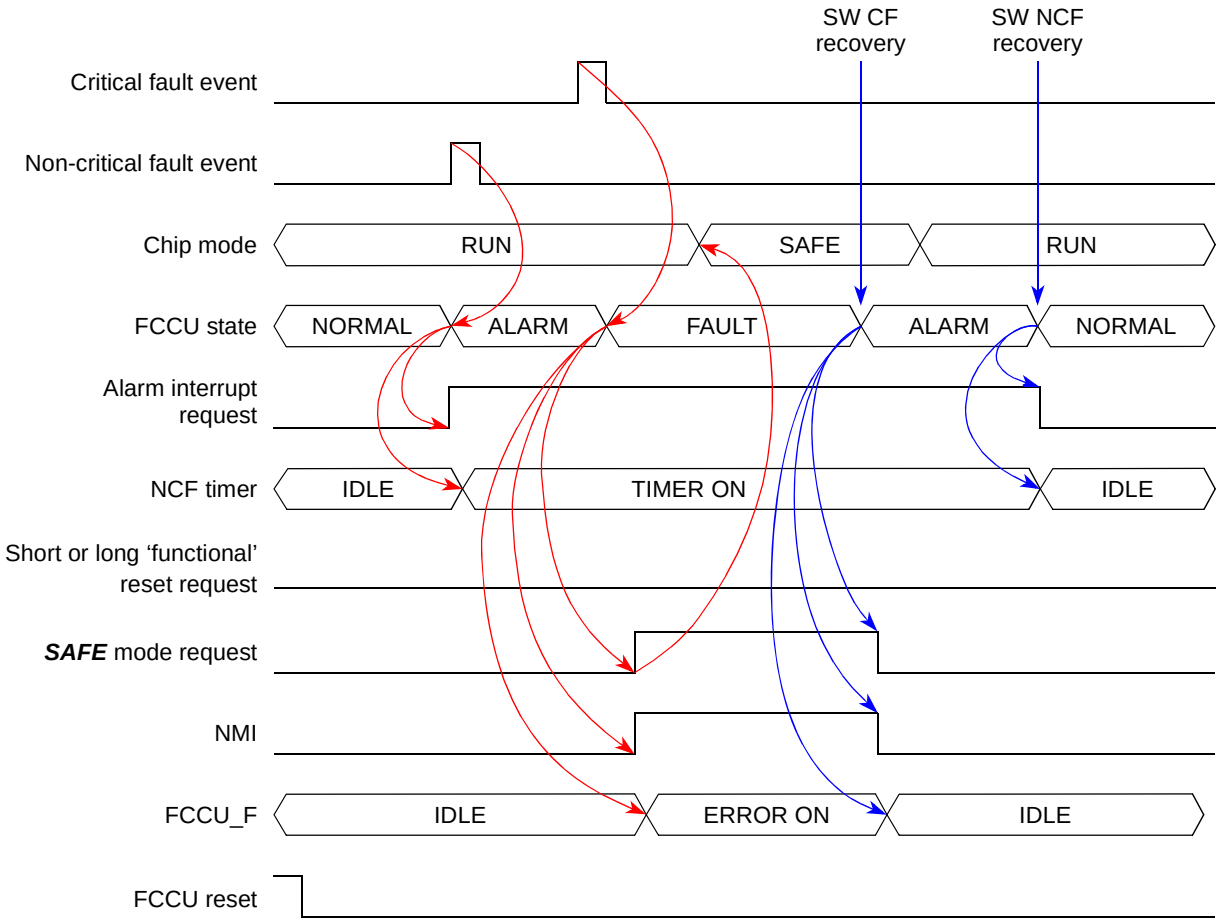


Figure 311. Critical FAULT (and non-critical FAULT nesting) recovery

23.7.7 WKUP/NMI interface

The NMI signal internally generated by FCCU is masked when:

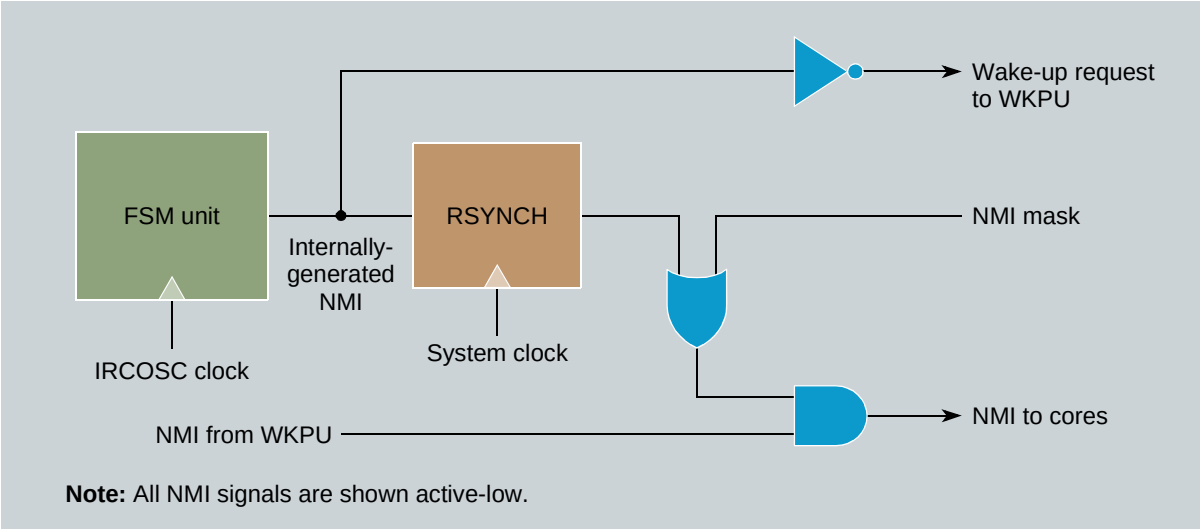
- FCCU asynchronous reset
- Chip mode = RESET

and un-masked when:

- A SW change request of the chip mode is triggered

The logical scheme is given in *Figure 312*.

Figure 312. NMI/WKUP scheme



23.7.8 STCU interface

The STCU interface includes:

- A set of signals resulting from the self-checking procedure connected externally at the FCCU critical/non-critical faults. The STCU fault signals are processed by the FCCU when the chip is re-booted following the self testing procedure. The STCU includes also a status register that stores the self-testing results (flags).
- A mask that inhibits the FCCU_F dummy signaling until the STCU self-checking procedure has been completed.
- During the self testing procedure, depending on the STCU results, 3 cases are applicable:
  - STCU completes the self testing procedure successfully. The chip re-boots and the FCCU is responsible to provide a reaction. (See *Figure 313*.)
  - STCU completes the self testing procedure with low severity failures. The FCCU is responsible to provide the proper reactions according to the fault occurred. (See *Figure 314*.)
  - STCU completes the self testing procedure with serious failures. The FCCU or other critical parts of the chip could not be able to provide the proper reaction. STCU should keep permanently the chip in RESET state. This feature is optionally programmable inside the STCU. (See *Figure 315*.)

Figure 313. STCU-FCCU (case a)

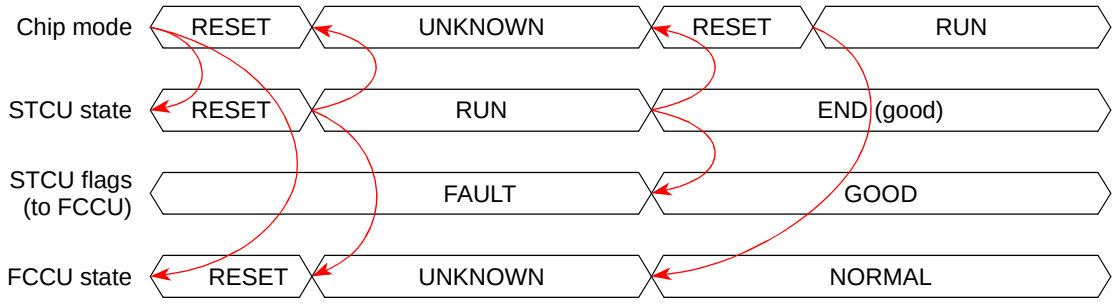


Figure 314. STCU-FCCU (case b)

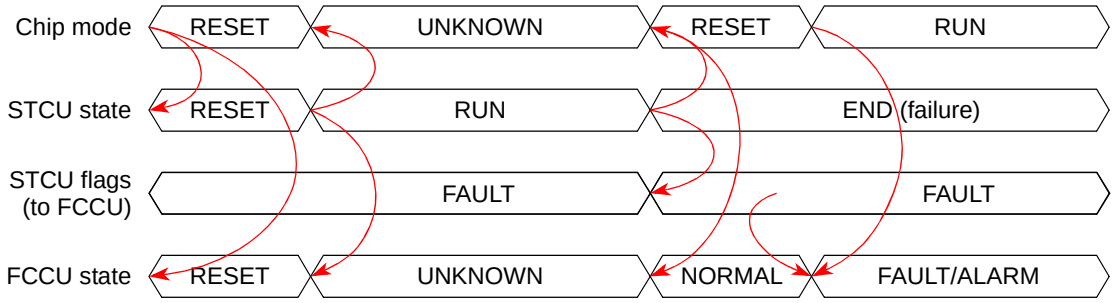
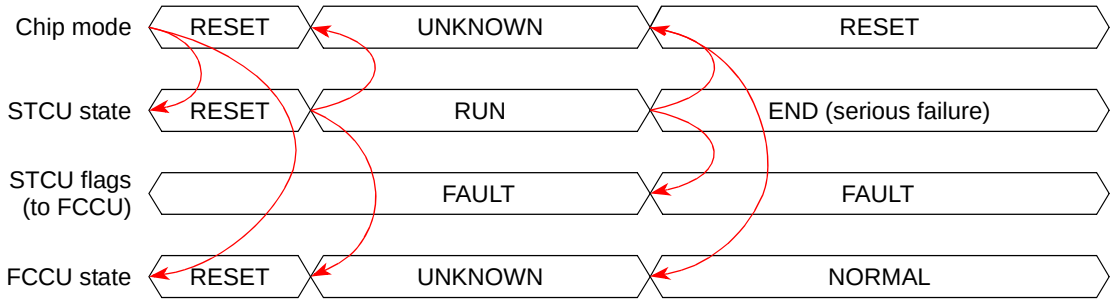


Figure 315. STCU-FCCU (case c)



For details related to the STCU refer to [3].

23.7.9 NVM interface

The NVM provides the FCCU with the initial configuration information shown in *Table 280*.

Table 280. NVM configuration

Flash memory option bit locations	Description
BIU4[20]	Initial FCCU_CFG.CM value
BIU4[21]	Initial FCCU_CFG.SM value
BIU4[22]	Initial FCCU_CFG.PS value
BIU4[23:25]	Initial FCCU_CFG.FOM value
BIU4[26:31]	Initial FCCU_CFG.FOP value

Figure 316 shows the FCCU configuration sequence after a power-on, ‘destructive’, or external reset.

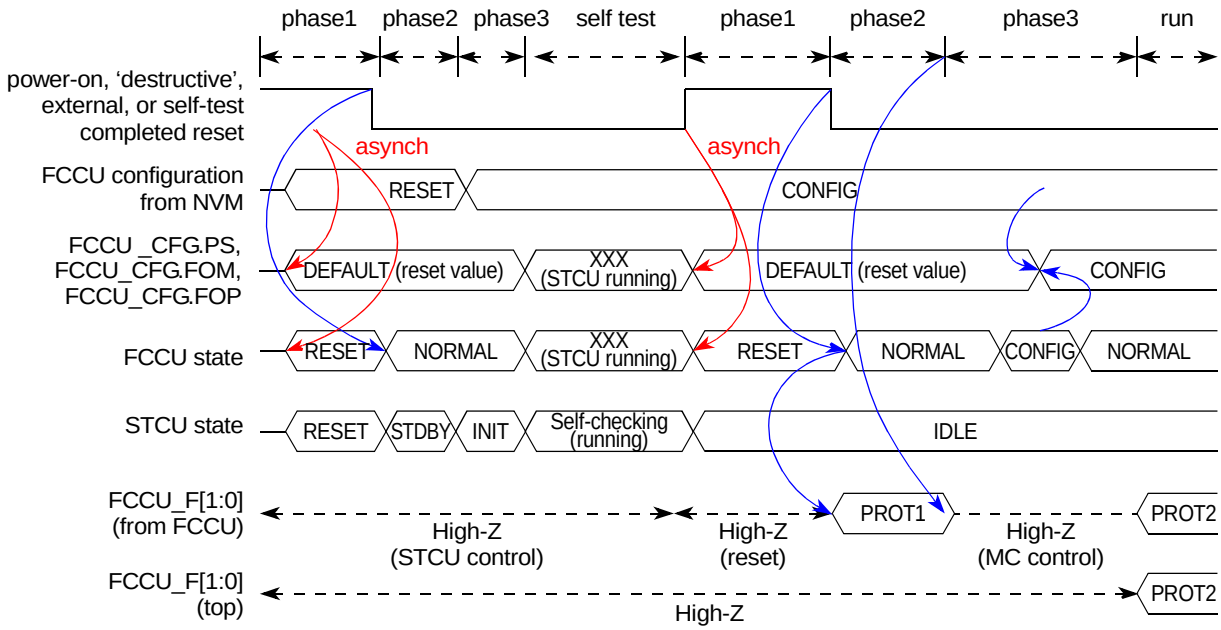


Figure 316. NVM interface

23.7.10 FCCU_F interface

The FCCU provides 2 external bidirectional signals (FCCU_F interface). Different protocols for the FCCU_F interface are supported, selecting the FCCU_CFG.FOM register field:

- Dual rail protocol
- Time switching protocol
- Bi-stable protocol
- Test mode

The signal polarity and the frequency can be programmed, setting the FCCU_CFG.PS and FCCU_CFG.FOP register fields. All the diagrams and tables are related to the default configuration selection (FCCU_CFG.FOP = 0b), switching mode (FCCU_CFG.SM = 0b) and config mode (FCCU_CFG.CM = 0b). In case of inverted polarity (FCCU_CFG.PS = 1b) all the values on the FCCU_F output pins are inverted.

2 modes can be programmed to define the FCCU_F protocol transitions in dual-rail or time-switching mode:

- Slow switching mode: no FCCU_F frequency violation during the FCCU state transition (NORMAL to ERROR or viceversa and CONFIG to NORMAL). The FCCU_F protocol transition occurs after a max delay equal to the duration of the semi-period of the FCCU_F frequency.
- Fast switching mode: The FCCU_F protocol transition (NORMAL to ERROR or viceversa and CONFIG to NORMAL) occurs immediately. A pulse with the minimum duration corresponding to 16 MHz / 1024 (IRCOSC clock) period can occurs in fast switching mode. It implies a frequency violation of the FCCU_F protocol.

Two modes, depending on the FCCU_CFG.CM bit setting, can be programmed to define the FCCU_F protocol in CONFIG state:

- configuration labelling: the CONFIG state is marked by a specific FCCU_F setting
- configuration transparency: the CONFIG and NORMAL state are equivalent

The FCCU_F frequency is programmable based on the IRCOSC clock frequency divided by a fixed prescaler (1024).

The external monitor of the FCCU_F protocol should oversample the FCCU_F signals in order to synchronize periodically the external clock (used by the monitor) and the IRCOSC clock detecting the edge transition of the FCCU_F protocol in dual-rail or time-switching mode.

Note: The initial values, after the reset phase, of the FCCU_CFG.CM, FCCU_CFG.SM, FCCU_CFG.PS, FCCU_CFG.FOP, FCCU_CFG.FOM registers are set by the NVM interface (see Section 23.7.9, NVM interface).

23.7.10.1 Dual-rail protocol

Dual-rail encoding is an alternate method for encoding bits. In contrast with classical encoding, where each signal carries a single-bit value, dual-rail encoded circuits use two wires to carry each bit. The encoding scheme is given in *Table 281* and the related timing diagram is given in *Figure 317* and *Figure 318*.

Table 281. Dual-rail encoding

Logical state	Dual-rail encoding (output pins FCCU_F[1:0])	Note
non-faulty	10	toggling
non-faulty	01	
faulty	00	toggling
faulty	11	
reset	high-Z	no toggling
configuration	high-Z	when FCCU_CFG.CM = 0
	= non-faulty	when FCCU_CFG.CM = 1

As long as FCCU is in NORMAL or ALARM state, output will show “no-faulty“signal. Output pins FCCU_F[0] and FCCU_F[1] will toggle between 01 and 10 with a given frequency. By default the frequency is the IRCOSC clock frequency divided by 18*1024.

During the RESET phase and during self testing the output pins are set as “high impedance”.

Note: Figure 317 and Figure 318 are formatted to display the behavior in all four phases (reset, normal, error, and config), not to imply transitions between one phase to another. In particular, transition from error phase to config phase is not possible.

Figure 317. Dual-rail protocol (slow switching mode)

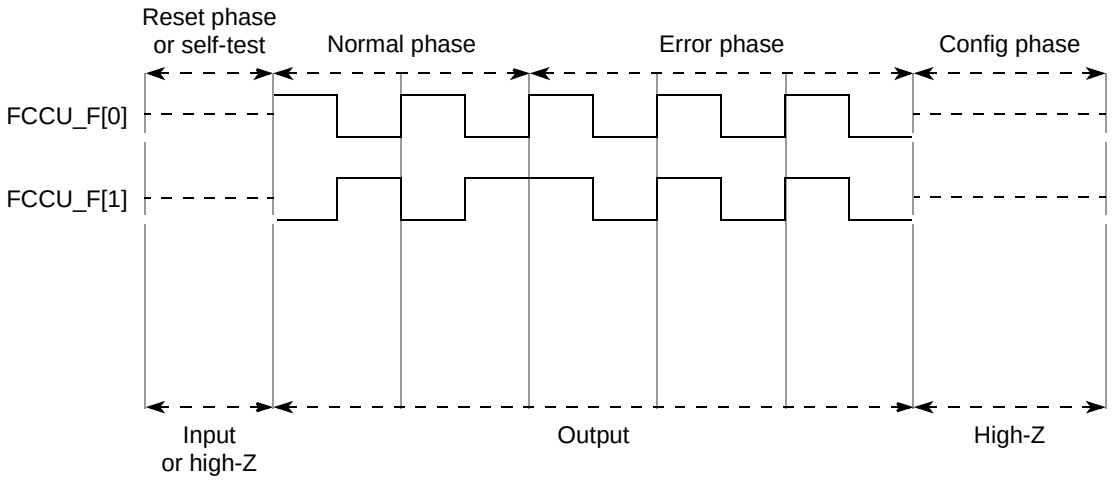
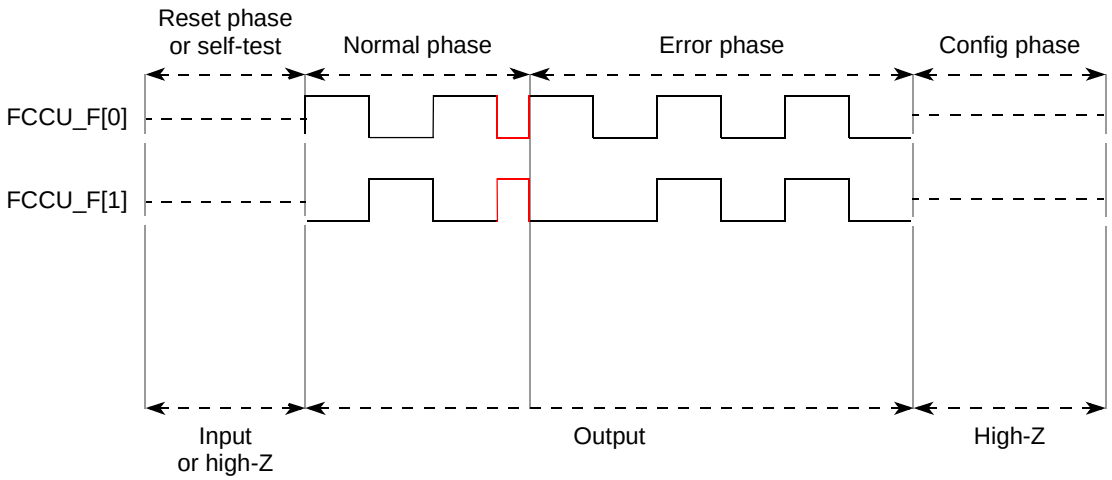


Figure 318. Dual-rail protocol (fast switching mode)



23.7.10.2 Time switching protocol

The encoding scheme is given in *Table 282* and the related timing diagram is given in *Figure 319*.

Table 282. Time switching encoding

Logical state	Time switching encoding (output pins FCCU_F[1:0])	Note
non-faulty	10	toggling
non-faulty	01	
faulty	10	no toggling
reset	high-Z	no toggling
configuration	01	when FCCU_CFG.CM = 0
	= non-faulty	when FCCU_CFG.CM = 1

As long as FCCU is in NORMAL or ALARM state, outputs will show “non-faulty“ signal. Output pins #0, #1 will toggle between “01” and “10” with a given frequency. By default the frequency is the IRCOSC clock frequency divided by 18*1024.

In the FAULT state, the output pin FCCU_F[0] is set as low.

In Time Switching mode the second output (FCCU_F[1]) is the inverted signal of first output (FCCU_F[0]). Values 00 on the outputs indicate a fault in the error out protocol itself. This state must be considered as critical fault, because no reliable error out indication is available any more.

In the RESET phase the output pins are set as “high impedance”.

Note: Figure 319 is formatted to display the behavior in all four phases (reset, normal, error, and config), not to imply transitions between one phase to another. In particular, transition from error phase to config phase is not possible.

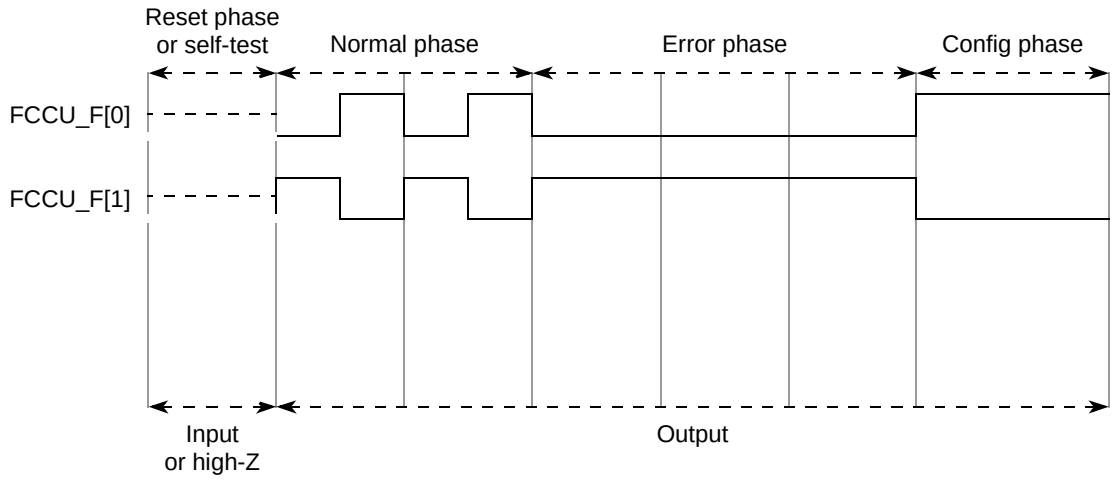


Figure 319. Time-switching protocol

23.7.10.3 Bi-stable protocol

The encoding scheme is given in *Table 283* and the related timing diagram is given in *Figure 320*.

Table 283. Bi-stable encoding

Logical state	Bi-stable encoding (output pins FCCU_F[1:0])	Note
non-faulty	01	no toggling
faulty	10	no toggling
reset	high-Z	no toggling
configuration	10	when FCCU_CFG.CM = 0
	= non-faulty	when FCCU_CFG.CM = 1

In the FAULT state, the faulty logical state is indicated. In NORMAL or ALARM state, “no-faulty” state is indicated. In Bi-stable mode the second output (FCCU_F[1]) is the inverted signal of first output (FCCU_F[0]).

In the RESET phase the output pins are set as “high impedance”.

Note: Figure 320 is formatted to display the behavior in all four phases (reset, normal, error, and config), not to imply transitions between one phase to another. In particular, transition from error phase to config phase is not possible.

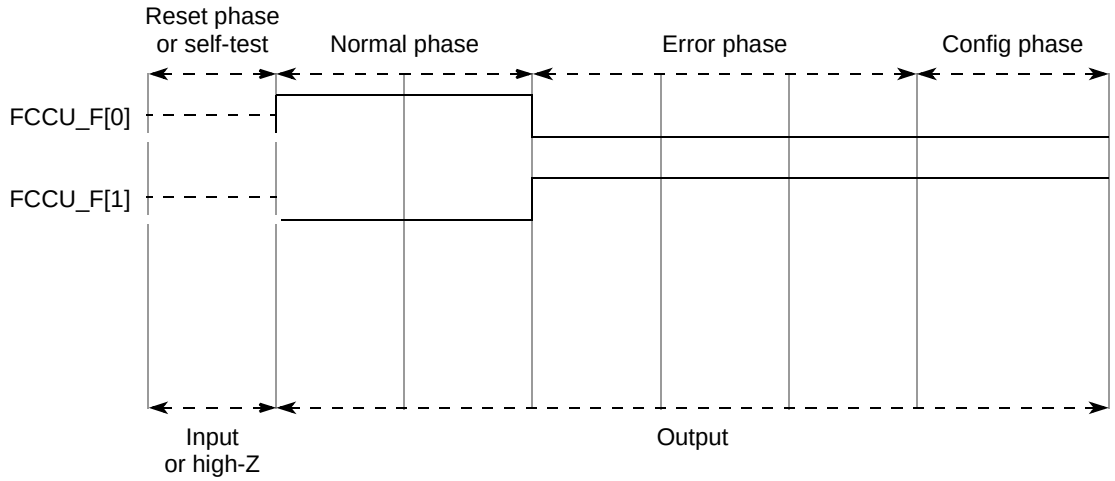


Figure 320. Bi-stable protocol

23.7.11 Fault mapping

Table 284 and *Table 285* show the source of the fault signals and the type of fault input these signals are connected to at the FCCU.

Table 284. FCCU mapping of critical faults

Critical fault	Source	Signal description	Short / long / none default func reset	Set / clear injection	NMI	Safe mode request
CF[0]	RCCUO[0]	Cores out of lock	Long	Yes	Yes	Yes
CF[1]	RCCU1[0]	Cores out of lock	Long	Yes	Yes	Yes
CF[2]	RCCUO[1]	DMA_MUXes out of lock	Long	Yes	Yes	Yes
CF[3]	RCCU1[1]	DMA_MUXes out of lock	Long	Yes	Yes	Yes
CF[4]	RCCUO[2]	PBRIDGEs out of lock	Long	Yes	Yes	Yes
CF[5]	RCCU1[2]	PBRIDGEs out of lock	Long	Yes	Yes	Yes
CF[6]	RCCUO[3]	XBARs out of lock	Long	Yes	Yes	Yes
CF[7]	RCCU1[3]	XBARs out of lock	Long	Yes	Yes	Yes

Table 284. FCCU mapping of critical faults (continued)

Critical fault	Source	Signal description	Short / long / none default func reset	Set / clear injection	NMI	Safe mode request
CF[8]	RCCUO[4]	SRAM arrays out of lock	Long	Yes	Yes	Yes
CF[9]	RCCU1[4]	SRAM arrays out of lock	Long	Yes	Yes	Yes
Note: CF[0] and CF[9] donot run in DP mode						
CF[10]	RCCUO[5]	PFLASHC out of lock (run in DP mode)	Long	Yes	Yes	Yes
CF[11]	RCCU1[5]	PFLASHC out of lock (run in DP mode)	Long	Yes	Yes	Yes
CF[12]	—	—	—	No	No	No
CF[13]	—	—	—	No	No	No
CF[14]	SWT_0	Software watchdog timer	Long	No	Yes	Yes
CF[15]	SWT_1	Software watchdog timer	Long	No	Yes	Yes
CF[16]	ECSM_NCE_0	Flash/SRAM ECC not correctable error	Long	No	Yes	Yes
CF[17]	ECSM_NCE_1	Flash/SRAM ECC not correctable error	Long	No	Yes	Yes
CF[18]	ADC_CF_0	Internal self test (critical fault)	—	Yes (by ADC itself)	Yes	Yes
CF[19]	ADC_CF_1	Internal self test (critical fault)	—	Yes (by ADC itself)	Yes	Yes
CF[20]	STCU	Bist results (critical faults)	—	No	No	No
Note: CF[20] is not supported in MPC5643L.						
CF[21]	LVD_HVD_ 1.2V	LVD/HVD BIST failure result in test mode	—	Yes	Yes	Yes
CF[22]	SSCM_XFER_ERR	SSCM transfer error (during the STCU config loading)	—	No	Yes	Yes
CF[23]	LSM_DPM_ERR0	LSM <-> DPM runtime switch	Long	Yes	Yes	Yes
CF[24]	LSM_DPM_ERR1	LSM <-> DPM runtime switch	Long	Yes	Yes	Yes
CF[25]	—	—	—	No	No	No
CF[26]	—	—	—	No	No	No
CF[27]	STCU	STCU fault condition (run in application mode)	Long	No	Yes	Yes

Table 284. FCCU mapping of critical faults (continued)

Critical fault	Source	Signal description	Short / long / none default func reset	Set / clear injection	NMI	Safe mode request
CF[28]	DFT0	Combination of safety critical signals from Test Control Unit (TCU)	Long	No	Yes	Yes
CF[29]	DFT1	Combination of safety critical signals from Test Control Unit (TCU)	Long	No	Yes	Yes
CF[30]	DFT2	Combination of safety critical signals from Test Control Unit (TCU)	Long	No	Yes	Yes
CF[31]	DFT3	Combination of safety critical signals from Test Control Unit (TCU)	Long	No	Yes	Yes
CF[37]	JTAG/NEXUS	Combination of safety critical signals from JTAG and NEXUS	Long	No	Yes	Yes

Table 285. FCCU mapping of non-critical faults

Non-critical fault	Source	Signal description	Short / long / none default func reset	Set / clear injection	Fault enabled	Time-out enabled
NCF[0]	Core_0 watchdog	p_wrs_core0[0]	Long	No	Yes	Yes
NCF[1]	Core_1 watchdog	p_wrs_core1[0]	Long	No	Yes	Yes
NCF[2]	FM_PLL_0	Loss of lock	Long	No	Yes	Yes
NCF[3]	FM_PLL_1	Loss of lock	Long	No	Yes	Yes
NCF[4]	CMU_0	Loss of XOSC clock	Long	No	Yes	Yes
NCF[5]	CMU_0	Sysclk frequency out of range	Long	No	Yes	Yes
NCF[6]	CMU_1	MOTC_CLK frequency out of range	Long	No	Yes	Yes
NCF[7]	CMU_2	FRPE_CLK frequency out of range	Long	No	Yes	Yes
NCF[8]	ECSM_ECN_0	ECC 1-bit error correction notification	—	No	No	Yes
NCF[9]	ECSM_ECN_1	ECC 1-bit error correction notification	—	No	No	Yes
NCF[10]	ADC_NCF_0	Internal self test (non critical fault)	—	Yes (by ADC itself)	Yes	Yes
NCF[11]	ADC_NCF_1	Internal self test (non critical fault)	—	Yes (by ADC itself)	Yes	Yes

Table 285. FCCU mapping of non-critical faults (continued)

Non-critical fault	Source	Signal description	Short / long / none default func reset	Set / clear injection	Fault enabled	Time-out enabled
NCF[12]	STCU_NCF	Bist results (non critical faults)	—	Yes	Yes	Yes
NCF[13]	LVD_ 1.2V	LVD BIST OK in test mode/ LVD NOK in user mode	—	Yes	Yes	Yes
NCF[14]	HVD_ 1.2V	HVD BIST OK in test mode/ HVD NOK in user mode	—	Yes	Yes	Yes
NCF[15]	LVD VREG	LVD VREG fault detected by self-checking. (refer *Table 610* for further clarifications)	—	Yes	Yes	Yes
NCF[16]	LVD FLASH	LVD FLASH fault detected by self-checking (refer *Table 610* for further clarifications)	—	Yes	Yes	Yes
NCF[17]	LVD IO	LVD IO fault detected by self-checking (refer *Table 610* for further clarifications)	—	Yes	Yes	Yes
NCF[18]	—	—	—	No	No	No
NCF[19]	FLEXR_ECN	ECC 1-bit error correction notification from flexray	—	No	No	Yes
NCF[20]	FLEXR_NCE	ECC not correctable error from flexray (combination of LRAM and DRAM ECC errors)	—	No	Yes	Yes
NCF[21]	MC_ME	Software device reset	—	No	No	Yes
NCF[22]	BP_BALLAST0(1)	Bypass Ballast0	—	No	Yes	Yes
NCF[23]	BP_BALLAST1(1)	Bypass Ballast1	—	No	Yes	Yes
NCF[24]	BP_BALLAST2(1)	Bypass Ballast2	—	No	Yes	Yes
NCF[25]	—	—	—	No	No	No
NCF[26]	—	—	—	No	No	No
NCF[27]	—	—	—	No	No	No
NCF[28]	—	—	—	No	No	No
NCF[29]	—	—	—	No	No	No
NCF[30]	—	—	—	No	No	No
NCF[31]	—	—	—	No	No	No

1. If one of these non-critical faults is triggered, ballast is no more available to supply the device and device enters "non-customer" test mode.

24 Flash Memory

24.1 Flash memory block (C90FL)

24.1.1 C90FL block overview

The primary function of the C90FL flash memory block is to serve as electrically programmable and erasable Non-Volatile Memory (NVM). The NVM can be used for instruction and/or data storage. The block is a non-volatile solid-state silicon memory device consisting of blocks 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 C90FL is addressable by word (32 bits) and page (128 bits).

The C90FL block is arranged as two functional units. The first functional unit is the C90FL Flash Core (FC). The FC is composed of arrayed non-volatile storage elements, sense amplifiers, row selects, column selects and charge pumps. The arrayed storage elements in the FC are subdivided into physically separate units referred to as blocks.

The second functional unit of the C90FL is the Memory Interface (MI). The MI contains the registers and logic which control the operation of the FC. The MI is also the interface to the PFlash Bus Interface Unit (PFLASH_C90FL).

The PFLASH_C90FL interfaces the system bus on this device to the C90FL memory block. The PFlash BIU is described in *Section 24.2, Dual-ported platform flash memory controller (PFLASH2P)*.

The base address for the flash bus and flash registers is 0xC3F8_8000.

There are three address spaces:

- Low address space (256 KB)
- Mid address space (256 KB)
- High address space (512 KB)

For block configurations overview see *Figure 321*.

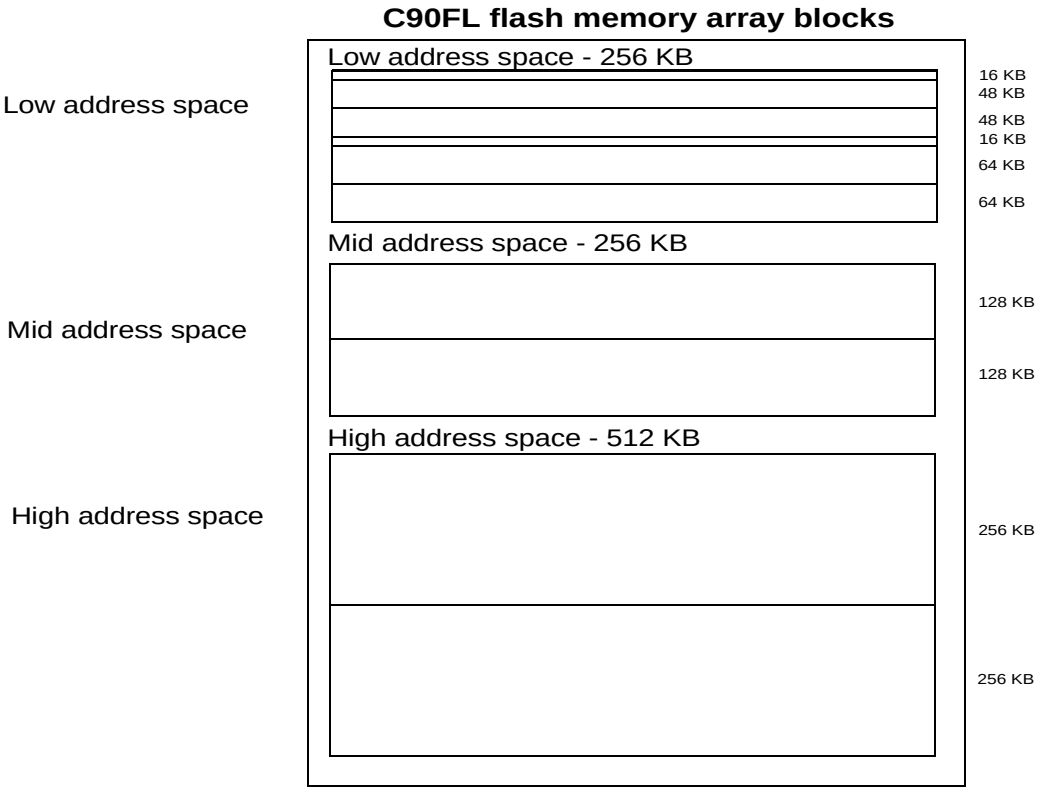


Figure 321. C90FL flash memory array diagram

24.1.2 C90FL block features

- Support for a 64-bit data bus for instruction fetch
- Support for a 32-bit data bus for CPU loads and DMA access. Byte, halfword, word and doubleword reads are supported. Only aligned word and doubleword writes are supported.
- Configurable read buffering and line prefetch support. Four line read buffers (128 bits wide) and a prefetch controller are used to support single-cycle read responses for hits in the buffers.
- Hardware and software configurable read and write access protections on a per-master basis
- Interface to the flash array controller is pipelined with a depth of 1, allowing overlapped accesses to proceed in parallel for interleaved or pipelined flash array designs
- Configurable access timing allowing use in a wide range of system frequencies
- Multiple-mapping support and mapping-based block access timing (0-31 additional cycles) allowing use for emulation of other memory types
- Software programmable block program/erase restriction control for low, mid and high address spaces
- Erase of selected block(s)
- Read page and program page size of 128 bits (4 words)
- ECC with single-bit correction, double-bit detection
- Minimum program size is 2 consecutive 32-bit words, aligned on a 0-modulo-8 byte address, due to ECC
- Embedded hardware program and erase algorithm
- Read while Write with multiple partitions
- Erase suspend, program suspend and erase-suspended program
- Automotive C90FL which meets automotive endurance and reliability requirements
- Shadow information stored in non-volatile shadow block
- Independent program/erase of the shadow block

24.1.3 C90FL modes of operation

24.1.3.1 C90FL User Mode

User mode is the default operating mode of the C90FL module. In this mode, it is possible to read and write, program and erase the C90FL module.

24.1.3.2 Stop Mode

In Stop mode, all DC current sources in the C90FL are disabled.

24.1.4 C90FL block diagram

Figure 322 shows a block diagram of the C90FL flash memory block.

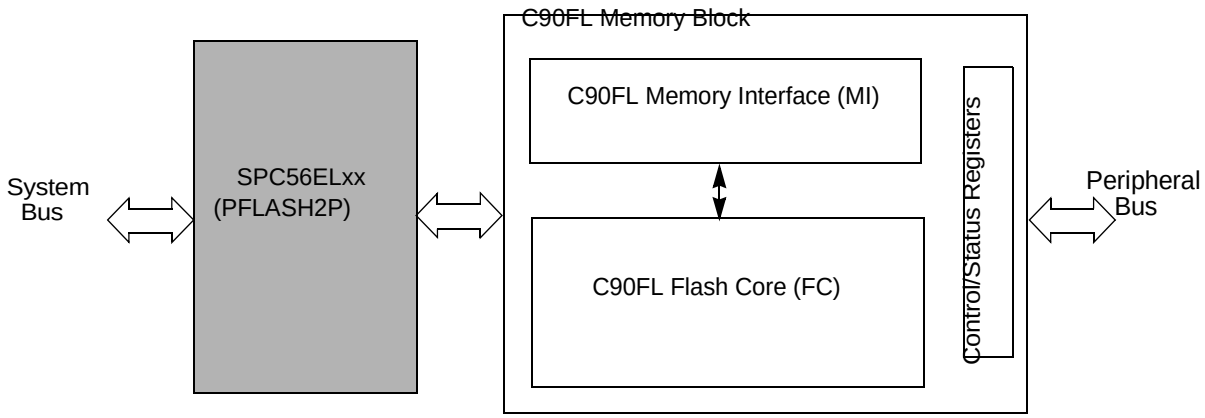


Figure 322. C90FL flash memory system block diagram

24.1.5 C90FL memory map and register definition

Table 286 shows how the array is memory mapped. *Table 287* shows how the registers are mapped.

Caution: Software executing from flash memory must not write to registers that control flash behavior (such as wait state settings or prefetch enable/disable). Doing so can cause data corruption. On this chip, these registers include PFCR0 and PFAPR.

Note: Flash memory configuration registers should be written only with 32-bit write operations to avoid any issues associated with register incoherency caused by bit fields spanning smaller size (8-, 16-bit) boundaries.

Table 286. C90FL flash memory map

FLASH_BASE address offset	Use	Block	Size (KB)	Partition
0x0	Low Address Space	L0	16	1
0x0000_4000		L1	48	
0x0001_0000		L2	48	
0x0001_C000		L3	16	
0x0002_0000		L4	64	2
0x0003_0000		L5	64	
0x0004_0000	Mid Address Space	M0	128	3
0x0006_0000		M1	128	
0x0008_0000	High Address Space	H0	256	4
0x000C_0000		H1	256	
0x0010_0000 – 0x00EF_FFFF	Reserved			

Table 286. C90FL flash memory map (continued)

FLASH_BASE address offset	Use	Block	Size (KB)	Partition
0x00F0_0000	Flash memory shadow block, for general use	S	16	All(1)
0x00F0_3DD8	System censoring passcode			
0x00F0_3DE0	System censoring			
0x00F0_3DE8	LML default			
0x00F0_3DF0	HBL default			
0x00F0_3DF8	SLL default			
0x00F0_3E00	PFAPR			
0x00F0_3E08	Reserved			
0x00F0_3E10	BIU4 default	S	16	All(1)
0x00F0_4000 – 0x00FF_FFFF	Reserved			
0x0100_0000 – 0x1FFF_FFFF	Flash memory emulation mapping	—	496	—

1. For Read while Write operations, shadow block behaves as if it is in all partitions.

Table 287. Register memory map

Address	Use	Location
FLASH_REGS_BASE + 0x0	MCR	on page 593
FLASH_REGS_BASE + 0x4	LML Register (LML)	on page 597
FLASH_REGS_BASE + 0x8	HBL Register (HBL)	on page 599
FLASH_REGS_BASE + 0xC	SLL Register (SLL)	on page 600
FLASH_REGS_BASE + 0x10	LMS Register (LMS)	on page 601
FLASH_REGS_BASE + 0x14	HBS Register (HBS)	on page 603
FLASH_REGS_BASE + 0x18	ADR Register (ADR)	on page 604
FLASH_REGS_BASE + 0x1C	PFlash Configuration Register 0 (PFCR0)	on page 628
FLASH_REGS_BASE + 0x20	Reserved	
FLASH_REGS_BASE + 0x24	PFlash Access Protection Register (PFAPR)	on page 632
FLASH_REGS_BASE + 0x2C	BIU4	on page 605
FLASH_REGS_BASE + 0x28	Reserved	
FLASH_REGS_BASE + 0x30-0x38	Reserved	
FLASH_REGS_BASE + 0x3C	UT0	on page 605
FLASH_REGS_BASE + 0x40	UT1	on page 607
FLASH_REGS_BASE + 0x44	UT2	on page 607
FLASH_REGS_BASE + 0x48	UM0	on page 608
FLASH_REGS_BASE + 0x4C	UM1	on page 608
FLASH_REGS_BASE + 0x50	UM2	on page 608

Table 287. Register memory map

Address	Use	Location
FLASH_REGS_BASE + 0x54	UM3	*on page 608*
FLASH_REGS_BASE + 0x58	UM4	*on page 608*
FLASH_SHADOW_BASE + 0x3DD8	NVPWD0	*on page 611*
FLASH_SHADOW_BASE + 0x3DDC	NVPWD1	*on page 612*
FLASH_SHADOW_BASE + 0x3DE0	NVSCI	*on page 612*

24.1.5.1 Module Configuration Register (MCR)

The MCR register is defined in *Figure 323* and *Table 288*.

Figure 323. Module Configuration Register (MCR)

Offset 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SIZE			0	LAS			0	0	0	MAS
W																
RESET:	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE	SBC	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
W	w1c	w1c	w1c													
RESET:	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Table 288. MCR Field Descriptions

Field	Description
SIZE	Array Space Size. The value of the SIZE field is dependent upon the size of the c90f module. SIZE is read only. 000 Reserved 001 Reserved 010 Reserved 011 1.0 MB (256 KB of LAS, 256 KB of MAS, and 512 KB of HAS) 100 Reserved 101 Reserved 110 Reserved 111 Reserved

Table 288. MCR Field Descriptions (continued)

Field	Description
LAS	Low Address Space. The value of the LAS field corresponds to the configuration of the Low Address Space. LAS is read only. 000 One 256 Kbyte Blocks 001 Two 128 Kbyte Blocks 010 Four 16 Kbyte, four 48 Kbyte Blocks 011 Reserved 100 Eight 16 Kbyte, two 64 Kbyte Blocks 101 Reserved 110 Two 16 Kbyte, two 48 Kbyte, two 64 Kbyte Blocks 111 Reserved
MAS	Mid Address Space. The value of the MAS field corresponds to the configuration of the Mid Address Space. MAS is read only. 0Two 128 Kbyte Blocks 1One 256 Kbyte Blocks (only available if LAS =0)
EER	ECC Event Error. EER provides information on previous reads. If a double bit detection occurred, the EER bit is set to a 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. This bit may not be set by the user. In the event of a single bit detection and correction, this bit is 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) are correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0Reads are occurring normally 1An ECC Error occurred during a previous read
RWE	Read While Write Event Error. RWE provides information on previous RWW reads. If a Read While Write error occurs, this bit is set to 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. This bit may not be written to a 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) are correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0Reads are occurring normally 1A Read While Write Error occurred during a previous read
SBC	Single Bit Correction. SBC provides information on previous reads provided the UT0[SPCE] is set. If a single bit correction occurred, the SBC bit is set to a 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. If SBC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of SBC) did not require a correction. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0Reads are occurring without corrections 1A Single Bit Correction occurred during a previous read
PEAS	Program/Erase Access Space. PEAS is used to indicate which space is valid for program and erase operations, either main array space or shadow space. PEAS = 0 indicates that the main address space is active for all FC program and erase operations. PEAS = 1 indicates the shadow address space is active for program/erase. The value in PEAS is captured and held when the shadow block is enabled with the first interlock write done for program or erase operations. The value of PEAS is retained between sampling events (i.e. subsequent first interlock writes). The value in PEAS may be changed during erase-suspended program, and reverts back to its' original state once the erase-suspended program is completed. PEAS is read only. 0Shadow address space is disabled for program/erase and main address space enabled 1Shadow address space is enabled for program/erase and main address space disabled

Table 288. MCR Field Descriptions (continued)

Field	Description
DONE	State Machine Status. DONE indicates if the flash module is performing a high voltage operation. DONE is set to a 1 on termination of the flash module reset. DONE is read only. DONE is cleared within Tdone (Appendix A) of a 0 to 1 transition of EHV which initiates a high voltage operation. DONE is cleared within Tres (Appendix A) of resuming a suspended operation. DONE is set to a 1 at the end of program and erase high voltage sequences. DONE is set to a 1 within Tdones (Appendix A) of a 1 to 0 transition of EHV which aborts a high voltage operation. 0Flash is executing a high voltage operation 1Flash is not executing a high voltage operation
PEG	Program/Erase Good. The PEG bit indicates the completion status of the last flash program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations. Aborting a program/erase high voltage operation causes PEG to be cleared, indicating the sequence failed. PEG is set to a 1 when the module is reset. PEG is read only. 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 PSUS or ESUS being set to logic 1. If PGM and ERS are both 1 when DONE makes a qualifying 0 to 1 transition the value of PEG indicates the completion status of the PGM sequence. This happens in an erase-suspended program operation. 0Program or erase operation failed 1Program or erase operation successful *Note: If program or erases are attempted on blocks that are locked, the response from c90fl is PEG = 1, indicating that the operation was successful, and the contents of the block are properly protected from the program or erase operation.*
PGM	Program. PGM is used to setup c90fl 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 one of the following conditions: – User mode read (ERS is low and UTE is low) – Erase suspend (ERS and ESUS are 1) with EHV low PGM can be cleared by the user only when PSUS and EHV are low and DONE is high. PGM is cleared on reset. 0Flash is not executing a program sequence 1Flash is executing a program sequence *Note: In an erase-suspended program, programming Flash locations in blocks which were being operated on in the erase may corrupt FC data. This should be avoided due to reliability implications.*
PSUS	Program Suspend. PSUS is used to indicate the flash module is in program suspend or in the process of entering a suspend state. The module is in program suspend when PSUS = 1 and DONE = 1. PSUS can be set high only when PGM and EHV are high. A 0 to 1 transition of PSUS starts the sequence which sets DONE and places the flash module in program suspend. The module enters suspend within Tpsus (Appendix A) of this transition. PSUS can be cleared only when DONE and EHV are high. A 1 to 0 transition of PSUS with EHV = 1 starts the sequence which clears DONE and returns the flash module to program. The module cannot exit program suspend and clear DONE while EHV is low. PSUS is cleared on reset. 0Program sequence is not suspended 1Program sequence is suspended

Table 288. MCR Field Descriptions (continued)

Field	Description
ERS	Erase. ERS is used to setup c90fl 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 only be set only in user mode read (PGM is low and UTE 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. 0Flash is not executing an erase sequence 1Flash is executing an erase sequence
ESUS	Erase Suspend. ESUS is used to indicate that the flash module is in erase suspend or in the process of entering a suspend state. The 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 in erase suspend. The flash module enters suspend within Tesus (Appendix A) 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 module cannot exit erase suspend and clear DONE while EHV is low. ESUS is cleared on reset. 0Erase sequence is not suspended 1Erase sequence is suspended
EHV	Enable High Voltage. The EHV bit enables the flash module for a high voltage program/erase operation. EHV is cleared on reset. EHV must be set after an interlock write to start a program/erase sequence. EHV may be set, initiating a program/erase, after an interlock under one of the following conditions: – Erase (ERS = 1, ESUS = 0). – Program (ERS = 0, ESUS = 0, PGM = 1, PSUS = 0). – Erase-suspended program (ERS = 1, ESUS = 1, PGM = 1, PSUS = 0). If a program operation is to be initiated while an erase is suspended the user must clear EHV while in erase suspend before setting PGM. In normal operation, a 1 to 0 transition of EHV with DONE high, PSUS 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 suspend bit for the current program/erase sequence low. An abort causes the value of PEG to be cleared, indicating a failed program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted. EHV may be written during suspend. EHV must be high for the flash module to exit suspend. EHV may not be written after a suspend bit is set high and before DONE transitions high. EHV may not be set low after the current suspend bit is set low and before DONE transitions low. 0Flash is not enabled to perform a high voltage operation 1Flash is enabled to perform a high voltage operation *Note: Aborting a high voltage operation leaves FC addresses in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.*

24.1.5.1.1 MCR Simultaneous Register Writes

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 section. The write locks detailed in the previous section do not consider the effects of trying to write two or more bits simultaneously. The effects of writing bits simultaneously which put the module in an illegal state are detailed here.

The flash module does not allow the user to write bits simultaneously which put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in *Table 289*.

Table 289. MCR bit set/clear priority levels

Priority level	MCR bit(s)
1	ERS
2	PGM
3	EHV
4	ESUS, PSUS

If the user attempts to write two or more MCR bits simultaneously then only the bit with the lowest priority level is written. Setting two bits with the same priority level is prevented by existing write locks or do not put the flash in an illegal state.

For example, setting ERS and PGM simultaneously results in only ERS being set. Attempting to clear EHV while setting PSUS results in EHV being cleared, while PSUS is unaffected.

24.1.5.2 Low/Mid Address Space Block Locking Register (LML)

The Low/Mid Address Block Locking Register (LML) provides a means to protect blocks from being modified. These bits, along with bits in the Secondary LLOCK (SLL), determine if the block is locked from program or erase. An “OR” of LML and SLL determine the final lock status.

Note: A reset value of 1 in Figure 324 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.*

Figure 324. LML Register

Offset 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	0	0	0	0	0	0	0	0	0	0	SLOCK	0	0	MLOCK	
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	1*	0	0	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	LLOCK									
W																
RESET:	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

= Unimplemented or Reserved

LML register functions are shown in *Table 290*.

Table 290. LML Field Descriptions

Field	Description
0 LME	Low/Mid Address Lock Enable. This bit is used to enable the Lock registers (SLOCK, MLOCK and LLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set this bit is to write a password, and if the password matches, the LME bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME, the password 0xA1A1_1111 must be written to the LML register. 0Low/Mid Address Locks are disabled, and can not be modified 1Low/Mid Address Locks are enabled to be written
1-10	Reserved, reset to 0
11 SLOCK	Shadow Lock. This bit is used to lock the shadow block from programs and erases. A value of 1 in the SLOCK register signifies that the shadow block is locked for program and erase. A value of 0 in the SLOCK register signifies that the shadow block is available to receive program and erase pulses. The SLOCK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, SLOCK register is not writable if a high voltage operation is suspended. SLOCK is also not writeable during UTest operations, when AIE is high. Upon reset, information from the shadow block is loaded into the SLOCK register. The SLOCK bit may be written as a register. Reset causes the bits to go back to their shadow block value. The default value of the SLOCK bits (assuming erased shadow location) is locked. SLOCK is not writable unless LME is high.
12-13	Reserved, reset to 0
14-15 MLOCK[1:0]	Mid Address Space Block Lock. A value of 1 in a bit of the lock register signifies that the corresponding block is locked for program and erase. A value of 0 in the lock register signifies that the corresponding block is available to receive program and erase pulses. The block numbering for Mid Address Space starts with MLOCK[0] and continues until all blocks are accounted. The lock register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the lock register is not writable if a high voltage operation is suspended. MLOCK is also not writeable during UTest operations, when AIE is high. Upon reset, information from the shadow block is loaded into the block registers. The LOCK bits may be written as a register. Reset causes the bits to go back to their shadow block value. The default value of the LOCK bits (assuming erased shadow location) is locked. In the event that blocks are not present (due to configuration or total memory size), the LOCK bits default to be locked, and are not writable. The reset value is always 1 (independent of the shadow block), and register writes have no effect. MLOCK is not writable unless LME is high.

Table 290. LML Field Descriptions (continued)

Field	Description
16-21	Reserved, reset to 0
22-31 LLOCK[9:0]	Low Address Space Block Lock. A value of 1 in a bit of the lock register signifies that the corresponding block is locked for program and erase. A value of 0 in the lock register signifies that the corresponding block is available to receive program and erase pulses. The block numbering for Low Address Space starts with LLOCK[0] and continues until all blocks are accounted. For more details on LLOCK, please see MLOCK bit description. LLOCK is not writable unless LME is high.

24.1.5.3 High Address Space Block Locking Register (HBL)

The High Address Space Block Locking Register (HBL) provides a means to protect blocks from being modified.

Note: A reset value of 1 in Figure 325 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.*

Figure 325. HBL Register

Offset 0x0008 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	HLOCK					
W																
RESET:	0	0	0	0	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*

= Unimplemented or Reserved

HBL register functions are shown in Table 291.

Table 291. HBL Field Descriptions

Field	Description
0 HBE	High Address Lock Enable This bit is used to enable the Lock registers (HLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set this bit is to provide a password, and if the password matches, the HBE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE, the password B2B2_2222h must be written to the HBL register. 0High Address Locks are disabled, and can not be modified 1High Address Locks are enabled to be written
1-25	Reserved, reset to 0
26-31 HLOCK[5:0]	High Address Space Block Lock. HLOCK has the same characteristics as LLOCK. Please see this description for more information. The block numbering for High Address Space starts with HLOCK[0] and continues until all blocks are accounted. HLOCK is not writable unless HBE is high.

24.1.5.4 Secondary Low/Mid Address Space Block Locking Register (SLL)

The Secondary Low/Mid Address Block Locking Register (SLL) provides an alternative means to protect blocks from being modified. This has the effect of creating a “tiered” locking scheme to enable different flash users to provide different default locking on blocks. These bits, along with bits in the LLOCK (LML), determine if the block is locked from program or erase. An “OR” of LML and SLL determine the final lock status.

Note: A reset value of 1 in Figure 326 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.*

Figure 326. SLL Register

Offset 0x000C — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	0	0	0	0	0	0	0	0	0	0	SSLOCK	0	0	SMLOCK	
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	1*	0	0	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	SLLOCK									
W																
RESET:	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

= Unimplemented or Reserved

SLL register functions are shown in Table 292.

Table 292. SLL Field Descriptions

Field	Description
0 SLE	Secondary Low/Mid Address Lock Enable. This bit is used to enable the Lock registers (SSLOCK, SMLOCK, and SLLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set this bit is to provide a password, and if the password matches, the SLE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE, the password 0xC3C3_3333 must be written to the SLL register. 0Secondary Low/Mid Address Locks are disabled, and can not be modified 1Secondary Low/Mid Address Locks are enabled to be written
1-10	Reserved, reset to 0
11 SSLOCK	Secondary Shadow Lock. This bit is an alternative method that may be used to lock the shadow block from programs and erases. SSLOCK has the same description as SLOCK. SSLOCK is not writable unless SLE is high.
12-13	Reserved, reset to 0
14-15 SMLOCK[1:0]	Secondary Mid Address Block Lock. This bit is an alternative method that may be used to lock the Mid Address Space blocks from programs and erases. SMLOCK has the same description as MLOCK. SMLOCK is not writable unless SLE is high.
16-21	Reserved, reset to 0
22-31 SLLOCK[9:0]	Secondary Low Address Block Lock. This bit is an alternative method that may be used to lock the Low Address Space blocks from programs and erases. SLLOCK has the same description as LLOCK. SLLOCK is not writable unless SLE is high.

24.1.5.5 Low/Mid Address Space Block Select Register (LMS)

The Low/Mid Address Space Block Select Register (LMS) provides a means to select blocks to be operated on during erase.

Figure 327. LMS Register

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	MSEL	
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	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	LSEL					
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

LMS register functions are shown in *Table 293*.

Table 293. LMS Field Descriptions

Field	Description
0-13	Reserved, reset to 0.
14-15 MSEL[1:0]	Mid Address Space Block Select. A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected. The reset value for the select registers is 0, or unselected. 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 until MCR[DONE] is set at the completion of the requested operation, or if a high voltage operation is suspended. MSEL is also not writeable during UTest operations, when AIE is high. In the event that blocks are not present (due to configuration or total memory size), the corresponding select bits default to unselected, and are not writable. The reset value is always 0, and register writes have no effect. M0 (128 Kb) : LMS.MSEL0 = LMS.b15 M1 (128 Kb) : LMS.MSEL1 = LMS.b14
16-25	Reserved, reset to 0.
26-31 LSEL[5:0]	Low Address Space Block Select. A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected. The reset value for the select registers is 0, or unselected. 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 until MCR[DONE] is set at the completion of the requested operation, or if a high voltage operation is suspended. LSEL is also not writeable during UTest operations, when AIE is high. In the event that blocks are not present (due to configuration or total memory size), the corresponding select bits default to unselected, and are not writable. The reset value is always 0, and register writes have no effect. L0 (16 Kb) : LMS.LSEL0 = LMS.b31 L1 (48 Kb) : LMS.LSEL1 = LMS.b30 L2 (48 Kb) : LMS.LSEL2 = LMS.b29 L3 (16 Kb) : LMS.LSEL3 = LMS.b28 L4 (64 Kb) : LMS.LSEL4 = LMS.b27 L5 (64 Kb) : LMS.LSEL5 = LMS.b26 Bits [5:0] correspond to LAS block L5, L4, L3, L2, L1 and L0, respectively.

24.1.5.6 High Address Space Block Select Register (HBS)

The High Address Space Block Select Register (HBS) provides a means to select blocks to be operated on during erase.

Figure 328. HBS Register

Offset 0x0014 Access: User read/write

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

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

= Unimplemented or Reserved

HBS register functions are shown in *Table 294*.

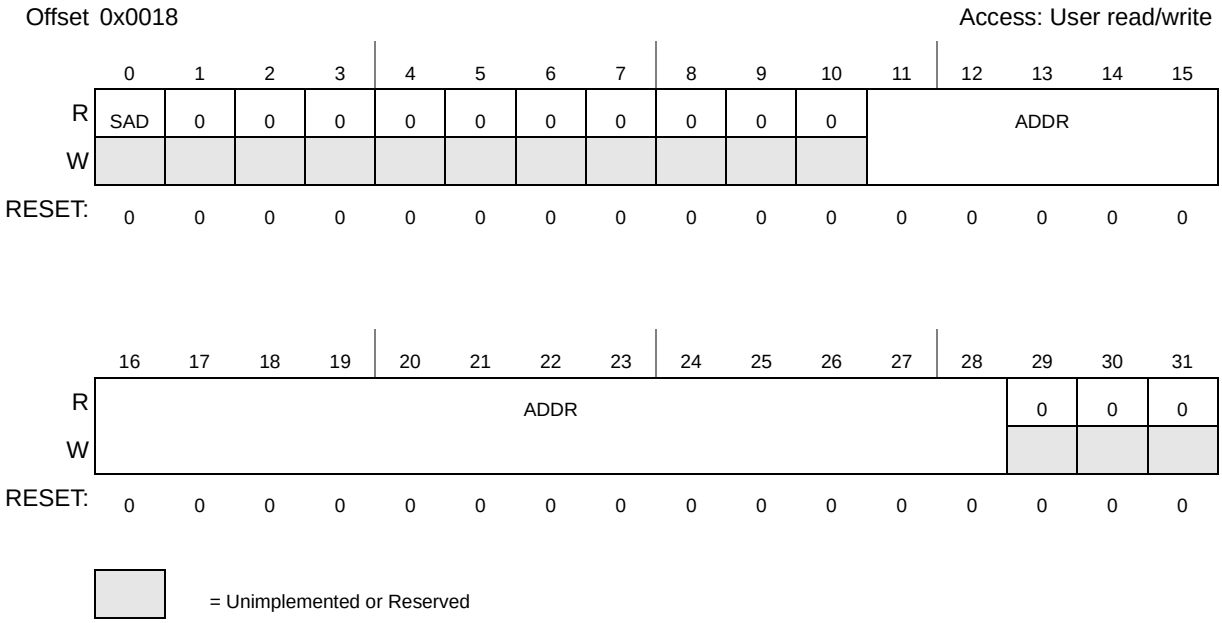
Table 294. HBS Field Descriptions

Field	Description
0-29	Reserved, reset to 0.
30-31 HSEL[1:0]	High Address Space Block Select. High Address Block Select has the same characteristics as LSEL. H0 (256 Kb) : HBS.HSEL0 = HBS.b31 H1 (256 Kb) : HBS.HSEL1 = HBS.b30

24.1.5.7 Address Register (ADR)

The Address register (ADR) provides the first failing address in the event module failures (ECC or PGM/Erase state machine)

Figure 329. ADR Register



ADR register functions are shown in *Table 295*.

Table 295. ADR Field Descriptions

Field	Description
0 SAD	Shadow Address. The SAD bit qualifies the address captured during an ECC Event Error, Single Bit Correction, or State Machine operation. The SAD register is not writable. 0Address Captured is from Main Array Space. 1Address Captured is from Shadow Array Space.
1-10	Reserved, reset to 0.
11-28 ADDR[17:0]	Address. The ADR register provides the first failing address in the event of ECC event error (MCR[EER] set), single bit correction (MCR[SBC] set), as well as providing the address of a failure that may have occurred in a state machine operation (MCR[PEG] cleared). ECC event errors take priority over single bit corrections, which take priority over state machine errors. This is especially valuable in the event of a RWW operation, where the read senses an ECC error or single bit correction, and the state machine fails simultaneously. This address is always a Double Word address that selects 64 bits. The ADR register is writable, and can be used in the UTEST ECC Logic Check. If the ECC logic check is enabled (UT0[EIE] = 1) then the ADR register will not update for ECC event error, single bit correction or state machine errors. If MCR[EER] or MCR[SBC] are set, the ADR register is locked from writing. MCR[PEG] does not affect the writability of the ADR register.
29-31	Reserved, reset to 0.

24.1.5.8 Bus Interface Unit 4 Register (BIU4)

The BIU4 register implements the user option bits described in *Section 24.1.7, User option bits*.

24.1.5.9 User Test 0 Register

The User Test 0 Register (UT0) provides a means to control UTest. The UTest mode gives the users of the flash module the ability to perform test features on the flash. This register is only writable when the flash is put into UTest mode by writing a passcode.

The following field and bit descriptions fully define the UT0 register.

Figure 330. UT0 Register

Offset 0x003C — Access: User read/write

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

	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	EA	0	MRE	MRV	EIE	AIS	AIE	AID
W																
Reset	0	0	0	0	0	0	0	0	*(1)	0	0	0	0	0	0	1

1. The reset value of this bit is configuration-specific. It is 0 for cut1, but for cut2 and cut3 it is dependent upon the ECC algorithm used.

UT0 register functions, as shown in *Table 296*.

Table 296. UT0 Field Descriptions

Field	Description
31 UTE	U-Test Enable. This status bit gives indication when U-Test is enabled. All bits in UT0, UT1, UT2, UM0, UM1, UM2, UM3, and UM4 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. The UTE password will only be accepted if MCR[PGM] = 0 and MCR [ERS] = 0 (program and erase are not being requested). UTE can only be cleared if UT0[AID] = 1, UT0[AIE] and UT0[EIE] = 0. While clearing UTE, writes to set AIE or set EIE will be ignored. For UTE, the password 0xF9F9_9999 must be written to the UT0 register.
31 SBCE	Single Bit Correction Enable. SBC enables Single Bit Correction results to be observed in MCR[SBC]. Also is used as an enable for interrupt signals created by the c90fl module. ECC corrections that occur when SBCE is cleared will not be logged. 0Single Bit Corrections observation is disabled. 1Single Bit Correction observation is enabled.
29–24	Reserved, reset to 0.
23-16 DSI[7:0]	Data Syndrome Input. These bits enable checks of ECC logic by allowing check bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DSI[7:0] correspond to the 8 ECC check bits on a double word.
15-6	Reserved, reset to 0.

Table 296. UT0 Field Descriptions (continued)

Field	Description
7 EA	Cut1: Reserved, reset to 0. Cut2/cut3: ECC Algorithm. EA is a status bit that provides information about the ECC algorithm used within the Flash. Either a modified Hamming code is used, or a modified Hsiao code is used. 0Default ECC Algorithm, modified Hamming algorithm. 1Optional/Alternative ECC Algorithm, modified Hsiao algorithm.
5 MRE	Margin Read Enable. MRE combined with MRV enables Factory Margin Reads to be done. Margin reads are only active during Array Integrity Checks. Normal user reads are not affected by MRE. MRE is not writable if AID is low. 0Margin reads are not enabled. 1Margin reads are enabled during Array Integrity Checks.
4 MRV	Margin Read Value. 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). In order for this value to be valid, MRE must also be set. MRV is not writable if AID is low. 0Zero's margin reads are requested. 1One's margin reads are requested.
3 EIE	ECC Data Input Enable. EIE enables the input registers (DSI and DAI) to be the source of data for the array. This is useful in the ECC logic check. If this bit is set, data read thru a BIU read request will be from the DSI and DAI registers when an address match is achieved to the ADR register. EIE is not simultaneously writable to a 1 as UTI is being cleared to a 0. 0Data read is from the flash array. 1 Data read is from the DSI and DAI registers.
2 AIS	Array Integrity Sequence. 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. If MRE is set, AIS has no effect. 0Array integrity sequence is proprietary sequence. 1 Array integrity sequence is sequential.
1 AIE	Array Integrity Enable. AIE set to one starts the array integrity check done on all selected and unlocked blocks. The address sequence selected is determined by AIS, and the MISR (UM0 through UM4) can be checked after the operation is complete, to determine if a correct signature is obtained. Once an Array Integrity operation is requested (AIE=1), it may be terminated by clearing AIE if the operation has finished (AID = 1) or aborted by clearing AIE if the operation is ongoing (AID = 0). AIE is not simultaneously writable to a 1 as UTI is being cleared to a 0. 0 Array integrity checks are not enabled. 1Array integrity checks are enabled.
0 AID	Array Integrity Done. AID is cleared upon an Array integrity check being enabled (to signify the operation is ongoing). Once completed, AID is set to indicate that the array integrity check is complete. At this time the MISR (UMR registers) can be checked. AID can not be written, and is status only. 0 Array integrity check is ongoing. 1Array integrity check is done.

24.1.5.10 User Test 1 Register

The User Test 1 Register (UT1) provides added controlability to UTest.

The following field and bit descriptions fully define the UT1 register.

Figure 331. UT1 Register

Offset 0x0040 Access: User read/write

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

UT1 register functions, as shown in *Table 297*.

Table 297. UT1 Field Descriptions

Field	Description
31-0 DAI[31:0]	Data Array Input. These bits enable checks of ECC logic by allowing data bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DAI[31:0] correspond to the 32 Array bits representing Word 0 of the double word selected in the ADR register.

24.1.5.11 User Test 2 Register

The following field and bit descriptions fully define the UT2 register.

Figure 332. UT2 Register

Offset 0x0044 Access: User read/write

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

UT2 register functions, as shown in *Table 298*.

Table 298. UT2 Field Descriptions

Field	Description
31-0 DAI[63:32]	Data Array Input. These bits enable checks of ECC logic by allowing data bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DAI[63:32] correspond to the 32 Array bits representing Word 1of the double word selected in the ADR register.

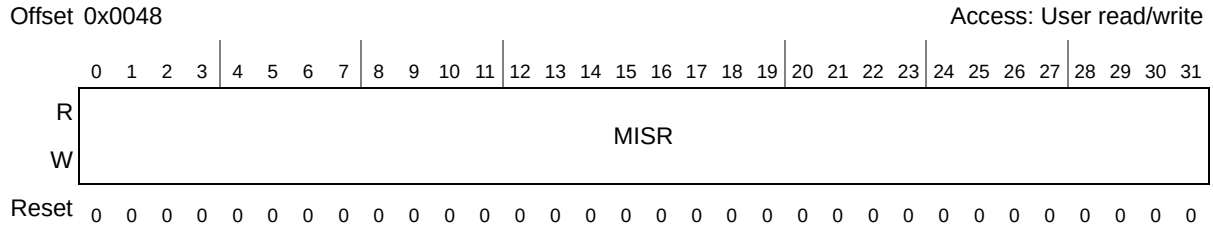
24.1.5.12 User Multiple Input Signature Registers

The Multiple Input Signature Registers (UM0, UM1, UM2, UM3 and UM4) provide a means to evaluate array integrity.

24.1.5.12.1 UM0 Register

The following field and bit descriptions fully define the UM0 register.

Figure 333. UM0 Register



MISR register functions are shown in *Table 299*.

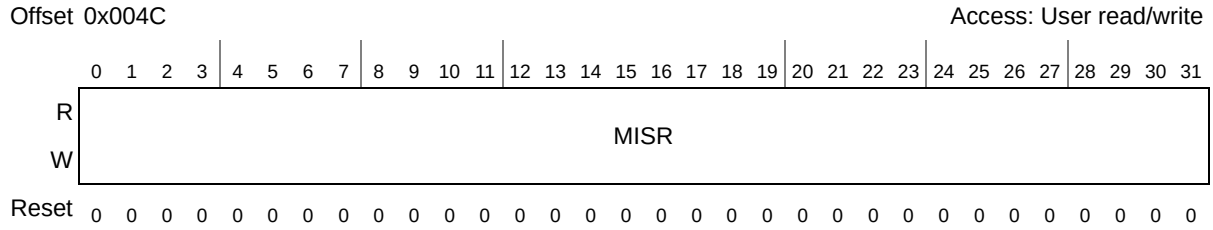
Table 299. UM0 Field Descriptions

Field	Description
0-31 MISR[31:0]	The MISR Registers accumulate a signature from an array integrity event. The MISR captures all data fields, as well as ECC fields, and the read transfer error signal. The MISR can be seeded to any value by writing the MISR registers. The MISR register provides a means to calculate a MISR during Array Integrity operations. The MISR can be represented by the following polynomial: $x^{145}+ x^6 + x^5+ x^1 + 1$ The MISR is calculated by taking the previous MISR value and then “exclusive ORing” the new data. In addition the most significant bit (in this case it is MISR[144]), is then “exclusive ORed” into input of MISR[6], MISR[5], MISR[1], and MISR[0]. The result of the “exclusive OR” is shifted left on each read. The MISR register is used in Array Integrity operations. If during address sequencing, reads extend into an invalid address location (i.e. greater than the maximum address for a given array size) or locked/unselected blocks, reads are still executed to the array but the results from the array read are not deterministic. In this instance, the MISR registers are not recalculated, and the previous value is retained. After running the user-test-mode margin read (also referenced as factory margin read) sequence on the C90fl flash module, the MISR registers cannot be written such that the following user-test-mode margin read sequence cannot seed the MISRs as desired. This will cause the generated MISRs to be unexpected for the following user margin read sequences, in case customers want to run the user margin read more than once. To be able to write the MISR registers: 1) Assert reset after each user margin read sequence so that MISRs can be written again. 2) Do a dummy program to a locked block after user margin read.

24.1.5.12.2 UM1 Register

The following field and bit descriptions fully define the UM1 register (*Figure 334*).

Figure 334. UM1 Register



MISR register functions are shown in *Table 300*.

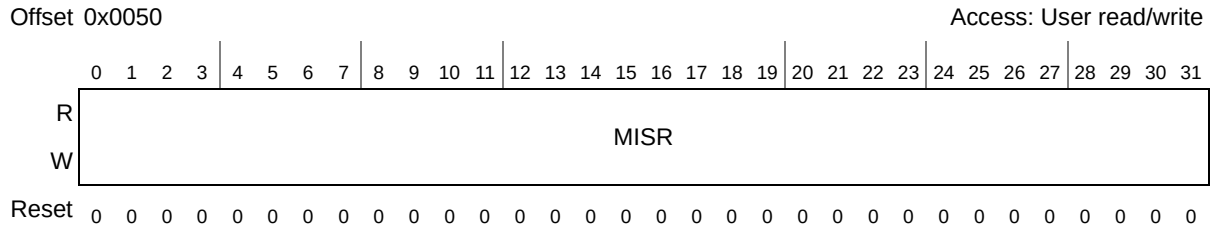
Table 300. UM1 Field Descriptions

Field	Description
0-31 MISR[63:32]	See the description of the MISR field in *Table 299*.

24.1.5.12.3 UM2 Register

The following field and bit descriptions fully define the UM2 register.

Figure 335. UM2 Register



MISR register functions are shown in *Table 301*.

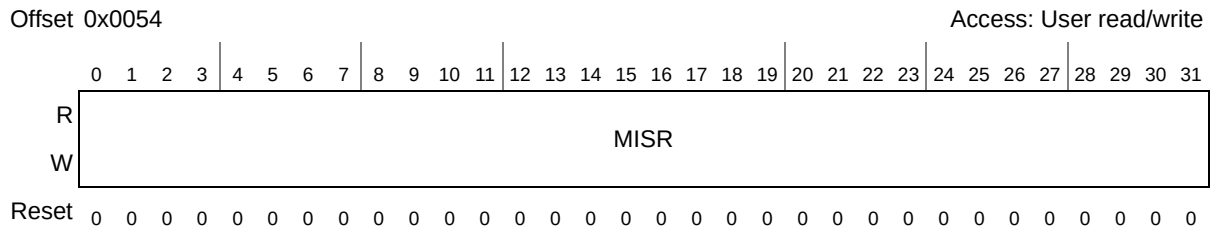
Table 301. UM2 Field Descriptions

Field	Description
0-31 MISR[95:64]	See the description of the MISR field in *Table 299*.

24.1.5.12.4 UM3 Register

The following field and bit descriptions fully define the UM3 register.

Figure 336. UM3 Register



MISR register functions are shown in *Table 302*.

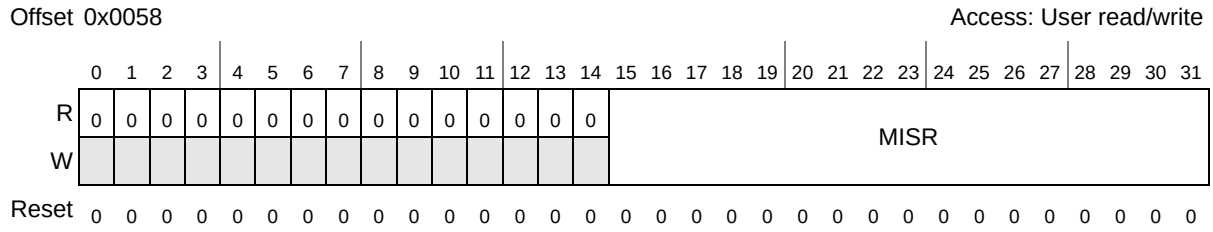
Table 302. UM3 Field Descriptions

Field	Description
0-31 MISR[127:96]	See the description of the MISR in *Table 299*.

24.1.5.12.5 UM4 Register

The following field and bit descriptions fully define the UM4 register.

Figure 337. UM4 register



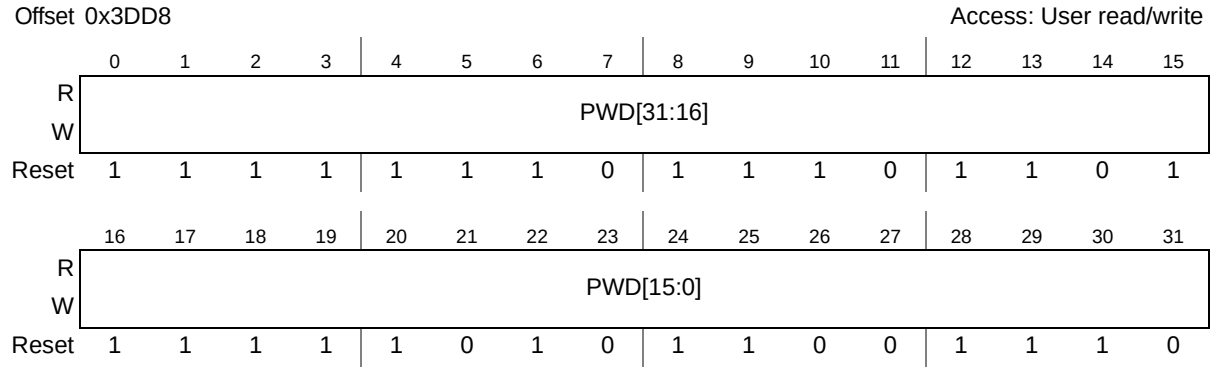
MISR register functions are shown in *Table 303*.

Table 303. UM4 field descriptions

Field	Description
MISR[144:128]	See the description of the MISR in *Table 299*.

24.1.5.13 Nonvolatile Private Censorship Password 0 register (NVPWD0)

Figure 338. Nonvolatile Private Censorship Password 0 register (NVPWD0)



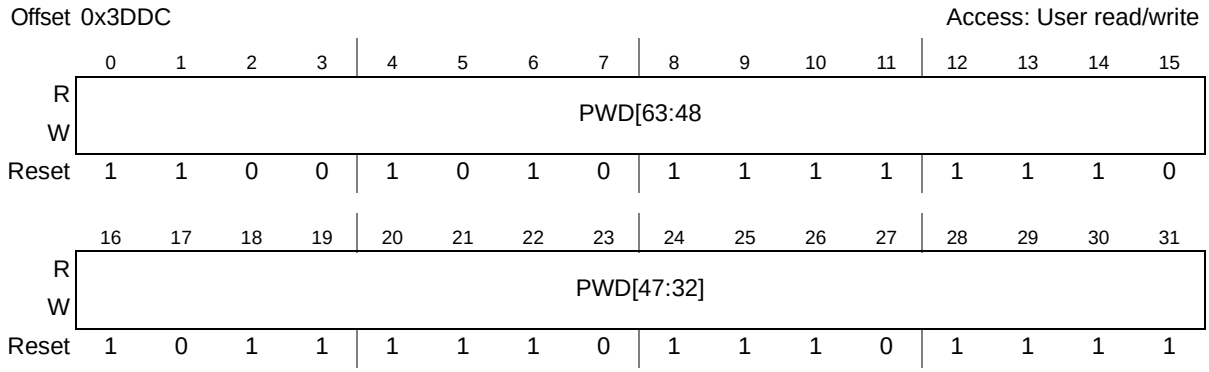
The Nonvolatile Private Censorship Password 0 register (NVPWD0) contains the 32 LSB of the password used to validate the censorship information contained in the NVSCI register.

Table 304. NVPWD0 field descriptions

Field	Description
PWD	These bits represent the private censorship password.

24.1.5.14 Nonvolatile Private Censorship Password 1 register (NVPWD1)

Figure 339. Nonvolatile Private Censorship Password 1 register (NVPWD1)



The Nonvolatile Private Censorship Password 1 register (NVPWD1) contains the 32 MSB of the password used to validate the censorship information contained in the NVSCI register.

Table 305. NVPWD1 field descriptions

Field	Description
PWD	These bits represent the private censorship password.

24.1.5.15 Nonvolatile System Censoring Information register (NVSCI)

Figure 340. Nonvolatile System Censoring Information register (NVSCI) for cut1

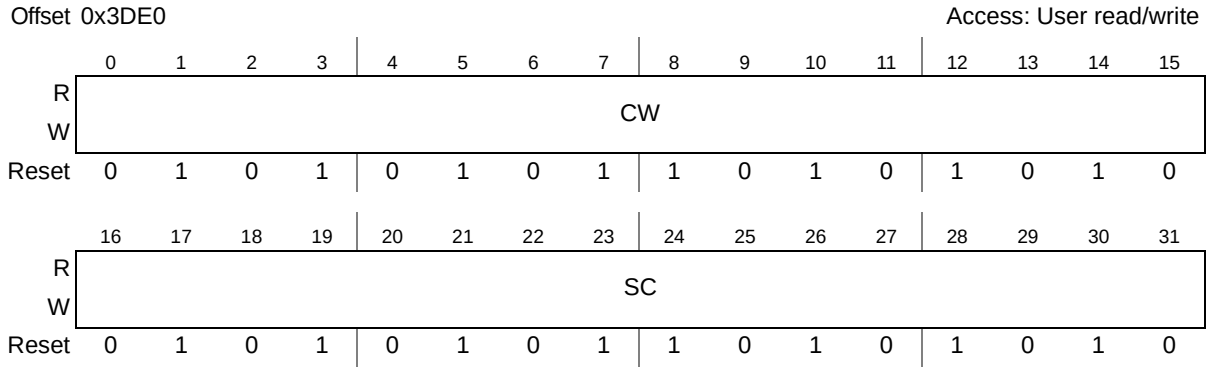
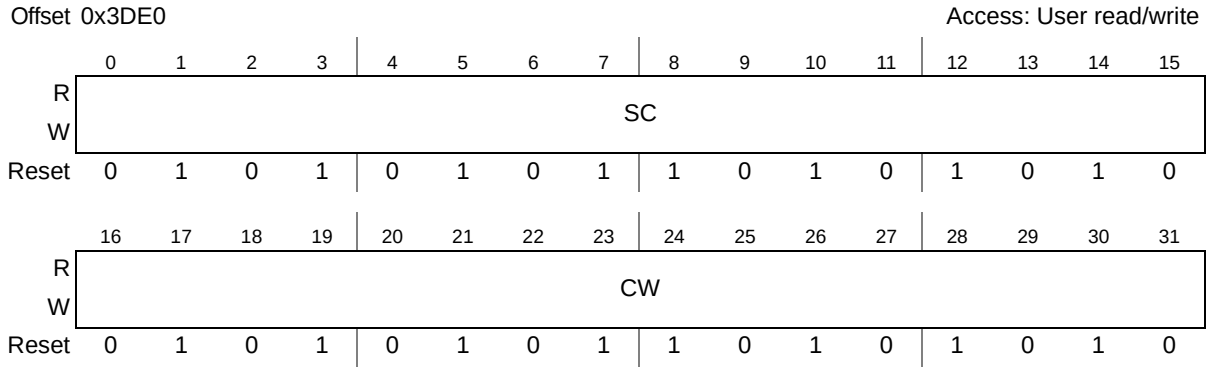


Figure 341. Nonvolatile System Censoring Information register (NVSCI) for cut2/3



The Nonvolatile System Censoring Information register (NVSCI) stores the censorship control word of the device. It is read during the reset phase of the flash memory module and the protection mechanisms are activated consequently. The devices are delivered uncensored to the user.

Table 306. NVSCI field descriptions

Field	Description
SC	Serial Censorship control word. These bits represent the Serial Censorship Control Word (SCCW). If SC[15:0] = 0x55AA, the Public Access is disabled. If SC[15:0] ≠ 0x55AA, the Public Access is enabled.
CW	Censorship control Word. These bits represent the Censorship Control Word (CCW). If CW = 0x55AA, the Censored Mode is disabled. If CW ≠ 0x55AA, the Censored Mode is enabled.

24.1.6 C90FL flash memory functional description (user mode)

In user mode the C90FL module can be read and written (register writes and interlock writes), programmed or erased. The following subsections define all actions that can be performed in user mode.

24.1.6.1 C90FL Read and Write

The default state of the C90FL module is read. The main and shadow address space can be read only in the read state. The Module Configuration Register (MCR) is always available for read. The C90FL module enters the read state on reset. The C90FL module is in the read state under four sets of conditions:

- The read state is active when PGM=1 or ERS=1 in the MCR (Read While Write)

Note: Reads done to the partition(s) being operated on (either erased or programmed) will result in an error and the RWE bit in the MCR will be set.

- The read state is active when PGM=1 and PSUS=1 in the MCR. (Program suspend)
- The read state is active when ERS=1 and ESUS=1 and PGM=0 in the MCR. (Erase suspend)

Note: FC reads are done through the BIU. In many cases the BIU will do "page buffering" to allow sequential reads to be done with higher performance. This can create a Data Coherency issue that must be handled with software. Data Coherency can be an issue after a program or erase operation, as well as shadow row operations.

In C90FL user mode, registers can be written. Array can be written to do interlock writes.

Reads attempted to invalid locations will result in indeterminate data. Invalid locations occur when addressing is done to blocks that do not exist in non 2^n array sizes.

Interlock writes attempted to invalid locations (due to blocks that do not exist in non 2^n array sizes), will result in an interlock occurring, but attempts to program or erase these blocks will not occur since they are forced to be locked.

24.1.6.2 Read While Write (RWW)

The Flash Core is divided into partitions. Partitions always comprise two or more blocks. Partitions are used to determine Read While Write (RWW) groupings. While a write (program or erase) is being done within a given partition, a read can be simultaneously executed to any other partition. Partitions are listed in *Table 286*. Note that the shadow block has unique Read While Write restrictions described in *Section 24.1.6.8, C90FL Shadow Block*.

The FC is also divided into blocks to implement independent erase or program protection. The shadow block exists outside the normal address space and is programmed, erased and read independently of the other blocks. The shadow block is included to support systems that require NVM for security or system initialization information.

A software mechanism is provided to independently lock or unlock each block in high, mid, and low address space against program and erase. In addition, two hardware locks are provided to enable/disable the FC for program/erase. See *Section 24.1.6.4, Software Locking*, for more information.

24.1.6.3 C90FL Program

A flash program sequence operates on any page within the FC. Up to 4 words within the page may be altered in a single program operation. Whenever the array is program, the ECC bits also get programmed. ECC is handled on a 64 bit boundary. Thus, if only 1 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 results in an operation failure (most likely). It is recommended that all programming operations be from 64 bits to 128 bits, and be 64 bit aligned. The programming operation should completely fill selected ECC segments within the page. Only one program is allowed per 64 bit ECC segment between erases.

Caution: The chosen ECC algorithm allows some bit manipulations so that a Double Word can be re-written several times without needing an erase of the sector. This allows to use a Double Word to store flags useful for the EEPROM Emulation. As an example the chosen ECC algorithm allows to start from an All '1's Double Word value and rewrite whichever of its four 16-bit Half-Words to an All '0's content by keeping the same ECC value.

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.

Note: If a logic 0 is attempted to be "over programmed" by a logic 1, the resulting operation will fail (MCR[PEG] = 0), and the 0's that are interlocked will be merged (ORed) with 0's that are already present in the 64 bit ECC segment.

Addresses in locked/disabled blocks cannot be programmed. The user may program the values in any or all of 4 words, within a page, with a single program sequence. Page-bound words have addresses which differ only in address bits [3:2]. The program operation consists of the following sequence of events:

1. Change the value in the MCR[PGM] bit from a 0 to a 1.

Note: Ensure the block that contains the address to be programmed is unlocked.

2. Write the first address to be programmed with the program data. The flash module latches address bits [20:4] and chip-specific shadow enable at this time. The flash 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.
3. If more than 1 word or double word is to be programmed, write each additional address in the page with data to be programmed. This is referred to as a program data write. The flash module ignores address bits [20:4] and chip-specific shadow enable for program data writes. All unwritten data words default to 0xffff ffff.
4. Write a logic 1 to the MCR[EHV] bit to start the internal program sequence or skip to step () 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 sequence.

The first write after a program is initiated determines the page address to be programmed. 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. This first write is referred to as an interlock write. The interlock write determines if the shadow or normal array space is to be programmed by sampling chip-specific shadow enable and causing MCR[PEAS] to be set/cleared.

In the case of an erase-suspended program, the values in MCR[PEAS] may be modified via the program interlock write, enabling erase-suspended programs to and from shadow space.

An interlock write must be performed before setting 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 bits [3:2]. Unwritten locations default to a data value of 0xffff_ffff. If multiple writes are done to the same location the data for the last write is used in programming.

While DONE is low, EHV is high and PSUS is low 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 results in PEG being set low, indicating a failed operation. The data space being operated on before the abort contains indeterminate data. The user may not abort a program sequence while in program suspend.

Caution: Aborting a program operation leaves the FC addresses being programmed in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

24.1.6.4 Software Locking

A software mechanism is provided to independently lock/unlock each high, mid, and low address space against program and erase.

Software Locking is done through the LBL (Low/Mid Address Space Block Lock) or HBL (High Address Space Block Lock) registers. These can be written through register writes, and can be read through register reads.

24.1.6.5 C90FL Program Suspend/Resume

The program sequence may be suspended to allow read access to the FC. It is not possible to erase during a program suspend, or program during a program suspend. Read While Write may also be used to read the array during a program sequence providing the read is to a different partition.

A program suspend can be initiated by changing the value of the MCR[PSUS] bit from a 0 to a 1. MCR[PSUS] can be set high at any time when MCR[PGM] and MCR[EHV] are high. A 0 to 1 transition of MCR[PSUS] causes the flash module to start the sequence to enter program suspend, which is a read state. The user must wait until MCR[DONE] = 1 before the module is suspended. At this time FC reads may be attempted. MCR[DONE] goes high no more than Tpsus after MCR[PSUS] is set to a 1. Once suspended, the FC may only be read. Reads to the block(s) being programmed/erased return indeterminate data.

The program sequence is resumed by writing a logic 0 to MCR[PSUS]. MCR[EHV] must be set to a 1 before clearing MCR[PSUS] to resume operation. When the operation resumes, the flash module continues the program sequence from one of a set of predefined points. This may extend the time required for the program operation.

Caution: Repeated suspends at a high frequency may result in the operation timing out, and the flash module will respond by completing the operation with a fail code (MCR[PEG] = 0). The minimum time between suspends to ensure this does not occur is 100 μs.

24.1.6.6 C90FL 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 in the Low, Mid or High Address Space, or the shadow block. The erase sequence is fully automated within the flash. The user only needs to select the blocks to be erased and initiate the erase sequence. Locked/disabled blocks cannot be erased. If multiple blocks are selected for erase during an erase sequence, the blocks are erased sequentially starting with the lowest numbered block and terminating with the highest. The erase sequence consists of the following sequence of events:

1. Change the value in the MCR[ERS] bit from 0 to a 1.
2. Select the block, or blocks to be erased by writing ones to the appropriate registers in LMS or HBS registers. If the shadow block is to be erased, this step may be skipped, and LMS and HBS are ignored. For shadow block erase, see *Section 24.1.6.8, C90FL Shadow Block*, for more information.

Note: Lock and Select are independent. If a block is selected and locked, no erase occurs.

3. Write to any address in flash. This is referred to as an erase interlock write. The interlock write causes the values of SOC specific shadow enable to be captured and causing MCR[PEAS] to be set/cleared.
4. Write a logic 1 to the MCR[EHV] bit to start an 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.

After setting ERS, one write, referred to as an interlock write, must be performed before EHV can be set to a 1. This interlock causes the values of SOC specific shadow enable to be captured. 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 EHV assuming DONE is low, EHV is high and ESUS is low. An erase abort forces the module to step (8) of the erase sequence. An aborted erase results in PEG being set low, indicating a failed operation. The block(s) being operated on before the abort contain indeterminate data. The user may not abort an erase sequence while in erase suspend.

Caution: Aborting an erase operation leaves the FC blocks being erased in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

24.1.6.7 C90FL Erase Suspend/Resume

The erase sequence may be suspended to allow read access to the FC. The erase sequence may also be suspended to program (Erase-Suspended Program) the FC. A program started during erase suspend can in turn be suspended. Only one erase suspend and one program suspend are allowed at a time during an operation. It is not possible to erase during an erase suspend, or program during a program suspend. During suspend, all reads to FC locations targeted for program and blocks targeted for erase return indeterminate data. Programming locations in blocks targeted for erase during erase-suspended program may result in corrupted data. Read While Write may also be used to read the array during an erase sequence providing the read is to a partition not selected for erase.

An erase suspend can be initiated by changing the value of the MCR[ESUS] bit from a 0 to a 1. MCR[ESUS] can be set to a 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] goes high no more than Tesus after MCR[ESUS] is set to a 1. Once suspended, the array may be read or a program sequence may be initiated (erase-suspended program). Before initiating a program sequence the user must first clear MCR[EHV]. If a program sequence is initiated the values of SOC specific shadow enable is recaptured. Once the erase-suspended program is completed, the value of PEAS is returned to its “erase” value. FC reads while MCR[ESUS] = 1 from the block(s) being erased return indeterminate data.

The erase sequence is resumed by writing a logic 0 to MCR[ESUS]. MCR[EHV] must be set to a 1 and MCR[PGM] must be cleared (in the event of an erase suspended program)

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.

Caution: Repeated suspends at a high frequency may result in the operation timing out, and the flash module will respond by completing the operation with a fail code (MCR[PEG] = 0). The minimum time between erase suspends to ensure this does not occur is 200 μs.

Caution: In an erase-suspended program, programming FC locations in blocks which were being operated on in the erase may corrupt FC data.

24.1.6.8 C90FL Shadow Block

The C90FL Shadow Block is a memory-mapped block in the C90FL memory map. Program and erase of the shadow block are enabled only when PEAS=1 in the MCR. Once the user has begun an erase operation on the shadow block, it cannot be suspended to program the main address space and vice-versa. The user must terminate the shadow erase operation to program or erase the main address space.

The Shadow block can not utilize the RWW feature. Once an operation is started in the shadow block, a read can not be done to the shadow block, or any other block. Likewise, once an operation is started in a block in Low/Mid/High Address Space, a read can not be done in the shadow block.

The shadow block contains information on how the lock registers are reset. The first and second words can be used for reset configuration words. All other words can be used for user defined functions or other configuration words.

24.1.6.9 C90FL Reset

A reset is the highest priority operation for the C90FL and terminates all other operations.

The C90FL uses reset to initialize register and status bits to their default reset values. If the C90FL is executing a program or erase operation and a reset is issued, the operation will be aborted and the C90FL will disable the high voltage logic without damage to the high voltage circuits. Reset aborts all operations and forces the C90FL into user mode ready to receive accesses.

After reset is negated, register accesses can be performed, although it should be noted that registers that require updating from shadow information, or other inputs, cannot be read until C90FL exits reset.

24.1.6.10 Factory Margin Read

Factory Margin Read must be done after the following “initial factory conditions” are met:

- < 100 program/erase cycles
- Nominal supply values
- Operation at 25 °C

One Factory Margin Read is allowed per erase.

Factory Margin Read may be done to selected and unlocked blocks by combining UT0[MRE] and UT0[MRV] with the Array Integrity check. If UT0[MRE] is set, UT0[AIS] has no affect, and the reads will be done sequentially.

The data to be read is customer specific. Thus, a customer can provide user code into the flash memory and the correct MISR value is calculated. The customer is free to provide any

random or non-random code, and a valid MISR signature is calculated. Once the operations is completed, the results of the reads can be checking by reading the MISR value. Factory Margin Read is a self timed event, and is independent of system clocks, or wait states selected. Margin ECC corrections or detections are not done during the Factory Margin Read test.

1. Enable UTest mode.
2. Select the block, or blocks to be receive margin read check by writing ones to the appropriate registers in LMS or HBS/EHS registers. Make sure that selected blocks are also unlocked.

Note: It is not possible to do UTest operations on the shadow block.

It is possible to do User Mode array reads during the Factory Margin Read test, if desired, but the partition rules for Read While Write used during program and erase are in effect during Factory Margin Reads.

3. Set the UT0[MRE] bit.
4. Set the UT0[MRV] bit to desired value depending on it is desired to do One's Margin or Zero's Margin.
5. Seed the MISR UM0 thru UM4 with desired values.
6. Set the UT0[AIE] bit.
   If desired, the Margin Read operation may be aborted prior to UT0[AID] going high. This may be done by clearing the UT0[AIE] bit and then continuing to the next step. In the event of an aborted Margin Read check the MISR registers will contain a signature for the portion of the operation that was completed prior to the abort, and will not be deterministic.
7. Wait until the UT0[AID] bit goes high.
8. Read values in the MISR registers (UM0 through UM4) to ensure correct signature.
9. Write a logic 0 to the UT0[AIE] bit.

Note: If it is desired to do 2 or more margin reads, and it is desired to re-seed the MISR, a reset must be done between operations. If the subsequent margin reads can be done with the previously calculated MISR value, then a reset is not required.

24.1.6.11 Array integrity self check

Array Integrity is checked using a pre-defined address sequence (based on UT0[AIS]), and this operation is executed on selected and unlocked blocks. The data to be read is customer specific, thus a customer can provide user code into the flash and the correct MISR value is calculated. The customer is free to provide any random or non-random code, and a valid MISR signature is calculated. After the operation is completed, the results of the reads can be checking by reading the MISR value, to determine if an incorrect read, or ECC detection was noted. Array integrity is controlled by the system clock, and it is required that the Read Wait States and Address Pipelined control registers in the BIU be set to match the user defined frequency being used.

The Array Integrity Check consists of the following sequence of events:

1. Enable UTest mode.
2. Select the block, or blocks to be receive array integrity check by writing ones to the appropriate registers in LMS or HBS registers.

Note: Locked Blocks can be tested with Array Integrity if selected in LMS and HBS.

It is not possible to do UTest operations on the shadow block.

While Array Integrity is being executed, flash memory array accesses thru the BIU should not be requested.

3. If desired, set the UT0[AIS] bit to 1 for sequential addressing only.

Note: For normal integrity checks of the flash memory, sequential addressing is recommended.

If it is required to more fully check the read path (in a diagnostic mode), it is recommend that AIS be left at 0, to use the address sequence that checks the read path more fully, and examine read transitions. This sequence takes more time.

4. Seed the MISR UM0 thru UM4 with desired values.
5. Set the UT0[AIE] bit.
   If desired, the Array Integrity operation may be aborted prior to UT0[AID] going high. This may be done by clearing the UT0[AIE] bit and then continuing to the next step. It should be noted that in the event of an aborted array integrity check the MISR registers will contain a signature for the portion of the operation that was completed prior to the abort, and will not be deterministic. Prior to doing another array integrity operation, the UM0, UM1, UM2 and UM3 registers may need to be initialized to the desired seed value by doing register writes.
6. Wait until the UT0[AID] bit goes high.
7. Read values in the MISR registers (UM0 through UM4) to ensure correct signature.
8. Write a logic 0 to the UT0[AIE] bit.

24.1.6.12 ECC logic check

ECC logic can be checked by providing data to be read in the UT0[DSI], UT1[DAI] and/or UT2[DAI] registers. Then array reads can be done, ensuring expected results. The ECC logic check consists of the following sequence of events:

1. Enable UTest mode.
2. Write UT0[EIE] to 1.
3. Write UT0[DSI], UT1[DAI] and/or UT2[DAI] bits to provide data and check bit values to be read. Single or Double bit detections/corrections can be simulated by properly choosing Data and Check Bit combinations.
4. Write double word address to receive the data inputted in step 3 into the ADR register.
5. Reads can now be done through the BIU in a Read Request type fashion. In the event of a BIU read requested from an address that matches the address in the ADR register, expected data, and corrections or detections should be observed based on data written into the UT0[DSI], UT1[DAI] and/or UT2[DAI] registers. MCR[EER] and MCR[SBC] can be checked to evaluate the status of reads done.

Note: In the event of an ECC error or Single Bit Correction, during the ECC logic check (UTO[EIE] high), the ADR register will not be loaded, and the address tagged to receive the UT0[DSI], UT1[DAI] and/or UT2[DAI] values will be persevered.

6. Once completed, clear the UT0[EIE] bit to 0.

24.1.7 User option bits

Table 307 describes the user option bits on this device. These are programmed in the BIU4 register (see *Section 24.1.5.8, Bus Interface Unit 4 Register (BIU4)*) and verified using the UOPS register in the SSCM (see *Section 49.3.1.7, User Option Status Register (UOPS)*).

Table 307. User option bits

Bit number	Function name	Description
31–20	FCCU_CFG	FCCU configuration Bits 26–31: FCCU_CFG.FOP Bits 23–25: FCCU_CFG.FOM Bit 22: FCCU_CFG.PS Bit 21: FCCU_CFG.SM Bit 20: FCCU_CFG.CM
19–10	Reserved	
9	LSM_DPMB	0 Selects single core/DPM 1 Selects LSM (default value)
8–2	Reserved	
1	XOSC_MARGIN	XOSC oscillation margin select 0 Selects lower XOSC consumption and lower oscillator margin 1 Selects higher XOSC consumption and higher oscillator margin
0	WATCHDOG_ENABLE	0 Disable SWT 1 Enable SWT

The default states of the SWT modules depend on the value of the WATCHDOG_ENABLE and LSM_DPMB bits as described in *Table 308*.

Table 308. SWT default states

LSM_DPMB (Mode)	WATCHDOG_ENABLE	SWT default state
1 (LS mode)	1	Enable SWT_0 Enable SWT_1
	0	Disable SWT_0 Disable SWT_1
0 (DP mode)	1	Enable SWT_0 Disable SWT_1
	0	Disable SWT_0 Disable SWT_1

24.1.8 Test flash memory

Test flash memory contains calibration and other chip-specific data. This information is summarized in *Table 309*.

Table 309. Test flash information

Address	Word name	Function	Note
0x0000	TSENS_0_CAL W2/W4	TSENS_0 Calibration data W2 (P2/PTAT) and W4 (C2/CTAT) at cold temperature	2x12bits words
0x0004	TSENS_1_CAL W2/W4	TSENS_1 Calibration data W2 (P2/PTAT) and W4 (C2/CTAT) at cold temperature	2x12bits words
0x0008	TSENS_0_CAL W1/W3	TSENS_0 Calibration data W1 (P1/PTAT) and W3 (C1/CTAT) at hot temperature	2x12bits words
0x000C	TSENS_1_CAL W1/W3	TSENS_1 Calibration data W1 (P1/PTAT) and W3 (C1/CTAT) at hot temperature	2x12bits words
0x0010	ADC0_CAL W1	ADC0 Self-Test calibration - RC algo	2x12bits words (STAW3RH/L)
0x0014	ADC0_CAL W2	ADC0 Self-Test calibration - C algo S0 step	2x12bits words (STAW4RH/L)
0x0018	ADC0_CAL W3	ADC0 Self-Test calibration - C algo Sn step	2x12bits words (STAW5RH/L)
0x001C	ADC0_CAL W4	ADC0 Self-Test calibration - S algo S0 step - 3.3 V	2x12bits words (STAW0RH/L)
0x0020	ADC0_CAL W5	ADC0 Self-Test calibration - S algo S0 step - 5.0 V	2x12bits words (STAW0RH/L)
0x0024	ADC0_CAL W6	ADC0 Self-Test calibration - S algo S1 step - integer	2x12bits words (STAW1ARH/L)
0x0028	ADC0_CAL W7	ADC0 Self-Test calibration - S algo S1 step - float	2x12bits words (STAW1BRH/L)
0x002C	ADC0_CAL W8	ADC0 Self-Test calibration - S algo S2 step	1x12bits words (STAW2R)
0x0030	ADC0 Reserved	ADC0 reserved	For future expansion
0x0034	ADC1_CAL W1	ADC1 Self-Test calibration - RC algo	2x12bits words (STAW3RH/L)
0x0038	ADC1_CAL W2	ADC1 Self-Test calibration - C algo S0 step	2x12bits words (STAW4RH/L)
0x003C	ADC1_CAL W3	ADC1 Self-Test calibration - C algo Sn step	2x12bits words (STAW5RH/L)
0x0040	ADC1_CAL W4	ADC1 Self-Test calibration - S algo S0 step - 3.3 V	2x12bits words (STAW0RH/L)
0x0044	ADC1_CAL W5	ADC1 Self-Test calibration - S algo S0 step - 5.0 V	2x12bits words (STAW0RH/L)
0x0048	ADC1_CAL W6	ADC1 Self-Test calibration - S algo S1 step - integer	2x12bits words (STAW1ARH/L)

Table 309. Test flash information (continued)

Address	Word name	Function	Note
0x004C	ADC1_CAL W7	ADC1 Self-Test calibration - S algo S1 step - float	2x12bits words (STAW1BRH/L)
0x0050	ADC1_CAL W8	ADC1 Self-Test calibration - S algo S2 step	1x12bits words (STAW2R)
0x0054	ADC1 Reserved	ADC1 Reserved	For future expansion
0x0058	PART ID1 Low	Plant and Lot Number ASCII format	Needed for KGD
0x005C	PART ID1 High	Plant and Lot Number ASCII format	Needed for KGD
0x0060	PART ID2	Wafer number and coordinates	Needed for KGD
0x0064	SPARE 1	Reserved	For future usage
0x0068	SPARE 2	Reserved	For future usage

24.2 Dual-ported platform flash memory controller (PFLASH2P)

24.2.1 Introduction

This section provides an overview of the Dual Ported Platform Flash Controller (PFLASH2P) for Standard Product Platforms (SPP). The PFLASH2P acts as an interface between the system bus (AHB-Lite 2.v6) and the integrated flash memory array. It intelligently converts the protocols between the system bus and the dedicated flash array interface.

The PFLASH2P block supports a 64-bit data bus width at the AHB ports, and a 128-bit read data interface from the flash memory array. The PFLASH2P has two AHB ports with dedicated line buffers for each interface. Each port has four, 128-bit line buffers and an associated controller which prefetches sequential lines of data from the flash array into the buffers. Line buffer hits support zero-wait AHB data phase responses. AHB read requests which miss the buffers generate the needed flash array access and are forwarded to the AHB upon completion, typically incurring multiple wait-states at the maximum operating frequency.

The PFLASH2P module supports SoC designs based on a AHB system bus. In particular, this memory controller is designed to support multiple core designs such as those that include an I/O processor. The design of the line buffers can accommodate processor cores with or without caches.

24.2.1.1 Features

The following list summarizes the key features of the PFLASH2P:

- The PFLASH2P has two AHB ports with buffering and arbitration logic to efficiently share a single flash array in a multi-core system.
- The PFLASH2P system bus AHB interfaces support 64-bit data buses. All AHB aligned and unaligned reads are supported.Only aligned word and doubleword writes are supported.
- The PFLASH2P array interface supports a 128-bit read data bus a 64-bit write data bus.
- The PFLASH2P provides configurable, independent read buffering for each AHB port. There are four line read buffers and a prefetch controller for each port. The buffers implement a least-recently-used replacement algorithm to maximize performance. The nomenclature “page buffers and “line buffers” are used interchangeably.
- The PFLASH2P provides read and write access protections on a per master basis.
- The PFLASH2P provides configurable access timing allowing use in a wide range of platforms and frequencies.
- The PFLASH2P provides multiple-mapping support and mapping-based block access timing allowing use for emulation of other memory types.
- The PFLASH2P uses one single asynchronous reset and one global clock.
- The PFLASH2P is implemented using a pos-edge, fully-synthesizable methodology, and uses a muxed-DFF scan methodology for testability.

24.2.1.2 Block Diagrams

Figure 342 provides a system block diagram showing a dual ported flash controller and *Figure 343* is a high-level block diagram for the PFLASH2P module.

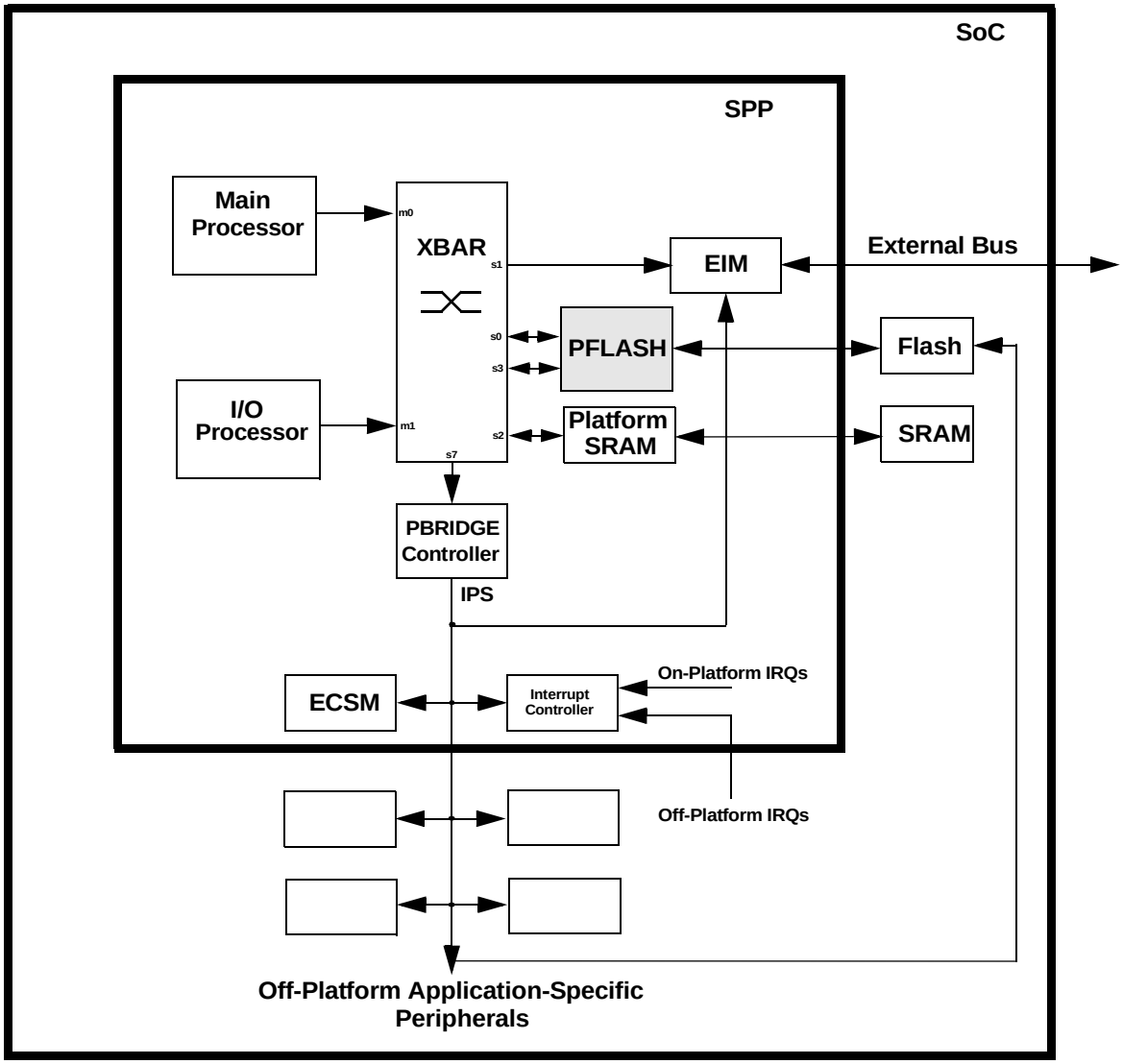


Figure 342. System Block Diagram with PFLASH Controller

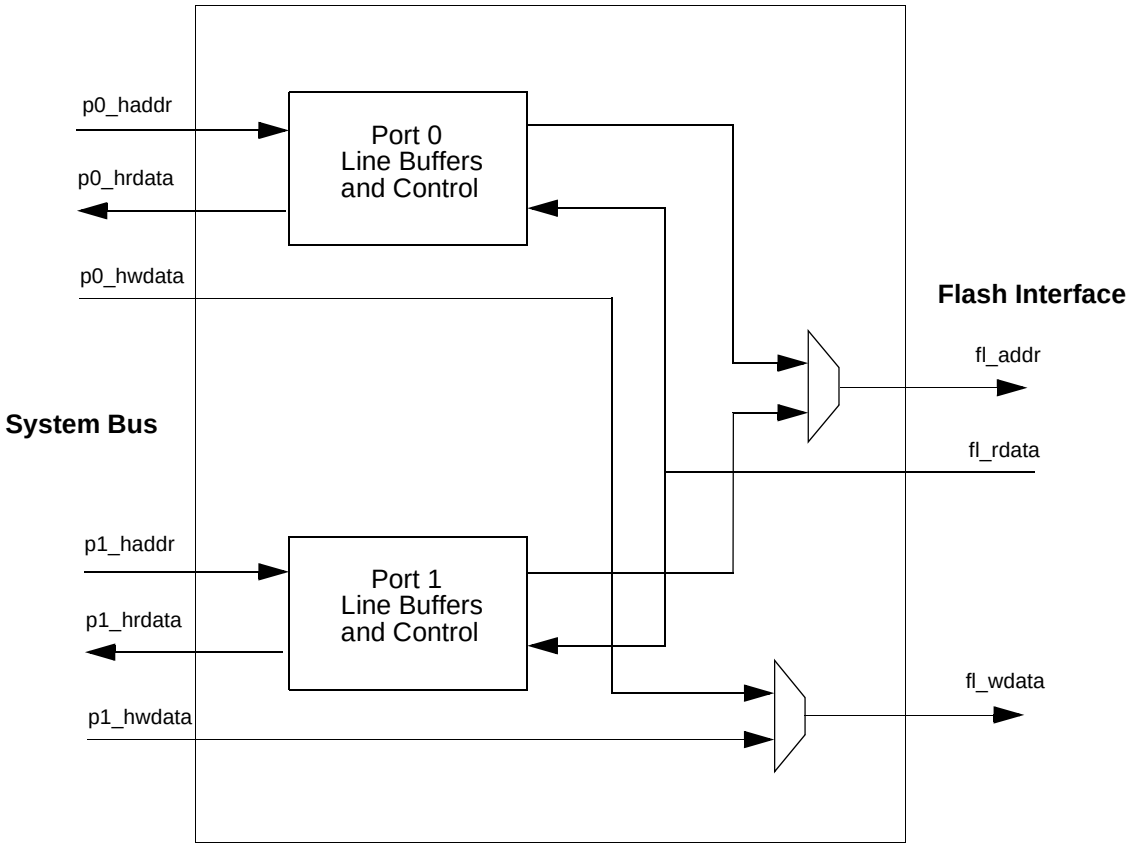


Figure 343. PFLASH2P High Level Block Diagram

24.2.1.3 General Operation

The PFLASH2P block provides two AHB-Lite slave ports for system bus masters to access the flash array. The PFLASH2P generates read and write enables, the flash array address, write size, and write data as inputs to the flash array controller. The PFLASH2P captures read data from the flash array interface and drives it onto the proper AHB port. Up to four lines of data are buffered by the PFLASH2P for each AHB port. Each line buffer holds 128 bits. Lines may be prefetched in advance of being requested by the AHB interface, allowing single-cycle read data (zero AHB wait-state) responses on buffer hits. Accesses may be overlapped on the flash array interface with the address and control signals transitioning to the next pending request (if any).

Read request addresses from the AHB are compared to the values stored in the tags in the prefetch controller. Access hits result in zero wait-state return of buffered data to the AHB. The prefetch controllers use one of several programmable algorithms to determine when to prefetch information from the flash array into one of the line buffers. Prefetching may be restricted to instruction-only triggering, data-only triggering, or instruction and data triggering. Line buffers are configured for use by instruction and/or data prefetches, allowing for exclusive or shared use by these access types.In addition, prefetching may be restricted to one or more AHB bus masters to improve the prefetch efficiency. In many systems, it may

be desirable to only enable prefetches for the processor core to limit the number of buffers used for other masters.

The PFLASH2P occupies a 512 MByte region of the address space. The actual flash array is multiply-mapped within this space. Upper address lines **haddr[28:24]** are used to provide additional control which allows the PFLASH2P responses on the AHB to be varied in order to provide for timing emulation of alternate memory types. See *Section 24.2.3.6, Wait-state emulation* for additional information. The PFLASH2P memory map is shown in *Figure 344*. Recall the PFLASH2P supports a 16 MByte (24 address bits) physical flash array size.

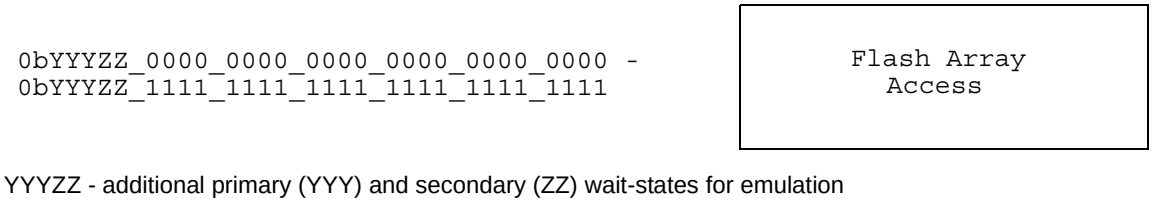


Figure 344. PFLASH2P Memory Map

Write accesses must be either word or doubleword in size, and must be aligned. Unaligned writes and byte or halfword writes result in an error termination on the AHB side, and no flash array write is initiated. Write addresses are captured from the AHB, and a write to the flash is initiated once data from the AHB is available on the following cycle. Write data is held valid until the flash write cycle completes.

24.2.2 Registers

Caution: Software executing from flash memory must not write to registers that control flash behavior (such as wait state settings or prefetch enable/disable). Doing so can cause data corruption.

Note: Flash memory configuration registers should be written only with 32-bit write operations to avoid any issues associated with register incoherency caused by bit fields spanning smaller size (8-, 16-bit) boundaries.

Within the module's programming model, there are a variety of control and configuration fields. Some are associated with the operating configuration of the flash memory array, while others are related to the behavior of the AHB master ports.

The PFLASH controller does *not* provide completely symmetric capabilities for the flash memory array.

First, consider the operating configuration of the flash memory array. In particular, there are 3 unique configuration fields that are associated with the array. These include all the parameters associated with the timing (read and write wait states, address pipeline control). Second, there are a total of 5 configuration fields that relate to the operation of the 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 memory array*.

24.2.2.1 Platform Flash Configuration Register 0 (PFCR0)

This register defines the configuration associated with the flash memory array. Additionally, it includes fields that provide specific information for the two separate AHB ports (p0 and p1).

Figure 345 shows the PFCR0. The fields in this register are described in *Table 310*.

Figure 345. PFLASH Configuration Register 0 (PFCR0) for port 0

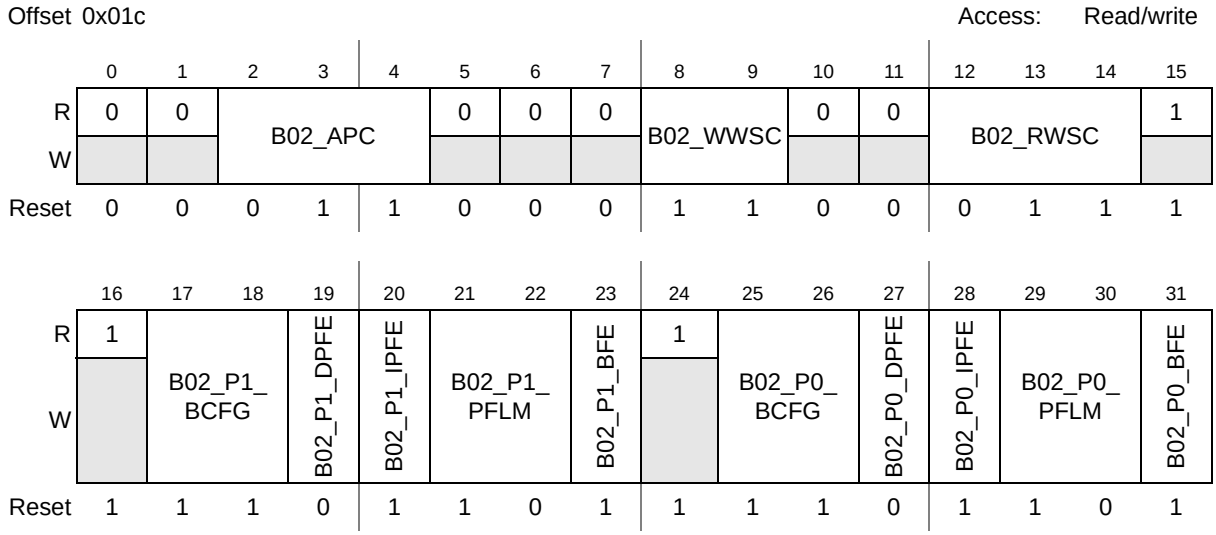


Table 310. PFCR0 field descriptions

Field	Description
B02_APC	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. The value of this field must be greater than or equal to the values in the WWSC and RWSC fields. Higher operating frequencies require non-zero settings for this field for proper flash operation. 000 Accesses may be initiated on consecutive (back-to-back) cycles 001 Access requests require one additional hold cycle 010 Access requests require two additional hold cycles ... 110 Access requests require six additional hold cycles 111 No Address Pipelining *Note: APC must equal RWSC.* Refer PFLASH Configuration Register 0 (PFCR0) settings for different frequencies table for values at different frequencie.
B02_WWSC	Write Wait State Control. This field is used to control the number of wait-states to be added to the flash array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFLASH. Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to an appropriate value by hardware reset. 00 No additional wait-states are added 01 1 additional wait-state is added 10 2 additional wait-states are added 11 3 additional wait-states are added Refer PFLASH Configuration Register 0 (PFCR0) settings for different frequencies table for values at different frequencies .
B02_RWSC	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. The integrator is strongly encouraged to verify these settings based on actual silicon results. 000 No additional wait-states are added 001 1 additional wait-state is added 010 2 additional wait-states are added ... 111 7 additional wait-states are added Refer PFLASH Configuration Register 0 (PFCR0) settings for different frequencies table for values at different frequencies.

Table 310. PFCR0 field descriptions (continued)

Field	Description
B02_P1_BCFG	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	Port 1 Data Prefetch Enable. This field enables or disables prefetching initiated by a data read access. This field is set by hardware reset. 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	Port 1 Instruction Prefetch Enable. This field enables or disables prefetching initiated by an instruction fetch read access. This field is cleared by hardware reset. 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	Port 1 Prefetch Limit. This field controls the prefetch algorithm used by the PFLASH controller. This field defines the prefetch behavior. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is set to 2b01 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_P1_BFE	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 310. PFCR0 field descriptions (continued)

Field	Description
B02_P0_BCFG	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	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	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	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	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.

Table 311. PFLASH Configuration Register 0 (PFCR0) settings for different frequencies

Frequency	Flash Wait State	APC field	WWSC field	RWSC field	RAM Wait State
<=120MHz	3	3	3	3	1
<=80MHz	2	2	2	2	0
<=60MHz	1	1	1	1	0

Note: The fields APC, WWSC and RWSC of the PFLASH Configuration Register 0 (PFCR0) should be set to the same value.

The RAM WS are configured by the MUDCR bit in the Miscellaneous User-Defined Control Register (MUDCR).

24.2.2.2 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 PFLASH2P_LCA. The register is described below in *Figure 346* and *Table 312*.

The contents of the register are loaded from location 0xF03E00 of the shadow region in the flash memory 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 0xF03E00 of the shadow region in the flash array must be programmed using the normal sequence of operations. The reset value shown in *Figure 346* reflects an erased or unprogrammed value from the shadow region.

Figure 346. PFLASH Access Protection Register (PFAPR)

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

Table 312. PFAPR field descriptions

Field	Description
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 memory array. 00 Fixed priority arbitration with AHB p0 > p1 01 Fixed priority arbitration with AHB p1 > p0 1*X* Round-robin arbitration
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
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

24.2.3 Functional description

The PFLASH2P has two AHB-Lite slave ports and a single flash array interface. The dual ported design of the PFLASH2P enables efficient use of a single flash memory array by two processor cores. Each AHB port has dedicated line buffers to support single-cycle read accesses and to limit accesses to the flash array.

The PFLASH2P generates read and write enables, the flash array address, write size, and write data as inputs to the flash array controller. The PFLASH2P captures read data from the flash array interface and drives it onto the proper AHB port. If line buffering is enabled, when data is read from the array it is stored in a line buffer. Up to four lines of data (128 bits) are buffered by the PFLASH2P for each AHB port.

If pre-fetching is enabled, data is read in advance and stored in the line buffers allowing single-cycle (zero AHB wait-states) read data responses on buffer hits. 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.

Arbitration between the two AHB ports for access to the flash interface is primarily based on the type of access; writes have priority over reads which have priority over prefetches. If both ports are doing the same type of access, priority is based on the settings of the arbitration and priority bits in the PFCRP0 register.

24.2.3.1 Basic interface protocol

The PFLASH2P interfaces to the flash array by driving addresses and read or write enable signals.

Accesses are terminated based on timing parameters contained in the PFCR0. The correct setting of the wait-state control bits in the PFCR0 is determined by the access time of the flash array and the platform clock frequency.

The PFLASH2P also has the capability of extending the normal AHB access timing by inserting additional wait states for reads. This capability is provided to allow emulation of other memories which have different access time characteristics. These wait-states are applied in addition to the normal wait-states incurred for flash accesses. Refer to *Section 24.2.3.6, Wait-state emulation* for more detail on wait-state emulation.

Prefetching of next sequential line can be blocked. Buffer hits can be blocked as well, regardless of whether the access corresponds to valid data in one of the line read buffers. These steps are taken to ensure that timing emulation is correct and that excessive prefetching is avoided.

24.2.3.2 Read cycles

Read cycles from the flash array are initiated by driving a valid access address and then asserting a read enable. The PFLASH2P then waits for the flash array to provide read data. This data is normally stored in the least-recently updated line read buffer in parallel with the requested data being forwarded to the AHB.

Single clock read responses to the AHB are possible with the PFLASH2P when the requested read access is buffered. In these cases, read data is returned to the AHB data phase with a zero wait-state response.

24.2.3.3 Write cycles

Write cycles to the flash array are initiated by driving a valid access address, driving write data, and indicating the size of the write data.The PFLASH2P then waits for the indicated number of wait states before the cycle has terminated.

24.2.3.4 Flash error response operation

The flash array may signal an error response. 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 PFLASH2P will not update or validate a line read buffer. An error response may be signaled on read or write operations.

24.2.3.5 Line Read Buffers and Prefetch Operation

The PFLASH2P contains four read buffers per AHB port which are used to hold line and ECC data read from the flash array. Each buffer operates independently, and is filled using a single array access. The buffers are used for both prefetch and normal demand fetches.

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 PFLASH2P may trigger a prefetch to the next sequential line of array data on the 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 line read 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:

- Invalid - the buffer contains no valid data
- Used - the buffer contains valid data which has been provided to satisfy an AHB burst type read
- Valid - the buffer contains valid data which has been provided to satisfy an AHB single type read
- Prefetched - the buffer contains valid data which has been prefetched to satisfy a potential future AHB access
- Busy AHB - the buffer is currently being used to satisfy an AHB burst read
- 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 reverse 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 line buffer has been selected, the flash array is accessed and read data loaded into the buffer. If the buffer load was in response to a miss, the just-loaded buffer is immediately marked as most-recently-used. If the buffer load was in response to a speculative fetch to the next-sequential line address after a buffer hit, *the recently-used status is not changed*. Rather, it is marked as most-recently-used only after a subsequent buffer hit.

This policy maximizes performance based on reference patterns of flash accesses and allows for prefetched data to remain valid when non-prefetch enabled bus masters are granted flash access.

Several algorithms are available for prefetch control which trade off performance for power. More aggressive prefetching increases power due to the number of wasted (discarded) prefetches, but may increase performance by lowering average read latency.

In order for prefetching to occur, PFCRPx[BFEN] must be set to ‘1’; PFCRPx[PFLIM] must be non-zero; either PFCRPx[IPFEN] or PFCRPx[DPFEN] must be ‘1’ and PFCRPx[MxPFE] must be ‘1’.

24.2.3.5.1 Inst/Data Prefetch Triggering

Prefetch triggering may be enabled for instruction reads via the PFCR0[IPFEN] control bit, while prefetching for data reads is enabled via the PFCR0[DPFEN] control bit. Additionally, the PFCRPx[PFLIM] must also be set to enable prefetching. Prefetches are never triggered by write cycles.

24.2.3.5.2 Per-Master Prefetch Triggering

Prefetch triggering may be controlled for individual bus masters via the PFCR0[MxPFD] control field.

24.2.3.5.3 Buffer Allocation

Allocation of the line read buffers is controlled via the PFCRPx control register for each AHB port. The LBCFG field of this register defines the operating organization of the four line buffers. The buffers can be organized as a “pool” of available resources with all four buffers available for either instruction or data. They can also be configured 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. In this configuration data prefetching is disabled.

24.2.3.5.4 Buffer invalidation

The line read buffers may be invalidated under hardware and software control. Assertion of the **fl_invbuf** input signal causes the line read buffers to be marked as invalid. To ensure that stale data is not read from the buffers, software should invalidate the buffers after writing to the array. This is done by clearing the PFCRPx[BFEN] bit, which also disables the buffers. Software may then restore the PFCRPx[BFEN] bit to its previous state, and the buffers will have been invalidated.

Also, the buffers are invalidated by hardware on any non-sequential (NSEQ) access with a non-zero wait state value to support wait-state emulation.

24.2.3.6 Wait-state emulation

Emulation of other memory array timings are supported by the PFLASH2P 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 PFLASH2P inserts additional wait-states according to the values of the PFCR0 Read Wait State Control fields plus the value on the address line bits 28-24. Wait-states are applied to the initial access of a burst fetch or to single-beat read accesses on the AHB system bus.

Note: There is an inherent 2-cycle delay added when using non-zero values for address line bits 28-24.

When the address line bits 28-24 are non-zero, normal AHB termination is extended only for read cycles. Write cycles are not affected. In addition, no line read buffer prefetches are initiated, and buffer hits are ignored. See the description of the Read Wait State Control field in the PFCR0 register for further information on read wait states.

Wait states are applied to the initial access of a burst fetch or to single-beat read accesses on the AHB system bus.

Table 313. Additional wait-state encoding

Memory address	Additional wait-stats
flash address + 0x0000_0000	0
flash address + 0x0100_0000	10
flash address + 0x0200_0000	18
flash address + 0x0300_0000	26

Table 313. Additional wait-state encoding (continued)

Memory address	Additional wait-stats
flash address + 0x0400_0000	3
flash address + 0x0500_0000	11
flash address + 0x0600_0000	19
flash address + 0x0700_0000	27
flash address + 0x0800_0000	4
flash address + 0x0900_0000	12
flash address + 0x0a00_0000	20
flash address + 0x0b00_0000	28
flash address + 0x0c00_0000	5
flash address + 0x0d00_0000	13
flash address + 0x0e00_0000	21
flash address + 0x0f00_0000	29
flash address + 0x1000_0000	6
flash address + 0x1100_0000	14
flash address + 0x1200_0000	22
flash address + 0x1300_0000	30
flash address + 0x1400_0000	7
flash address + 0x1500_0000	15
flash address + 0x1600_0000	23
flash address + 0x1700_0000	31
flash address + 0x1800_0000	8
flash address + 0x1900_0000	16
flash address + 0x1a00_0000	24
flash address + 0x1b00_0000	32
flash address + 0x1c00_0000	9
flash address + 0x1d00_0000	17
flash address + 0x1e00_0000	25
flash address + 0x1f00_0000	33

25 FlexCAN Module

25.1 Introduction

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. A general block diagram is shown in *Figure 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. This device supports 32 Message Buffers. The functions of the sub-modules are described in subsequent sections.

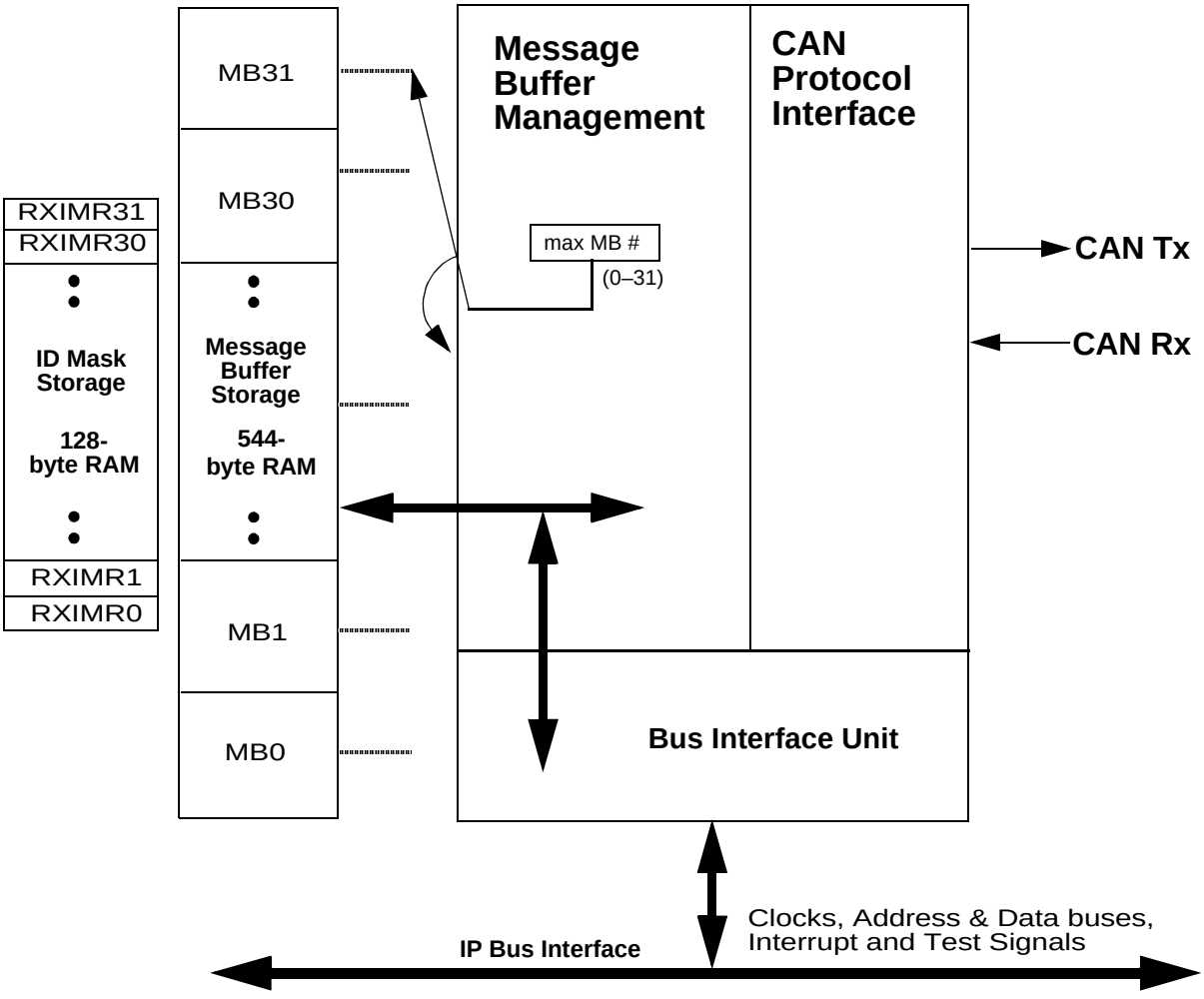


Figure 347. FlexCAN block diagram

25.1.1 Overview

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B, which supports both standard and extended message frames. 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.

25.1.2 FlexCAN module features

The FlexCAN module includes these distinctive features:

- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Zero to eight bytes data length
  - Programmable bit rate up to 1 Mbit/s
  - Content-related addressing
- 32 Message Buffers 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 544 bytes of RAM used for MB storage
- Includes 128 bytes (32 MBs) of RAM used for individual Rx Mask Registers
- Full featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard or 32 partial (8 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

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs.

25.1.3 Modes of operation

The FlexCAN module has four functional modes: Normal Mode (User and Supervisor), Freeze Mode, Listen-Only Mode and Loop-Back Mode. There are also two low power modes: Disable Mode and Stop Mode.

25.1.3.1 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.

25.1.3.2 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 25.4.9.1, Freeze Mode,* for more information.

25.1.3.3 Listen-Only Mode

The module enters this mode when the LOM bit in the Control Register is asserted. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.

25.1.3.4 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.

25.1.3.5 Module Disable Mode:

This low power mode is entered when the MDIS bit in the MCR Register is asserted. When disabled, the module shuts down 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 25.4.9.2, Module Disable Mode,* for more information.

25.1.3.6 Stop Mode

This low power mode is entered when Stop Mode is requested at MCU level. When in Stop Mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally. Exit from this mode happens when the Stop Mode request is removed or when activity is detected on the CAN bus. See *Section 25.4.9.3, Stop Mode,* for more information.

25.2 External signal description

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in *Table 314* and described in more detail in the next subsections.

Table 314. 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. Please consult the Device User Guide for the actual signal names.

25.2.1 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’.

25.2.2 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’.

25.3 Memory map and 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.

25.3.1 FlexCAN memory mapping

The memory map for the FlexCAN module is shown in *Table 315*. Each individual register is identified by its complete name and the corresponding mnemonic. The access type can be Supervisor (S) or Unrestricted (U). Most of the registers can be configured to have either Supervisor or Unrestricted access by programming the SUPV bit in the MCR Register. These registers are identified as S/U in the Access column of *Table 315*.

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. 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 315. Module Memory Map

Address	Use	Access Type	Affected by Hard Reset	Affected by Soft Reset
Base + 0x0000	Module Configuration (MCR)	S	Yes	Yes
Base + 0x0004	Control Register (CTRL)	S/U	Yes	No
Base + 0x0008	Free Running Timer (TIMER)	S/U	Yes	Yes
Base + 0x000C	Reserved			
Base + 0x0010	Rx Global Mask (RXGMASK)	S/U	Yes	No
Base + 0x0014	Rx Buffer 14 Mask (RX14MASK)	S/U	Yes	No
Base + 0x0018	Rx Buffer 15 Mask (RX15MASK)	S/U	Yes	No
Base + 0x001C	Error Counter Register (ECR)	S/U	Yes	Yes
Base + 0x0020	Error and Status Register (ESR)	S/U	Yes	Yes
Base + 0x0024	Reserved			
Base + 0x0028	Interrupt Masks 1 (IMASK1)	S/U	Yes	Yes
Base + 0x002C	Reserved			
Base + 0x0030	Interrupt Flags 1 (IFLAG1)	S/U	Yes	Yes
Base + 0x0034–0x005F	Reserved			
Base + 0x0060–0x007F	Reserved			
Base + 0x0080–0x017F	Message Buffers MB0–MB15	S/U	No	No
Base + 0x0180–0x027F	Message Buffers MB16–MB31	S/U	No	No
Base + 0x0280-087F	Reserved			
Base + 0x0880-0x08BF	Rx Individual Mask Registers RXIMR0-RXIMR15	S/U	No	No
Base + 0x08C0-0x08FF	Rx Individual Mask Registers RXIMR16-RXIMR31	S/U	No	No
Base + 0x0900-0x097F	Reserved			

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 316*. *Table 316* shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

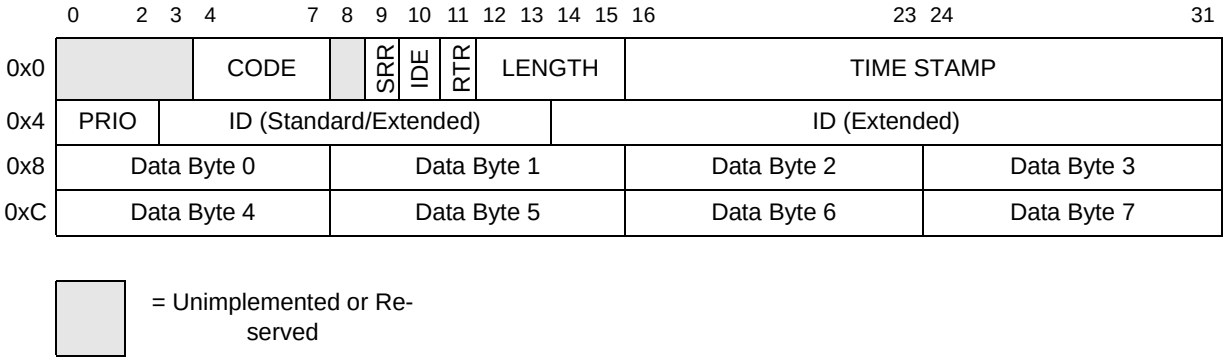
Table 316. Message Buffer (MB0) Memory Mapping

Address Offset	MB Field
0x80	Control and Status (C/S)
0x84	Identifier Field
0x88–0x8F	Data Field 0 – Data Field 7 (1 byte each)

25.3.2 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



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 317* and *Table 318*. See *Section 25.4, Functional Description,* for additional information.

Table 317. 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.

Table 317. Message Buffer Code for Rx buffers

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to *Section 25.4.5, Matching Process,* for details about overrun behavior.
0110	OVERRUN: a frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB will be overwritten again, and the code will remain OVERRUN. Refer to *Section 25.4.5, Matching Process,* for details about overrun behavior.
0XY1(1)	BUSY: Flexcan is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

1. Note that for Tx MBs (see *Table 318*), the BUSY bit should be ignored upon read, except when AEN bit is set in the MCR register.

Table 318. Message Buffer Code for Tx buffers

RTR	Initial Tx code	Code after successful transmission	Description
X	1000	—	INACTIVE: MB does not participate in the arbitration process.
X	1001	—	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when AEN bit in MCR is asserted. MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.

Table 318. Message Buffer Code for Tx buffers

RTR	Initial Tx code	Code after successful transmission	Description
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.

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

RTR — Remote Transmission Request

This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as '1' (recessive) and receives it as '0' (dominant), it is interpreted as arbitration loss. If this bit is transmitted as '0' (dominant), then if it is received as '1' (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission.

1 = Indicates the current MB has a Remote Frame to be transmitted
0 = Indicates the current MB has a Data Frame to be transmitted

LENGTH — Length of Data in Bytes

This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the MB space (see *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 25.4.3, Arbitration process.*

ID — Frame Identifier

In Standard Frame format, only the 11 most significant bits (3 to 13) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In Extended Frame format, all bits are used for frame identification in both receive and transmit cases.

DATA — Data Field

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.

25.3.3 Rx FIFO Structure

When the FEN bit is set in the MCR, the memory area from 0x80 to 0xFC (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. *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 25.4.7, Rx FIFO,* for more information.

Figure 349. Rx FIFO Structure

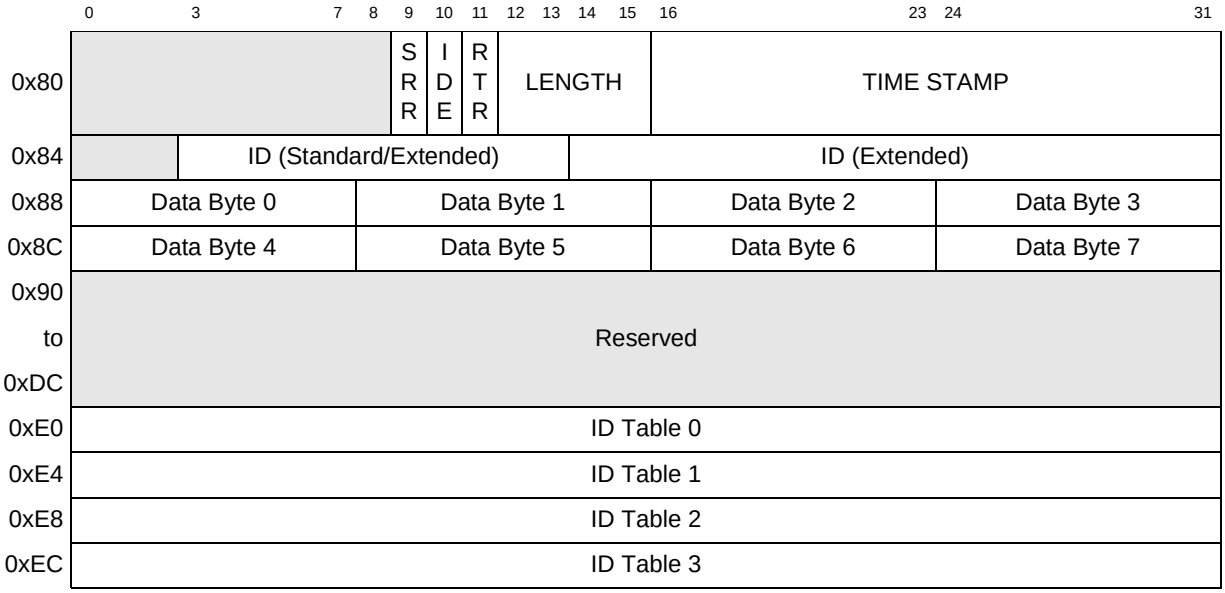


Figure 349. Rx FIFO Structure (continued)

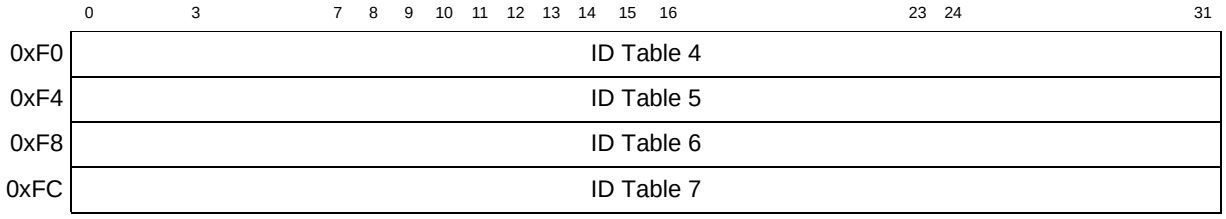
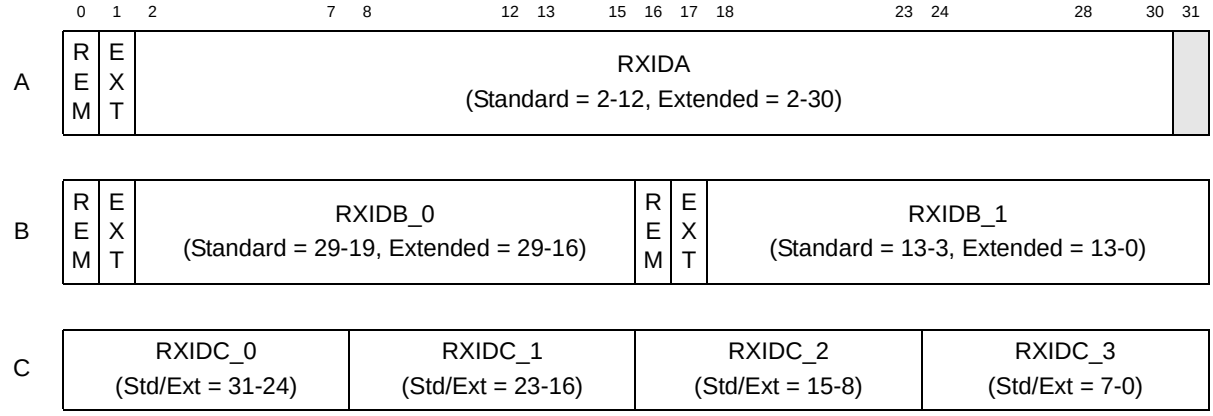


Figure 350. ID Table 0 - 7



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 (2 to 12) 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) (2 to 12 and 18 to 28) 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.

25.3.4 Register descriptions

The FlexCAN registers are described in this section in ascending address order.

25.3.4.1 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.

Base + 0x0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FRZ	FEN	HALT	NOT_RDY	0	SOFT_RST	FRZ_ACK	SUPV	0	WRN_EN	LPM_ACK	0	0	SRX_DIS	BCC
W																
RESET:	Note (1)	1	0	1	1	0	0	Note (2)	1	0	0	Note(3)	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	LPRIO_EN	AEN	0	0	IDAM		0	0	MAXMB					
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

= Unimplemented or Reserved

1. Reset value of this bit is different on various platforms. Consult the specific MCU documentation to determine its value.
2. Different on various platforms, but it is always the opposite of the MDIS reset value.
3. Different on various platforms, but it is always the same as the MDIS reset value.

Figure 351. Module Configuration Register (MCR)

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 25.4.9.2, 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 25.3.3, Rx FIFO Structure* and *Section 25.4.7, Rx FIFO,* for more information. This bit must be written in Freeze mode only.

1 = FIFO enabled
0 = FIFO not enabled

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 25.4.9.1, 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 either in Disable Mode, Stop Mode or Freeze Mode. It is negated once FlexCAN has exited these modes.

1 = FlexCAN module is either in Disable Mode, Stop Mode or Freeze Mode
0 = FlexCAN module is either in Normal Mode, Listen-Only Mode or Loop-Back Mode

SOFT_RST — Soft Reset

When this bit is asserted, FlexCAN resets its internal state machines and some of the memory mapped registers. The following registers are reset: MCR (except the MDIS bit), TIMER, ECR, ESR, IMASK1, IFLAG1. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected:

- CTRL
- RXIMR0–RXIMR31
- RXGMASK, RX14MASK, RX15MASK
- all Message Buffers

The SOFT_RST bit can be asserted directly by the CPU when it writes to the MCR 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 315*
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 25.4.9.1, 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 315*. Reset value of this bit is ‘1’, so the affected registers start with Supervisor access restrictions. This bit should be written in Freeze mode only.

1 = Affected registers are in Supervisor memory space. Any access without supervisor permission behaves as though the access was done to an unimplemented register location
0 = Affected registers are in Unrestricted memory space

WRN_EN — Warning Interrupt Enable

When asserted, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the Error and Status Register. If WRN_EN is negated, the TWRN_INT and RWRN_INT flags will always be zero, independent of the values of the error counters, and no warning interrupt will ever be generated. This bit must be written in Freeze mode only.

1 = TWRN_INT and RWRN_INT bits are set when the respective error counter transition from <96 to ≥ 96.
0 = TWRN_INT and RWRN_INT bits are zero, independent of the values in the error counters.

LPM_ACK — Low Power Mode Acknowledge

This read-only bit indicates that FlexCAN is either in Disable Mode or Stop Mode. Either of these low power modes can not be entered until all current transmission or reception processes have finished, so the CPU can poll the LPM_ACK bit to know when FlexCAN has actually entered low power mode. See *Section 25.4.9.2, Module Disable Mode,* and *Section 25.4.9.3, Stop Mode,* for more information.

1 = FlexCAN is either in Disable Mode or Stop mode

0 = FlexCAN not in any of the low power modes

SRX_DIS — Self Reception Disable

This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module will not be stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal will be generated due to the frame reception. This bit must be written in Freeze mode only.

1 = Self reception disabled
0 = Self reception enabled

BCC — Backwards Compatibility Configuration

This bit is provided to support Backwards Compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied:

- For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per MB, FlexCAN uses its previous masking scheme with RXGMASK, RX14MASK and RX15MASK.
- The reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, FlexCAN will not look for another matching MB. It will override this MB with the new message and set the CODE field to ‘0110’ (overrun).

Upon reset this bit is negated, allowing legacy software to work without modification. This bit must be written in Freeze mode only.

1 = Individual Rx masking and queue feature are enabled.
0 = Individual Rx masking and queue feature are disabled.

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

AEN— Abort Enable

This bit is supplied for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification. This bit must be written in Freeze mode only.

1 = Abort enabled
0 = Abort disabled

IDAM — ID Acceptance Mode

This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in *Table 319*. Note that all elements of the table are configured at the same time by this field (they are all the same format). See *Section 25.3.3, Rx FIFO Structure*. This bit must be written in Freeze mode only.

Table 319. 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.

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.

25.3.4.2 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. This register can be accessed at any time, however some fields must be changed only during either Disable Mode or Freeze Mode. Find more information in the fields descriptions ahead.

Base + 0x0004

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRESDIV								RJW		PSEG1			PSEG2		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BOFF_MSK	ERR_MSK	CLK_SRC	LPB	TWRN_MSK	RWRN_MSK	0	0	SMP	BOFF_REC	TSYN	LBUF	LOM	PROPSEG		
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 352. Control Register (CTRL)

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 25.4.8.4, 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[j] 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

CLK_SRC — CAN Engine Clock Source

This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the peripheral clock (driven by the FMPLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit must only be changed while the module is in Disable Mode. See *Section 25.4.8.4, 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: This clock selection feature may not be available in all MCUs. A particular MCU may not have a FMPLL, in which case it would have only the oscillator clock, or it may use only the FMPLL clock feeding the FlexCAN module. In these cases, this bit has no effect on the module operation.

j. One time quantum is equal to the Sclock period.

In order to guarantee reliable operation, the selected CAN Protocol Interface (CPI) clock should not be faster as the the peripheral clock.

If the application needs to go into STOP mode (FMPLL turned off), the FlexCAN must be moved into freeze mode before the device goes into STOP mode. The application can move the FlexCAN out of freeze mode as soon as the PLL is locked again. If this recommendation is not fulfilled, the FlexCAN could violate the CAN specification or transmit incorrect data.

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

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. 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.

25.3.4.3 Free Running Timer (TIMER)

This register represents a 16-bit free running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Base + 0x0008

	0	1	2	3	4	5	6	7	8	9	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

= Unimplemented or Reserved

Figure 353. Free Running Timer (TIMER)

25.3.4.4 Rx Global Mask (RXGMASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in MCR causes the RXGMASK Register to have no effect on the module operation. For MCUs not supporting individual masks per MB, this register is always effective.

RXGMASK is used as acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in MCR is set (FIFO enabled), the RXGMASK also applies to all elements of the ID filter table, except elements 6-7, which have individual masks.

Refer to *Section 25.4.7, 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.

Base + 0x0010

	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

= Unimplemented or Reserved

Figure 354. Rx Global Mask Register (RXGMASK)

MI31–MI0 — Mask Bits

For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR).

1 = The corresponding bit in the filter is checked against the one received
0 = The corresponding bit in the filter is “don’t care”

25.3.4.5 Rx 14 Mask (RX14MASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in MCR causes the RX14MASK Register to have no effect on the module operation.

RX14MASK is used as acceptance mask for the Identifier in Message Buffer 14. When the FEN bit in MCR is set (FIFO enabled), the RXG14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register.

Refer to *Section 25.4.7, 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

25.3.4.6 Rx 15 Mask (RX15MASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in MCR causes the RX15MASK Register to have no effect on the module operation.

When the BCC bit is negated, RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When the FEN bit in MCR is set (FIFO enabled), the 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 25.4.7, 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

25.3.4.7 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.

Base + 0x001C

	0	1	2	3	4	5	6	7	8	9	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

= Unimplemented or Reserved

Figure 355. Error Counter Register (ECR)

25.3.4.8 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 25.4.10, Interrupts,* for more details.

Base + 0x0020

	0	1	2	3	4	5	6	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															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	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														w1c	w1c	
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 356. Error and Status Register (ESR)

TWRN_INT — Tx Warning Interrupt Flag

If the WRN_EN bit in MCR is asserted, the TWRN_INT bit is set when the TX_WRN flag transition from ‘0’ to ‘1’, meaning that the Tx error counter reached 96. If the corresponding mask bit in the Control Register (TWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect.

1 = The Tx error counter transition from < 96 to ≥ 96
0 = No such occurrence

RWRN_INT — Rx Warning Interrupt Flag

If the WRN_EN bit in MCR is asserted, the RWRN_INT bit is set when the RX_WRN flag transition from ‘0’ to ‘1’, meaning that the Rx error counters reached 96. If the corresponding mask bit in the Control Register (RWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect.

1 = The Rx error counter transition from < 96 to ≥ 96
0 = No such occurrence

BIT1_ERR — Bit1 Error

This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message.

1 = At least one bit sent as recessive is received as dominant
0 = No such occurrence

Note: .This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.

BIT0_ERR — Bit0 Error

This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message.

1 = At least one bit sent as dominant is received as recessive
0 = No such occurrence

ACK_ERR — Acknowledge Error

This bit indicates that an Acknowledge Error has been detected by the transmitter node, i.e., a dominant bit has not been detected during the ACK SLOT.

1 = An ACK error occurred since last read of this register
0 = No such occurrence

CRC_ERR — Cyclic Redundancy Check Error

This bit indicates that a CRC Error has been detected by the receiver node, i.e., the calculated CRC is different from the received.

1 = A CRC error occurred since last read of this register.
0 = No such occurrence

FRM_ERR — Form Error

This bit indicates that a Form Error has been detected by the receiver node, i.e., a fixed-form bit field contains at least one illegal bit.

1 = A Form Error occurred since last read of this register
0 = No such occurrence

STF_ERR — Stuffing Error

This bit indicates that a Stuffing Error has been detected.

1 = A Stuffing Error occurred since last read of this register.
0 = No such occurrence.

TX_WRN — TX Error Warning

This bit indicates when repetitive errors are occurring during message transmission.

1 = TX_Err_Counter ≥ 96
0 = No such occurrence

RX_WRN — Rx Error Warning

This bit indicates when repetitive errors are occurring during message reception.

1 = Rx_Err_Counter ≥ 96
0 = No such occurrence

IDLE — CAN bus IDLE state

This bit indicates when CAN bus is in IDLE state.

1 = CAN bus is now IDLE
0 = No such occurrence

TXRX — Current FlexCAN status (transmitting/receiving)

This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted.

1 = FlexCAN is transmitting a message (IDLE=0)
0 = FlexCAN is receiving a message (IDLE=0)

FLT_CONF — Fault Confinement State

This 2-bit field indicates the Confinement State of the FlexCAN module, as shown in *Table 320*. If the LOM bit in the Control Register is asserted, the FLT_CONF field will indicate “Error Passive”. Since the Control Register is not affected by soft reset, the FLT_CONF field will not be affected by soft reset if the LOM bit is asserted.

Table 320. Fault Confinement State

Value	Meaning
00	Error Active
01	Error Passive
1X	Bus Off

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

25.3.4.9 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).

Base + 0x0028

	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 357. Interrupt Masks 1 Register (IMASK1)

BUF31M–BUF0M — Buffer MB_i Mask

Each bit enables or disables the respective FlexCAN Message Buffer (MB0 to MB31) Interrupt.

1 = The corresponding buffer Interrupt is enabled
0 = The corresponding buffer Interrupt is disabled

Note: Setting or clearing a bit in the IMASK1 Register can assert or negate an interrupt request, if the corresponding IFLAG1 bit is set.

25.3.4.10 Interrupt Flags 1 Register (IFLAG1)

This register defines the flags for 32 Message Buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG1 bit. If the corresponding IMASK1 bit is set, an interrupt will be generated. The Interrupt flag must be cleared by writing it to ‘1’. Writing ‘0’ has no effect.

When the AEN bit in the MCR is set (Abort enabled), while the IFLAG1 bit is set for an MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

When the FEN bit in the MCR is set (FIFO enabled), the function of the 8 least significant interrupt flags (BUF7I - BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Base + 0x0030

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	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
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	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
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 358. Interrupt Flags 1 Register (IFLAG1)

BUF31I–BUF8I — Buffer MB_i Interrupt

Each bit flags the respective FlexCAN Message Buffer (MB8 to MB31) interrupt.

1 = The corresponding MB has successfully completed transmission or reception
0 = No such occurrence

BUF7I — Buffer MB7 Interrupt or “FIFO Overflow”

If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full).

1 = MB7 completed transmission/reception or FIFO overflow
0 = No such occurrence

BUF6I — Buffer MB6 Interrupt or “FIFO Warning”

If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 5 out of 6 buffers of the FIFO are already occupied (FIFO almost full).

1 = MB6 completed transmission/reception or FIFO almost full
0 = No such occurrence

BUF5I — Buffer MB5 Interrupt or “Frames available in FIFO”

If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO.

1 = MB5 completed transmission/reception or frames available in the FIFO
0 = No such occurrence

BUF4I–BUF0I — Buffer MB_i Interrupt or “reserved”

If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations.

1 = Corresponding MB completed transmission/reception
0 = No such occurrence

25.3.4.11 Rx Individual Mask Registers (RXIMR0–RXIMR31)

These registers are used as acceptance masks for ID filtering in Rx MBs and the FIFO. If the FIFO is not enabled, one mask register is provided for each available Message Buffer, providing ID masking capability on a per Message Buffer basis. When the FIFO is enabled (FEN bit in MCR is set), the first 8 Mask Registers apply to the 8 elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze Mode. Out of Freeze Mode, write accesses are blocked and read accesses will return “all zeros”. Furthermore, if the BCC bit in the MCR Register is negated, any read or write operation to these registers results in access error.

Base + 0x0880–0x08FF

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0

Figure 359. Rx Individual Mask Registers (RXIMR0 - RXIMR31)

MI31–MI0 — Mask Bits

For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR).

1 = The corresponding bit in the filter is checked against the one received
0 = the corresponding bit in the filter is “don’t care”

25.4 Functional Description

25.4.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of up to 64 Message Buffers (MB) that store configuration and control data, time stamp, message ID and data (see *Section 25.3.2, Message Buffer Structure*). The memory corresponding to the first 8 MBs can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (up to 8 extended IDs or 16 standard IDs or 32 8-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB ordering.

Before proceeding with the functional description, an important concept must be explained. A Message Buffer is said to be “active” at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a ‘0000’ code is inactive (refer to *Table 317*). Similarly, a Tx MB with a ‘1000’ or ‘1001’ code is also inactive (refer to *Table 318*). 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 25.4.6.2, Message Buffer Deactivation*).

25.4.2 Transmit Process

In order to transmit a CAN frame, the CPU must prepare a Message Buffer for transmission by executing the following procedure:

- If the MB is active (transmission pending), write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see *Section 25.4.6.1, Transmission Abort Mechanism*). If backwards compatibility is desired (AEN in MCR negated), just write ‘1000’ to the Code field to inactivate the MB but then the pending frame may be transmitted without notification (see *Section 25.4.6.2, Message Buffer Deactivation*).
- Write the ID word.
- Write the data bytes.
- Write the Length, Control and Code fields of the Control and Status word to activate the MB.

Once the MB is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit. The new Code field after transmission depends on the code that was used to activate the MB in step four (see *Table 317* and *Table 318* in *Section 25.3.2, Message Buffer Structure*). When the Abort feature is enabled (AEN in MCR is asserted), after the Interrupt Flag is asserted for an MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to update it until the Interrupt Flag be negated by CPU. It means that the CPU must clear the corresponding IFLAG before starting to prepare this MB for a new transmission or reception.

25.4.3 Arbitration process

The arbitration process is an algorithm executed by the MBM that scans the whole MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers will be scanned to find the lowest ID[k] or the lowest MB number or the

k. 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.

highest priority, depending on the LBUF and LPRIO_EN bits on the Control Register. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During Intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- When MBM is in Idle or Bus Off state and the CPU writes to the C/S word of any MB
- Upon leaving Freeze Mode

When LBUF is asserted, the LPRIO_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO_EN are both negated, the MB with the lowest ID is transmitted first but. If LBUF is negated and LPRIO_EN is asserted, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define which MB should be transmitted first, therefore MBs with PRIO = 000 have higher priority. If two or more MBs have the same priority, the regular ID will determine the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB will be transmitted first.

Once the highest priority MB is selected, it is transferred to a temporary storage space called Serial Message Buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called “move-out” and after it is done, write access to the corresponding MB is blocked (if the AEN bit in MCR is asserted). The write access is released in the following events:

- After the MB is transmitted
- FlexCAN enters in HALT or BUS OFF
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits up to eight data bytes, even if the DLC (Data Length Code) value is bigger.

25.4.4 Receive Process

To be able to receive CAN frames into the mailbox MBs, the CPU must prepare one or more Message Buffers for reception by executing the following steps:

- If the MB has a pending transmission, write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see *Section 25.4.6.1, Transmission Abort Mechanism*). If backwards compatibility is desired (AEN in MCR negated), just write ‘1000’ to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification (see *Section 25.4.6.2, Message Buffer Deactivation*). If the MB already programmed as a receiver, just write ‘0000’ to the Code field of the Control and Status word to keep the MB inactive.
- Write the ID word
- Write ‘0100’ to the Code field of the Control and Status word to activate the MB

Once the MB is activated in the third step, it will be able to receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

- The value of the Free Running Timer is written into the Time Stamp field
- The received ID, Data (8 bytes at most) and Length fields are stored
- The Code field in the Control and Status word is updated (see *Table 317* and *Table 318* in *Section 25.3.2, Message Buffer Structure*)
- A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

- Read the Control and Status word (mandatory – activates an internal lock for this buffer)
- Read the ID field (optional – needed only if a mask was used)
- Read the Data field
- Read the Free Running Timer (optional – releases the internal lock)

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the MB until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see *Section 25.4.6, Data Coherence*).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the IFLAG Registers and not by the Code field of that MB. Polling the Code field does not work because once a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the Code field will not return to EMPTY. It will remain FULL, as explained in *Table 317*. 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 25.4.7, Rx FIFO*). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

- Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)
- Read the ID field (optional – needed only if a mask was used)
- Read the Data field
- Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

25.4.5 Matching Process

The matching process is an algorithm executed by the MBM that scans the MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first and then, if a match is not found within the FIFO table, the other MBs are scanned. In the event that the FIFO is full, the matching algorithm will always look for a matching MB outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary MB called Serial Message Buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular MBs, the contents of the SMB will be transferred to the FIFO or to the matched MB during the 6th bit of the End-Of-Frame field of the CAN protocol. This operation is called “move-in”. If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

For the regular mailbox MBs, an MB is said to be “free to receive” a new frame if the following conditions are satisfied:

- The MB is not locked (see *Section 25.4.6.3, Message Buffer Lock Mechanism*)
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not “free to receive” the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it can not find one that is free, then it will overwrite the last matching MB (unless it is locked) and set the Code field to OVERRUN (refer to *Table 317* and *Table 318*). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see *Section 25.4.6.3, Message Buffer Lock Mechanism*).

Suppose, for example, that the FIFO is disabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm will find the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm will find MB number 2 again, but it is not “free to receive”, so it will keep looking and find MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are “free to receive”, so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the Code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the MBs. By programming more than one MB with the same ID, received messages will be queued into the MBs. The CPU can examine the Time Stamp field of the MBs to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the BCC bit in MCR is negated, the matching algorithm stops at the first MB with a matching ID that it founds, whether this MB is free or not. As a result, the message queueing feature does not work if the BCC bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. Please refer to *Section 25.3.4.11, Rx Individual Mask Registers (RXIMR0–RXIMR31).* During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is

"don't care". Please note that the Individual Mask Registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the BCC bit is asserted and while the module is in Freeze Mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, RX14MASK and RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the BCC bit in the MCR Register is negated.

25.4.6 Data Coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in *Section 25.4.2, Transmit Process* and *Section 25.4.4, Receive Process*. Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

25.4.6.1 Transmission Abort Mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. In order to maintain backwards compatibility, the abort mechanism must be explicitly enabled by asserting the AEN bit in the MCR.

In order to abort a transmission, the CPU must write a specific abort code (1001) to the Code field of the Control and Status word. When the abort mechanism is enabled, the active MBs configured as trasmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that was already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into Freeze Mode

If none of conditions above are reached, the MB is transmitted correctly, the interrupt flag is set in the IFLAG register and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. In the other hand, if one of the above conditions is reached, the frame is not transmitted, therefore the abort code is written into the Code field, the interrupt flag is set in the IFLAG and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked, therefore the MB is updated and no interrupt flag is set. In this way the CPU just needs to read the abort code to make sure the active MB was deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the MB is deactivated and not aborted, because the transmission did not start yet. One MB is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

- CPU writes 1001 into the code field of the C/S word
- CPU reads the CODE field and compares it to the value that was written
- If the CODE field that was read is different from the value that was written, the CPU must read the corresponding IFLAG to check if the frame was transmitted or it is being

currently transmitted. If the corresponding IFLAG is set, the frame was transmitted. If the corresponding IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the MB was aborted (CODE=1001) or it was transmitted (CODE=1000).

Note: An abort request to a TxMB can block any write operation to its CODE field. Therefore, the TxMB cannot be aborted or deactivated until it completes a transmission by winning the CAN bus arbitration.

25.4.6.2 Message Buffer Deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the Control and Status word of active MBs out of Freeze Mode. Any CPU write access to the Control and Status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active MBs when not in Freeze Mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was “free to receive”.
- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the Code field is not updated. In order to avoid this situation, the abort procedures described in *Section 25.4.6.1, Transmission Abort Mechanism* should be used.

25.4.6.3 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[I] ('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.

25.4.7 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the MCR. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80-0xFF) is now reserved for use of the FIFO engine (see *Section 25.3.3, Rx FIFO Structure*). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a Message Buffer structure at the beginning of the memory.

The FIFO can store up to six frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80 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

I. 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.

following formats (see also *Section 25.3.3, Rx FIFO Structure*):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

Note: A chosen format is applied to all 8 registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight Individual Mask Registers (RXIMR0 - RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIM8, continue to affect the regular MBs, starting from MB8. If the BCC bit is negated (or if the RXIMR are not available for the particular MCU), then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by RX14MASK, element 7 is affected by RX15MASK and the other elements (0 to 5) are affected by RXGMASK.

25.4.7.1 Precautions When Using Global Mask and Individual Mask Registers

Mask filtering alignment is affected based on the setting of the FEN and BCC of MCR. The following table shows recommended actions depending on FEN and BCC settings.

Table 321. Recommended FEN and BCC settings

Case	MCR[FEN] RxFIFO	MCR[BCC] RxIndividual Mask	Notes
Case1	FEN=0	BCC=0	RXGMASK, RX14MASK and RX15MASK can safely be used.This allows backwards compatibility to older devices (e.g. devices without the individual masks feature). In this case, individual masks are not used.
Case 2	FEN=1	BCC=0	1st alternative: Don't use RXGMASK, RX14MASK and RX15MASK in this case, leave the masks in their reset state.
Case 3	FEN=1	BCC=0	2nd alternative: Do not configure any MB as Rx (i.e. let all MBs as either Tx or inactive). In this case, the masks RXGMASK, RX14MASK and RX15MASK can be used to affect ID Tables without affecting the filtering process for Rx MBs
Case 4	Don’t Care	BCC=1	If MCR[BCC] = 1, then the RXIMRs are enabled an thus the masks RXGMASK, RX14MASK and RX15MASK are not used.Particularly, when MCR[FEN] = 0, RxFifo is disabled; RXGMASK, RX14MASK and RX15MASK don't affect filtering. Individual masks used.

25.4.8 CAN Protocol Related Features

25.4.8.1 Remote Frames

Remote frame is a special kind of frame. The user can program an 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).

25.4.8.2 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

25.4.8.3 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 25.3.4.2, Control Register (CTRL).*

25.4.8.4 Protocol Timing

Figure 360 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 FMPLL). In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (bit MDIS set in the Module Configuration Register).

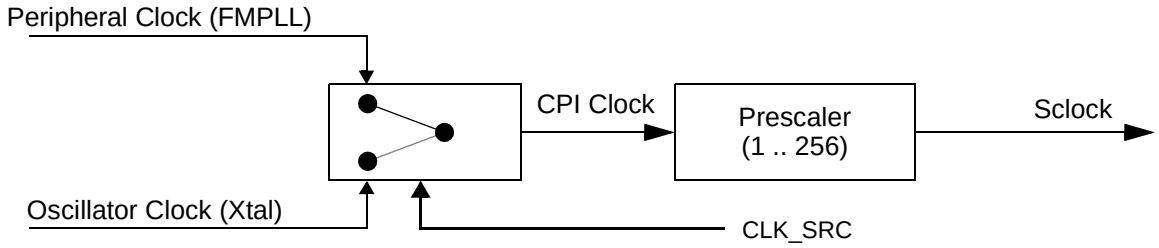


Figure 360. CAN Engine Clocking Scheme

The crystal oscillator clock should be selected whenever a tight tolerance (up to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than FMPLL generated clocks.

Note: This clock selection feature may not be available in all MCUs. A particular MCU may not have a FMPLL, in which case it would have only the oscillator clock, or it may use only the FMPLL clock feeding the FlexCAN module. In these cases, the CLK_SRC bit in the CTRL Register has no effect on the module operation.

In order to guarantee reliable operation, the selected CAN Protocol Interface (CPI) clock should not be faster as the the peripheral clock.

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 25.3.4.2, 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[(m)] (reference *Figure 361* and *Table 322*):

- 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

m. 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

$$Bit\ Rate = \frac{f_{Tq}}{(number\ of\ Time\ Quanta)}$$

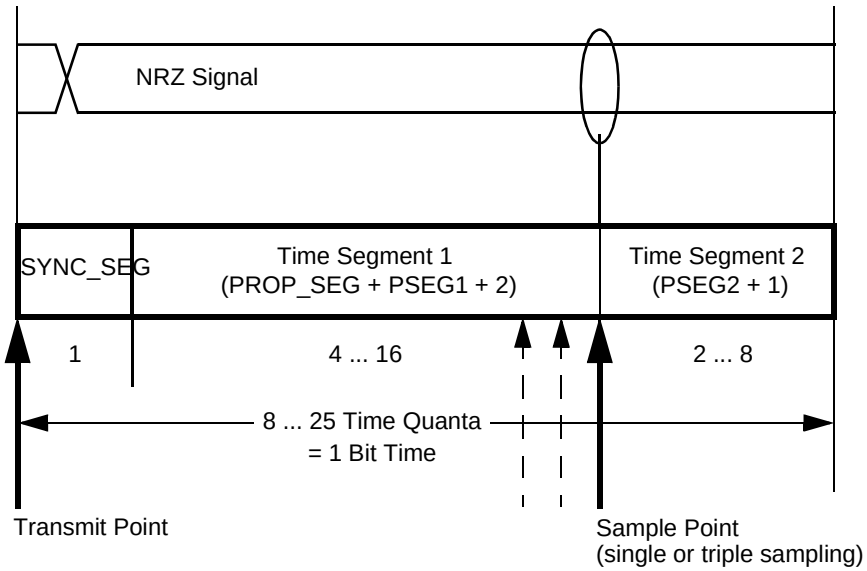


Figure 361. Segments within the Bit Time

Table 322. 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 323 gives an overview of the CAN compliant segment settings and the related parameter values.

Table 323. CAN 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
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

Note: It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module. The Information Processing Time (IPT) is the time required for the logic to determine the bit level of a sampled bit. The IPT begins at the sample point, is measured in TQ and is fixed at 2TQ for the FlexCAN module. The length of the time quantum (TQ), which is the basic time unit of the bit time, is defined by the CAN controller's system clock fsys and the Baud Rate Prescaler (BRP): tq = BRP/fsys. Typical system clocks are: fsys = fosc or fsys = fosc/2.

25.4.8.5 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 362*.

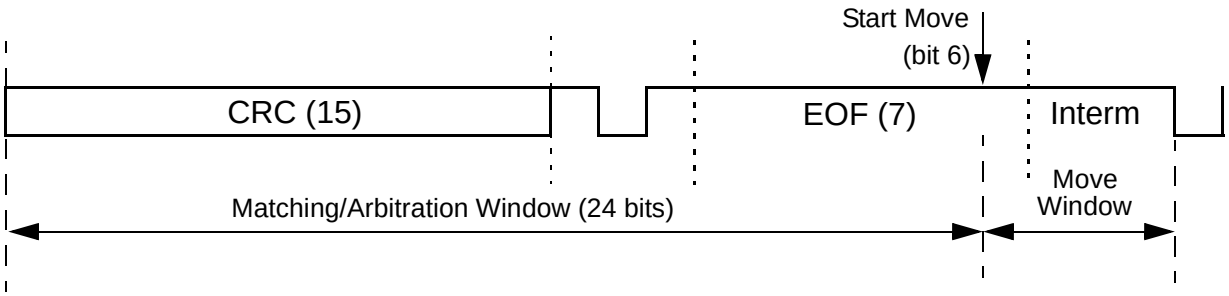


Figure 362. Arbitration, Match and Move Time Windows

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 323*
- The peripheral clock frequency can not be smaller than the oscillator clock frequency, i.e. the FMPLL 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 324*

Table 324. 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 324* 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.

25.4.9 Modes of Operation Details

25.4.9.1 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 any of the low power modes (Disable, Stop). 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 re-synchronize to the CAN bus by waiting for 11 consecutive recessive bits.

25.4.9.2 Module Disable Mode

This low power mode is entered when the MDIS bit in the MCR Register is asserted. If the module is disabled during Freeze Mode, it shuts down the clocks to the CPI and MBM sub-

modules, sets the LPM_ACK bit and negates the FRZ_ACK bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT_RDY and LPM_ACK bits in MCR

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Free Running Timer, the Error Counter Register and the Message Buffers, which cannot be accessed when the module is in Disable Mode. Exiting from this mode is done by negating the MDIS bit, which will resume the clocks and negate the LPM_ACK bit.

25.4.9.3 Stop Mode

This is a system low power mode in which all MCU clocks are stopped for maximum power savings. If FlexCAN receives the global Stop Mode request during Freeze Mode, it sets the LPM_ACK bit, negates the FRZ_ACK bit and then sends a Stop Acknowledge signal to the CPU, in order to shut down the clocks globally. If Stop Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Sets the NOT_RDY and LPM_ACK bits in MCR
- Sends a Stop Acknowledge signal to the CPU, so that it can shut down the clocks globally

Exiting Stop Mode is done in one of the following ways:

- CPU resuming the clocks and removing the Stop Mode request

25.4.10 Interrupts

The module can generate up to 70 interrupt sources (64 interrupts due to message buffers and 5 interrupts due to Ored interrupts from MBs, Bus Off, Error, Tx Warning, and Rx Warning). The number of actual sources depends on the configured number of Message Buffers.

Each one of the message buffers can be an interrupt source, if its corresponding IMASK bit is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the IFLAG Registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to ‘1’ (unless another interrupt is generated at the same time).

Note: It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags which are set after entering the current interrupt service routine.

If the Rx FIFO is enabled (bit FEN on MCR set), the interrupts corresponding to MBs 0 to 7 have a different behavior. Bit 7 of the IFLAG1 becomes the “FIFO Overflow” flag; bit 6 becomes the FIFO Warning flag, bit 5 becomes the “Frames Available in FIFO flag” and bits 4-0 are unused. See *Section 25.3.4.10, 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 interrupt sources (Bus Off, Error, Tx Warning, and Rx Warning) generate interrupts like the MB ones, and can be read from the Error and Status Register. The Bus Off, Error, Tx Warning and Rx Warning interrupt mask bits are located in the Control Register.

25.4.11 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.

25.5 Initialization/Application Information

This section provide instructions for initializing the FlexCAN module.

25.5.1 FlexCAN Initialization Sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- MCU level soft reset, which resets some of the memory mapped registers synchronously (refer to *Table 315* 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 25.4.9.1, 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 abort mechanism by setting the AEN bit
  - Enable the local priority feature by setting the LPRIO_EN bit
- Initialize the Control Register
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
  - Determine the bit rate by programming the PRESDIV field
  - Determine the internal arbitration mode (LBUF bit)
- Initialize the Message Buffers
  - The Control and Status word of all Message Buffers must be initialized
  - If FIFO was enabled, the 8-entry ID table must be initialized
  - Other entries in each Message Buffer should be initialized as required
- Initialize the Rx Individual Mask Registers
- Set required interrupt mask bits in the IMASK Registers (for all MB interrupts) and in CTRL Register (for Bus Off and Error interrupts)
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

25.5.2 FlexCAN addressing and RAM size configurations

The RAM configuration is as follows:

- 544 bytes for MB memory
- 128 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. MAXMB can be any number between 0–31.

26 Flexible Motor Control Pulse Width Modulator Module (FlexPWM)

26.1 Introduction

26.1.1 Overview

The pulse width modulator module (FlexPWM) contains 4 PWM submodules each of which is set up to control a single half-bridge power stage. There are 4 fault channels.

This PWM is capable of controlling most motor types: AC induction motors (ACIM), Permanent Magnet AC motors (PMAC), both brushless (BLDC) and brush DC motors (BDC), switched (SRM) and variable reluctance motors (VRM), and stepper motors.

This device contains two FlexPWM modules. The modules interact with the CTU as described in *Section 14.4.1, Interaction with other peripherals*.

26.1.2 Features

- 16 bits of resolution for center, edge aligned, and asymmetrical PWMs
- PWM outputs can operate as complimentary pairs or independent channels
- Can accept signed numbers for PWM generation
- Independent control of both edges of each PWM output
- Synchronization to external hardware or other PWM supported
- Double buffered PWM registers
  - Integral reload rates from 1 to 16
  - Half cycle reload capability
- Multiple output trigger events can be generated per PWM cycle via hardware
- Support for double switching PWM outputs
- Fault inputs can be assigned to control multiple PWM outputs
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom deadtime insertion
- Each complementary pair can operate with its own PWM frequency and deadtime values
- Individual software-control for each PWM output
- All outputs can be programmed to change simultaneously via a "Force Out" event
- PWMX pin can optionally output a third PWM signal from each submodule
- Channels not used for PWM generation can be used for buffered output compare functions
- Channels not used for PWM generation can be used for input capture functions
- Enhanced dual edge capture functionality
- The option to supply the source for each complementary PWM signal pair from any of the following:
  - external digital pin
  - internal timer channel
  - external ADC input, taking into account values set in ADC high and low limit registers.

26.1.3 Modes of operation

Care must be exercised when using this module in certain chip operating modes. Some motors (such 3-phase AC motors) require regular software updates for proper operation. Failure to do so could result in destroying the motor or inverter. Because of this, PWM outputs are placed in their inactive states in STOP mode, and optionally under WAIT and debug modes. PWM outputs will be reactivated (assuming they were active to begin with) when these modes are exited.

Table 325. Modes when PWM operation is restricted

Mode	Description
STOP	Peripheral and CPU clocks are stopped. PWM outputs are driven inactive.
WAIT	CPU clocks are stopped while peripheral clocks continue to run. PWM outputs are driven inactive as a function of the WAITEN bit.
DEBUG	CPU and peripheral clocks continue to run, but CPU maybe stalled for periods of time. PWM outputs are driven inactive as a function of the DBGEN bit.

26.1.4 Block diagrams

26.1.4.1 Module level

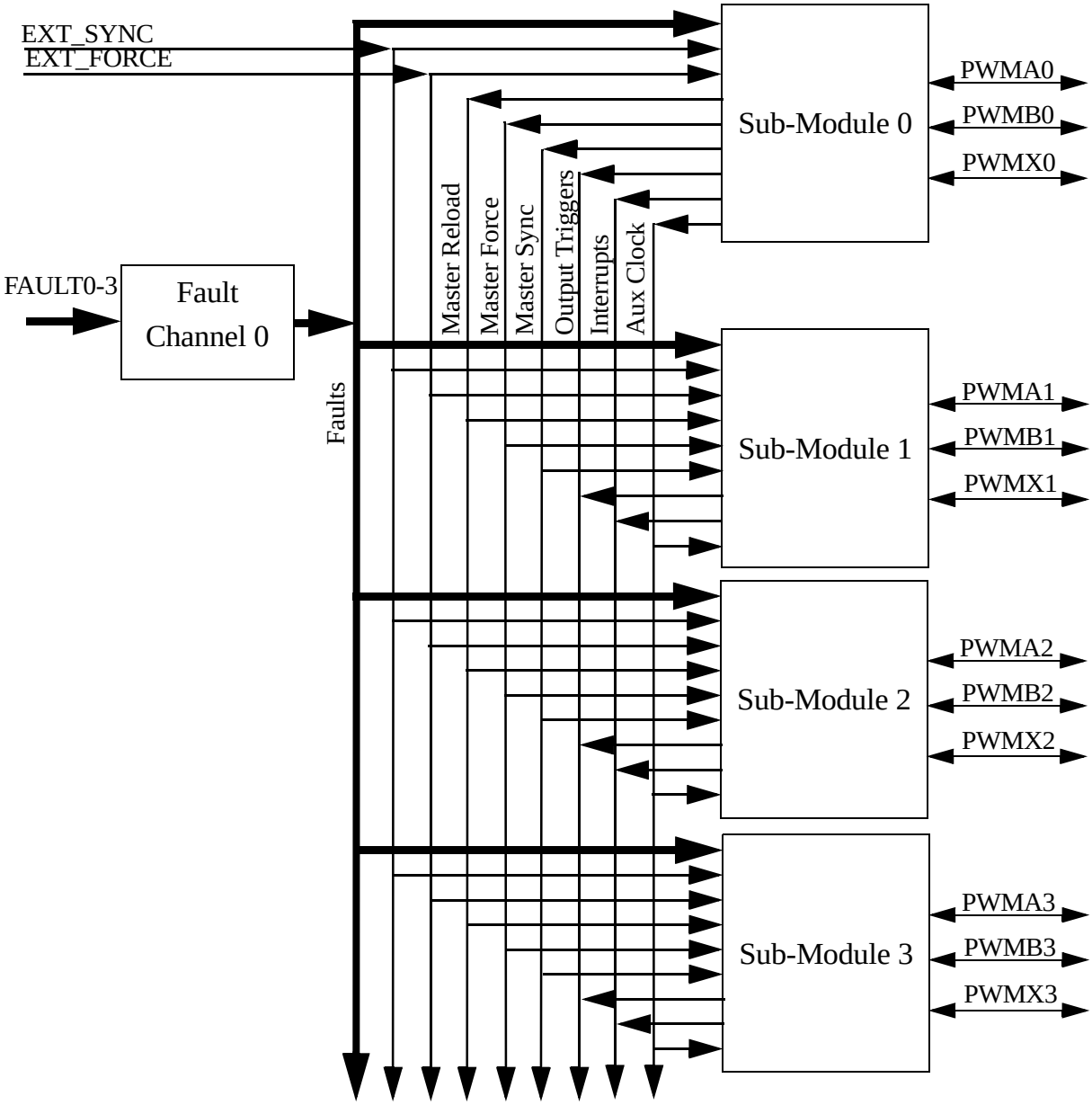


Figure 363. PWM block diagram

26.1.4.2 PWM Submodule

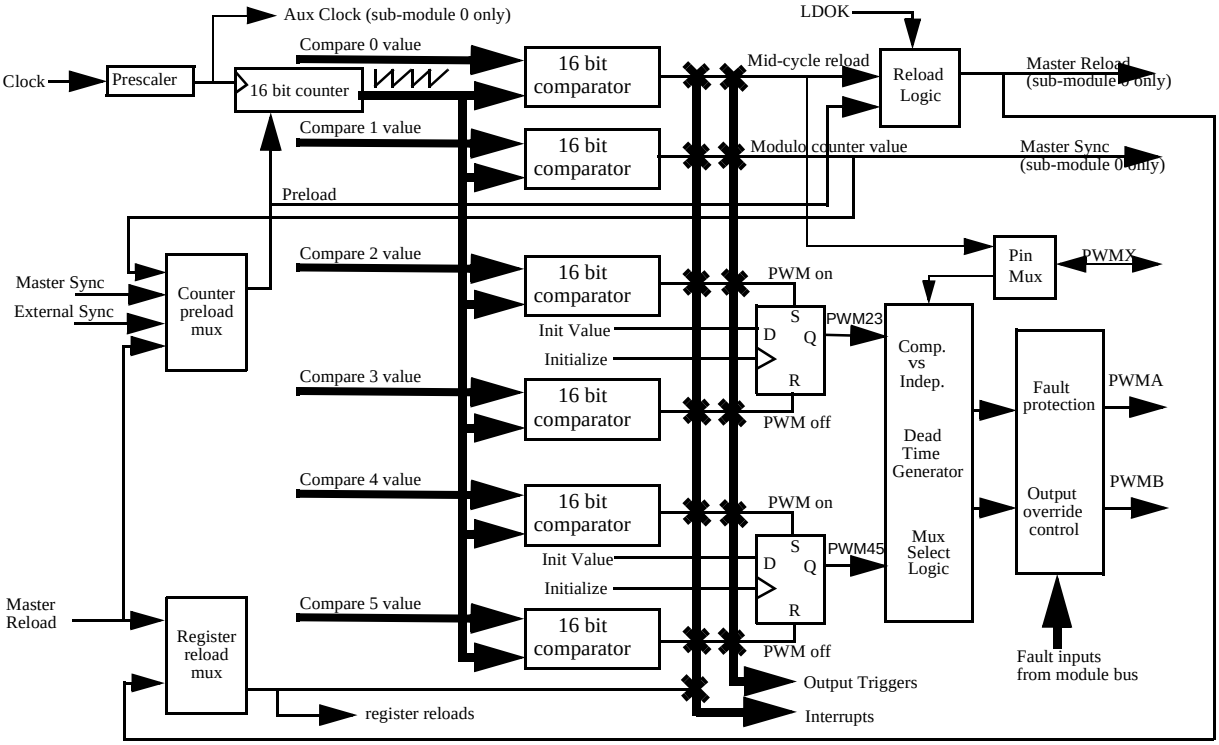


Figure 364. PWM Submodule Block Diagram

26.2 External signal descriptions

The PWM module has external pins named PWMA[n], PWMB[n], PWMX[n], FAULT[n], EXT_SYNC, EXT_FORCE, EXTA[n], and EXTB[n]. The PWM module also has an on chip input called EXT_CLK and an on-chip output called OUT_TRIG[n]. The number of these signals can vary depending on PWM configuration for each chip. (See *Figure 98*.)

26.2.1 PWMA[n] and PWMB[n] — External PWM Pair

These pins are the output pins of the PWM channels. They can be independent PWM signals or a complementary pair.

26.2.2 PWMX[n] — Auxiliary PWM signal

These pins are the auxiliary output pins of the PWM channels. They can be independent PWM signals. When not needed as an output, they can be used as inputs to the input capture circuitry or they can be used to generate the IPOL bit during deadtime correction.

26.2.3 FAULT[n] — Fault inputs

These are input pins for disabling selected PWM outputs.

26.2.4 EXT_SYNC — External Synchronization Signal

This input signal allows a source external to the PWM to initialize the PWM counter. In this manner the PWM can be synchronized to external circuitry.

26.2.5 EXT_FORCE — External Output Force Signal

This input signal allows a source external to the PWM to force an update of the PWM outputs. In this manner the PWM can be synchronized to external circuitry. An example would be to simultaneously switch all of the PWM outputs on a commutation boundary for trapezoidal control of a BLDC motor. The boundary can be established via external logic or an on-chip timer.

26.2.6 EXTA[n] and EXTB[n] — Alternate PWM Control Signals

These pins allow an alternate source to control the PWMA and PWMB outputs although typically, the EXTA input will be used for the generation of a complementary pair. Typical connections include ADC results registers, TMR outputs, GPIO inputs, and comparator outputs. The SIM chapter of the chip system spec will describe the possible connections and the bits to control any muxing.

26.2.7 OUT_TRIG0[n] and OUT_TRIG1[n] — Output Triggers

These outputs allow the PWM submodules to control timing of the ADC conversions. See *26.4.3.15, "Output Trigger Control Register (TCTRL)"* for a description of how to enable these outputs and how the compare registers match up to the output triggers.

26.2.8 EXT_CLK — External Clock Signal

This on-chip input signal allows an on-chip source external to the PWM (typically a Timer) to control the PWM clocking. In this manner the PWM can be synchronized to the Timer. This signal must be generated synchronously to the PWM's clock since it is not resynchronized in the PWM.

26.3 Functional Description

26.3.1 Block Diagram

A block diagram of the PWM is shown in *Figure 363*.

26.3.2 PWM Capabilities

This section describes some capabilities of the PWM module.

26.3.2.1 Center-Aligned PWMs

Each submodule has its own timer that is capable of generating PWM signals on two output pins. The edges of each of these signals are controlled independently as shown in *Figure 365*.

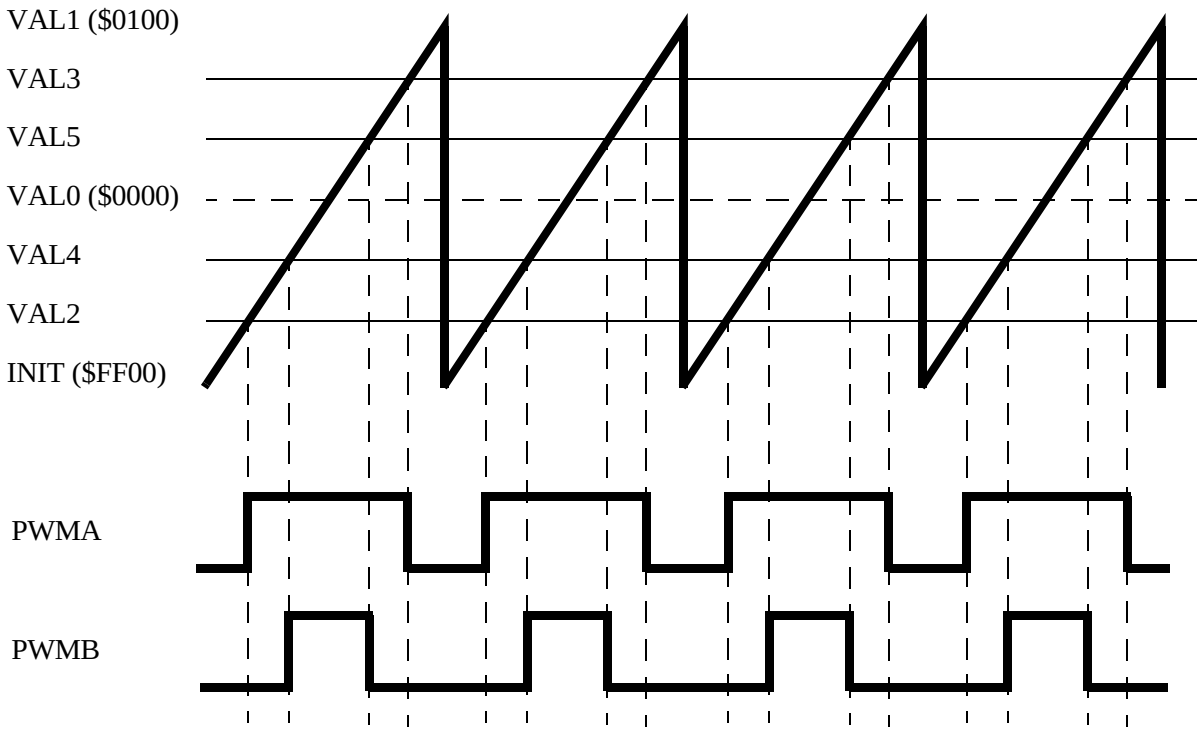


Figure 365. Center Aligned Example

The submodule timers only count in the up direction and then reset to the INIT value. Instead of having a single value that determines pulse width, there are two values that must be specified: the turn on edge and the turn off edge. This double action edge generation not only gives the user control over the pulse width, but over the relative alignment of the signal as well. As a result, there is no need to support separate PWM alignment modes since the PWM alignment mode is inherently a function of the turn on and turn off edge values.

Figure 365 also illustrates an additional enhancement to the PWM generation process. When the counter resets, it is reloaded with a user specified value, which may or may not be zero. If the value chosen happens to be the 2's complement of the modulus value, then the PWM generator operates in "signed" mode. This means that if each PWM's turn on and turn off edge values are also the same number but only different in their sign, the "on" portion of the output signal will be centered around a count value of zero. Therefore, only one PWM value needs to be calculated in software and then this value and its negative are provided to the submodule as the turn off and turn on edges respectively. This technique will result in a pulse width that always consists of an odd number of timer counts. If all PWM signal edge calculations follow this same convention, then the signals will be center aligned with respect to each other, which is the goal. Of course, center alignment between the signals is not restricted to symmetry around the zero count value, as any other number would also work. However, centering on zero provides the greatest range in signed mode and also simplifies the calculations.

26.3.2.2 Edge-Aligned PWMs

When the turn on edge for each pulse is specified to be the INIT value, then edge aligned operation results, as illustrated in *Figure 366*. Therefore, only the turn off edge value needs to be periodically updated to change the pulse width.

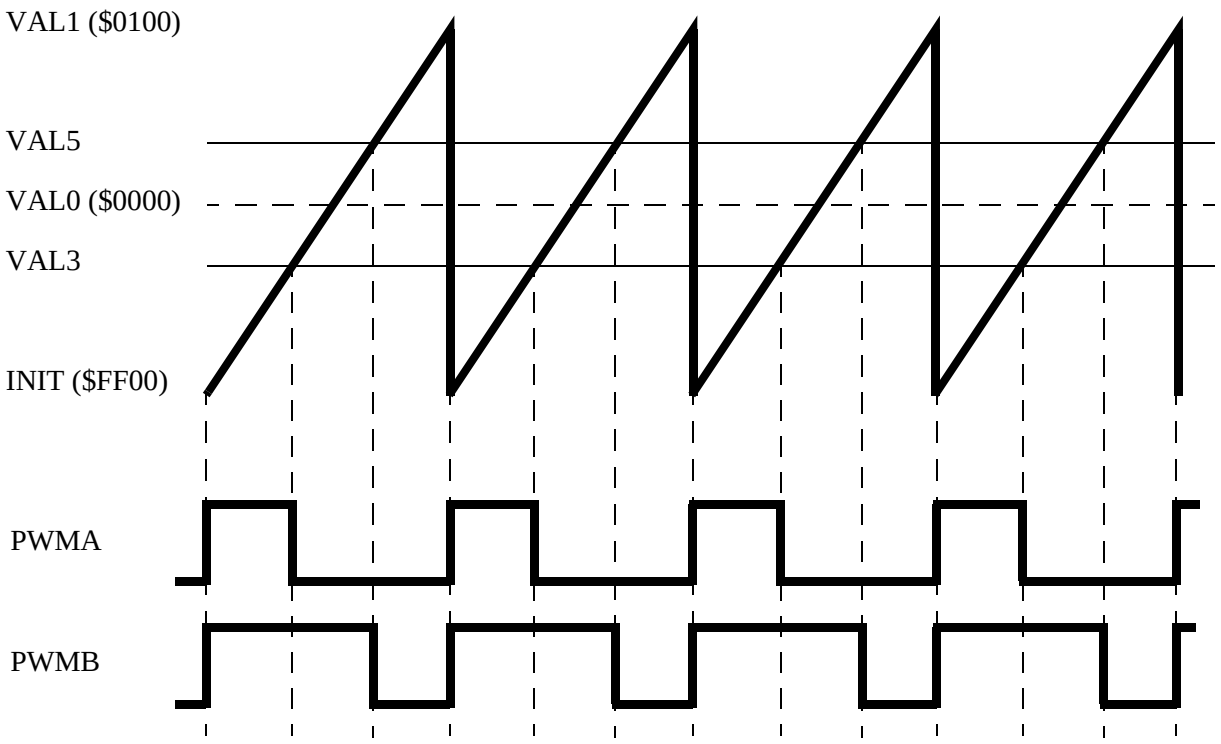


Figure 366. Edge Aligned Example (INIT=VAL2=VAL4)

With edge aligned PWMs, another example of the benefits of signed mode can be seen. A common way to drive an H-bridge is to use a technique called "bipolar" PWMs where a 50% duty cycle results in zero volts on the load. Duty cycles less than 50% result in negative load voltages and duty cycles greater than 50% generate positive load voltages. If the module is set to signed mode operation (the INIT and VAL1 values are the same number with opposite signs), then there is a direct proportionality between the PWM turn off edge value and the motor voltage, INCLUDING the sign. So once again, signed mode of operation simplifies the software interface to the PWM module since no offset calculations are required to translate the output variable control algorithm to the voltage on an H-Bridge load.

26.3.2.3 Phase-Shifted PWMs

In the previous sections, the benefits of signed mode of operation were discussed in the context of simplifying the required software calculations by eliminating the requirement to bias up signed variables before applying them to the module. However, if numerical biases ARE applied to the turn on and turn off edges of different PWM signal, the signals will be phase shifted with respect to each other, as illustrated in *Figure 367*. This results in certain advantages when applied to a power stage. For example, when operating a multi-phase inverter at a low modulation index, all of the PWM switching edges from the different phases

occur at nearly the same time. This can be troublesome from a noise standpoint, especially if ADC readings of the inverter must be scheduled near those times. Phase shifting the PWM signals can open up timing windows between the switching edges to allow a signal to be sampled by the ADC. However, phase shifting does NOT affect the duty cycle so average load voltage is not affected.

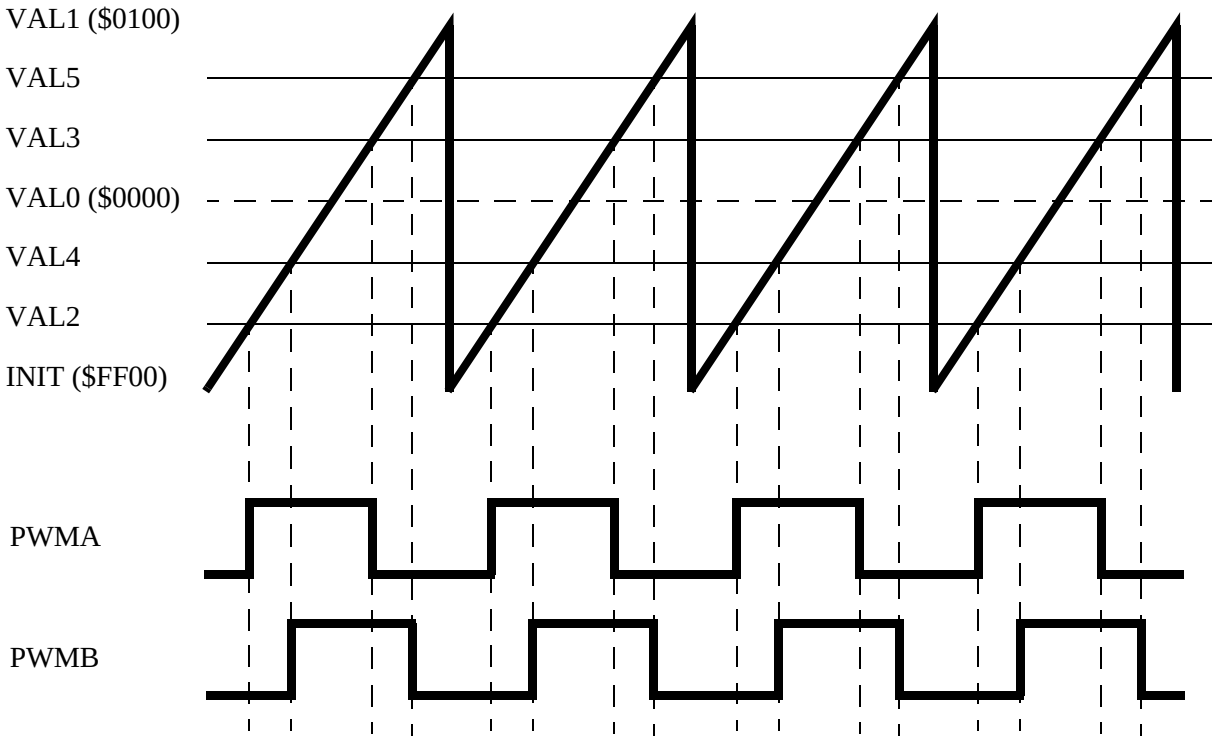


Figure 367. Phase Shifted Outputs Example

An additional benefit of phase shifted PWMs can be seen in *Figure 368*. In this case, an H-Bridge circuit is driven by 4 PWM signals to control the voltage waveform on the primary of a transformer. Both left and right side PWMs are configured to always generate a square wave with 50% duty cycle. This works out nicely for the H-Bridge since no narrow pulse widths are generated reducing the high frequency switching requirements of the transistors. Notice that the square wave on the right side of the H-Bridge is phase shifted compared to the left side of the H-Bridge. As a result, the transformer primary sees the bottom waveform across its terminals. The RMS value of this waveform is directly controlled by the amount of phase shift of the square waves. Regardless of the phase shift, no DC component appears in the load voltage as long as the duty cycle of each square wave remains at 50% making this technique ideally suited for transformer loads. As a result, this topology is frequently used in industrial welders to adjust the amount of energy delivered to the weld arc.

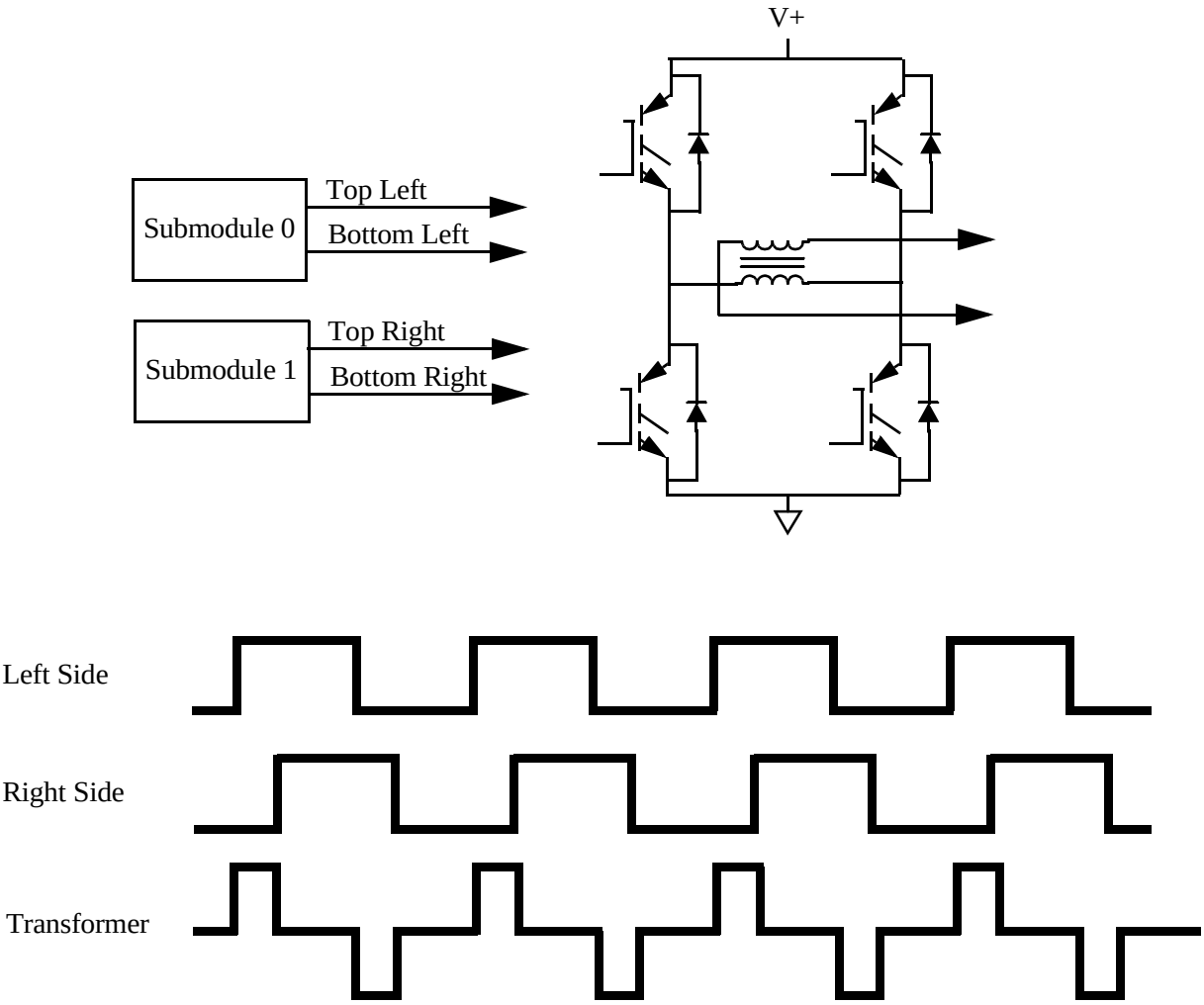


Figure 368. Phase Shifted PWMs Applied to a Transformer Primary

26.3.2.4 Double Switching PWMs

Double switching PWM output is supported to aid in single shunt current measurement and three phase reconstruction. This method support two independent rising edges and two independent falling edges per PWM cycle. The VAL2 and VAL3 registers are used to generate the even channel (labelled as PWMA in the figure) while VAL4 and VAL5 are used to generate the odd channel. The two channels are combined using XOR logic (see *Figure 379*) as shown in *Figure 369*. The DBLPWM signal can be run through the deadtime insertion logic.

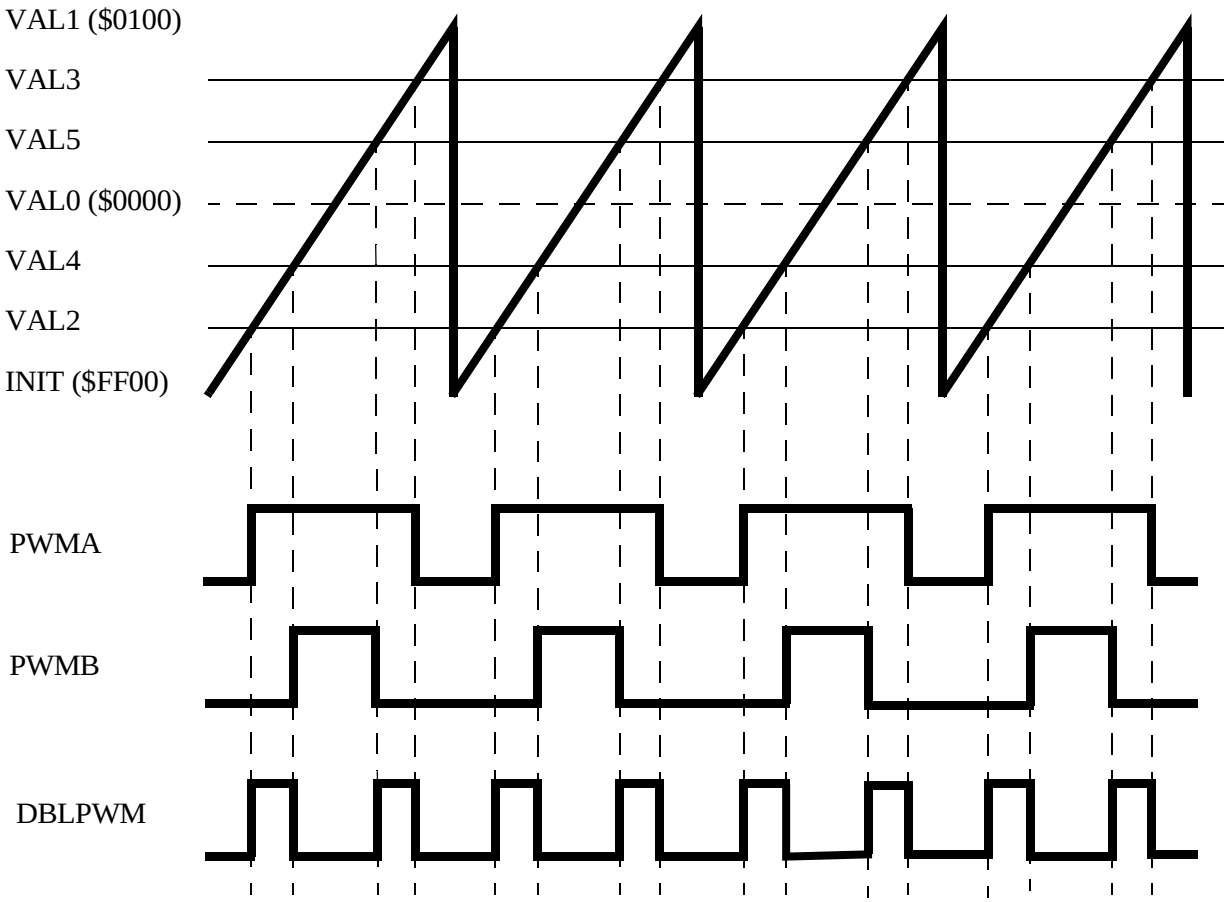


Figure 369. Double Switching Output Example

26.3.2.5 ADC Triggering

In cases where the timing of the ADC triggering is critical, it must be scheduled as a hardware event instead of software activated. With this PWM module, multiple ADC triggers can be generated in hardware per PWM cycle without the requirement of another timer module. *Figure 370* shows how this is accomplished. When specifying complimentary mode of operation, only two edge comparators are required to generate the output PWM signals for a given submodule. This means that the other comparators are free to perform other functions. In this example, the software doesn't have to quickly respond after the first conversion to set up other conversions that must occur in the same PWM cycle.

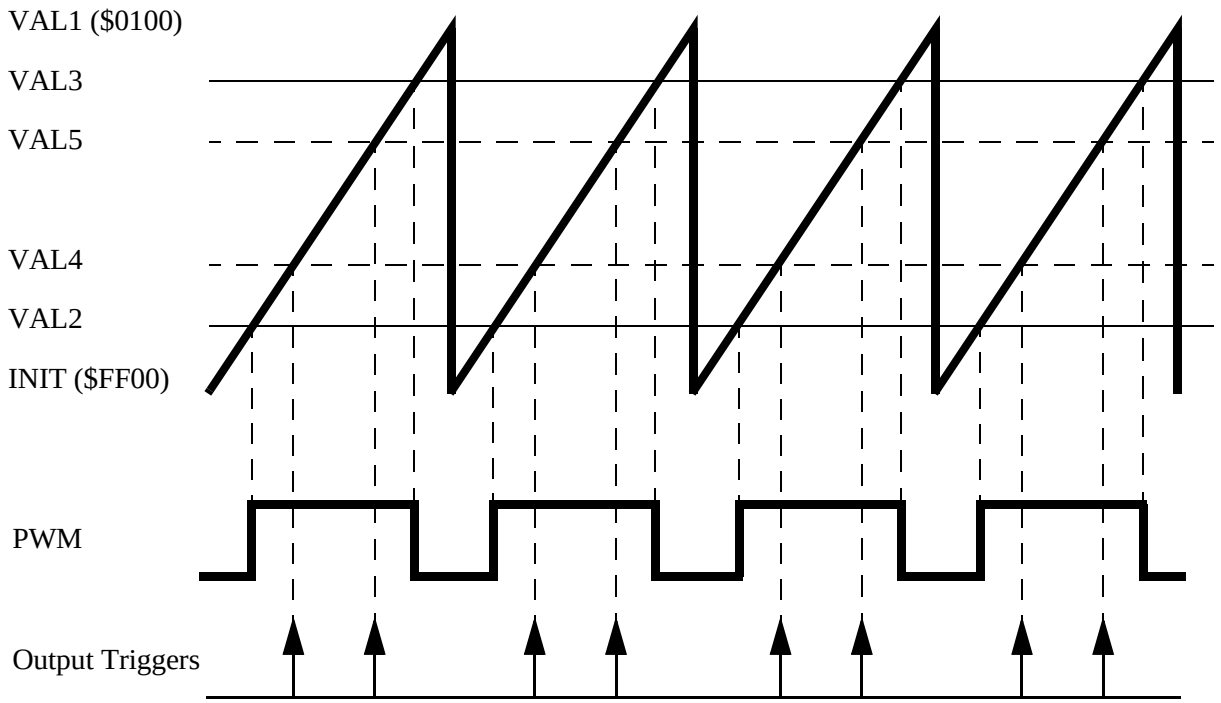


Figure 370. Multiple Output Trigger Generation in Hardware

Since each submodule has its own timer, it is possible for each submodule to run at a different frequency. One of the options possible with this PWM module is to have one or more submodules running at a lower frequency, but still synchronized to the timer in submodule0. *Figure 371* shows how this feature can be used to schedule ADC triggers over multiple PWM cycles. A suggested use for this configuration would be to use the lower frequency submodule to control the sampling frequency of the software control algorithm where multiple ADC triggers can now be scheduled over the entire sampling period. In *Figure 371*, ALL submodule comparators are shown being used for ADC trigger generation.

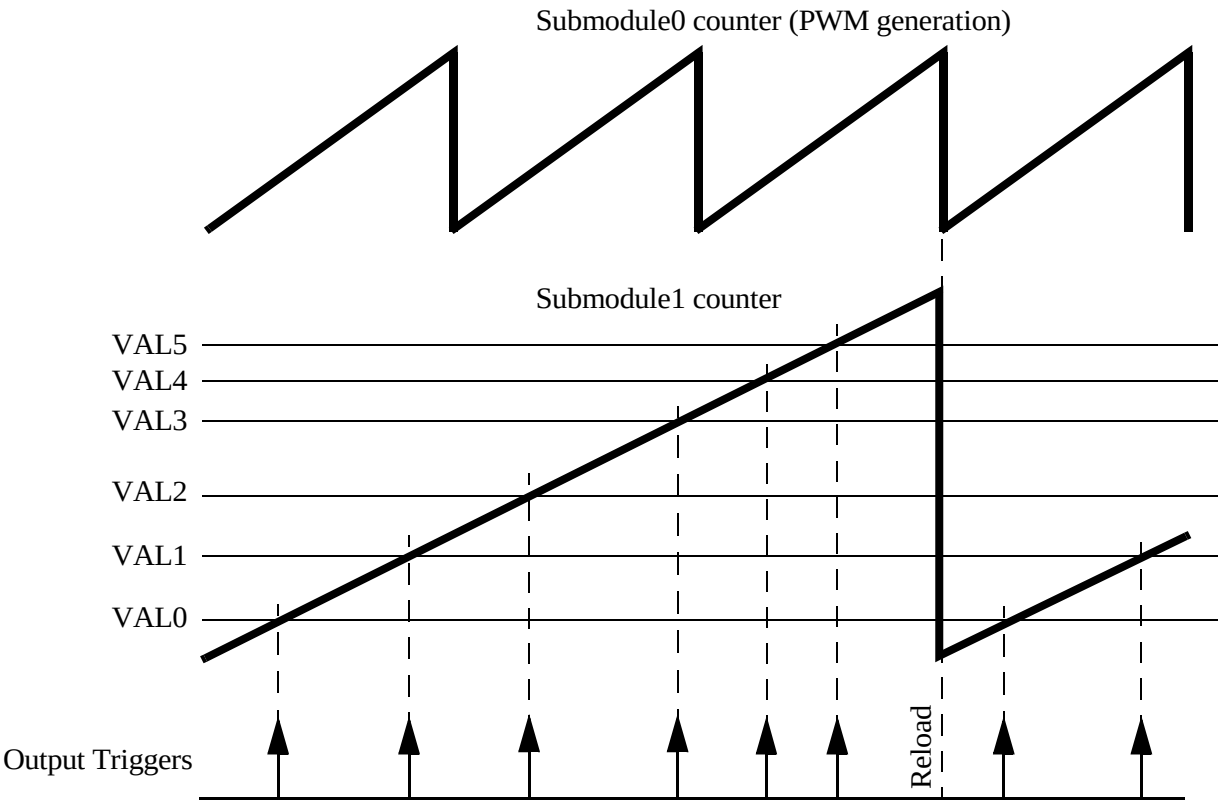


Figure 371. Multiple Output Triggers Over Several PWM Cycles

26.3.2.6 Enhanced Capture Capabilities (E-Capture)

When a PWM pin is not being used for PWM generation, it can be used to perform input captures. Recall that for PWM generation BOTH edges of the PWM signal are specified via separate compare register values. When programmed for input capture, both of these registers work on the same pin to capture multiple edges, toggling from one to the other in either a free running or one-shot fashion. By simply programming the desired edge of each capture circuit, period and pulse width of an input signal can easily be measured without the requirement to re-arm the circuit. In addition, each edge of the input signal can clock an 8 bit counter where the counter output is compared to a user specified value (EDGCMP). When the counter output equals EDGCMP, the value of the submodule timer is captured and the counter is automatically reset. This feature allows the module to count a specified number of edge events and then perform a capture and interrupt. *Figure 372* illustrates some of the functionality of the E-Capture circuit.

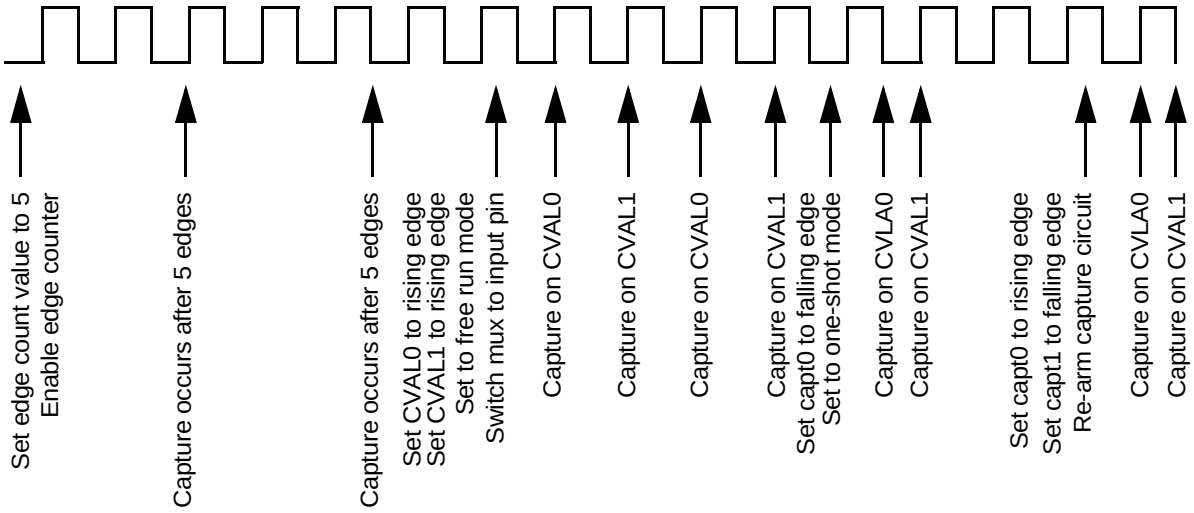


Figure 372. Capture Capabilities of the E-Capture Circuit

When a submodule is being used for PWM generation, its timer counts up to the modulus value used to specify the PWM frequency and then is re-initialized. Therefore, using this timer for input captures on one of the other pins (e.g, PWMX) has limited utility since it does not count through all of the numbers and the timer reset represents a discontinuity in the 16 bit number range. However, when measuring a signal that is synchronous to the PWM frequency, the timer modulus range is perfectly suited for the application. As an example, consider *Figure 373*. In this application the output of a PWM power stage is connected to the PWMX pin that is configured for free running input captures. Specifically, the CVAL0 capture circuitry is programmed for rising edges and the CVAL1 capture circuitry is set for falling edges. This will result in new load pulse width data being acquired every PWM cycle. To calculate the pulse width, simply subtract the CVAL0 register value from the CVAL1 register value. This measurement is extremely beneficial when performing dead-time distortion correction on a half bridge circuit driving an inductive load. Also, these values can be directly compared to the VALx registers responsible for generating the PWM outputs to obtain a measurement of system propagation delays.

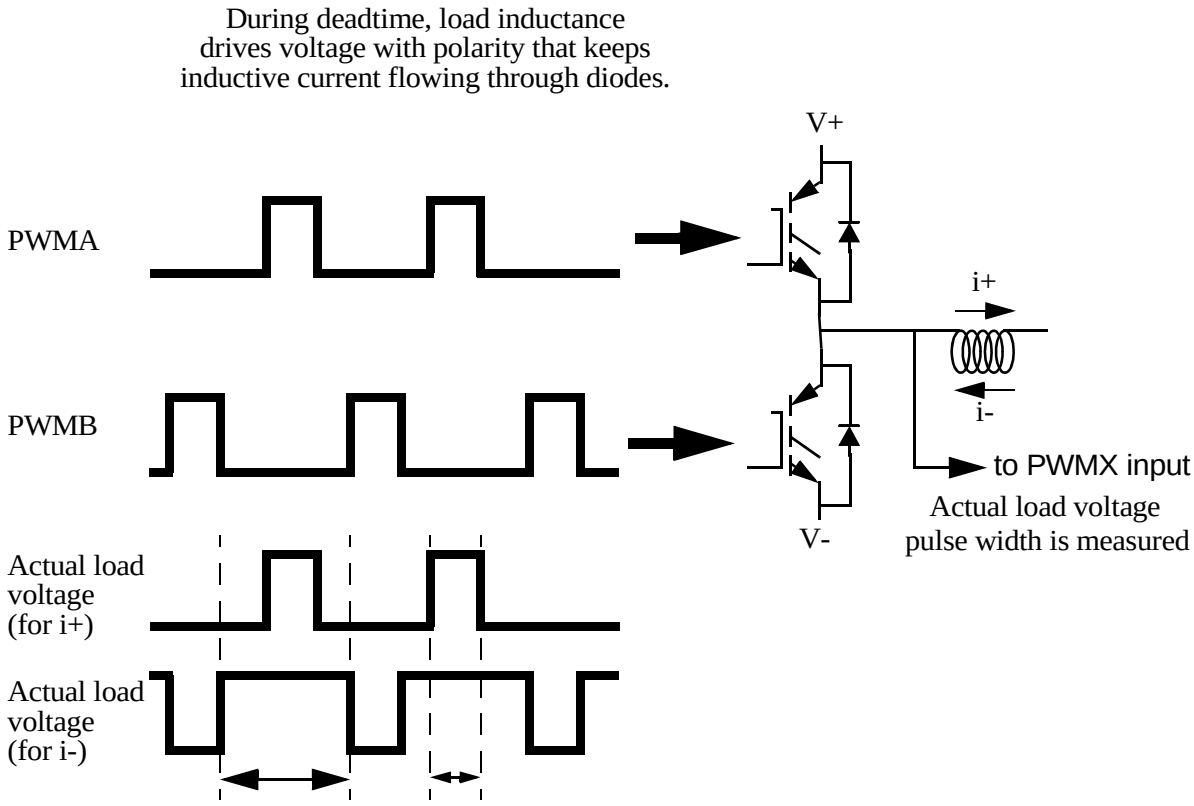


Figure 373. Output Pulse Width Measurement Possible with the E-Capture Circuit

26.3.2.7 Synchronous Switching of Multiple Outputs

Before the PWM signals are routed to the output pins, they are processed by a hardware block that permits all submodule outputs to be switched synchronously. This feature can be extremely useful in commutated motor applications where the next commutation state can be laid in ahead of time and then immediately switched to the outputs when the appropriate condition or time is reached. Not only do all the changes occur synchronously on all submodule outputs, but they occur IMMEDIATELY after the trigger event occurs eliminating any interrupt latency.

The synchronous output switching is accomplished via a signal called FORCE_OUT. This signal originates from the local FORCE bit within the submodule, from submodule0, or from external to the PWM module and, in most cases, is supplied from an external timer channel configured for output compare. In a typical application, software sets up the desired states of the output pins in preparation for the next FORCE_OUT event. This selection lays dormant until the FORCE_OUT signal transitions and then all outputs are switched simultaneously. The signal switching is performed upstream from the deadtime generator so that any abrupt changes that might occur do not violate deadtime on the power stage when in complementary mode.

Figure 374 shows a popular application that can benefit from this feature. On a brushless DC motor it is desirable on many cases to spin the motor without need of hall-effect sensor feedback. Instead, the back EMF of the motor phases is monitored and this information is used to schedule the next commutation event. The top waveforms of *Figure 374* are a

simplistic representation of these back EMF signals. Timer compare events (represented by the long vertical lines in the diagram) are scheduled based on the zero crossings of the back-EMF waveforms. The PWM module is configured via software ahead of time with the next state of the PWM pins in anticipation of the compare event. When it happens, the output compare of the timer drives the FORCE_OUT signal which immediately changes the state of the PWM pins to the next commutation state with no software latency.

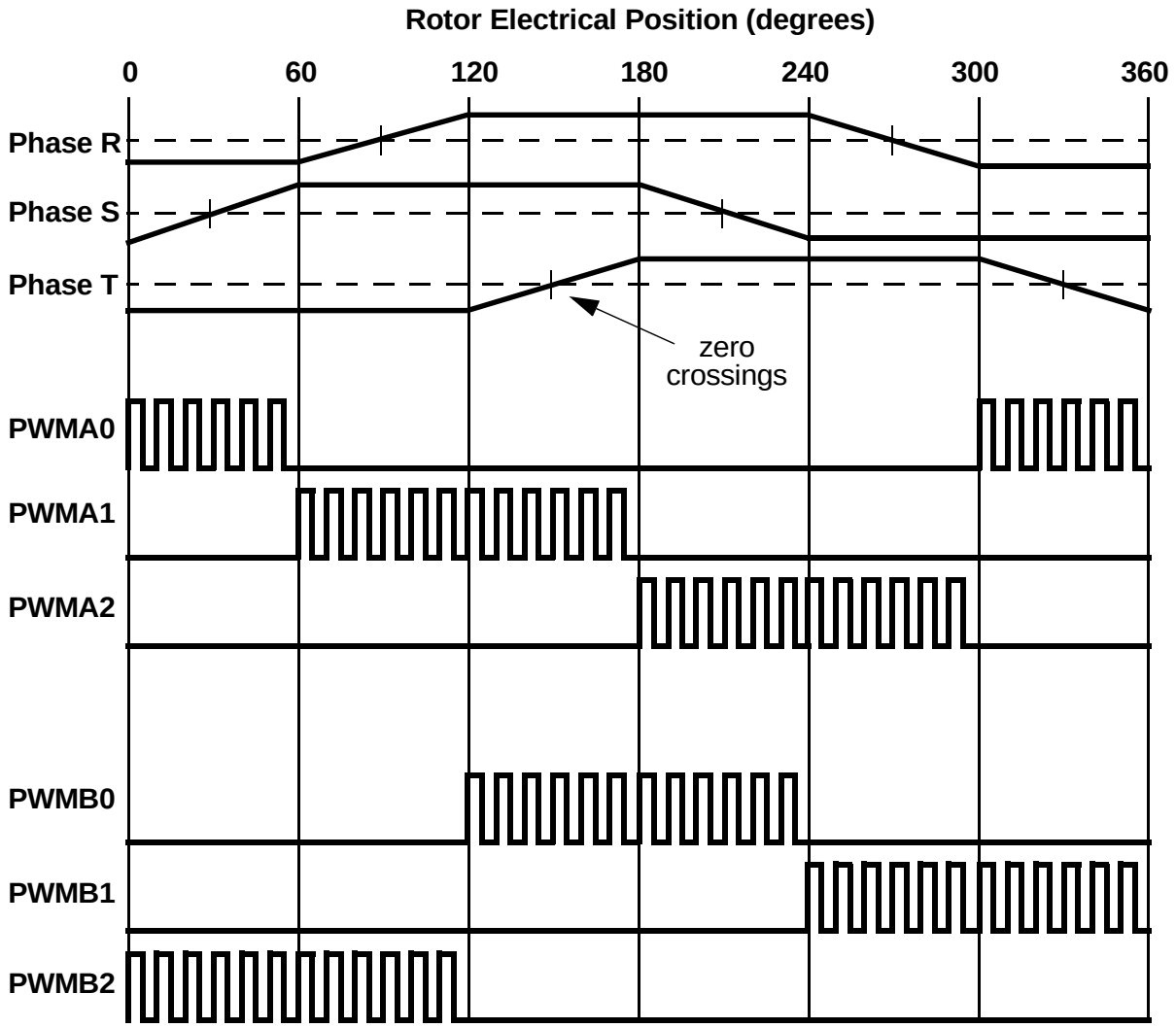


Figure 374. Sensorless BLDC Commutation Using the Force Out Function

26.3.3 Functional details

This section describes the implementation of various sections of the PWM in greater detail.

26.3.3.1 PWM clocking

Figure 375 shows the logic used to generate the main counter clock. Each submodule can select between three clock signals: the IPBus clock, EXT_CLK, and AUX_CLK. The

EXT_CLK is generated by an on-chip resource such as a Timer module and goes to all of the submodules. The AUX_CLK signal is broadcast from submodule0 and can be selected as the clock source by other submodules so that the 8 bit prescaler and RUN bit from submodule0 can control all of the submodules. When AUX_CLK is selected as the source for the submodule clock, the RUN bit from submodule0 is used instead of the local RUN bit from this submodule.

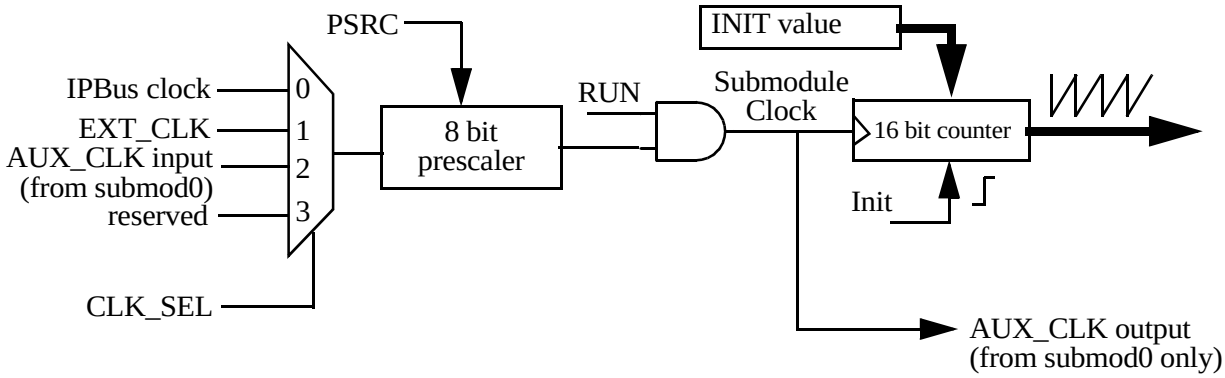


Figure 375. Clocking block diagram for each PWM submodule

To permit lower PWM frequencies, the prescaler produces the PWM clock frequency by dividing the IPBus clock frequency by 1-128. The prescaler bits, PRSC, in the control register (CTRL1), select the prescaler divisor. This prescaler is buffered and will not be used by the PWM generator until the LDOK bit is set and a new PWM reload cycle begins or LDMOD is set.

26.3.3.2 Register reload logic

The register reload logic is used to determine when the outer set of registers for all double buffered register pairs will be transferred to the inner set of registers. The register reload event can be scheduled to occur every "n" PWM cycles using the LDFQ bits and the FULL bit. A half cycle reload option is also supported (HALF) where the reload can take place in the middle of a PWM cycle. The half cycle point is defined by the VAL0 register and does not have to be exactly in the middle of the PWM cycle.

As illustrated in *Figure 376* the reload signal from submodule0 can be broadcast as the Master Reload signal allowing the reload logic from submodule0 to control the reload of registers in other submodules.

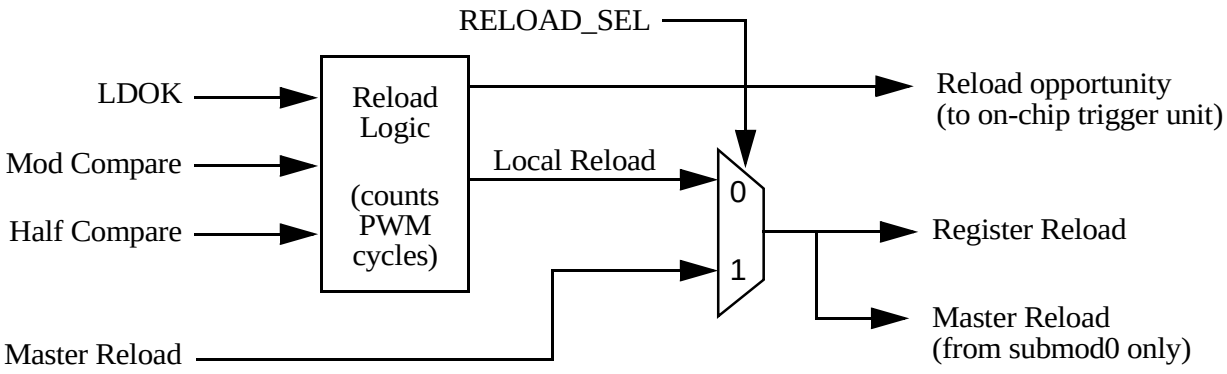


Figure 376. Register Reload Logic

26.3.3.3 Counter synchronization

Referring to *Figure 377*, the 16 bit counter will count up until its output equals VAL1 which is used to specify the counter modulus value. The resulting compare causes a rising edge to occur on the Local Sync signal which is one of four possible sources used to cause the 16 bit counter to be initialized with INIT. If Local Sync is selected as the counter initialization signal, then VAL1 within the submodule effectively controls the timer period (and thus the PWM frequency generated by that submodule) and everything works on a local level.

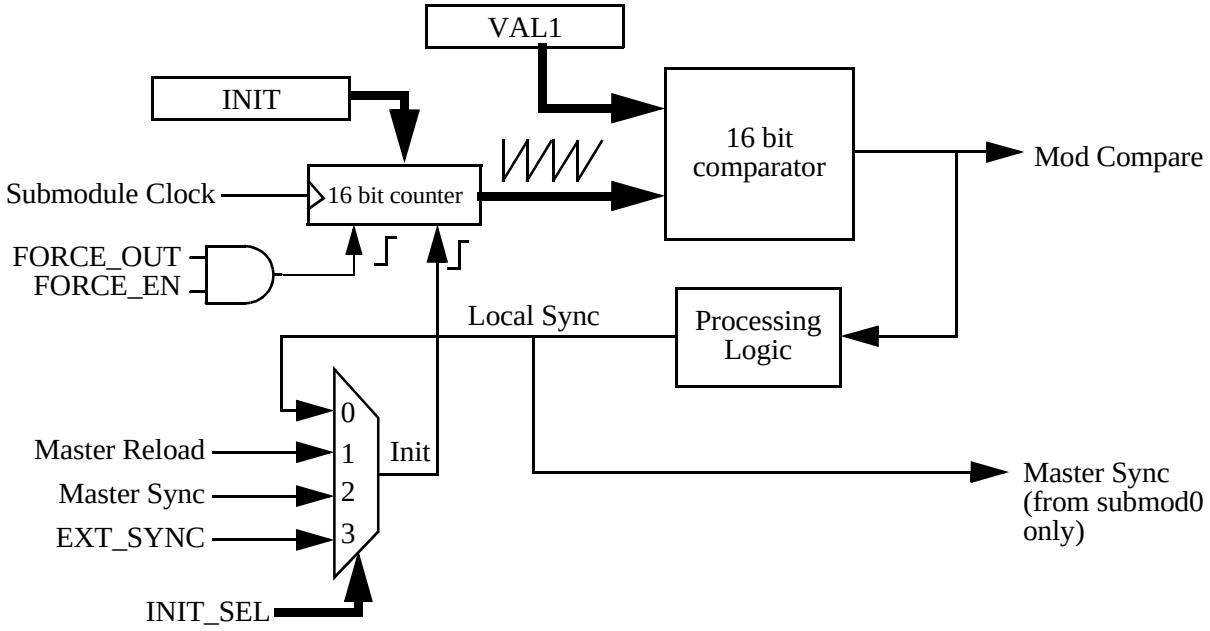


Figure 377. Submodule timer synchronization

The Master Sync signal originates as the Local Sync from submodule0. If configured to do so, the timer period of any submodule can be locked to the period of the timer in submodule0. The VAL1 register and associated comparator of the other submodules can then be freed up for other functions such as PWM generation, input captures, output compares, or output triggers.

The EXT_SYNC signal originates on chip or off chip depending on the system architecture. This signal may be selected as the source for counter initialization so that an external source can control the period of all submodules.

If the Master Reload signal is selected as the source for counter initialization, then the period of the counter will be locked to the register reload frequency of submodule0. Since the reload frequency is usually commensurate to the sampling frequency of the software control algorithm, the submodule counter period will therefore equal the sampling period. As a result, this timer can be used to generate output compares or output triggers over the entire sampling period which may consist of several PWM cycles. The Master Reload signal can only originate from submodule0.

The counter can optionally initialize upon the assertion of the FORCE_OUT signal assuming that the FORCE_EN bit is set. As indicated by *Figure 377*, this constitutes a second init input into the counter which will cause the counter to initialize regardless of which signal is selected as the counter init signal. The FORCE_OUT signal is provided mainly for commutated applications. When PWM signals are commutated on an inverter controlling a brushless DC motor, it is necessary to restart the PWM cycle at the beginning of the commutation interval. This action effectively resynchronizes the PWM waveform to the commutation timing. Otherwise, the average voltage applied to a motor winding integrated over the entire commutation interval will be a function of the timing between the asynchronous commutation event with respect to the PWM cycle. The effect is more critical at higher motor speeds where each commutation interval may consist of only a few PWM cycles. If the counter is not initialized at the start of each commutation interval, the result will be an oscillation caused by the beating between the PWM frequency and the commutation frequency.

26.3.3.4 PWM generation

Figure 378 illustrates how PWM generation is accomplished in each submodule. In each case, two comparators and associated VALx registers are utilized for each PWM output signal. One comparator and VALx register is used to control the turn on edge while a second comparator and VALx register control the turn off edge.

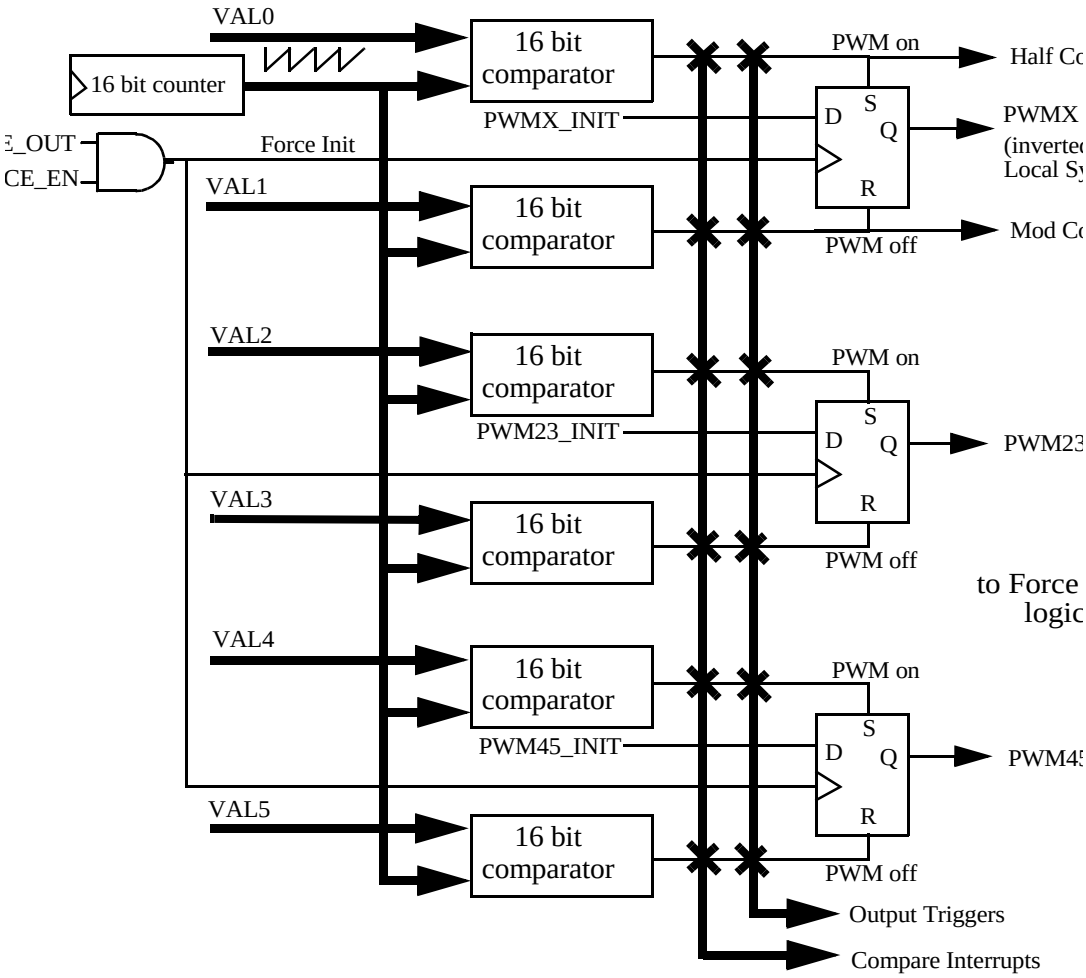


Figure 378. PWM Generation Hardware

The generation of the Local Sync signal is performed exactly the same way as the other PWM signals in the submodule. While comparator 0 causes a rising edge of the Local Sync signal, comparator 1 generates a falling edge. Comparator 1 is also hardwired to the reload logic to generate the half cycle reload indicator.

If VAL1 is controlling the modulus of the counter and VAL0 is half of the VAL1 register minus the INIT value, then the half cycle reload pulse will occur exactly half way through the timer count period and the Local Sync will have a 50% duty cycle. On the other hand, if the VAL1 and VAL0 registers are not required for register reloading or counter initialization, they can be used to modulate the duty cycle of the Local Sync signal effectively turning it into an auxiliary PWM signal (PWMX) assuming that the PWMX pin is not being used for another function such as input capture or deadtime distortion correction. Including the Local Sync signal, each submodule is capable of generating 3 PWM signals where software has complete control over each edge of each of the signals.

If the comparators and edge value registers are not required for PWM generation, they can also be used for other functions such as output compares, generating output triggers, or generating interrupts at timed intervals.

The 16-bit comparators shown in *Figure 378* are "equal to or greater than" not just "equal to" comparators. In addition, if both the set and reset of the flip-flop are both asserted, then the flop output goes to 0.

26.3.3.5 Output compare capabilities

By using the VALx registers in conjunction with the submodule timer and 16 bit comparators, buffered output compare functionality can be achieved with no additional hardware required. Specifically, the following output compare functions are possible:

- An output compare sets the output high
- An output compare sets the output low
- An output compare generates an interrupt
- An output compare generates an output trigger

Referring again to *Figure 378*, an output compare is initiated by programming a VALx register for a timer compare which in turn causes the output of the D flip-flop to either set or reset. For example, if an output compare is desired on the PWMA signal that sets it high, VAL2 would be programmed with the counter value where the output compare should take place. However, to prevent the D flip-flop from being reset again after the compare has occurred, the VAL3 register must be programmed to a value outside of the modulus range of the counter. Therefore, a compare that would result in resetting the D flip-flop output would never occur. Conversely, if an output compare is desired on the PWMA signal that sets it low, the VAL3 register is programmed with the appropriate count value and the VAL2 register is programmed with a value outside the counter modulus range. Regardless of whether a high compare or low compare is programmed, an interrupt or output trigger can be generated when the compare event occurs.

26.3.3.6 Force out logic

For each submodule software can select between seven signal sources for the FORCE_OUT signal: the local FORCE bit, the Master Force signal from submodule0, the local Reload signal, the Master Reload signal from submodule0, the Local Sync signal, the Master Sync signal from submodule0, or the EXT_FORCE signal from on or off chip depending on the chip architecture. The local signals are used when the user simply wants to change the signals on the output pins of the submodule without regard for synchronization with other submodules. However, if it is required that all signals on all submodule outputs change at the same time, the Master signals or EXT_FORCE signal should be selected.

Figure 379 illustrates the Force logic. The SEL23 and SEL45 fields each choose from one of four signals that can be supplied to the submodule outputs: the PWM signal, the inverted PWM signal, a binary level specified by software via the OUT23 and OUT45 bits, or the EXTA or EXTB alternate external control signals. The selection can be determined ahead of time and, when a FORCE_OUT event occurs, these values are presented to the signal selection mux which immediately switches the requested signal to the output of the mux for further processing downstream.

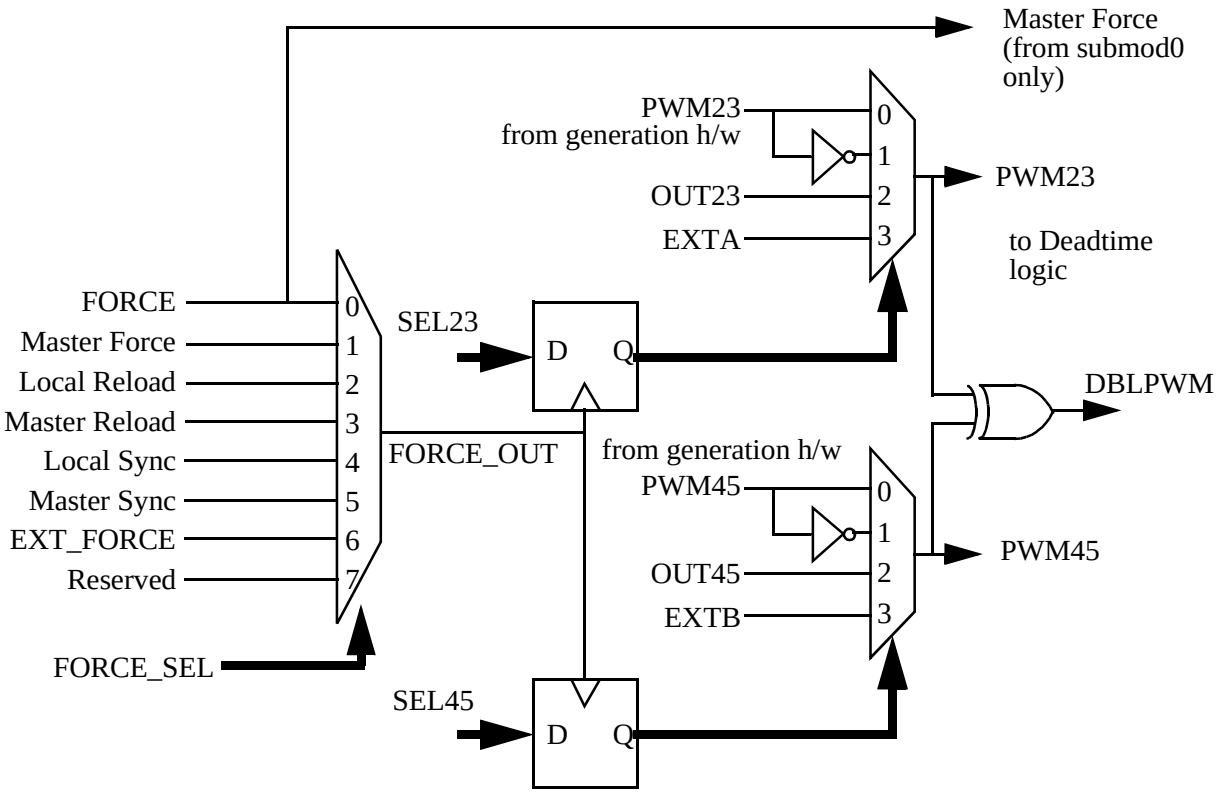


Figure 379. Force Out Logic

The local FORCE signal of submodule0 can be broadcast as the Master Force signal to other submodules. This feature allows the local FORCE bit of submodule0 to synchronously update all of the submodule outputs at the same time. The EXT_FORCE signal originates from outside the PWM module from a source such as a timer or digital comparators in the Analog-to-Digital Converter.

26.3.3.7 Independent or complementary channel operation

Writing a logic one to the INDEP bit of the CNFG register configures the pair of PWM outputs as two independent PWM channels. Each PWM output is controlled by its own VALx pair operating independently of the other output.

Writing a logic zero to the INDEP bit configures the PWM output as a pair of complementary channels. The PWM pins are paired as shown in ***Figure 380*** in complementary channel operation. Which signal is connected to the output pin (PWM23 or PWM45) is determined by the IPOL bit.

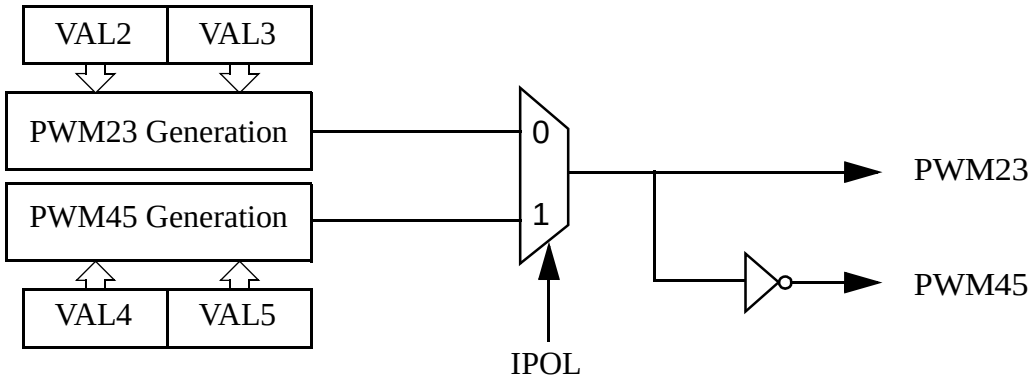


Figure 380. Complementary Channel Pair

The complementary channel operation is for driving top and bottom transistors in a motor drive circuit, such as the one in ***Figure 381***.

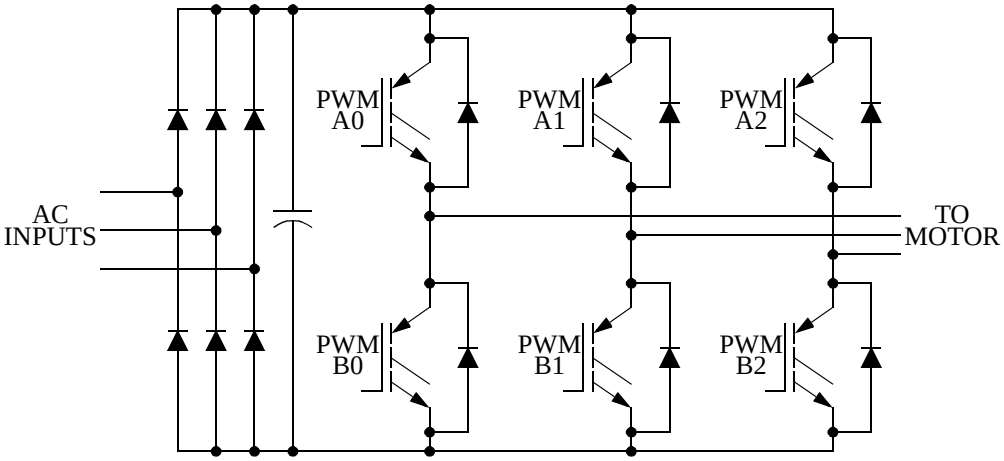


Figure 381. Typical 3 Phase AC Motor Drive

Complementary operation allows the use of the deadtime insertion feature.

26.3.3.8 Deadtime insertion logic

Figure 382 shows the deadtime insertion logic of each submodule which is used to create non-overlapping complementary signals when not in independent mode.

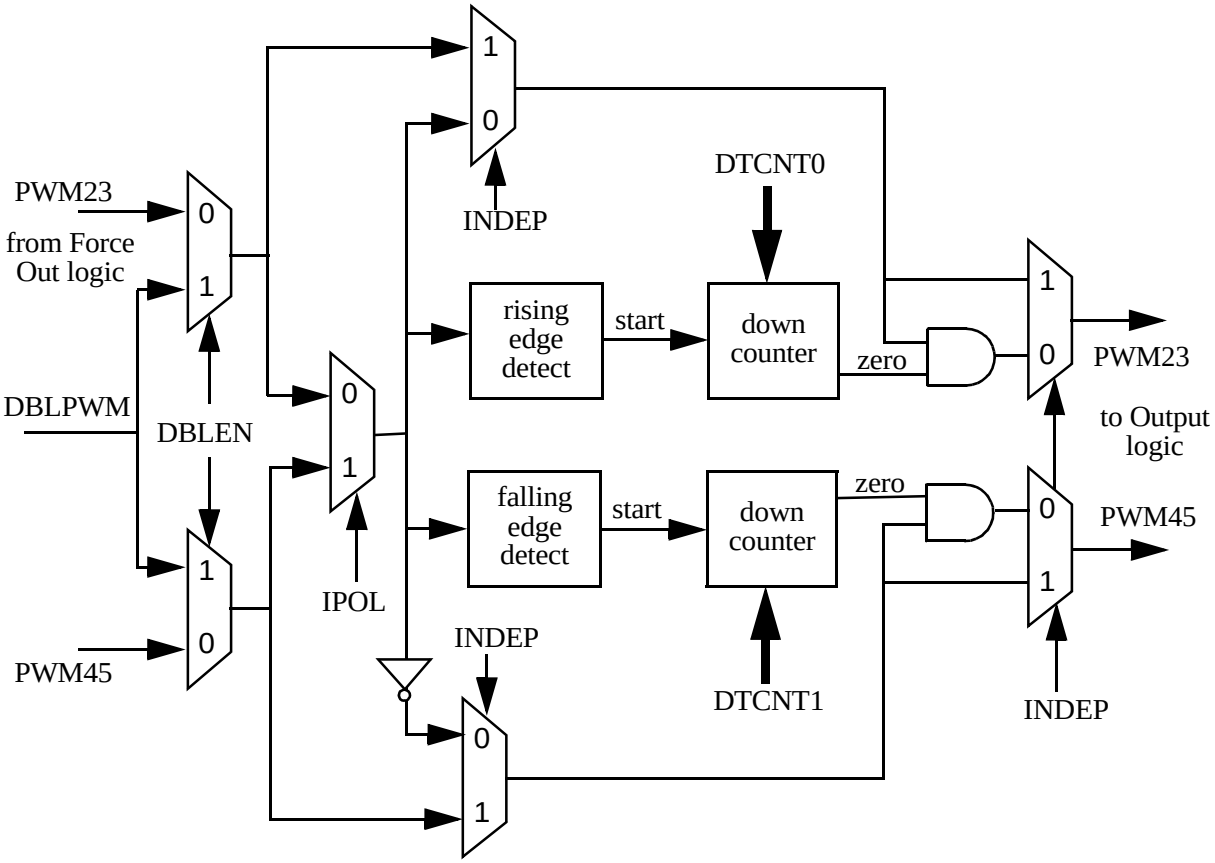


Figure 382. Deadtime Insertion and Fine Control Logic

While in the complementary mode, a PWM pair can be used to drive top/bottom transistors, as shown in *Figure 382*. When the top PWM channel is active, the bottom PWM channel is inactive, and vice versa.

Note: To avoid short circuiting the DC bus and endangering the transistor, there must be no overlap of conducting intervals between top and bottom transistor. But the transistor's characteristics may make its switching-off time longer than switching-on time. To avoid the conducting overlap of top and bottom transistors, deadtime needs to be inserted in the switching period (as illustrated in Figure 383).

The deadtime generators automatically insert software-selectable activation delays into the pair of PWM outputs. The deadtime registers (DTCNT0 and DTCNT1) specify the number of IPBus clock cycles to use for deadtime delay. Every time the deadtime generator inputs change state, deadtime is inserted. Deadtime forces both PWM outputs in the pair to the inactive state.

When deadtime is inserted in complementary PWM signals connected to an inverter driving an inductive load, the PWM waveform on the inverter output will have a different duty cycle than what appears on the output pins of the PWM module. This results in a distortion in the voltage applied to the load. A method of correcting this, adding to or subtracting from the PWM value used, is discussed next.

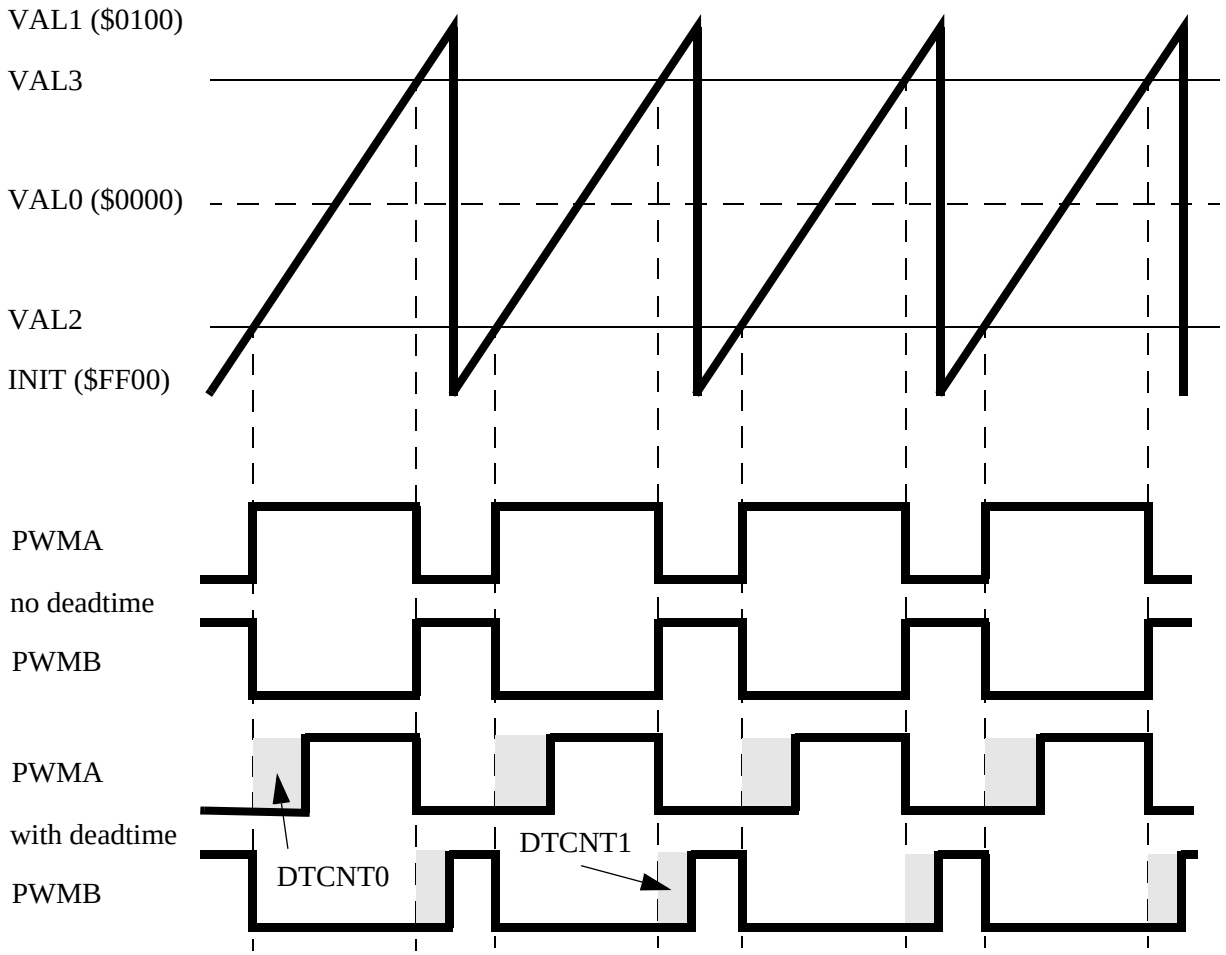


Figure 383. Deadtime Insertion

26.3.3.8.1 Top/Bottom Correction

In complementary mode, either the top or the bottom transistor controls the output voltage. However, deadtime has to be inserted to avoid overlap of conducting interval between the top and bottom transistor. Both transistors in complementary mode are off during deadtime, allowing the output voltage to be determined by the current status of load and introduce distortion in the output voltage. See *Figure 384*. On AC induction motors running open-loop, the distortion typically manifests itself as poor low-speed performance, such as torque ripple and rough operation.

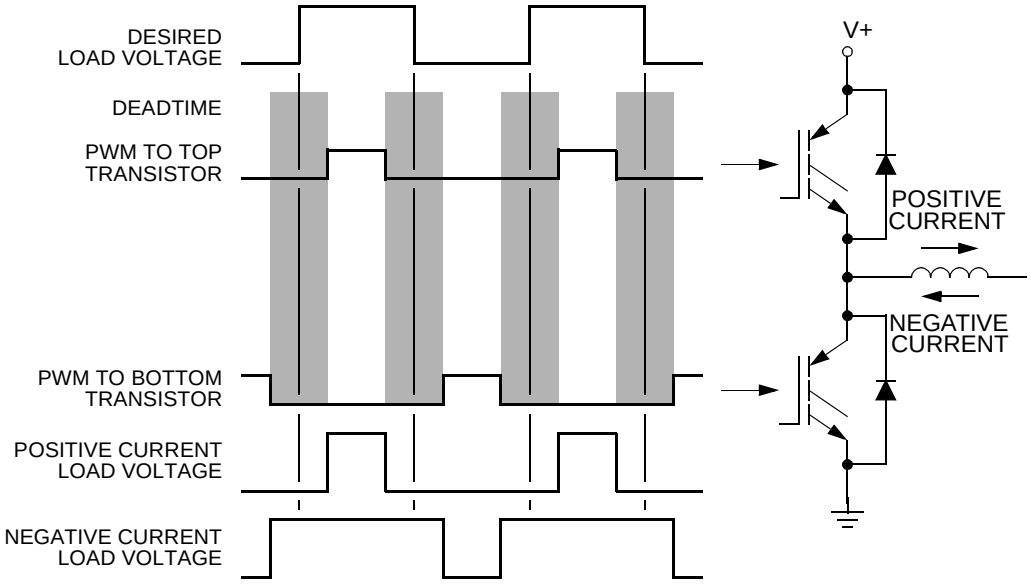


Figure 384. Deadtime Distortion

During deadtime, load inductance distorts output voltage by keeping current flowing through the diodes. This deadtime current flow creates a load voltage that varies with current direction. With a positive current flow, the load voltage during deadtime is equal to the bottom supply, putting the top transistor in control. With a negative current flow, the load voltage during deadtime is equal to the top supply putting the bottom transistor in control.

Remembering that the original PWM pulse widths where shortened by deadtime insertion, the averaged sinusoidal output will be less than the desired value. However, when deadtime is inserted, it creates a distortion in the motor current waveform. This distortion is aggravated by dissimilar turn-on and turn-off delays of each of the transistors. By giving the PWM module information on which transistor is controlling at a given time this distortion can be corrected.

For a typical circuit in complementary channel operation, only one of the transistors will be effective in controlling the output voltage at any given time. This depends on the direction of the motor current for that pair. See *Figure 384*. To correct distortion one of two different factors must be added to the desired PWM value, depending on whether the top or bottom transistor is controlling the output voltage. Therefore, the software is responsible for calculating both compensated PWM values prior to placing them in the VALx registers. Either the VAL2/VAL3 or the VAL4/VAL5 register pair controls the pulse width at any given time. For a given PWM pair, whether the VAL2/VAL3 or VAL4/VAL5 pair is active depends on either:

- The state of the current status pin, PWMx, for that driver
- The state of the odd/even correction bit, IPOL, for that driver

To correct deadtime distortion, software can decrease or increase the value in the appropriate VALx register.

- In edge-aligned operation, decreasing or increasing the PWM value by a correction value equal to the deadtime typically compensates for deadtime distortion.
- In center-aligned operation, decreasing or increasing the PWM value by a correction value equal to one-half the deadtime typically compensates for deadtime distortion.

26.3.3.8.2 Manual Correction

To detect the current status, the voltage on each PWMx pin is sampled twice in a PWM period, at the end of each deadtime. The value is stored in the DTx bits in the CTRL1 register. The DTx bits are a timing marker especially indicating when to toggle between PWM value registers. Software can then set the IPOL bit to switch between VAL2/VAL3 and VAL4/VAL5 register pairs according to DTx values.

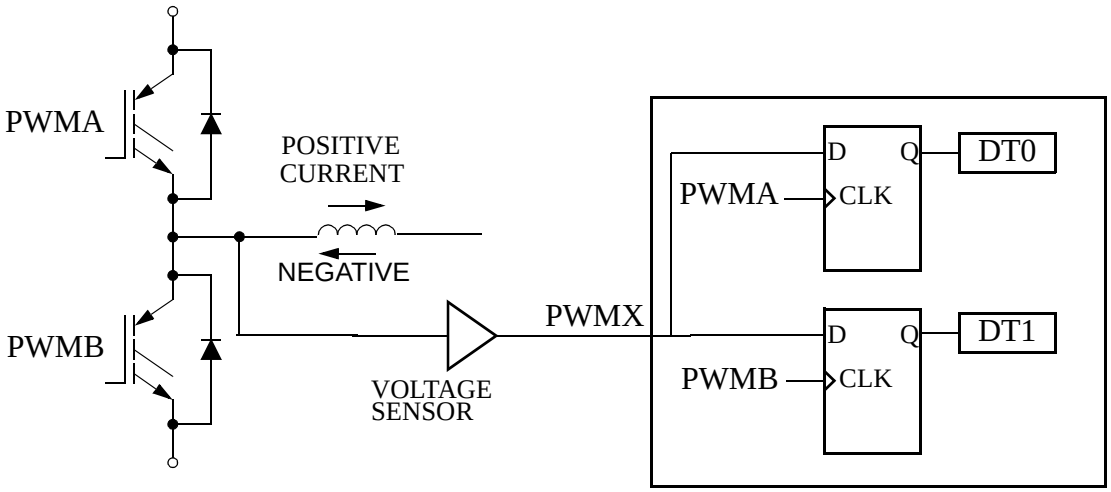


Figure 385. Current-status Sense Scheme for Deadtime Correction

Both D flip-flops latch low, DT0 = 0, DT1 = 0, during deadtime periods if current is large and flowing out of the complementary circuit. See *Figure 385*. Both D flip-flops latch the high, DT0 = 1, DT1 = 1, during deadtime periods if current is also large and flowing into the complementary circuit.

However, under low-current, the output voltage of the complementary circuit during deadtime is somewhere between the high and low levels. The current cannot free-wheel through the opposition anti-body diode, regardless of polarity, giving additional distortion when the current crosses zero. Sampled results will be DT0 = 0 and DT1 = 1. Thus, the best time to change one PWM value register to another is just before the current zero crossing.

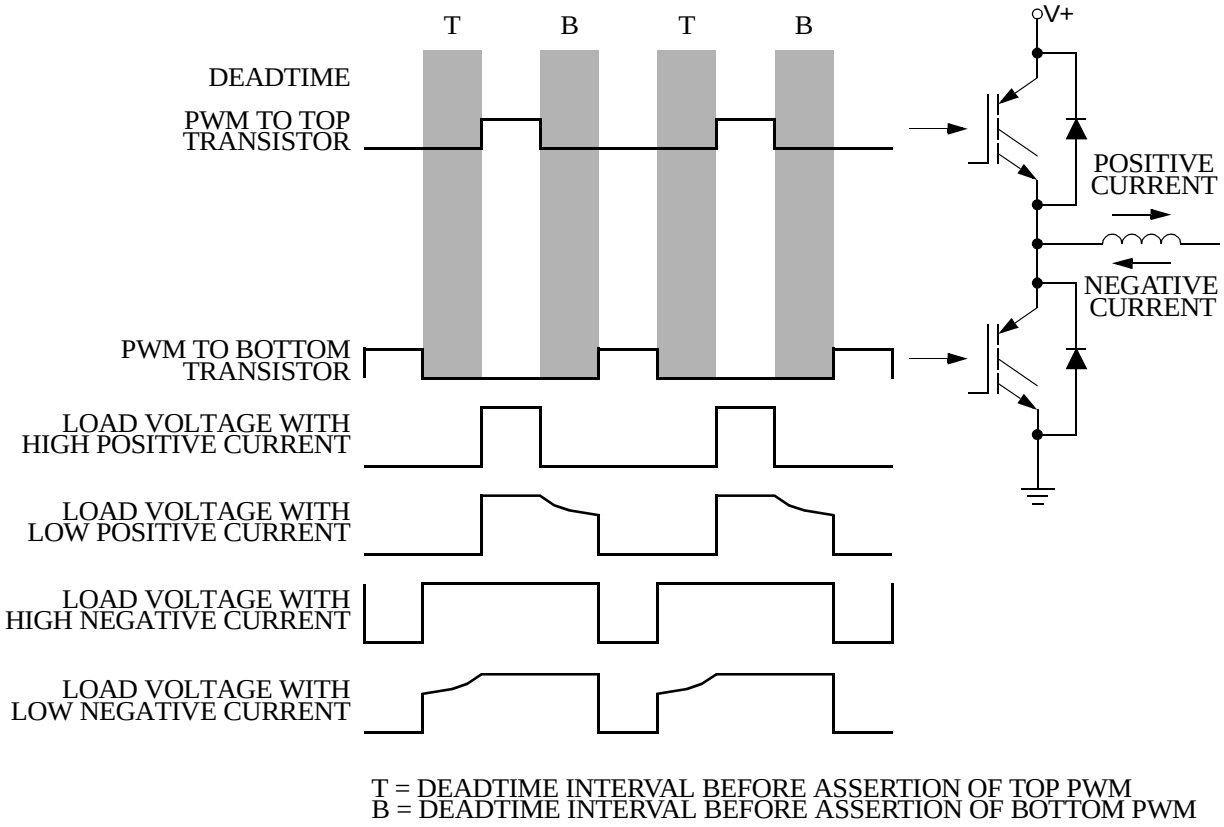


Figure 386. Output Voltage Waveforms

26.3.3.9 Output logic

Figure 387 shows the output logic of each submodule including how each PWM output has individual fault disabling, polarity control, and output enable. This allows for maximum flexibility when interfacing to the external circuitry.

The PWM23 and PWM45 signals which are output from the deadtime logic in *Figure 387* are positive true signals. In other words, a high level on these signals should result in the corresponding transistor in the PWM inverter being turned ON. The voltage level required at the PWM output pin to turn the transistor ON or OFF is a function of the logic between the pin and the transistor. Therefore, it is imperative that the user program the POLA and POLB bits before enabling the output pins. A fault condition can result in the PWM output being tristated, forced to a logic 1, or forced to a logic 0 depending on the values programmed into the PWMxFS fields.

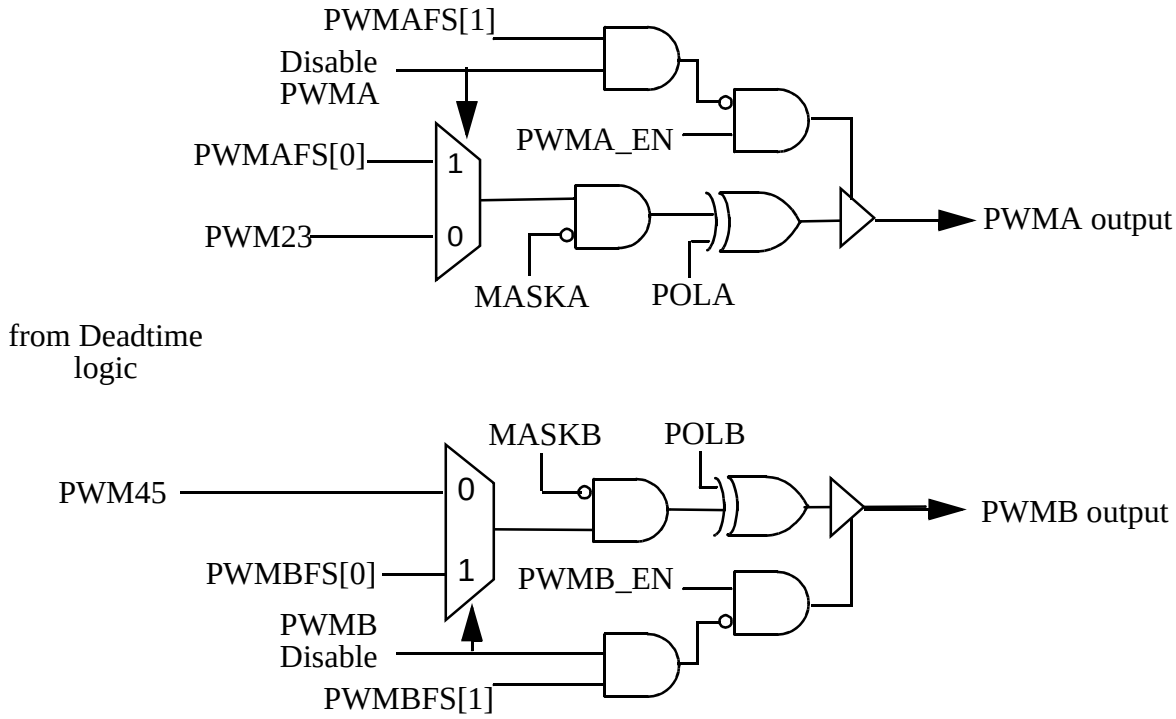


Figure 387. Output Logic Section

26.3.3.10 E-Capture

Commensurate with the idea of controlling both edges of an output signal, the Enhanced Capture (E-Capture) logic is designed to measure both edges of an input signal. As a result, when a submodule pin is configured for input capture, the CVALx registers associated with that pin are used to record the edge values.

Figure 388 illustrates the block diagram of the E-Capture circuit. Upon entering the pin input, the signal is split into two paths. One goes straight to a mux input where software can select to pass the signal directly to the capture logic for processing. The other path connects the signal to an 8 bit counter which counts both the rising and falling edges of the signal. The output of this counter is compared to an 8 bit value that is specified by the user (EDGCMPx) and when the two values are equal, the comparator generates a pulse that resets the counter. This pulse is also supplied to the mux input where software can select it to be processed by the capture logic. This feature permits the E-Capture circuit to count up to 256 edge events before initiating a capture event. this feature is useful for dividing down high frequency signals for capture processing so that capture interrupts don't overwhelm the CPU. Also, this feature can be used to simply generate an interrupt after "n" events have been counted.

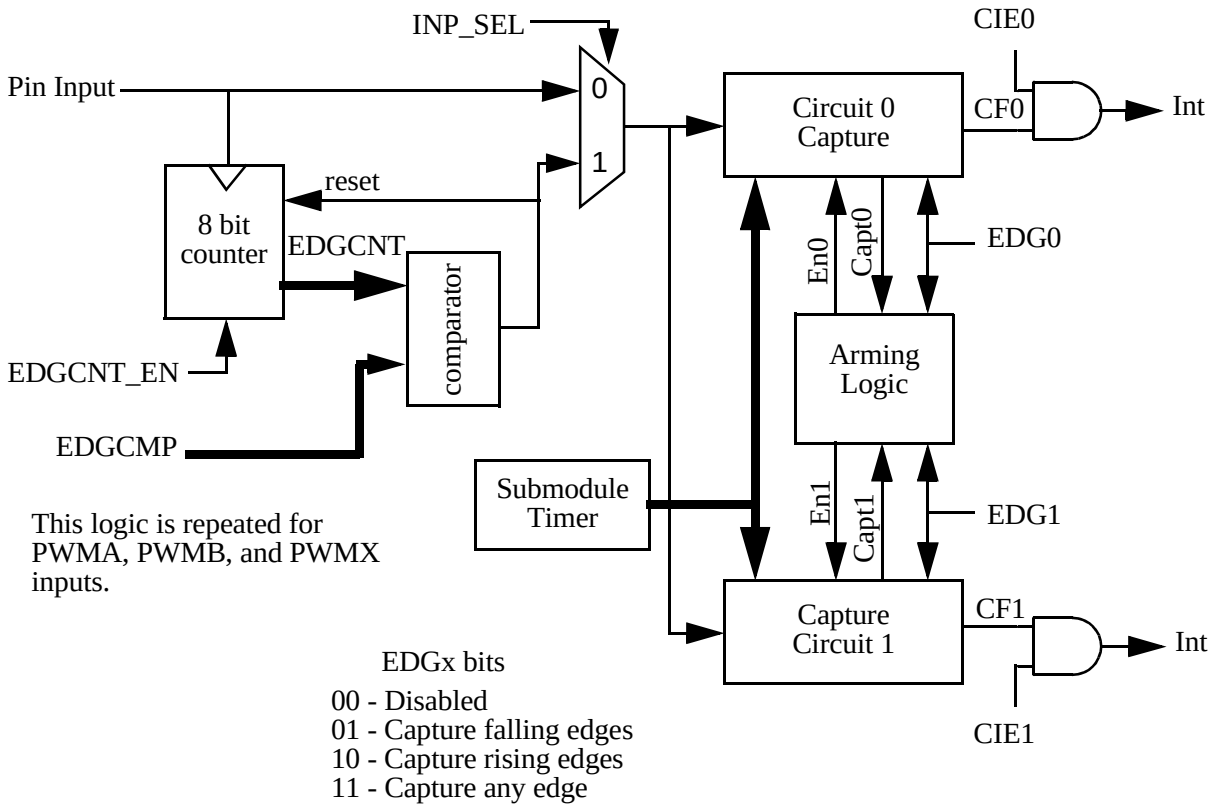


Figure 388. Enhanced Capture (E-Capture) Logic

Based on the mode selection, the mux selects either the pin input or the compare output from the count/compare circuit to be processed by the capture logic. The selected signal is routed to two separate capture circuits which work in tandem to capture sequential edges of the signal. The type of edge to be captured by each circuit is determined by the EDGx1 and EDGx0 bits whose functionality is listed in *Figure 388*. Also, controlling the operation of the capture circuits is the arming logic which allows captures to be performed in a free running (continuous) or one shot fashion. In free running mode, the capture sequences will be performed indefinitely. If both capture circuits are enabled, they will work together in a ping-pong style where a capture event from one circuit leads to the arming of the other and vice versa. In one shot mode, only one capture sequence will be performed. If both capture circuits are enabled, capture circuit 0 is first armed and when a capture event occurs, capture circuit 1 is armed. Once the second capture occurs, further captures are disabled until another capture sequence is initiated. Both capture circuits are also capable of generating an interrupt to the CPU.

26.3.3.11 Fault protection

Fault protection can control any combination of PWM output pins. Faults are generated by a logic one on any of the FAULTx pins. This polarity can be changed via the FLVL bits. Each FAULTx pin can be mapped arbitrarily to any of the PWM outputs. When fault protection hardware disables PWM outputs, the PWM generator continues to run, only the output pins are forced to logic 0, logic 1, or tristated depending the values of the PWMxFS bits.

The fault decoder disables PWM pins selected by the fault logic and the disable mapping register (DISMAP). See *Figure 389* for an example of the fault disable logic. Each bank of bits in DISMAP control the mapping for a single PWM pin. Refer to *Table 326*.

The fault protection is enabled even when the PWM module is not enabled; therefore, a fault will be latched in and must be cleared in order to prevent an interrupt when the PWM is enabled.

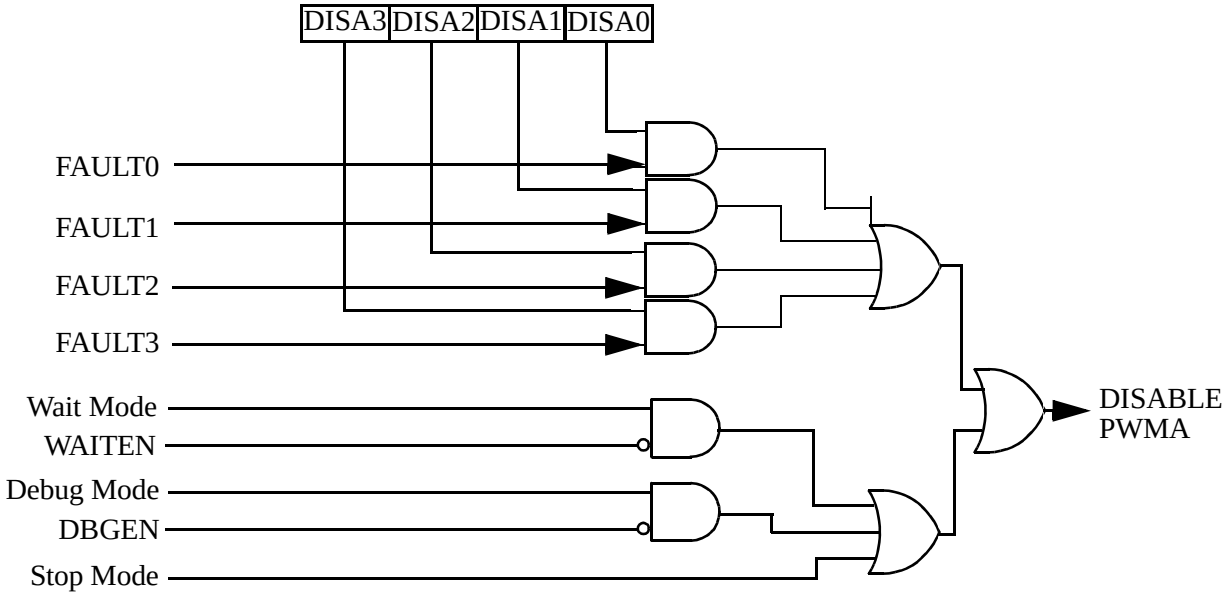


Figure 389. Fault Decoder for PWMA

Table 326. Fault Mapping

PWM Pin	Controlling Register Bits
PWMA	DISA[3:0]
PWMB	DISB[3:0]
PWMX	DISX[3:0]

26.3.3.11.1 Fault Pin Filter

Each fault pin has a programmable filter that can be bypassed. The sampling period of the filter can be adjusted with the FILT_PER field of the FFILTx register. The number of consecutive samples that must agree before an input transition is recognized can be adjusted using the FILT_CNT field of the same register. Setting FILT_PER to all 0 disables the input filter for a given FAULTx pin.

Upon detecting a logic 0 on the filtered FAULTx pin (or a logic 1 if FLVLx is set), the corresponding FFPINx and fault flag, FFLAGx, bits are set. The FFPINx bit remains set as long as the filtered FAULTx pin is zero. Clear FFLAGx by writing a logic 1 to FFLAGx.

If the FIEx, FAULTx pin interrupt enable bit is set, the FFLAGx flag generates a CPU interrupt request. The interrupt request latch remains set until:

- Software clears the FFLAGx flag by writing a logic one to the bit
- Software clears the FIEx bit by writing a logic zero to it
- A reset occurs

Even with the filter enabled, there is a combinational path from the FAULTx inputs to the PWM pins. This logic is also capable of holding a fault condition in the event of loss of clock to the PWM module.

26.3.3.11.2 Automatic Fault Clearing

Setting an automatic clearing mode bit, FAUTOx, configures faults from the FAULTx pin for automatic clearing.

When FAUTOx is set, disabled PWM pins are enabled when the FAULTx pin returns to logic one and a new PWM full or half cycle begins. See *Figure 390*. If FFULLx is set, then the disabled PWM pins are enabled only at the start of a full cycle and not at the half cycle. Clearing the FFLAGx flag does not affect disabled PWM pins when FAUTOx is set.

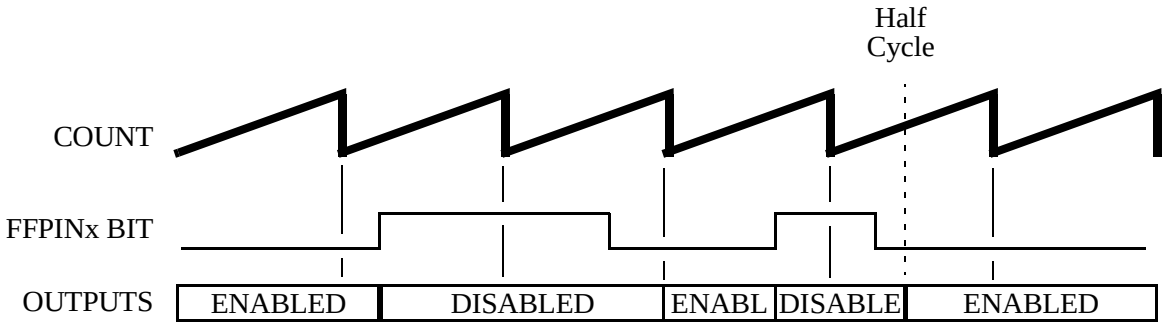


Figure 390. Automatic Fault Clearing

26.3.3.11.3 Manual Fault Clearing

Clearing the automatic clearing mode bit, FAUTOx, configures faults from the FAULTx pin for manual clearing:

- If the fault safety mode bits, FSAFEx, are clear, then PWM pins disabled by the FAULTx pins are enabled when:
  - Software clears the corresponding FFLAGx flag
  - The pins are enabled when the next PWM full or half cycle begins regardless of the logic level detected by the filter at the FAULTx pin. See *Figure 391*. If FFULLx is set, then the disabled PWM pins are enabled only at the start of a full cycle and not at the half cycle.
- If the fault safety mode bits, FSAFEx, are set, then PWM pins disabled by the FAULTx pins are enabled when:
  - Software clears the corresponding FFLAGx flag
  - The filter detects a logic one on the FAULTx pin at the start of the next PWM full or half cycle boundary. See *Figure 392*. If FFULLx is set, then the disabled PWM pins are enabled only at the start of a full cycle and not at the half cycle.

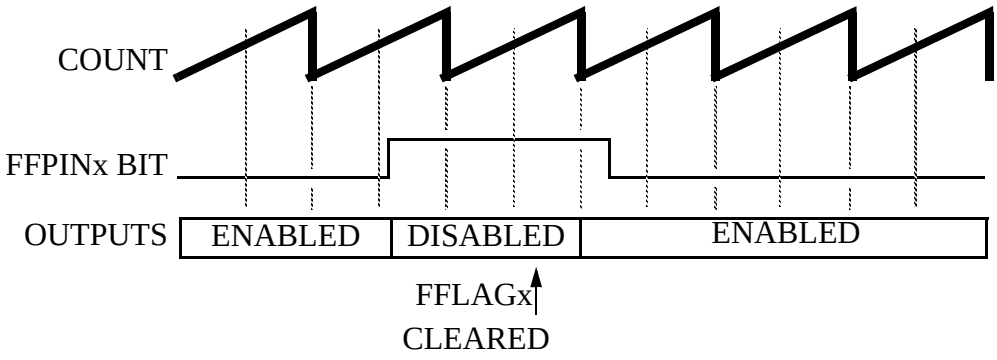


Figure 391. Manual Fault Clearing (FSAFE=0)

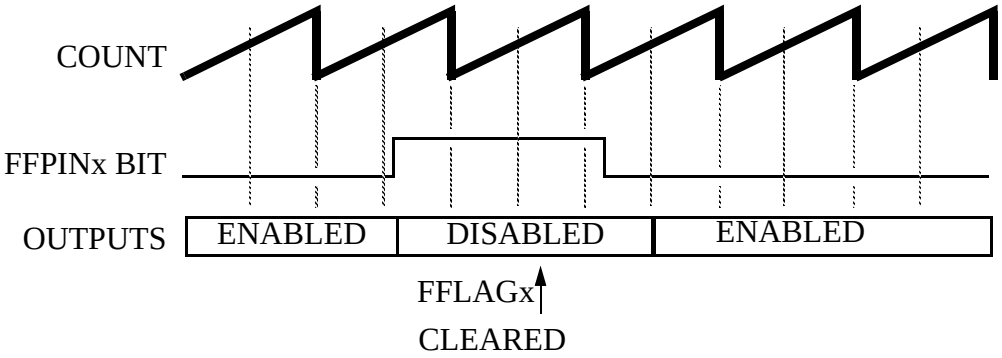


Figure 392. Manual Fault Clearing (FSAFE=1)

Note: *Fault protection also applies during software output control when the SEL23 and SEL45 fields are set to select OUT23 and OUT45 bits or EXTA and EXTB. Fault clearing still occurs at half PWM cycle boundaries while the PWM generator is engaged, RUN equals one. But the OUTx bits can control the PWM pins while the PWM generator is off, RUN equals zero. Thus, fault clearing occurs at IPBus cycles while the PWM generator is off and at the start of PWM cycles when the generator is engaged.*

26.3.3.11.4 Fault Testing

The FTEST bit is used to simulate a fault condition on each of the fault inputs.

26.3.4 PWM generator loading

26.3.4.1 Load enable

The LDOK bit enables loading of the following PWM generator parameters:

- The prescaler divisor—from the PRSC bits in the CTRL1 register
- The PWM period and pulse width—from the INIT and VALx registers

LDOK allows software to finish calculating all of these PWM parameters so they can be synchronously updated. The PSRC, INIT, and VALx registers are loaded by software into a set of outer buffers. When LDOK is set, these values are transferred to an inner set of registers at the beginning of the next PWM reload cycle to be used by the PWM generator. These values can be transfered to the inner set of registers immediately upon setting LDOK

if LDMOD is set. Set LDOK by reading it when it is a logic zero and then writing a logic one to it. After loading, LDOK is automatically cleared.

26.3.4.2 Load frequency

The LDFQ bits in the CTRL1 register select an integral loading frequency of one to 16 PWM reload opportunities. The LDFQ bits take effect at every PWM reload opportunity, regardless the state of the LDOK bit. The HALF and FULL bits in the CTRL1 register control reload timing. If FULL is set, a reload opportunity occurs at the end of every PWM cycle when the count equals VAL1. If HALF is set, a reload opportunity occurs at the half cycle when the count equals VAL0. If both HALF and FULL are set, a reload opportunity occurs twice per PWM cycle when the count equals VAL1 and when it equals VAL0.

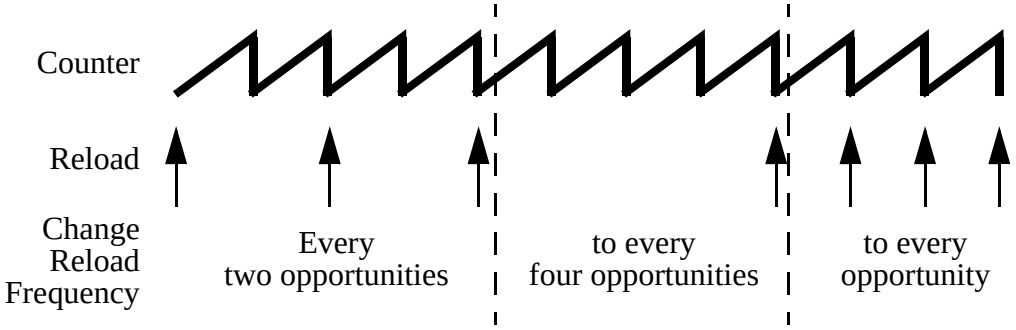


Figure 393. Full Cycle Reload Frequency Change

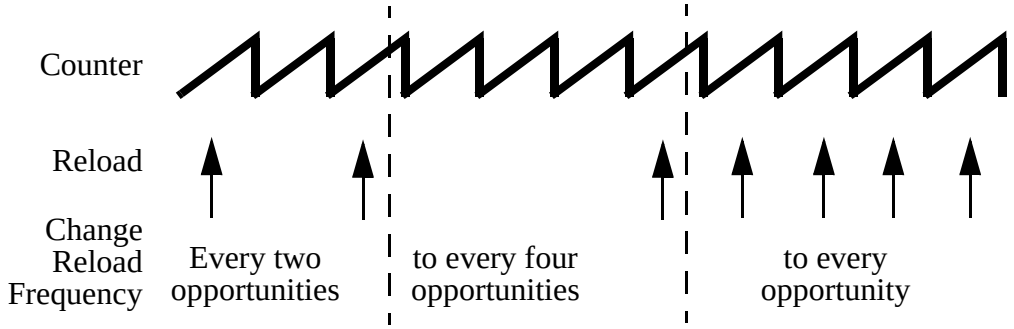


Figure 394. Half Cycle Reload Frequency Change

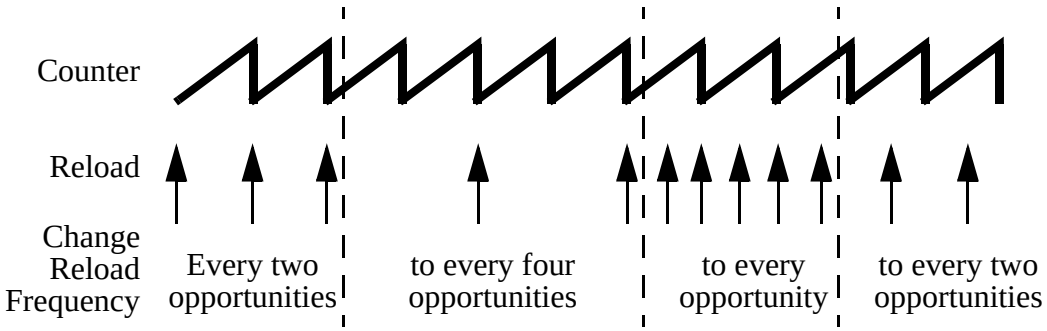


Figure 395. Full and Half Cycle Reload Frequency Change

26.3.4.3 Reload flag

At every reload opportunity the PWM Reload Flag (RF) in the CTRL1 register is set. Setting RF happens even if an actual reload is prevented by the LDOK bit. If the PWM reload interrupt enable bit, RIE is set, the RF flag generates CPU interrupt requests allowing software to calculate new PWM parameters in real time. When RIE is not set, reloads still occur at the selected reload rate without generating CPU interrupt requests.

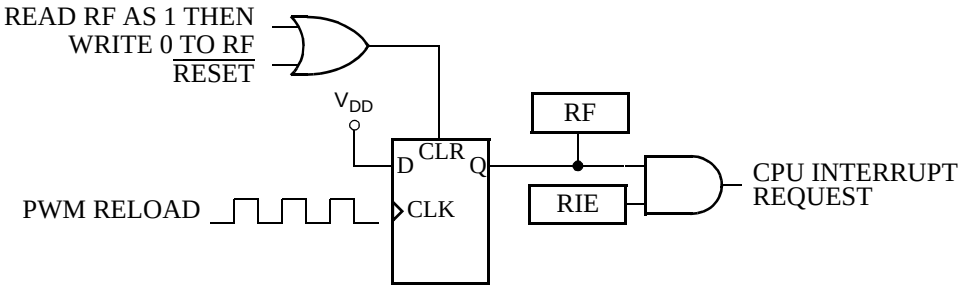


Figure 396. PWMF Reload Interrupt Request

26.3.4.4 Reload errors

Whenever one of the VALx or PSRC registers is updated, the RUF flag is set to indicate that the data is not coherent. RUF will be cleared by a successful reload which consists of the reload signal while LDOK is set. If RUF is set and LDOK is clear when the reload signal occurs, a reload error has taken place and the REF bit is set. If RUF is clear when a reload signal asserts, then the data is coherent and no error will be flagged.

26.3.4.5 Initialization

Initialize all registers and set the LDOK bit before setting the RUN bit.

Note: Even if LDOK is not set, setting RUN also sets the RF flag. To prevent a CPU interrupt request, clear the RIE bit before setting RUN.

The PWM generator uses the last values loaded if RUN is cleared and then set while LDOK equals zero.

When the RUN bit is cleared:

- The RF flag and pending CPU interrupt requests are not cleared
- All fault circuitry remains active
- Software/external output control remains active
- Deadtime insertion continues during software/external output control

26.4 Memory Map and Registers

26.4.1 Module Memory Map

Table 327. PWM Registers

Address	Reg Name	Description
PWM submodule registers (repeated for each submodule as SUB goes from 0 to 3)		
mcPWM_BASE + ($50 * SUB) + $00	CNT	Counter Register
mcPWM_BASE + ($50 * SUB) + $02	INIT	Initial Count Register
mcPWM_BASE + ($50 * SUB) + $04	CTRL2	Control 2 Register
mcPWM_BASE + ($50 * SUB) + $06	CTRL1	Control 1 Register
mcPWM_BASE + ($50 * SUB) + $08	VAL0	Value Register 0
mcPWM_BASE + ($50 * SUB) + $0A	VAL1	Value Register 1
mcPWM_BASE + ($50 * SUB) + $0C	VAL2	Value Register 2
mcPWM_BASE + ($50 * SUB) + $0E	VAL3	Value Register 3
mcPWM_BASE + ($50 * SUB) + $10	VAL4	Value Register 4
mcPWM_BASE + ($50 * SUB) + $12	VAL5	Value Register 5
mcPWM_BASE + ($50 * SUB) + $14	Reserved	
mcPWM_BASE + ($50 * SUB) + $16	Reserved	
mcPWM_BASE + ($50 * SUB) + $18	OCTRL	Output Control Register
mcPWM_BASE + ($50 * SUB) + $1A	STS	Status Register
mcPWM_BASE + ($50 * SUB) + $1C	INTEN	Interrupt Enable Register
mcPWM_BASE + ($50 * SUB) + $1E	DMAEN	DMA Enable Register
mcPWM_BASE + ($50 * SUB) + $20	TCTRL	Output Trigger Control Register
mcPWM_BASE + ($50 * SUB) + $22	DISMAP	Fault Disable Mapping Register
mcPWM_BASE + ($50 * SUB) + $24	DTCNT0	Deadtime Count Register 0
mcPWM_BASE + ($50 * SUB) + $26	DTCNT1	Deadtime Count Register 1
mcPWM_BASE + ($50 * SUB) + $28	Reserved	
mcPWM_BASE + ($50 * SUB) + $2A	Reserved	
mcPWM_BASE + ($50 * SUB) + $2C	Reserved	
mcPWM_BASE + ($50 * SUB) + $2E	Reserved	
mcPWM_BASE + ($50 * SUB) + $30	CAPTCTRLX	Capture Control Register X

Table 327. PWM Registers (continued)

Address	Reg Name	Description
mcPWM_BASE + ($50 * SUB) + $32	CAPTCOMPX	Capture Compare Register X
mcPWM_BASE + ($50 * SUB) + $34	CVAL0	Capture Value 0 Register
mcPWM_BASE + ($50 * SUB) + $36	CVAL0C	Capture Value 0 Cycle Register
mcPWM_BASE + ($50 * SUB) + $38	CVAL1	Capture Value 1 Register
mcPWM_BASE + ($50 * SUB) + $3A	CVAL1C	Capture Value 1 Cycle Register
mcPWM_BASE + ($50 * SUB) + $3C	Reserved	
mcPWM_BASE + ($50 * SUB) + $3E	Reserved	
mcPWM_BASE + ($50 * SUB) + $40	Reserved	
mcPWM_BASE + ($50 * SUB) + $42	Reserved	
mcPWM_BASE + ($50 * SUB) + $44	Reserved	
mcPWM_BASE + ($50 * SUB) + $46	Reserved	
mcPWM_BASE + ($50 * SUB) + $48	Reserved	
mcPWM_BASE + ($50 * SUB) + $4A	Reserved	
Configuration Logic Registers		
mcPWM_BASE + $140	OUTEN	Output Enable Register
mcPWM_BASE + $142	MASK	Output Mask Register
mcPWM_BASE + $144	SWCOUT	Software Controlled Output Register
mcPWM_BASE + $146	DTSRCSEL	Deadtime Source Select Register
mcPWM_BASE + $148	MCTRL	Master Control Register
Fault Channel registers		
mcPWM_BASE + $14C	FCTRL	Fault Control Register
mcPWM_BASE + $14E	FSTS	Fault Status Register
mcPWM_BASE + $150	FFILT	Fault Filter Register

26.4.2 Register Descriptions

The address of a register is the sum of a base address and an address offset. The base address is defined at the core level and the address offset is defined at the module level. There are a set of registers for each PWM submodule, for the configuration logic, and for each Fault channel.

26.4.3 Submodule Registers

These registers are repeated for each PWM submodule. The base address of submodule 0 is the same as the base address for the PWM module as a whole. The base address of submodule 1 is $50. This is the base address of the PWM module plus an offset based on the number of registers in a submodule. The base address of submodule 2 is equal to the base address of submodule 1 plus this same offset.

26.4.3.1 Counter Register (CNT)

PWM_SUB-_BASE+$0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	CNT															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 397. Counter Register (CNT)

This read-only register displays the state of the signed 16-bit submodule counter. This register is not byte accessible.

26.4.3.2 Initial Count Register (INIT)

PWM_SUB-_BASE+$2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	INIT															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 398. Initial Count Register (INIT)

The 16-bit signed value in this buffered, read/write register defines the initial count value for the PWM in PWM clock periods. This is the value loaded into the submodule counter when local sync, master sync, or master reload is asserted (based on the value of INIT_SEL) or when FORCE is asserted and force init is enabled. For PWM operation, the buffered contents of this register are loaded into the counter at the start of every PWM cycle. This register is not byte accessible.

Note: *The INIT register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. This register cannot be written when LDOK is set. Reading INIT reads the value in a buffer and not necessarily the value the PWM generator is currently using.*

26.4.3.3 Control 2 Register (CTRL2)

PWM_SUB-_BASE+$4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	DBGEN	WAITEN	INDEP	PWM23_INIT	PWM45_INIT	PWMX_INIT	INIT_SEL		FRCEN	0	FORCE_SEL			RELOAD_SEL	CLK_SEL	
Write										FORCE						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 399. Control 2 Register (CTRL2)

DBGEN — Debug Enable

When set to one, the PWM will continue to run while the chip is in debug mode. If the device enters debug mode and this bit is zero, then the PWM outputs will be disabled

until debug mode is exited. At that point the PWM pins will resume operation as programmed in the PWM registers.

For certain types of motors (such as 3-phase AC), it is imperative that this bit be left in its default state (in which the PWM is disabled in debug mode). Failure to do so could result in damage to the motor or inverter. For other types of motors (example: DC motors), this bit might safely be set to one, enabling the PWM in debug mode. The key point is PWM parameter updates will not occur in debug mode. Any motors requiring such updates should be disabled during debug mode. If in doubt, leave this bit set to zero.

WAITEN — WAIT Enable

When set to one, the PWM will continue to run while the chip is in WAIT mode. In this mode, the peripheral clock continues to run but the CPU clock does not. If the device enters WAIT mode and this bit is zero, then the PWM outputs will be disabled until WAIT mode is exited. At that point the PWM pins will resume operation as programmed in the PWM registers.

For certain types of motors (such as 3-phase AC), it is imperative that this bit be left in its default state (in which the PWM is disabled in WAIT mode). Failure to do so could result in damage to the motor or inverter. For other types of motors (example: DC motors), this bit might safely be set to one, enabling the PWM in WAIT mode. The key point is PWM parameter updates will not occur in this mode. Any motors requiring such updates should be disabled during WAIT mode. If in doubt, leave this bit set to zero.

INDEP — Independent or Complementary Pair Operation

This bit determines if the PWMA and PWMB channels will be independent PWMs or a complementary PWM pair.

1 = PWMA and PWMB outputs are independent PWMs.
0 = PWMA and PWMB form a complementary PWM pair.

PWM23_INIT — PWM23 Initial Value

This read/write bit determines the initial value for PWM23 and the value to which it is forced when FORCE_INIT (*Figure 378*) is asserted.

PWM45_INIT — PWM45 Initial Value

This read/write bit determines the initial value for PWM45 and the value to which it is forced when FORCE_INIT (*Figure 378*) is asserted.

PWMX_INIT — PWMX Initial Value

This read/write bit determines the initial value for PWMX and the value to which it is forced when FORCE_INIT (*Figure 378*) is asserted.

INIT_SEL — Initialization Control Select

These read/write bits control the source of the INIT signal which goes to the counter.

- 00 = Local sync (PWMX) causes initialization.
- 01 = Master reload from submodule 0 causes initialization. This setting should not be used in submodule 0 as it will force the INIT signal to logic 0.
- 10 = Master sync from submodule 0 causes initialization. This setting should not be used in submodule 0 as it will force the INIT signal to logic 0.
- 11 = EXT_SYNC causes initialization.

FRCEN — Force Initialization Enable

This bit allows the FORCE_OUT signal to initialize the counter without regard to the signal selected by INIT_SEL. This is a software controlled initialization.

1 = Initialization from a Force Out event is enabled.
0 = Initialization from a Force Out event is disabled.

FORCE — Force Initialization

If the FORCE_SEL bits are set to 000, writing a 1 to this bit results in a Force Out event. This causes the following actions to be taken:

- The PWMA and PWMB output pins will assume values based on the SEL23 and SEL45 bits.
- If the FRCEN bit is set, the counter value will be initialized with the INIT register value.

FORCE_SEL — Force Source Select

This read/write bit determines the source of the FORCE OUTPUT signal for this submodule.

- 000 = The local force signal, FORCE, from this submodule is used to force updates.
- 001 = The master force signal from submodule 0 is used to force updates. This setting should not be used in submodule 0 as it will hold the FORCE OUTPUT signal to logic 0.
- 010 = The local reload signal from this submodule is used to force updates.
- 011 = The master reload signal from submodule0 is used to force updates. This setting should not be used in submodule0 as it will hold the FORCE OUTPUT signal to logic 0.
- 100 = The local sync signal from this submodule is used to force updates.
- 101 = The master sync signal from submodule0 is used to force updates. This setting should not be used in submodule0 as it will hold the FORCE OUTPUT signal to logic 0.
- 110 = The external force signal, EXT_FORCE, from outside the PWM module causes updates.
- 111 = reserved

RELOAD_SEL — Reload Source Select

This read/write bit determines the source of the RELOAD signal for this submodule. When this bit is set, the LDOK bit in submodule 0 should be used since the local LDOK bit will be ignored.

1 = The master RELOAD signal (from submodule 0) is used to reload registers. This setting should not be used in submodule 0 as it will force the RELOAD signal to logic 0.
0 = The local RELOAD signal is used to reload registers.

CLK_SEL — Clock Source Select

These read/write bits determine the source of the clock signal for this submodule.

- 00 = The IPBus clock is used as the clock for the local prescaler and counter.
- 01 = EXT_CLK is used as the clock for the local prescaler and counter.
- 10 = Submodule 0's clock (AUX_CLK) is used as the source clock for the local prescaler and counter. This setting should not be used in submodule 0 as it will force the clock to logic 0.
- 11 = reserved

26.4.3.4 Control 1 Register (CTRL1)

PWM_SUB-_BASE+$6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	LDFQ				HALF	FULL	DT		0	PRSC			0	LDMOD	0	DBLEN
Write																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 400. Control 1 Register (CTRL1)

LDFQ — Load Frequency

These buffered read/write bits select the PWM load frequency according to *Table 328*. Reset clears the LDFQ bits, selecting loading every PWM opportunity. A PWM opportunity is determined by the HALF and FULL bits.

Note: *The LDFQx bits take effect when the current load cycle is complete regardless of the state of the LDOK bit. Reading the LDFQx bits reads the buffered values and not necessarily the values currently in effect.*

Table 328. PWM Reload Frequency

LDFQ	PWM reload frequency
0000	Every PWM opportunity
0001	Every 2 PWM opportunities
0010	Every 3 PWM opportunities
0011	Every 4 PWM opportunities
0100	Every 5 PWM opportunities
0101	Every 6 PWM opportunities
0110	Every 7 PWM opportunities
0111	Every 8 PWM opportunities
1000	Every 9 PWM opportunities
1001	Every 10 PWM opportunities
1010	Every 11 PWM opportunities
1011	Every 12 PWM opportunities
1100	Every 13 PWM opportunities

Table 328. PWM Reload Frequency

LDFQ	PWM reload frequency
1101	Every 14 PWM opportunities
1110	Every 15 PWM opportunities
1111	Every 16 PWM opportunities

HALF — Half Cycle Reload

This read/write bit enables half-cycle reloads. A half cycle is defined by when the submodule counter matches the VAL0 register and does not have to be half way through the PWM cycle.

1 = Half-cycle reloads enabled.
0 = Half-cycle reloads disabled.

FULL — Full Cycle Reload

This read/write bit enables full-cycle reloads. A full cycle is defined by when the submodule counter matches the VAL1 register. Either the HALF or FULL bit must be set in order to move the buffered data into the registers used by the PWM generators. If both the HALF and FULL bits are set, then reloads can occur twice per cycle.

1 = Full-cycle reloads enabled.
0 = Full-cycle reloads disabled.

DT — Deadtime

These read only bits reflect the sampled values of the PWMX input at the end of each deadtime. Sampling occurs at the end of deadtime 0 for DT[0] and the end of deadtime 1 for DT[1]. Reset clears these bits.

PRSC — Prescaler

These buffered read/write bits select the divide ratio of the PWM clock frequency selected by CLK_SEL as illustrated in *Table 329*.

Table 329. PWM Prescaler

PRSC	PWM clock frequency
000	f_{clk}
001	$f_{clk}/2$
010	$f_{clk}/4$
011	$f_{clk}/8$
100	$f_{clk}/16$
101	$f_{clk}/32$
110	$f_{clk}/64$
111	$f_{clk}/128$

Note: *Reading the PRSCx bits reads the buffered values and not necessarily the values currently in effect. The PRSCx bits take effect at the beginning of the next PWM cycle and only when*

the load okay bit, LDOK, is set or LDMOD is set. This field cannot be written when LDOK is set.

LDMOD — Load Mode Select

This read/write bit selects the timing of loading the buffered registers for this submodule.

1 = Buffered registers of this submodule are loaded and take effect immediately upon LDOK being set.
0 = Buffered registers of this submodule are loaded and take effect at the next PWM reload if LDOK is set.

DBLEN — Double Switching Enable

This read/write bit enables the double switching PWM behavior.

1 = Double switching enabled.
0 = Double switching disabled.

26.4.3.5 Value Register 0 (VAL0)

PWM_SUB-_BASE+$8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Write	VAL0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 401. Value Register 0 (VAL0)

The 16-bit signed value in this buffered, read/write register defines the mid-cycle reload point for the PWM in PWM clock periods. This value also defines when the PWMX signal is set and the local sync signal is reset. This register is not byte accessible.

Note: *The VAL0 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL0 cannot be written when LDOK is set. Reading VAL0 reads the value in a buffer. It is not necessarily the value the PWM generator is currently using.*

26.4.3.6 Value Register 1 (VAL1)

PWM_SUB-_BASE+$A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Write	VAL1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 402. Value Register 1 (VAL1)

The 16-bit signed value written to this buffered, read/write register defines the modulo count value (maximum count) for the submodule counter. Upon reaching this count value, the counter will reload itself with the contents of the INIT register and assert the local sync signal while resetting PWMX. This register is not byte accessible.

Note: *If VAL1 register defines the timer period (Local Sync is selected as the counter initialization signal), a 100% duty cycle cannot be achieved on the PWMX output. The PWMX output will be low for a minimum of one count every cycle after the count reaches VAL1.*

When the Master Sync signal (only originated by the Local Sync from sub-module 0) is used to control the timer period, the VAL1 register can be free for other functions such as PWM.

26.4.3.7 Value Register 2 (VAL2)

PWM_SUB-_BASE+$C	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Write	VAL2															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 403. Value Register 2 (VAL2)

The 16-bit signed value in this buffered, read/write register defines the count value to set PWM23 high (*Figure 364*). This register is not byte accessible.

Note: *The VAL2 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL2 cannot be written when LDOK is set. Reading VAL2 reads the value in a buffer and not necessarily the value the PWM generator is currently using.*

26.4.3.8 Value Register 3 (VAL3)

PWM_SUB-_BASE+$E	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Write	VAL3															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 404. Value Register 3 (VAL3)

The 16-bit signed value in this buffered, read/write register defines the count value to set PWM23 low (*Figure 364*). This register is not byte accessible.

Note: *The VAL3 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL3 cannot be written when LDOK is set. Reading VAL3 reads the value in a buffer and not necessarily the value the PWM generator is currently using.*

26.4.3.9 Value Register 4 (VAL4)

PWM_SUB-_BASE+$10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Write	VAL4															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 405. Value Register 4 (VAL4)

The 16-bit signed value in this buffered, read/write register defines the count value to set PWM45 high (*Figure 364*). This register is not byte accessible.

Note: The VAL4 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL4 cannot be written when LDOK is set. Reading VAL4 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

26.4.3.10 Value Register 5 (VAL5)

PWM_SUB-_BASE+$12	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Write	VAL5															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 406. Value Register 5 (VAL5)

The 16-bit signed value in this buffered, read/write register defines the count value to set PWM45 low (*Figure 364*). This register is not byte accessible.

Note: The VAL5 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL5 cannot be written when LDOK is set. Reading VAL5 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

26.4.3.11 Output Control Register (OCTRL)

PWM_-SUB-_BASE+$18	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	PWMA_IN	PWMB_IN	PWMX_IN	0	0	POLA	POLB	POLX	0	0	PWMAFS		PWMBFS		PWMXFS	
Write																
Reset	U	U	U	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 407. Output Control Register (OCTRL)

PWMA_IN — PWMA Input

This read only bit shows the logic value currently being driven into the PWMA input.

PWMB_IN — PWMB Input

This read only bit shows the logic value currently being driven into the PWMB input.

PWMX_IN — PWMX Input

This read only bit shows the logic value currently being driven into the PWMX input.

POLA — PWMA Output Polarity

This bit inverts the PWMA output polarity.

1 = PWMA output inverted. A low level on the PWMA pin represents the "on" or "active" state.
0 = PWMA output not inverted. A high level on the PWMA pin represents the "on" or "active" state.

POLB — PWMB Output Polarity

This bit inverts the PWMB output polarity.

1 = PWMB output inverted. A low level on the PWMB pin represents the "on" or "active" state.
0 = PWMB output not inverted. A high level on the PWMB pin represents the "on" or "active" state.

POLX — PWMX Output Polarity

This bit inverts the PWMX output polarity.

1 = PWMX output inverted. A low level on the PWMX pin represents the "on" or "active" state.
0 = PWMX output not inverted. A high level on the PWMX pin represents the "on" or "active" state.

PWMAFS — PWMA Fault State

These bits determine the fault state for the PWMA output during fault conditions and STOP mode. It may also define the output state during WAIT and DEBUG modes depending on the settings of WAITEN and DBGEN.

- 00 = Output is forced to logic 0 state prior to consideration of output polarity control.
- 01 = Output is forced to logic 1 state prior to consideration of output polarity control.
- 1x = Output is tristated.

PWMBFS — PWMB Fault State

These bits determine the fault state for the PWMB output during fault conditions and STOP mode. It may also define the output state during WAIT and DEBUG modes depending on the settings of WAITEN and DBGEN.

- 00 = Output is forced to logic 0 state prior to consideration of output polarity control.
- 01 = Output is forced to logic 1 state prior to consideration of output polarity control.
- 1x = Output is tristated.

PWMXFS — PWMX Fault State

These bits determine the fault state for the PWMX output during fault conditions and STOP mode. It may also define the output state during WAIT and DEBUG modes depending on the settings of WAITEN and DBGEN.

- 00 = Output is forced to logic 0 state prior to consideration of output polarity control.
- 01 = Output is forced to logic 1 state prior to consideration of output polarity control.
- 1x = Output is tristated.

26.4.3.12 Status Register (STS)

PWM_SUB-_BASE+$1A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	RUF	REF	RF	0	0	0	0	CFX1	CFX0	CMPF					
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 408. Status Register (STS)

RUF — Registers Updated Flag

This read only flag is set when one of the INIT, VALx, or PRSC registers has been written resulting in non-coherent data in the set of double buffered registers. Clear RUF by a proper reload sequence consisting of a reload signal while LDOK = 1. Reset clears RUF.

1 = At least one of the double buffered registers has been updated since the last reload.
0 = No register update has occurred since last reload.

REF — Reload Error Flag

This read/write flag is set when a reload cycle occurs while LDOK is 0 and the double buffered registers are in a non-coherent state (RUF = 1). Clear REF by writing a logic one to the REF bit. Reset clears REF.

1 = Reload signal occurred with non-coherent data and LDOK = 0.
0 = No reload error occurred.

RF — Reload Flag

This read/write flag is set at the beginning of every reload cycle regardless of the state of the LDOK bit. Clear RF by writing a logic one to the RF bit when VALDE is clear (non-DMA mode). RF can also be cleared by the DMA done signal when VALDE is set (DMA mode). Reset clears RF.

1 = New reload cycle since last RF clearing
0 = No new reload cycle since last RF clearing

CFX1 — Capture Flag X1

This bit is set when the word count of the Capture X1 FIFO (CX1CNT) exceeds the value of the CFXWM field. This bit is cleared by writing a one to this bit position if CX1DE is clear (non-DMA mode) or by the DMA done signal if CX1DE is set (DMA mode). Reset clears this bit.

CFX0 — Capture Flag X0

This bit is set when the word count of the Capture X0 FIFO (CX0CNT) exceeds the value of the CFXWM field. This bit is cleared by writing a one to this bit position if CX0DE is clear (non-DMA mode) or by the DMA done signal if CX0DE is set (DMA mode). Reset clears this bit.

CMPF — Compare Flags

These bits are set when the submodule counter value matches the value of one of the VALx registers. Clear these bits by writing a 1 to a bit position. Bits [5:0] of this field correspond to VAL5, VAL4, VAL3, VAL2, VAL1, and VAL0, respectively.

1 = A compare event has occurred for a particular VALx value.
0 = No compare event has occurred for a particular VALx value.

26.4.3.13 Interrupt Enable Register (INTEN)

PWM_SUB-_BASE+$1C	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	REIE	RIE	0	0	0	0	CX1IE	CX0IE	CMPIE					
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 409. Interrupt Enable Register (INTEN)

REIE — Reload Error Interrupt Enable

This read/write bit enables the reload error flag (REF) to generate CPU interrupt requests. Reset clears RIE.

1 = REF CPU interrupt requests enabled
0 = REF CPU interrupt requests disabled

RIE — Reload Interrupt Enable

This read/write bit enables the reload flag (RF) to generate CPU interrupt requests. Reset clears RIE.

1 = RF CPU interrupt requests enabled
0 = RF CPU interrupt requests disabled

CX1IE — Capture X 1 Interrupt Enable

This bit allows the CFX1 flag to create an interrupt request to the CPU. Do not set both this bit and the CX1DE bit.

1 = Interrupt request enabled for CFX1.
0 = Interrupt request disabled for CFX1.

CX0IE — Capture X 0 Interrupt Enable

This bit allows the CFX0 flag to create an interrupt request to the CPU. Do not set both this bit and the CX0DE bit.

1 = Interrupt request enabled for CFX0.
0 = Interrupt request disabled for CFX0.

CMPIE — Compare Interrupt Enables

These bits enable the CMPF flags to cause a compare interrupt request to the CPU. Bits [5:0] of this field correspond to VAL5, VAL4, VAL3, VAL2, VAL1, and VAL0, respectively.

1 = The corresponding CMPF bit will cause an interrupt request.

0 = The corresponding CMPF bit will not cause an interrupt request.

26.4.3.14 DMA Enable Register (DMAEN)

PWM_SUB-_BASE+$1E	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	0	0	VALDE	FAND	CAPTDE		0	0	0	0	CX1DE	CX0DE
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 410. DMA Enable Register (DMAEN)

VALDE — Value Registers DMA Enable

This read/write bit enables DMA write requests for the VALx register when RF is set. Reset clears VALDE.

1 = DMA write requests for the VALx register enabled
0 = DMA write requests disabled

FAND — FIFO Watermark AND Control

This read/write bit works in conjunction with the CAPTDE field when it is set to watermark mode (CAPTDE = 01). While the CXxDE bits determine which FIFO watermarks the DMA read request is sensitive to, this bit determines if the selected watermarks are AND'ed together or OR'ed together in order to create the request.

1 = Selected FIFO watermarks are AND'ed together.
0 = Selected FIFO watermarks are OR'ed together.

CAPTDE — Capture DMA Enable Source Select

These read/write bits select the source of enabling the DMA read requests for the capture FIFOs. Reset clears these bits.

- 00 = Read DMA requests disabled.
- 01 = Exceeding a FIFO watermark sets the DMA read request. This requires at least 1 of the CX1DE or CX0DE bits to also be set in order to determine which watermark(s) the DMA request is sensitive to.
- 10 = A local sync (VAL1 matches counter) sets the read DMA request.
- 11 = A local reload (RF being set) sets the read DMA request.

CX1DE — Capture X1 FIFO DMA Enable

This read/write bit enables DMA read requests for the Capture X1 FIFO data when CFX1 is set. Reset clears CX1DE. Do not set both this bit and the CX1IE bit.

CX0DE — Capture X0 FIFO DMA Enable

This read/write bit enables DMA read requests for the Capture X0 FIFO data when CFX0 is set. Reset clears CX0DE. Do not set both this bit and the CX0IE bit.

26.4.3.15 Output Trigger Control Register (TCTRL)

PWM_SUB-_BASE+$20	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	0	0	0	0	0	0	OUT_TRIG_EN					
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 411. Output Trigger Control Register (TCTRL)

OUT_TRIG_EN — Output Trigger Enables

These bits enable the generation of OUT_TRIG0 and OUT_TRIG1 outputs based on the counter value matching the value in one or more of the VAL0-5 registers. VAL0, VAL2, and VAL4 are used to generate OUT_TRIG0 and VAL1, VAL3, and VAL5 are used to generate OUT_TRIG1. The OUT_TRIGx signals are only asserted as long as the counter value matches the VALx value, therefore up to six triggers can be generated (three each on OUT_TRIG0 and OUT_TRIG1) per PWM cycle per submodule. Bits [5:0] of this field correspond to VAL5, VAL4, VAL3, VAL2, VAL1, and VAL0, respectively.

1 = OUT_TRIGx will set when the counter value matches the VALx value.
0 = OUT_TRIGx will not set when the counter value matches the VALx value.

26.4.3.16 Fault Disable Mapping Register (DISMAP)

This register determines which PWM pins are disabled by the fault protection inputs, illustrated in *Table 389* in *Section 26.3.3.11, "Fault protection*. Reset sets all of the bits in the fault disable mapping register.

PWM_SUB-_BASE+$22	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	1	1	1	1	DISX				DISB				DISA			
Write																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 412. Fault Disable Mapping Register (DISMAP)

Bits [3:0] of the fields in this register correspond to FAULT3, FAULT2, FAULT1, and FAULT0, respectively.

DISX — PWMX Fault Disable Mask

Each of the four bit of this read/write field is one-to-one associated with the four FAULTx inputs. The PWMX output will be turned off if there is a logic 1 on a FAULTx input and a 1 in the corresponding bit of the DISX field. A reset sets all DISX bits.

DISB — PWMB Fault Disable Mask

Each of the four bit of this read/write field is one-to-one associated with the four FAULTx inputs. The PWMB output will be turned off if there is a logic 1 on a FAULTx input and a 1 in the corresponding bit of the DISB field. A reset sets all DISB bits.

DISA — PWMA Fault Disable Mask

Each of the four bit of this read/write field is one-to-one associated with the four FAULTx inputs. The PWMA output will be turned off if there is a logic 1 on a FAULTx input and a 1 in the corresponding bit of the DISA field. A reset sets all DISA bits.

26.4.3.17 Deadtime Count Registers (DTCNT0, DTCNT1)

Deadtime operation is only applicable to complementary channel operation. The 11-bit values written to these registers are in terms of IPBus clock cycles regardless of the setting of PRSC and/or CLK_SEL. Reset sets the deadtime count registers to a default value of 0x07FF, selecting a deadtime of 2047 IPBus clock cycles. These registers are not byte accessible.

PWM_SUB-_BASE+$24	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	0	DTCNT0										
Write																
Reset	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1

Figure 413. Deadtime Count Register 0 (DTCNT0)

PWM_SUB-_BASE+$26	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	0	DTCNT1										
Write																
Reset	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1

Figure 414. Deadtime Count Register 1 (DTCNT1)

The DTCNT0 field is used to control the deadtime during 0 to 1 transitions of the PWMA output (assuming normal polarity). The DTCNT1 field is used to control the deadtime during 0 to 1 transitions of the complementary PWMB output.

26.4.3.18 Capture Control X Register (CAPTCTRLX)

PWM_SUB-_BASE+$30	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	CX1CNT			CX0CNT			CFXWM		EDGCNTX_EN	INP_SELX	EDGX1		EDGX0		ONESHOTX	ARMX
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 415. Capture Control X Register (CAPTCTRLX)

CX1CNT — Capture X1 FIFO Word Count

This field reflects the number of words in the Capture X1 FIFO.

CX0CNT — Capture X0 FIFO Word Count

This field reflects the number of words in the Capture X0 FIFO.

CFXWM — Capture X FIFOs Water Mark

This field represents the water mark level for capture X FIFOs. The capture flags, CFX1 and CFX0, won't be set until the word count of the corresponding FIFO is greater than this water mark level.

EDGCNTX_EN — Edge Counter X Enable

This bit enables the edge counter which counts rising and falling edges on the PWMX input signal.

1 = Edge counter enabled.
0 = Edge counter disabled and held in reset.

INPSELX — Input Select X

This bit selects between the raw PWMX input signal and the output of the edge counter/compare circuitry as the source for the input capture circuit.

1 = Output of edge counter/compare selected as source.
0 = Raw PWMX input signal selected as source.

Note: When INPSELX = 1, the internal edge counter is enabled and the rising and/or falling edges specified by the EDGX0 and EDGX1 fields are ignored. The software must still place a value other than 00 in either or both of the EDGX0 and/or EDGX1 fields in order to enable one or both of the capture registers.

EDGX1 — Edge X 1

These bits control the input capture 1 circuitry by determining which input edges cause a capture event.

- 00 = Disabled.
- 01 = Capture falling edges.
- 10 = Capture rising edges.
- 11 = Capture any edge.

EDGX0 — Edge X 0

These bits control the input capture 0 circuitry by determining which input edges cause a capture event.

- 00 = Disabled.
- 01 = Capture falling edges.
- 10 = Capture rising edges.
- 11 = Capture any edge.

ONESHOTX — One Shot Mode Aux

This bit selects between free running and one shot mode for the input capture circuitry.

1 = One shot mode is selected.

If both capture circuits are enabled, then capture circuit 0 is armed first after the ARMX bit is set. Once a capture occurs, capture circuit 0 is disarmed and capture

circuit 1 is armed. After capture circuit 1 performs a capture, it is disarmed and the ARMX bit is cleared. No further captures will be performed until the ARMX bit is set again.

If only one of the capture circuits is enabled, then a single capture will occur on the enabled capture circuit and the ARMX bit is then cleared.

0 = Free running mode is selected

If both capture circuits are enabled, then capture circuit 0 is armed first after the ARMX bit is set. Once a capture occurs, capture circuit 0 is disarmed and capture circuit 1 is armed. After capture circuit 1 performs a capture, it is disarmed and capture circuit 0 is re-armed. The process continues indefinitely.

If only one of the capture circuits is enabled, then captures continue indefinitely on the enabled capture circuit.

ARMX — Arm X

Setting this bit high starts the input capture process. This bit can be cleared at any time to disable input capture operation. This bit is self cleared when in one shot mode and the enabled capture circuit(s) has had a capture event(s).

1 = Input capture operation as specified by the EDGX bits is enabled.
0 = Input capture operation is disabled.

26.4.3.19 Capture Compare X Register (CAPTCMPX)

PWM_SUB-_BASE+$32	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	EDGCNTX								EDGCMPX							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 416. Capture Compare X Register (CAPTCMPX)

EDGCNTX — Edge Counter X

This read only field contains the edge counter value for the PWMX input capture circuitry.

EDGCMPX — Edge Compare X

This read/write field is the compare value associated with the edge counter for the PWMX input capture circuitry.

26.4.3.20 Capture Value 0 Register (CVAL0)

PWM_SUB-_BASE+$34	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	CAPTVAL0															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 417. Capture Value 0 Register (CVAL0)

This read only register stores the value captured from the submodule counter. Exactly when this capture occurs is defined by the EDGX0 bits. This is actually a 4 deep FIFO and not a single register. This register is not byte accessible.

26.4.3.21 Capture Value 0 Cycle Register (CVAL0CYC)

PWM_SUB-_BASE+$36	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	0	0	0	0	0	0	0	0	0	CVAL0CYC		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 418. Capture Value 0 Cycle Register (CVAL0CYC)

This read only register stores the cycle number corresponding to the value captured in CVAL0. The PWM cycle is reset to 0 and is incremented each time the counter is loaded with the INIT value. This is actually a 4 deep FIFO and not a single register.

26.4.3.22 Capture Value 1 Register (CVAL1)

PWM_SUB-_BASE+$38	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	CAPTVAL1															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 419. Capture Value 1 Register (CVAL1)

This read only register stores the value captured from the submodule counter. Exactly when this capture occurs is defined by the EDGX1 bits. This is actually a 4 deep FIFO and not a single register. This register is not byte accessible.

26.4.3.23 Capture Value 1 Cycle Register (CVAL1CYC)

PWM_SUB-_BASE+$3A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	0	0	0	0	0	0	0	0	0	CVAL1CYC		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 420. Capture Value 1 Cycle Register (CVAL1CYC)

This read only register stores the cycle number corresponding to the value captured in CVAL1. The PWM cycle is reset to 0 and is incremented each time the counter is loaded with the INIT value. This is actually a 4 deep FIFO and not a single register.

26.4.4 Configuration registers

The base address of the configuration registers is equal to the base address of the mcPWM plus as offset of $140.

26.4.4.1 Output Enable Register (OUTEN)

PWM_BASE+$140	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	PWMA_EN				PWMB_EN				PWMX_EN			
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 421. Output Enable Register (OUTEN)

Note: *In Figure 421 (Output Enable Register (OUTEN)), Field[3:0] refers to submodule 3,2,1,0.*

PWMA_EN — PWMA Output Enables

These bits enable the PWMA outputs of each submodule. These bits should be set to 0 (output disabled) when a PWMA pin is being used for input capture.

1 = PWMA output enabled.
0 = PWMA output disabled.

PWMB_EN — PWMB Output Enables

These bits enable the PWMB outputs of each submodule. These bits should be set to 0 (output disabled) when a PWMB pin is being used for input capture.

1 = PWMB output enabled.
0 = PWMB output disabled.

PWMX_EN — PWMX Output Enables

These bits enable the PWMX outputs of each submodule. These bits should be set to 0 (output disabled) when a PWMX pin is being used for input capture or deadtime correction.

1 = PWMX output enabled.
0 = PWMX output disabled.

26.4.4.2 Mask Register (MASK)

PWM_BASE+$142	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	MASKA				MASKB				MASKX			
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 422. Mask Register (MASK)

Note: *In Figure 422 (Mask Register (MASK)), Field[3:0] refers to submodule 3,2,1,0.*

Note: The MASKx bits are double buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 379 to see how FORCE_OUT is generated. Reading the MASK bits reads the buffered value and not necessarily the value currently in effect.

MASKA — PWMA Masks

These bits mask the PWMA outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity.

1 = PWMA output masked.
0 = PWMA output normal.

MASKB — PWMB Masks

These bits mask the PWMB outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity.

1 = PWMB output masked.
0 = PWMB output normal.

MASKX — PWMX Masks

These bits mask the PWMX outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity.

1 = PWMX output masked.
0 = PWMX output normal.

26.4.4.3 Software Controlled Output Register (SWCOUT)

PWM_BASE+$144	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	0	0	0	0	0	OUT23_3	OUT45_3	OUT23_2	OUT45_2	OUT23_1	OUT45_1	OUT23_0	OUT45_0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 423. Software Controlled Output Register (SWCOUT)

Note: *These bits are double buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 379 to see how FORCE_OUT is generated. Reading these bits reads the buffered value and not necessarily the value currently in effect.*

OUT23_3 — Software Controlled Output 23_3

This bit is only used when SEL23 for submodule 3 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule.

1 = A logic 1 is supplied to the deadtime generator of submodule 3 instead of PWM23.
0 = A logic 0 is supplied to the deadtime generator of submodule 3 instead of PWM23.

OUT45_3 — Software Controlled Output 45_3

This bit is only used when SEL45 for submodule 3 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule.

1 = A logic 1 is supplied to the deadtime generator of submodule 3 instead of PWM45.
0 = A logic 0 is supplied to the deadtime generator of submodule 3 instead of PWM45.

OUT23_2 — Software Controlled Output 23_2

This bit is only used when SEL23 for submodule 2 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule.

1 = A logic 1 is supplied to the deadtime generator of submodule 2 instead of PWM23.
0 = A logic 0 is supplied to the deadtime generator of submodule 2 instead of PWM23.

OUT45_2 — Software Controlled Output 45_2

This bit is only used when SEL45 for submodule 2 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule.

1 = A logic 1 is supplied to the deadtime generator of submodule 2 instead of PWM45.
0 = A logic 0 is supplied to the deadtime generator of submodule 2 instead of PWM45.

OUT23_1 — Software Controlled Output 23_1

This bit is only used when SEL23 for submodule 1 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule.

1 = A logic 1 is supplied to the deadtime generator of submodule 1 instead of PWM23.
0 = A logic 0 is supplied to the deadtime generator of submodule 1 instead of PWM23.

OUT45_1 — Software Controlled Output 45_1

This bit is only used when SEL45 for submodule 1 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule.

1 = A logic 1 is supplied to the deadtime generator of submodule 1 instead of PWM45.
0 = A logic 0 is supplied to the deadtime generator of submodule 1 instead of PWM45.

OUT23_0 — Software Controlled Output 23_0

This bit is only used when SEL23 for submodule 0 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule.

1 = A logic 1 is supplied to the deadtime generator of submodule 0 instead of PWM23.
0 = A logic 0 is supplied to the deadtime generator of submodule 0 instead of PWM23.

OUT45_0 — Software Controlled Output 45_0

This bit is only used when SEL45 for submodule 0 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule.

1 = A logic 1 is supplied to the deadtime generator of submodule 0 instead of PWM45.
0 = A logic 0 is supplied to the deadtime generator of submodule 0 instead of PWM45.

26.4.4.4 Deadtime Source Select Register (DTSRCSEL)

PWM_BASE+$146	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Write	SEL23_3		SEL45_3		SEL23_2		SEL45_2		SEL23_1		SEL45_1		SEL23_0		SEL45_0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 424. Deadtime Source Select Register (DTSRCSEL)

Note: *The deadtime source select bits are double buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 379 to see how FORCE_OUT is generated. Reading these bits reads the buffered value and not necessarily the value currently in effect.*

SEL23_3 — PWM23_3 Control Select

This field selects possible over-rides to the generated PWM23 signal in submodule 3 that will be passed to the deadtime logic upon the occurrence of a "Force Out" event in that submodule.

- 00 = Generated PWM23_3 signal is used by the deadtime logic.
- 01 = Inverted generated PWM23_3 signal is used by the deadtime logic.
- 10 = OUT23_3 bit is used by the deadtime logic.
- 11 = EXTA[3] signal is used by the deadtime logic.

SEL45_3 — PWM45_3 Control Select

This field selects possible over-rides to the generated PWM45 signal in submodule 3 that will be passed to the deadtime logic upon the occurrence of a "Force Out" event in that submodule.

- 00 = Generated PWM45_3 signal is used by the deadtime logic.
- 01 = Inverted generated PWM45_3 signal is used by the deadtime logic.
- 10 = OUT45_3 bit is used by the deadtime logic.
- 11 = EXTB[3] signal is used by the deadtime logic.

SEL23_2 — PWM23_2 Control Select

This field selects possible over-rides to the generated PWM23 signal in submodule 2 that will be passed to the deadtime logic upon the occurrence of a "Force Out" event in that submodule.

- 00 = Generated PWM23_2 signal is used by the deadtime logic.
- 01 = Inverted generated PWM23_2 signal is used by the deadtime logic.
- 10 = OUT23_2 bit is used by the deadtime logic.
- 11 = EXTA[2] signal is used by the deadtime logic.

SEL45_2 — PWM45_2 Control Select

This field selects possible over-rides to the generated PWM45 signal in submodule 2 that will be passed to the deadtime logic upon the occurrence of a "Force Out" event in that submodule.

- 00 = Generated PWM45_2 signal is used by the deadtime logic.
- 01 = Inverted generated PWM45_2 signal is used by the deadtime logic.
- 10 = OUT45_2 bit is used by the deadtime logic.
- 11 = EXTB[2] signal is used by the deadtime logic.

SEL23_1 — PWM23_1 Control Select

This field selects possible over-rides to the generated PWM23 signal in submodule 1 that will be passed to the deadtime logic upon the occurrence of a "Force Out" event in that submodule.

- 00 = Generated PWM23_1 signal is used by the deadtime logic.
- 01 = Inverted generated PWM23_1 signal is used by the deadtime logic.

– 10 = OUT23_1 bit is used by the deadtime logic.
– 11 = EXTA[1] signal is used by the deadtime logic.

SEL45_1 — PWM45_1 Control Select

This field selects possible over-rides to the generated PWM45 signal in submodule 1 that will be passed to the deadtime logic upon the occurrence of a "Force Out" event in that submodule.

– 00 = Generated PWM45_1 signal is used by the deadtime logic.
– 01 = Inverted generated PWM45_1 signal is used by the deadtime logic.
– 10 = OUT45_1 bit is used by the deadtime logic.
– 11 = EXTB[1] signal is used by the deadtime logic.

SEL23_0 — PWM23_0 Control Select

This field selects possible over-rides to the generated PWM23 signal in submodule 0 that will be passed to the deadtime logic upon the occurrence of a "Force Out" event in that submodule.

– 00 = Generated PWM23_0 signal is used by the deadtime logic.
– 01 = Inverted generated PWM23_0 signal is used by the deadtime logic.
– 10 = OUT23_0 bit is used by the deadtime logic.
– 11 = EXTA[0] signal is used by the deadtime logic.

SEL45_0 — PWM45_0 Control Select

This field selects possible over-rides to the generated PWM45 signal in submodule 0 that will be passed to the deadtime logic upon the occurrence of a "Force Out" event in that submodule.

– 00 = Generated PWM45_0 signal is used by the deadtime logic.
– 01 = Inverted generated PWM45_0 signal is used by the deadtime logic.
– 10 = OUT45_0 bit is used by the deadtime logic.
– 11 = EXTB[0] signal is used by the deadtime logic.

26.4.4.5 Master Control Register (MCTRL)

PWM_BASE+$148	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	IPOL				RUN				0	0	0	0	LDOK			
Write									CLDOK							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 425. Master Control Register (MCTRL)

The relationship between the fields of MCTRL and the submodules is as follows:

- *Field*[3] refers to submodule 3
- *Field*[2] refers to submodule 2
- *Field*[1] refers to submodule 1
- *Field*[0] refers to submodule 0

IPOL — Current Polarity

This buffered read/write bit is used to select between PWM23 and PWM45 as the source for the generation of the complementary PWM pair output. IPOL is ignored in independent mode.

1 = PWM45 (*Figure 364*) is used to generate complementary PWM pair.
0 = PWM23 (*Figure 364*) is used to generate complementary PWM pair.

Note: *The IPOL bit does not take effect until a FORCE_OUT event takes place in the appropriate submodule. Reading the IPOL bit reads the buffered value and not necessarily the value currently in effect.*

RUN — Run

This read/write bit enables the clocks to the PWM generator. When RUN equals zero, the submodule counter is reset. In submodules other than 0, the local RUN bit is ignored when CLK_SEL is 1 because this indicates that the AUX_CLK from submod0 is being used by this submodule. A reset clears RUN.

1 = PWM generator enabled.
0 = PWM generator disabled.

Note: *For proper initialization of the LDOK and RUN bits, see Section 26.3.4.5, "Initialization.*

CLDOK — Clear Load Okay

This write only bit is used to clear the LDOK bit. Write a 1 to this location to clear the corresponding LDOK. If a reload occurs with LDOK set at the same time that CLDOK is written, then the reload will not be performed and LDOK will be cleared. This bit is self clearing and always reads as a 0.

LDOK — Load Okay

This read/set bit loads the PRSC bits of CTRL1 and the INIT and VALx registers into a set of buffers. The buffered prescaler divisor, submodule counter modulus value, and PWM pulse width take effect at the next PWM reload if LDMOD is clear or immediately if LDMOD is set. Set LDOK by reading it when it is logic zero and then writing a logic one to it. The VALx, INIT, and PRSC registers cannot be written while LDOK is set. LDOK is automatically cleared after the new values are loaded, or can be manually cleared before a reload by writing a logic 1 to CLDOK. This bit cannot be written with a zero. LDOK can be set in DMA mode when the DMA indicates that it has completed the update of all PRSC, INIT and VALx registers. Reset clears LDOK.

1 = Load prescaler, modulus, and PWM values.
0 = Do not load new values.

Note: *For proper initialization of the LDOK and RUN bits, see Section 26.3.4.5, "Initialization.*

26.4.5 Fault Channel Registers

The base address of the fault logic is equal to the base address of the mcPWM plus an offset of $14C.

26.4.5.1 Fault Control Register (FCTRL)

PWM_BASE+$14C	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read / Write	FLVL				FAUTO				FSAFE				FIE			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 426. Fault Control Register (FCTRL)

Bits [3:0] of the fields in this register correspond to FAULT3, FAULT2, FAULT1, and FAULT0, respectively.

FLVL — Fault Level

These read/write bits select the active logic level of the individual fault inputs. A reset clears FLVL.

1 = A logic 1 on the fault input indicates a fault condition.
0 = A logic 0 on the fault input indicates a fault condition.

FAUTO — Automatic Fault Clearing

These read/write bits select automatic or manual clearing of faults. A reset clears FAUTO.

1 = Automatic fault clearing. PWM outputs disabled by this fault are enabled when the FFPINx bit is clear at the start of a half cycle or full cycle depending on the state of the FFULL bits without regard to the state of FFLAGx bit.
0 = Manual fault clearing. PWM outputs disabled by this fault are not enabled until the FFLAGx bit is clear at the start of a half cycle or full cycle depending the state of the FFULL bits. This is further controlled by the FSAFE bits.

FSAFE — Fault Safety Mode

These read/write bits select the safety mode during manual fault clearing. A reset clears FSAFE.

1 = Safe mode. PWM outputs disabled by this fault are not enabled until the FFLAGx bit is clear and the FFPINx bit is clear at the start of a half cycle or full cycle depending on the state of the FFULL bits.
0 = Normal mode. PWM outputs disabled by this fault are not enabled until the FFLAGx bit is clear at the start of a half cycle or full cycle depending on the state of the FFULL bits without regard to the state of the FFPINx bit. The PWM outputs disabled by this fault input will not be re-enabled until the actual FAULTx input signal de-asserts since the fault input will combinationally disable the PWM outputs (as programmed in DISMAP).

Note: *The FFPINx bit may indicate a fault condition still exists even though the actual fault signal at the FAULTx pin is clear due to the fault filter latency.*

FIE — Fault Interrupt Enables

This read/write bit enables CPU interrupt requests generated by the FAULTx pins. A reset clears FIE.

1 = FAULTx CPU interrupt requests enabled.
0 = FAULTx CPU interrupt requests disabled.

Note: *The fault protection circuit is independent of the FIEx bit and is always active. If a fault is detected, the PWM outputs are disabled according to the disable mapping register.*

26.4.5.2 Fault Status Register (FSTS)

PWM_BASE+$14E	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	0	0	0	FTEST	FFPIN				FFULL				FFLAG			
Write																
Reset	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1

Figure 427. Fault Status Register (FSTS)

Note: *In Figure 427 (Fault Status Register (FSTS)), Field[3:0] refers to subumodule 3,2,1,0.*

FTEST — Fault Test

These read/write bit is used to simulate a fault condition. Setting this bit will cause a simulated fault to be sent into all of the fault filters. The condition will propagate to the fault flags and possibly the PWM outputs depending on the DISMAP settings. Clearing this bit removes the simulated fault condition.

1 = Cause a simulated fault.
0 = No fault.

FFPIN — Filtered Fault Pins

These read-only bits reflect the current state of the filtered FAULTx pins converted to high polarity. A logic 1 indicates a fault condition exists on the filtered FAULTx pin. A reset has no effect on FFPIN.

FFULL — Full Cycle

These read/write bits are used to control the timing for re-enabling the PWM outputs after a fault condition. These bits apply to both automatic and manual clearing of a fault condition.

1 = PWM outputs are re-enabled only at the start of a full cycle.
0 = PWM outputs are re-enabled at the start of a full or half cycle.

FFLAG — Fault Flags

These read-only flag is set within two CPU cycles after a transition to active on the FAULTx pin. Clear FFLAGx by writing a logic one to it. A reset clears FFLAG.

1 = Fault on the FAULTx pin.
0 = No fault on the FAULTx pin.

26.4.5.3 Fault Filter Register (FFILT)

PWM_BASE+$150	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read	GSTR	0	0	0	0	FILT_CNT			FILT_PER							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 428. Fault Filter Register (FFILT)

The settings in this register are shared among each of the fault input filters.

GSTR — Fault Glitch Stretch Enable

This bit is used to enable the fault glitch stretching logic. This logic ensures that narrow fault glitches are stretched to be at least 2 IPBus clock cycles wide. In some cases a narrow fault input can cause problems due to the short PWM output shutdown/re-activation time. The stretching logic ensures that a glitch on the fault input, when the fault filter is disabled, will be registered in the fault flags.

1 = Input fault signals will be stretched to at least 2 IPBus clock cycles.
0 = Fault input glitch stretching is disabled.

FILT_CNT — Fault Filter Count

These bits represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0 represents 3 samples. A value of 7 represents 10 samples. The value of FILT_CNT affects the input latency as described in *Section 26.4.5.3.1, "Input Filter Considerations"*.

FILT_PER — Fault Filter Period

These bits represent the sampling period (in IPBus clock cycles) of the fault pin input filter. Each input is sampled multiple times at the rate specified by FILT_PER. If FILT_PER is 0x00 (default), then the input filter is bypassed. The value of FILT_PER affects the input latency as described in *Section 26.4.5.3.1, "Input Filter Considerations"*.

26.4.5.3.1 Input Filter Considerations

The FILT_PER value should be set such that the sampling period is larger than the period of the expected noise. This way a noise spike will only corrupt one sample. The FILT_CNT value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The probability of an incorrect transition is defined as the probability of an incorrect sample raised to the FILT_CNT+3 power.

The values of FILT_PER and FILT_CNT must also be traded off against the desire for minimal latency in recognizing input transitions. Turning on the input filter (setting FILT_PER to a non-zero value) introduces a latency of ((FILT_CNT+4) x FILT_PER x IPBus clock period). Note that even when the filter is enabled, there is a combinational path to disable the PWM outputs. This is to ensure rapid response to fault conditions and also to ensure fault response if the PWM module loses its clock. The latency induced by the filter will be seen in the time to set the FFLAG and FFPIN bits of the FSTS register.

26.5 Interrupts

Each of the submodules within the FlexPWM can generate an interrupt from several sources. The fault logic can also generate interrupts. The interrupt service routine (ISR) must check the related interrupt enables and interrupt flags to determine the actual cause of the interrupt.

Table 330. Interrupt summary

Core Interrupt	Interrupt Flag	Interrupt Enable	Name	Description
COF0	CMPF_0	CMPIE_0	Submodule 0 compare interrupt	Compare event has occurred
CAF0	CFX1_0, CFX0_0	CFX1IE_0, CFX0IE_0	Submodule 0 input capture interrupt	Input capture event has occurred
RF0	RF_0	RIE_0	Submodule 0 reload interrupt	Reload event has occurred
COF1	CMPF_1	CMPIE_1	Submodule 1 compare interrupt	Compare event has occurred
CAF1	CFX1_1, CFX0_1	CFX1IE_1, CFX0IE_1	Submodule 1 input capture interrupt	Input capture event has occurred
RF1	RF_1	RIE_1	Submodule 1 reload interrupt	Reload event has occurred
COF2	CMPF_2	CMPIE_2	Submodule 2 compare interrupt	Compare event has occurred
CAF2	CFX1_2, CFX0_2	CFX1IE_2, CFX0IE_2	Submodule 2 input capture interrupt	Input capture event has occurred
RF2	RF_2	RIE_2	Submodule 2 reload interrupt	Reload event has occurred
COF3	CMPF_3	CMPIE_3	Submodule 3 compare interrupt	Compare event has occurred
CAF3	CFX1_3, CFX0_3	CFX1IE_3, CFX0IE_3	Submodule 3 input capture interrupt	Input capture event has occurred
RF3	RF_3	RIE_3	Submodule 3 reload interrupt	Reload event has occurred
REF	REF_0	REIE_0	Submodule 0 reload error interrupt	Reload error has occurred
	REF_1	REIE_1	Submodule 1 reload error interrupt	
	REF_2	REIE_2	Submodule 2 reload error interrupt	
	REF_3	REIE_3	Submodule 3 reload error interrupt	
FFLAG	FFLAG	FIE	Fault input interrupt	Fault condition has been detected

26.6 DMA

Each submodule can request a DMA read access for its capture FIFOs and a DMA write request for its double buffered VALx registers.

Table 331. DMA Summary

DMA Request	DMA Enable	Name	Description
Submodule 0 read request	CX0DE_0	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_0	Capture FIFO X! read request	CVAL1 contains a value to be read
	CAPTDE_0	Capture FIFO read request source select	Selects source of read DMA request
Submodule 0 write request	VALDE_0	VALx write request	VALx registers need to be updated
Submodule 1 read request	CX0DE_1	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_1	Capture FIFO X! read request	CVAL1 contains a value to be read
	CAPTDE_1	Capture FIFO read request source select	Selects source of read DMA request
Submodule 1 write request	VALDE_1	VALx write request	VALx registers need to be updated
Submodule 2 read request	CX0DE_2	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_2	Capture FIFO X! read request	CVAL1 contains a value to be read
	CAPTDE_2	Capture FIFO read request source select	Selects source of read DMA request
Submodule 2 write request	VALDE_2	VALx write request	VALx registers need to be updated
Submodule 3 read request	CX0DE_3	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_3	Capture FIFO X! read request	CVAL1 contains a value to be read
	CAPTDE_3	Capture FIFO read request source select	Selects source of read DMA request
Submodule 3 write request	VALDE_3	VALx write request	VALx registers need to be updated

27 FlexRay Communication Controller

27.1 Introduction

27.1.1 Reference

The following documents are referenced.

- FlexRay Communications System Protocol Specification, Version 2.1 Rev A[n]
- FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A

27.1.2 Glossary

This section provides a list of terms used in this chapter.

Table 332. 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	FlexRay Communication Controller, module described in this chapter.
CDC	Clock Domain Crosser
CHI	Controller Host Interface
Cycle length in µT	The actual length of a cycle in µT for the ideal CC (+/- 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 contains the FlexRay memory area.
System Bus	Bus that connects the CC and System Memory
FSS	Frame Start Sequence
HIF	Host Interface. Provides host access to the CC.
Host	The FlexRay CC host CPU.
LUT	Look Up Table. Stores message buffer header index value.
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.

n. The FlexRay Specifications have been developed for automotive applications.The FlexRay Specifications have been neither developed nor tested for non-automotive applications.

Table 332. List of Terms (continued)

Term	Definition
MCU	Microcontroller Unit
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

27.1.3 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*.

27.1.4 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 429*.

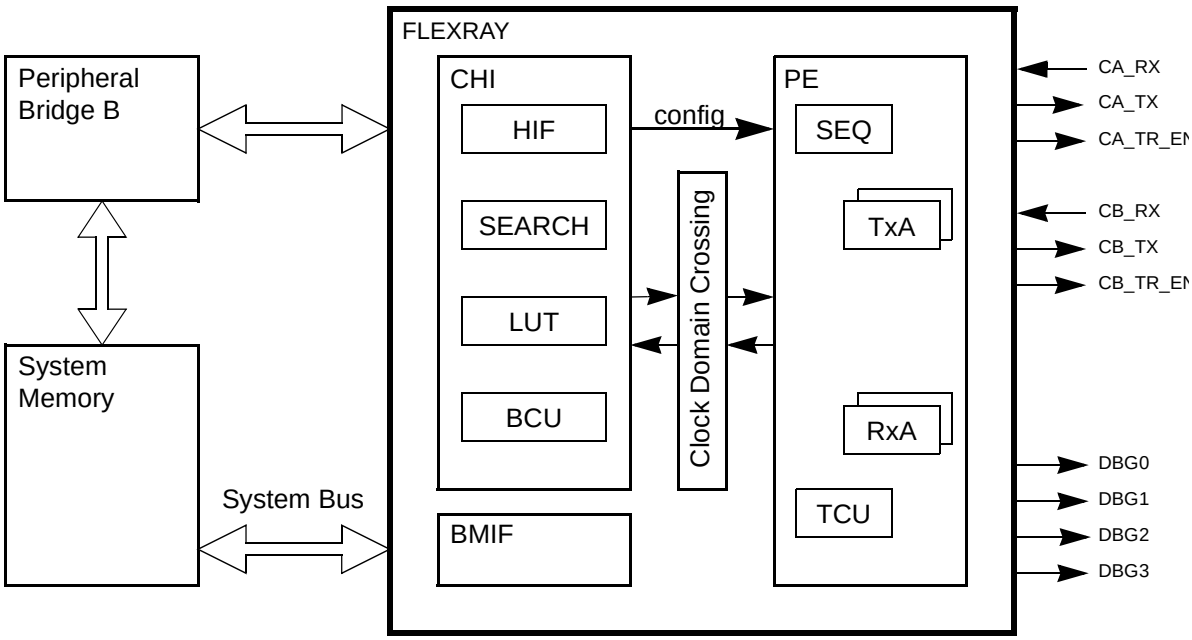


Figure 429. FLEXRAY block diagram

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 CC 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 of the MCU system memory is calculated using the offset values the FlexRay memory area base address.

FlexRay interacts with the CTU as described in *Section 14.4.1, "Interaction with other peripherals."*

Note: The CC does not provide a memory protection scheme for the FlexRay memory area.

27.1.5 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
- 64 configurable message buffers with
  - individual frame ID filtering
  - individual channel ID filtering
  - individual cycle counter filtering
- message buffer header, status and payload data stored in dedicated FlexRay memory 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
  - single buffered transmit message buffer
  - double buffered transmit message buffer (combines two single buffered message buffer)
- individual message buffer reconfiguration supported
  - means provided to safely disable individual message buffers
  - disabled message buffers can be reconfigured

- 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

27.1.6 Modes of operation

This section describes the basic operational power modes of the CC.

27.1.6.1 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)*.

In the Disabled Mode 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 27.5.2, "Register Descriptions"*.

The application configures the CC by accessing the configuration bits and fields in the *Module Configuration Register (FR_MCR)*.

27.1.6.1.1 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: Once the CC is enabled it can only be disabled via a device reset.

27.1.6.2 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)*.

27.1.6.2.1 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 *27.7.1.2, "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.

27.2 External Signal Description

This section lists and describes the CC signals, connected to external pins. These signals are summarized in *Table 333* and described in detail in *Section 27.2.1, "Detailed Signal Descriptions"*.

Note: The off chip signals CA_RX, CA_TX, and CA_TR_EN are available on each package option. The availability of the other off chip signals depends on the package option.

Table 333. External Signal Properties

Name	Direction	Active	Reset	Function
CA_RX	Input	—	—	Receive Data Channel A
CA_TX	Output	—	1	Transmit Data Channel A
CA_TR_EN	Output	Low	1	Transmit Enable Channel A
CB_RX	Input	—	—	Receive Data Channel B
CB_TX	Output	—	1	Transmit Data Channel B
CB_TR_EN	Output	Low	1	Transmit Enable Channel B
DBG0	Output	—	0	Debug Strobe Signal 0
DBG1	Output	—	0	Debug Strobe Signal 1
DBG2	Output	—	0	Debug Strobe Signal 2
DBG3	Output	—	0	Debug Strobe Signal 3

27.2.1 Detailed Signal Descriptions

This section provides a detailed description of the CC signals, connected to external pins.

27.2.1.1 CA_RX — Receive Data Channel A

The CA_RX signal carries the receive data for channel A from the corresponding FlexRay bus driver.

27.2.1.2 CA_TX — Transmit Data Channel A

The CA_TX signal carries the transmit data for channel A to the corresponding FlexRay bus driver.

27.2.1.3 CA_TR_EN — Transmit Enable Channel A

The CA_TR_EN signal indicates to the FlexRay bus driver that the CC is attempting to transmit data on channel A.

27.2.1.4 CB_RX — Receive Data Channel B

The CB_RX signal carries the receive data for channel B from the corresponding FlexRay bus driver.

27.2.1.5 CB_TX — Transmit Data Channel B

The CB_TX signal carries the transmit data for channel B to the corresponding FlexRay bus driver

27.2.1.6 CB_TR_EN — Transmit Enable Channel B

The CB_TR_EN signal indicates to the FlexRay bus driver that the CC is attempting to transmit data on channel B.

27.2.1.7 DBG3, DBG2, DBG1, DBG0 — Strobe Signals

These signals provide the selected debug strobe signals. For details on the debug strobe signal selection refer to *Section 27.6.16, "Strobe Signal Support"*.

27.3 Controller Host Interface Clocking

The clock for the CHI is derived from the system bus clock and has the same phase and frequency as the system bus clock. Since the FlexRay protocol requires data delivery at fixed points in time, the memory read cycles from the FlexRay memory area must be finished after a fixed amount of time. To ensure this, a minimum frequency f_{chi} of the CHI clock is required, which is given in *Equation 15*.

Equation 15 $f_{chi} \geq 32MHz$

Additional requirements for the minimum frequency of the CHI clock result from the number of message buffers. These requirements are provided in *Section 27.7.5, "Number of Usable Message Buffers*"

27.4 Protocol Engine Clocking

The clock for the protocol engine can be generated by two sources. The first source is the internal crystal oscillator and the second source is an internal FMPLL. The clock source to be used is selected by the clock source select bit CLKSEL in the *Module Configuration Register (FR_MCR)*.

If the protocol engine is clocked by the internal crystal oscillator, an 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.*

27.5 Memory Map and Register Description

The CC occupies 768 bytes of address space starting atthe base address of the CC is defined by the memory map of the MCU.

27.5.1 Memory Map

The complete memory map of the CC is shown in *Table 334*. The addresses presented here are the offsets relative to the CC base address which is defined by the MCU address map.

Table 334. FlexRay Memory Map

Offset	Register	Access
Module Configuration and Control		
0x0000	*Module Version Register (FR_MVR)*	R
0x0002	*Module Configuration Register (FR_MCR)*	R/W
0x0004	*System Memory Base Address High Register (FR_SYMBADHR)*	R/W
0x0006	*System Memory Base Address Low Register (FR_SYMBADLR)*	R/W
0x0008	*Strobe Signal Control Register (FR_STBSCR)*	R/W
0x000A	Reserved	R
0x000C	*Message Buffer Data Size Register (FR_MBDSR)*	R/W
0x000E	*Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*	R/W
PE Access Registers		
0x0010	*PE DRAM Access Register (FR_PEDRAR)*	R/W
0x0012	*PE DRAM Data Register (FR_PEDRDR)*	R/W
Interrupt and Error Handling		
0x0014	*Protocol Operation Control Register (FR_POCR)*	R/W
0x0016	*Global Interrupt Flag and Enable Register (FR_GIFER)*	R/W
0x0018	*Protocol Interrupt Flag Register 0 (FR_PIFR0)*	R/W
0x001A	*Protocol Interrupt Flag Register 1 (FR_PIFR1)*	R/W
0x001C	*Protocol Interrupt Enable Register 0 (FR_PIER0)*	R/W
0x001E	*Protocol Interrupt Enable Register 1 (FR_PIER1)*	R/W
0x0020	*CHI Error Flag Register (FR_CHIERFR)*	R/W
0x0022	*Message Buffer Interrupt Vector Register (FR_MBIVEC)*	R
0x0024	*Channel A Status Error Counter Register (FR_CASERCR)*	R
0x0026	*Channel B Status Error Counter Register (FR_CBSERCR)*	R
Protocol Status		
0x0028	*Protocol Status Register 0 (FR_PSR0)*	R
0x002A	*Protocol Status Register 1 (FR_PSR1)*	R
0x002C	*Protocol Status Register 2 (FR_PSR2)*	R

Table 334. FlexRay Memory Map (continued)

Offset	Register	Access
0x002E	*Protocol Status Register 3 (FR_PSR3)*	R/W
0x0030	*Macrotick Counter Register (FR_MTCTR)*	R
0x0032	*Cycle Counter Register (FR_CYCTR)*	R
0x0034	*Slot Counter Channel A Register (FR_SLTCTAR)*	R
0x0036	*Slot Counter Channel B Register (FR_SLTCTBR)*	R
0x0038	*Rate Correction Value Register (FR_RTCORVR)*	R
0x003A	*Offset Correction Value Register (FR_OFCORVR)*	R
0x003C	*Combined Interrupt Flag Register (FR_CIFR)*	R
0x003E	*System Memory Access Time-Out Register (FR_SYMATOR)*	R/W
Sync Frame Counter and Tables		
0x0040	*Sync Frame Counter Register (FR_SFCNTR)*	R
0x0042	*Sync Frame Table Offset Register (FR_SFTOR)*	R/W
0x0044	*Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)*	R/W
Sync Frame Filter		
0x0046	*Sync Frame ID Rejection Filter Register (FR_SFIDRFR)*	R/W
0x0048	*Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)*	R/W
0x004A	*Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)*	R/W
Network Management Vector		
0x004C	*Network Management Vector Register 0 (FR_NMVR0)*	R
0x004E	*Network Management Vector Register 1 (FR_NMVR1)*	R
0x0050	*Network Management Vector Register 2 (FR_NMVR2)*	R
0x0052	*Network Management Vector Register 3 (FR_NMVR3)*	R
0x0054	*Network Management Vector Register 4 (FR_NMVR4)*	R
0x0056	*Network Management Vector Register 5 (FR_NMVR5)*	R
0x0058	*Network Management Vector Length Register (FR_NMVLR)*	R/W
Timer Configuration		
0x005A	*Timer Configuration and Control Register (FR_TICCR)*	R/W
0x005C	*Timer 1 Cycle Set Register (FR_TI1CYSR)*	R/W
0x005E	*Timer 1 Macrotick Offset Register (FR_TI1MTOR)*	R/W
0x0060	*Timer 2 Configuration Register 0 (FR_TI2CR0)*	R/W
0x0062	*Timer 2 Configuration Register 1 (FR_TI2CR1)*	R/W
Slot Status Configuration		
0x0064	*Slot Status Selection Register (FR_SSSR)*	R/W
0x0066	*Slot Status Counter Condition Register (FR_SSCCR)*	R/W

Table 334. FlexRay Memory Map (continued)

Offset	Register	Access
Slot Status		
0x0068	*Slot Status Register 0 (FR_SSR0)*	R
0x006A	*Slot Status Register 1 (FR_SSR1)*	R
0x006C	*Slot Status Register 2 (FR_SSR2)*	R
0x006E	*Slot Status Register 3 (FR_SSR3)*	R
0x0070	*Slot Status Register 4 (FR_SSR4)*	R
0x0072	*Slot Status Register 5 (FR_SSR5)*	R
0x0074	*Slot Status Register 6 (FR_SSR6)*	R
0x0076	*Slot Status Register 7 (FR_SSR7)*	R
0x0078	*Slot Status Counter Register 0 (FR_SSCR0)*	R
0x007A	*Slot Status Counter Register 1 (FR_SSCR1)*	R
0x007C	*Slot Status Counter Register 2 (FR_SSCR2)*	R
0x007E	*Slot Status Counter Register 3 (FR_SSCR3)*	R
MTS Generation		
0x0080	*MTS A Configuration Register (FR_MTSACFR)*	R/W
0x0082	*MTS B Configuration Register (MTSBCFR)*	R/W
Shadow Buffer Configuration		
0x0084	*Receive Shadow Buffer Index Register (FR_RSBIR)*	R/W
Receive FIFO — Configuration		
0x0086	*Receive FIFO Watermark and Selection Register (FR_RFWMSR)*	R/W
0x0088	*Receive FIFO Start Index Register (FR_RFSIR)*	R/W
0x008A	*Receive FIFO Depth and Size Register (RFDSR)*	R/W
Receive FIFO - Control		
0x008C	*Receive FIFO A Read Index Register (FR_RFARIR)*	R
0x008E	*Receive FIFO B Read Index Register (FR_RFBRIR)*	R
Receive FIFO - Filter		
0x0090	*Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)*	R/W
0x0092	*Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)*	R/W
0x0094	*Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)*	R/W
0x0096	*Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)*	R/W
0x0098	*Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)*	R/W
0x009A	*Receive FIFO Range Filter Control Register (FR_RFRFCTR)*	R/W
Dynamic Segment Status		

Table 334. FlexRay Memory Map (continued)

Offset	Register	Access
0x009C	*Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)*	R
0x009E	*Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)*	R
Protocol Configuration		
0x00A0 ... 0x00DC	*Protocol Configuration Register 0 (FR_PCR0)* ... *Protocol Configuration Register 30 (FR_PCR30)*	R/W – R/W
0x00DE ... 0x00E6	Reserved	R
Receive FIFO — Configuration (cont.)		
0x00E8	*Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR)*	R/W
0x00EA	*Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR)*	R/W
0x00EC	*Receive FIFO Periodic Timer Register (FR_RFPTR)*	R/W
Receive FIFO - Control (cont.)		
0x00EE	*Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)*	R/W
ECC Registers		
0x00F0	*ECC Error Interrupt Flag and Enable Register (FR_EEIFER)*	R/W
0x00F2	*ECC Error Report and Injection Control Register (FR_EERICR)*	R/W
0x00F4	*ECC Error Report Address Register (FR_EERAR)*	R
0x00F6	*ECC Error Report Data Register (FR_EERDR)*	R
0x00F8	*ECC Error Report Code Register (FR_EERCR)*	R
0x00FA	*ECC Error Injection Address Register (FR_EEIAR)*	R/W
0x00FC	*ECC Error Injection Data Register (FR_EEIDR)*	R/W
0x00FE	*ECC Error Injection Code Register (FR_EEICR)*	R/W
Message Buffers Configuration, Control, Status		
0x0100	*Message Buffer Configuration, Control, Status Register 0 (FR_MBCCSR0)*	R/W
0x0102	*Message Buffer Cycle Counter Filter Register 0 (FR_MBCCFR0)*	R/W
0x0104	*Message Buffer Frame ID Register 0 (FR_MBFIDR0)*	R/W
0x0106	*Message Buffer Index Register 0 (FR_MBIDXR0)*	R/W
...	...	...
0x02F8	*Message Buffer Configuration, Control, Status Register 63 (FR_MBCCSR63)*	R/W
0x02FA	*Message Buffer Cycle Counter Filter Register 63 (FR_MBCCFR63)*	R/W
0x02FC	*Message Buffer Frame ID Register 63 (FR_MBFIDR63)*	R/W
0x02FE	*Message Buffer Index Register 63 (FR_MBIDXR63)*	R/W

27.5.2 Register Descriptions

This section provides detailed descriptions of all registers in ascending address order, presented as 16-bit wide entities

Table 335 provides a key for the register figures and register tables.

Table 335. 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.

27.5.2.1 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 336*.

Table 336. 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.

27.5.2.2 Register write access

This section describes the write access restriction terms that apply to all registers.

27.5.2.2.1 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 337*. 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 337. 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

27.5.2.2.2 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

27.5.2.2.3 Internal Register Access

The following memory mapped registers are used to access multiple internal registers.

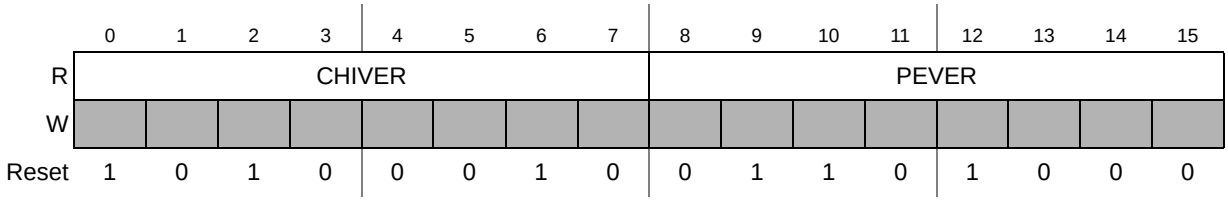
- *Strobe Signal Control Register (FR_STBSCR)*
- *Slot Status Selection Register (FR_SSSR)*
- *Slot Status Counter Condition Register (FR_SSCCR)*
- *Receive Shadow Buffer Index Register (FR_RSBIR)*

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.

27.5.2.3 Module Version Register (FR_MVR)

Figure 430. Module Version Register (FR_MVR)

Base + 0x0000



This register provides the CC version number. The module version number is derived from the CHI version number and the PE version number.

Table 338. FR_MVR Field Descriptions

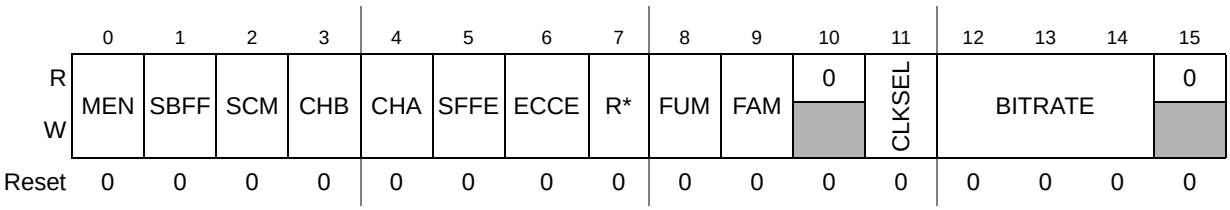
Field	Description
CHIVER	**CHI Version Number** — This field provides the version number of the CC host interface.
PEVER	**PE Version Number** — This field provides the version number of the protocol engine.

27.5.2.4 Module Configuration Register (FR_MCR)

Figure 431. 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*



This register defines the global configuration of the CC.

Table 339. 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 27.1.6, "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 27.6.10, "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 340*.
SFFE	**Synchronization Frame Filter Enable** — This bit controls the filtering for received synchronization frames. For details see *Section 27.6.15, "Sync Frame Filtering"*. 0 Synchronization frame filtering disabled 1 Synchronization frame filtering enabled
ECCE	**ECC Functionality Enable** — This bit controls the ecc memory error detection functionality. For details see *Section 27.6.24, "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 27.6.9.8, "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 27.6.9.2, "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)*

Table 339. FR_MCR Field Descriptions (continued)

Field	Description
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 FMPLL.
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 340. FlexRay Channel Selection

SCM	CHB	CHA	Description
Dual Channel Device Modes			
0	0	0	ports CA_RX, CA_TX, and CA_TR_EN not driven by CC ports CB_RX, CB_TX, and CA_TR_EN not driven by CC
	0	1	ports CA_RX, CA_TX, and CA_TR_EN driven by CC - connected to FlexRay channel A ports CB_RX, CB_TX, and CA_TR_EN not driven by CC
	1	0	ports CA_RX, CA_TX, and CA_TR_EN not driven by CC ports CB_RX, CB_TX, and CA_TR_EN driven by CC - connected to FlexRay channel B
	1	1	ports CA_RX, CA_TX, and CA_TR_EN driven by CC - connected to FlexRay channel A ports CB_RX, CB_TX, and CA_TR_EN driven by CC - connected to FlexRay channel B
Single Channel Device Mode			
1	0	0	ports CA_RX, CA_TX, and CA_TR_EN not driven by CC ports CB_RX, CB_TX, and CA_TR_EN not driven by CC
	0	1	ports CA_RX, CA_TX, and CA_TR_EN driven by CC - connected to FlexRay channel A ports CB_RX, CB_TX, and CA_TR_EN not driven by CC
	1	0	ports CA_RX, CA_TX, and CA_TR_EN driven by CC - connected to FlexRay channel B ports CB_RX, CB_TX, and CA_TR_EN not driven by CC
	1	1	reserved

27.5.2.5 System Memory Base Address Register (FR_SYMBADR)

Figure 432. System Memory Base Address High Register (FR_SYMBADHR)

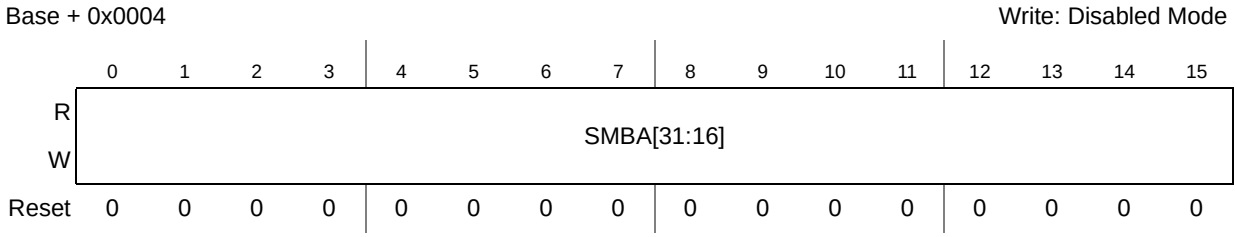
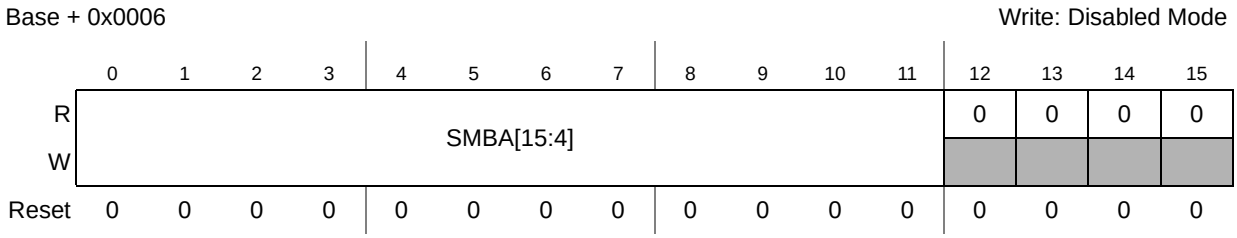


Figure 433. System Memory Base Address Low Register (FR_SYMBADLR)



Note: The system memory base address must be set before the CC is enabled.

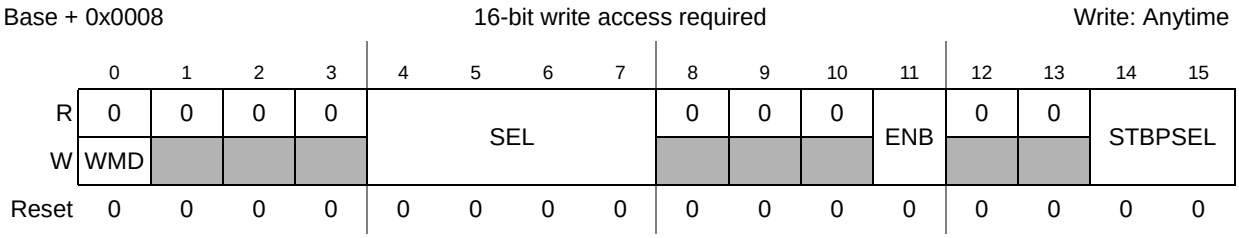
The system memory base address registers define the base address of the FlexRay memory area within the system memory. The base address is used by the BMIF to calculate the physical memory address for system memory accesses.

Table 341. 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.

27.5.2.6 Strobe Signal Control Register (FR_STBSCR)

Figure 434. Strobe Signal Control Register (FR_STBSCR)



This register is used to assign the individual protocol timing related strobe signals given in *Table 343* 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 27.6.16, "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 342. 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 343* to be enabled or disabled and assigned to one of the four strobe ports given in *Table 343*.
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 DBG0 01 assign selected signal to DBG1 10 assign selected signal to DBG2 11 assign selected signal to DBG3

Table 343. 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
4	0x4	slot start	A	pulse	0	MT start
5	0x5		B			
6	0x6	receive data after glitch filtering	A	value	+4	CA_RX
7	0x7		B			CB_RX
8	0x8	channel idle indicator	A	level	+5	CA_RX
9	0x9		B			CB_RX
10	0xA	syntax error detected	A	pulse	+4	CA_RX
11	0xB		B			CB_RX
12	0xC	content error detected	A	level	+4	CA_RX
13	0xD		B			CB_RX

Table 343. Strobe signal mapping (continued)

SEL		Description	Channel	Type	Offset(1)	Reference
dec	hex					
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.

27.5.2.7 Message Buffer Data Size Register (FR_MBDSR)

Figure 435. 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

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 344. 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.

27.5.2.8 Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)

Figure 436. 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	0	LAST_MB_SEG1						0	0	LAST_MB_UTIL					
W																
Reset	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1

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 345. 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 < 64. *Note: If LAST_MB_SEG1 = 63 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.*

27.5.2.9 PE DRAM Access Register (FR_PEDRAR)

Figure 437. 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

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 27.6.24, "Memory Content Error Detection*"

Table 346. 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

27.5.2.10 PE DRAM Data Register (FR_PEDRDR)

Figure 438. 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	DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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)*.

27.5.2.11 Protocol Operation Control Register (FR_POCR)

Figure 439. 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

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 27.7.6, "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 27.6.11, "External Clock Synchronization"*.

Table 347. 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.
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

Table 347. FR_POCR Field Descriptions

Field	Description
BSY	**Protocol Control Command Write Busy** — This status bit indicates the acceptance of the protocol control command issued by the application via the POCCMD field. The 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.
WMC	**Write Mode Command** — This bit controls the write mode of the POCCMD field. 0 Write to POCCMD field on register write. 1 Do not write to POCCMD field on register write.
POCCMD	**Protocol Control Command** — The application writes to this field to issue a protocol control command to the PE. The 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.

27.5.2.12 Global Interrupt Flag and Enable Register (FR_GIFER)

Figure 440. 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

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 588*. For more details on interrupt generation, see *Section 27.6.20, "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 348. FR_GIFER field descriptions

Field	Description
MIF	**Module Interrupt Flag** — This flag is asserted if at least one of the other interrupt flags in this register and its related interrupt enable is asserted. 0 No interrupt flag is asserted or no interrupt enable is set 1 At least one of the other interrupt flags in this register is asserted and the related interrupt bit is asserted, too
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* and *Protocol Interrupt Flag Register 1 (FR_PIFR1)* is asserted and the related interrupt enable flag is asserted. 0 All individual protocol interrupt flags are equal to 0 or no interrupt enable bit is set. 1 At least one of the individual protocol interrupt flags and the related interrupt enable is equal to 1.
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the *CHI Error Flag Register (FR_CHIERFR)* is asserted and the chi error interrupt enable FR_GIFER[CHIE] is asserted. 0 All CHI error flags are equal to 0 or the chi error interrupt is disabled 1 At least one CHI error flag is asserted and chi error interrupt is enabled
WUPIF	**Wakeup Interrupt Flag** — This flag is set when the 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 condition or interrupt disabled 1 Wakeup symbol received on FlexRay bus and interrupt enabled
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — This 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
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — This 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

Table 348. FR_GIFER field descriptions (continued)

Field	Description
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) 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 RBIF flag directly. This flag 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 interrupt enables bit MBIE. 0 None of the individual receive message buffers has the MBIF and MBIE flag asserted. 1 At least one individual receive message buffer has the MBIF and MBIE flag asserted.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual single or double transmit 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 TBIF flag directly. This 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 host has cleared the interrupt enables bit MBIE. 0 None of the individual transmit message buffers has the MBIF and MBIE flag asserted. 1 At least one individual transmit message buffer has the MBIF and MBIE flag asserted.
MIE	**Module Interrupt Enable** — This flag controls if the Module Interrupt line is asserted when the MIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
PRIE	**Protocol Interrupt Enable** — This flag controls if the Protocol Interrupt line is asserted when the PRIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
CHIE	**CHI Interrupt Enable** — This flag controls if the CHI Interrupt line is asserted when the CHIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
WUPIE	**Wakeup Interrupt Enable** — This flag controls if the Wakeup Interrupt line is asserted when the WUPIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
FAFBIE	**Receive FIFO Channel B Almost Full Interrupt Enable** — This flag 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 flag 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

Table 348. FR_GIFER field descriptions (continued)

Field	Description
RBIE	**Receive Message Buffer Interrupt Enable** — This flag 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 flag controls if the Transmit Message Buffer Interrupt line is asserted when the TBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

27.5.2.13 Protocol Interrupt Flag Register 0 (FR_PIFR0)

Figure 441. 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

The register holds one set of the protocol-related individual interrupt flags.

Table 349. 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.

Table 349. FR_PIFR0 field descriptions (continued)

Field	Description
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.*
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.

Table 349. FR_PIFR0 field descriptions (continued)

Field	Description
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.

27.5.2.14 Protocol Interrupt Flag Register 1 (FR_PIFR1)

Figure 442. 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

The register holds one set of the protocol-related individual interrupt flags.

Table 350. 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 27.7.6, "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 protocol engine and the CC host interface 0 No such event. 1 Protocol Engine Communication Failure detected.
PSC_IF	**Protocol State Changed Interrupt Flag** — This flag is set when the protocol state in the PROTSTATE field in the *Protocol Status Register 0 (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

27.5.2.15 Protocol Interrupt Enable Register 0 (FR_PIER0)

Figure 443. 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

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 351. 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

Table 351. FR_PIER0 field descriptions (continued)

Field	Description
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
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

27.5.2.16 Protocol Interrupt Enable Register 1 (FR_PIER1)

Figure 444. 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

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 352. 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
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

27.5.2.17 CHI Error Flag Register (FR_CHIERFR)

Figure 445. 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	DBL_EF	SBCF_EF(1)	FID_EF	DPL_EF	SPL_EF	NML_EF	NMF_EF	ILSA_EF(1)
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

1. The FlexRay controller should be stopped via a FREEZE or HALT command and subsequently restarted when any of the error flags CHIERFR[SBCF_EF] or CHIERFR[ILSA_EF] is set.

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 353. 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
MSB_EF	**Message Buffer Search Error Flag** — This flag is set if the message buffer search engine is still running while the next search cycle must be started due to the FlexRay protocol timing. In this case, not all message buffers are considered while searching. 0 No such event 1 Search engine active while search start appears
MBU_EF	Message Buffer Utilization Error Flag — This flag is asserted if the application writes to a message buffer control field that is beyond the number of utilized message buffers programmed in the *Message Buffer Segment Size and Utilization Register (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

Table 353. FR_CHIERFR field descriptions (continued)

Field	Description
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
DBL_EF	**Double Transmit Message Buffer Lock Error Flag** — This flag is set if the application tries to lock the transmit side of a double transmit message buffer. In this case, the CC does not grant the lock to the transmit side of a double transmit message buffer. 0 No such event 1 Double transmit buffer lock error occurred
SBCF_EF	**System Bus Communication Failure Error Flag** — This flag is set if a system bus access was not finished within the required amount of time (see *Section 27.6.19.2, "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 single or double transmit message buffer assigned to the dynamic segment is greater than the maximum payload length for the dynamic segment as it is configured in the corresponding protocol configuration register field max_payload_length_dynamic in the *Protocol Configuration Register 24 (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 single or double transmit message buffer assigned to the static segment is different from the payload length for the static segment as it is configured in the corresponding protocol configuration register field payload_length_static in the *Protocol Configuration Register 19 (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

Table 353. FR_CHIERFR field descriptions (continued)

Field	Description
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 27.6.19.1, "System Bus Illegal Address Access*"). 0 No such event 1 Illegal system bus address accessed

27.5.2.18 Message Buffer Interrupt Vector Register (FR_MBIVEC)

Figure 446. 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	0	TBIVEC						0	0	RBIVEC					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

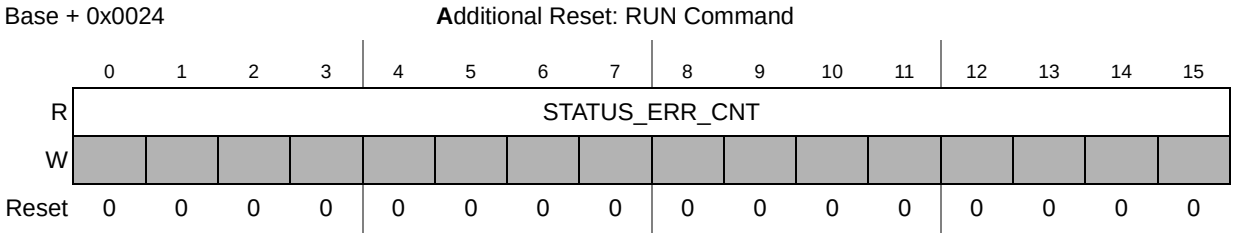
This register indicates the lowest numbered receive message buffer and the lowest numbered transmit message buffer that have their interrupt status flag MBIF and interrupt enable MBIE bits asserted. This means that message buffers with lower message buffer numbers have higher priority.

Table 354. 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.

27.5.2.19 Channel A Status Error Counter Register (FR_CASERCR)

Figure 447. Channel A Status Error Counter Register (FR_CASERCR)



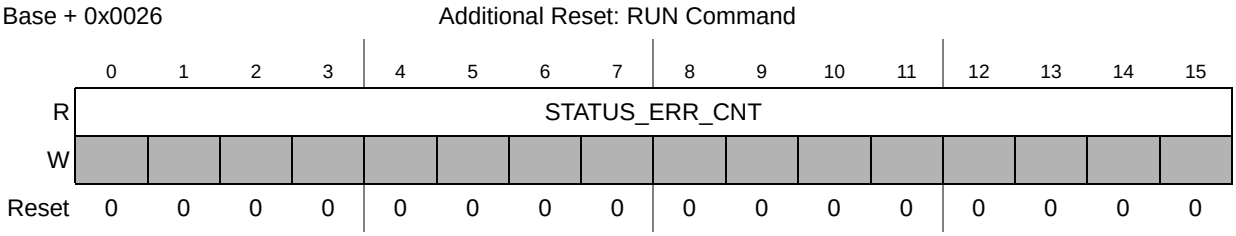
This register provides the channel status error counter for channel A. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError, vSS!ContentError, 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 27.6.18, "Slot Status Monitoring"*.

Table 355. 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.

27.5.2.20 Channel B Status Error Counter Register (FR_CBSERCR)

Figure 448. 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 27.6.18, "Slot Status Monitoring"*.

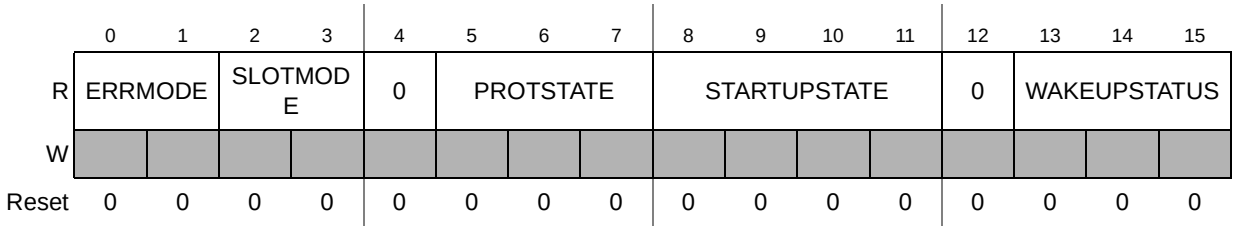
Table 356. FR_CBSERCR field descriptions

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.

27.5.2.21 Protocol Status Register 0 (FR_PSR0)

Figure 449. Protocol Status Register 0 (FR_PSR0)

Base + 0x0028



This register provides information about the current protocol status.

Table 357. 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*

Table 357. FR_PSR0 field descriptions (continued)

Field	Description
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

27.5.2.22 Protocol Status Register 1 (FR_PSR1)

Figure 450. 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

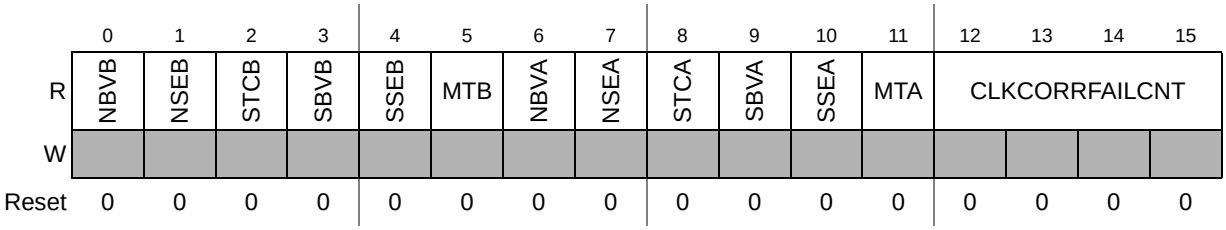
Table 358. FR_PSR1 Field Descriptions

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)*.

27.5.2.23 Protocol Status Register 2 (FR_PSR2)

Figure 451. Protocol Status Register 2 (FR_PSR2)

Base + 0x002C Additional Reset: RUN Command



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 359. 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

Table 359. FR_PSR2 field descriptions (continued)

Field	Description
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
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

Table 359. FR_PSR2 field descriptions (continued)

Field	Description
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.

27.5.2.24 Protocol Status Register 3 (FR_PSR3)

Figure 452. 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

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 360. 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

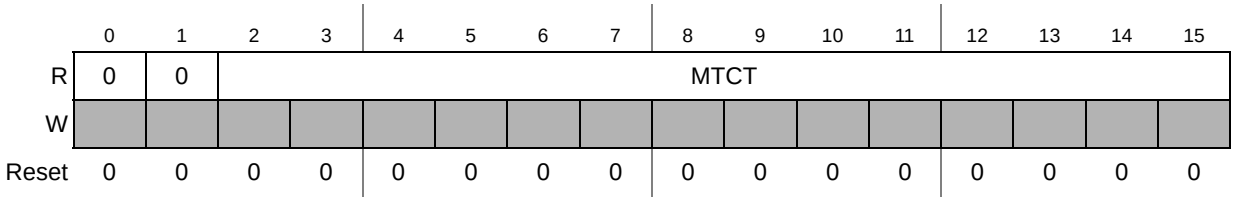
Table 360. FR_PSR3 field descriptions (continued)

Field	Description
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
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

27.5.2.25 Macrotick Counter Register (FR_MTCTR)

Figure 453. Macrotick Counter Register (FR_MTCTR)

Base + 0x0030



This register provides the macrotick count of the current communication cycle.

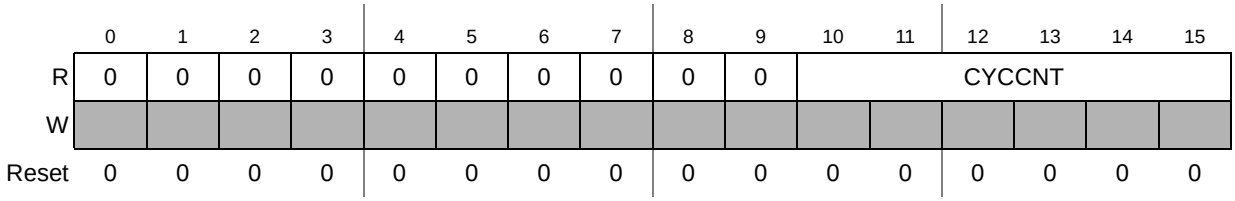
Table 361. FR_MTCTR Field Descriptions

Field	Description
MTCT	**Macrotick Counter** — protocol related variable: *vMacrotick* This field provides the macrotick count of the current communication cycle.

27.5.2.26 Cycle Counter Register (FR_CYCTR)

Figure 454. Cycle Counter Register (FR_CYCTR)

Base + 0x0032



This register provides the number of the current communication cycle.

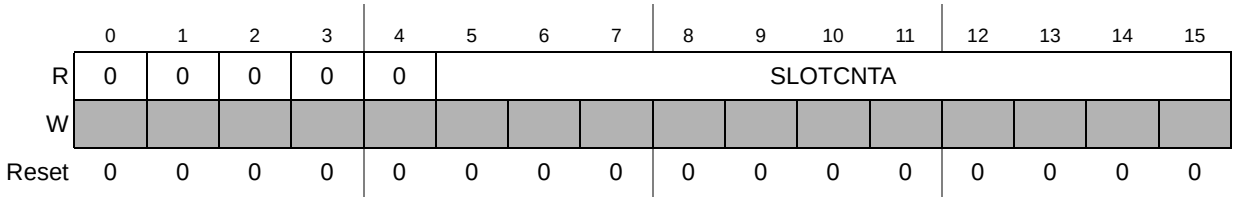
Table 362. FR_CYCTR Field Descriptions

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.

27.5.2.27 Slot Counter Channel A Register (FR_SLTCTAR)

Figure 455. Slot Counter Channel A Register (FR_SLTCTAR)

Base + 0x0034



This register provides the number of the current slot in the current communication cycle for channel A.

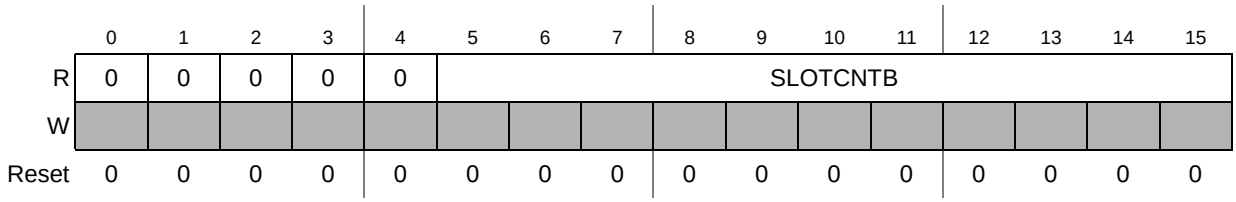
Table 363. FR_SLTCTAR Field Descriptions

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.

27.5.2.28 Slot Counter Channel B Register (FR_SLTCTBR)

Figure 456. Slot Counter Channel B Register (FR_SLTCTBR)

Base + 0x0036



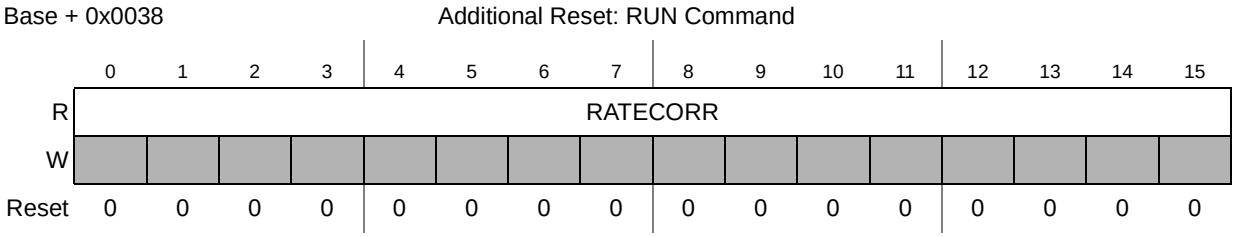
This register provides the number of the current slot in the current communication cycle for channel B.

Table 364. 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.

27.5.2.29 Rate Correction Value Register (FR_RTCORVR)

Figure 457. Rate Correction Value Register (FR_RTCORVR)



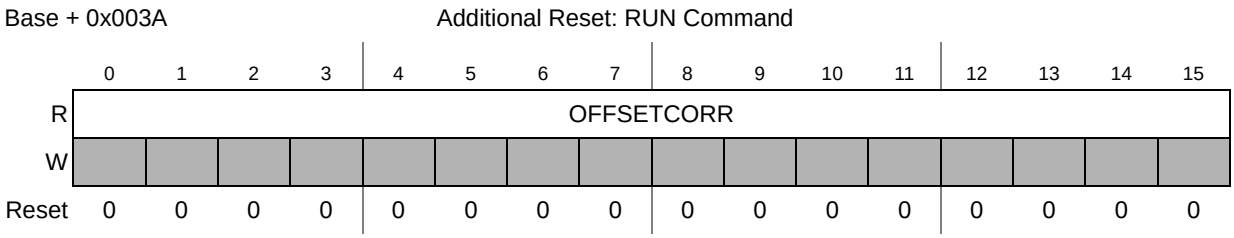
This register provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The CC updates this register during the NIT of each odd numbered communication cycle.

Table 365. 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.*

27.5.2.30 Offset Correction Value Register (FR_OFCORVR)

Figure 458. 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 366. 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.*

27.5.2.31 Combined Interrupt Flag Register (FR_CIFR)

Figure 459. 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

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 590*. 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 367. 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

Table 367. FR_CIFR Field Descriptions

Field	Description
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 single or double 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.

27.5.2.32 System Memory Access Time-Out Register (FR_SYMATOR)

Figure 460. 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

Table 368. FR_SYMATOR Field Descriptions

Field	Description
TIMEOUT	**System Memory Access Time-Out** — This value is related to the maximum amount of time to finish a system bus access in order to ensure correct frame transmission and reception. For a detailed description see *Section 27.6.19.2, "System Bus Access Timeout"*.

27.5.2.33 Sync Frame Counter Register (FR_SFCNTR)

Figure 461. 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

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.

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 369. FR_SFCNTR Field Descriptions

Field	Description
SFEVB	Sync Frames Channel B, even cycle — protocol related variable: size of (*vSyncIdListB* 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 (*vSyncIdListA* 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 (*vSyncIdListB* 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 (*vSyncIdListA* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.

27.5.2.34 Sync Frame Table Offset Register (FR_SFTOR)

Figure 462. 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

This register defines the FlexRay memory area related offset for sync frame tables. For more details, see *Section 27.6.12, "Sync Frame ID and Sync Frame Deviation Tables"*.

Table 370. 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.

27.5.2.35 Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)

Figure 463. 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	SDVEN	SIDEN
W	ELKT	OLKT												OPT		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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 27.6.12, "Sync Frame ID and Sync Frame Deviation Tables"*.

Table 371. FR_SFTCCSR Field Descriptions

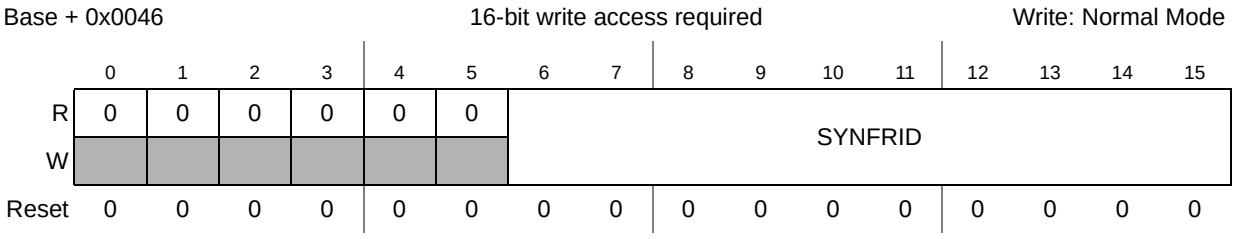
Field	Description
ELKT	**Even Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the even cycle tables. 0 No effect 1 Triggers lock/unlock of the even cycle tables.
OLKT	**Odd Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the odd cycle 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.

Table 371. FR_SFTCCSR Field Descriptions

Field	Description
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).
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

27.5.2.36 Sync Frame ID Rejection Filter Register (FR_SFIDRFR)

Figure 464. 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 372. 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 27.6.15.2, "Sync Frame Rejection Filtering"*.

27.5.2.37 Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)

Figure 465. 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

This register defines the sync frame acceptance filter value. For details on filtering, see *Section 27.6.15, "Sync Frame Filtering"*.

Table 373. FR_SFIDAFVR Field Descriptions

Field	Description
FVAL	**Filter Value** — This field defines the value for the sync frame acceptance filtering.

27.5.2.38 Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)

Figure 466. 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

This register defines the sync frame acceptance filter mask. For details on filtering see *Section 27.6.15.1, "Sync Frame Acceptance Filtering"*.

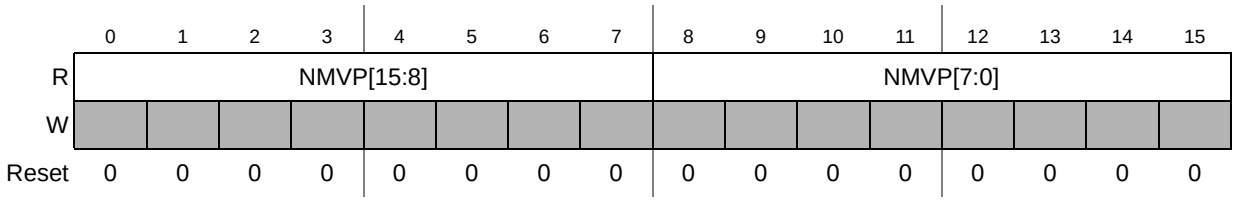
Table 374. FR_SFIDAFMR Field Descriptions

Field	Description
FMSK	**Filter Mask** — This field defines the mask for the sync frame acceptance filtering.

27.5.2.39 Network Management Vector Registers (FR_NMVR0–FR_NMVR5)

Figure 467. 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)



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 375. 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 376***.

Table 376. 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

27.5.2.40 Network Management Vector Length Register (FR_NMVLR)

Figure 468. 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

This register defines the length of the network management vector in bytes.

Table 377. 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.

27.5.2.41 Timer Configuration and Control Register (FR_TICCR)

Figure 469. 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

This register is used to configure and control the two timers T1 and T2. For timer details, see *Section 27.6.17, "Timer Support"*. The Timer T1 is an absolute timer. The Timer T2 can be configured as an absolute or relative timer.

Table 378. 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

Table 378. FR_TICCR Field Descriptions

Field	Description
T2TR	**Timer T2 Trigger** — This trigger bit is used to start timer T2. 0 no effect 1 start timer T2
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.

27.5.2.42 Timer 1 Cycle Set Register (FR_TI1CYSR)

Figure 470. 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

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 27.6.17.1, "Absolute Timer T1"*.

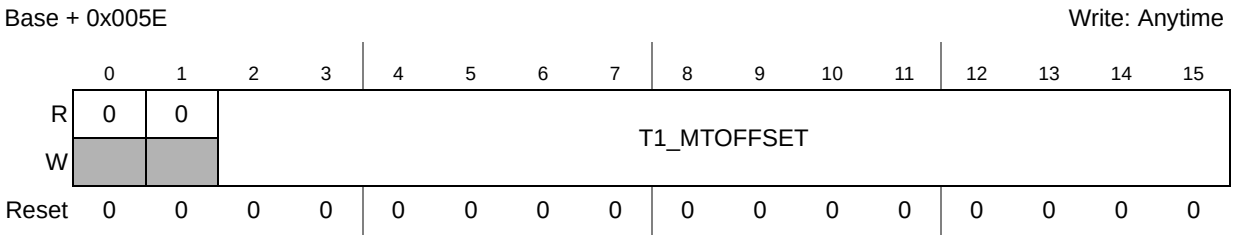
Table 379. FR_TI1CYSR Field Descriptions

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.

27.5.2.43 Timer 1 Macrotick Offset Register (FR_TI1MTOR)

Figure 471. 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 27.6.17.1, "Absolute Timer T1"*.

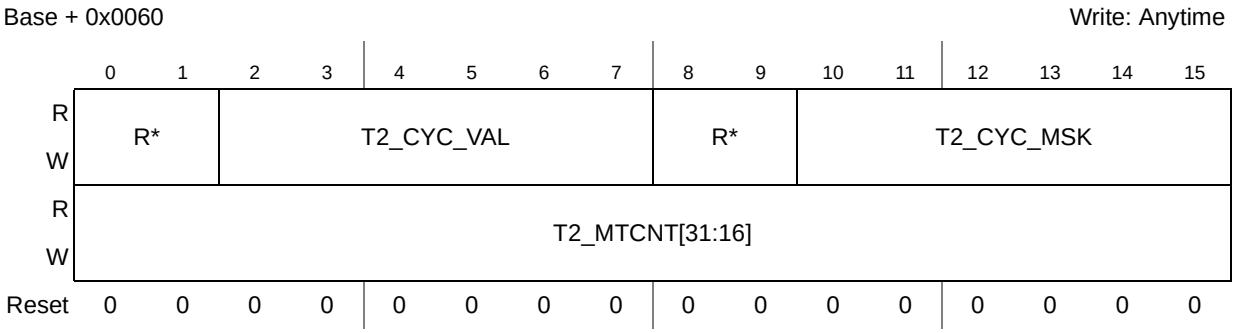
Table 380. FR_TI1MTOR Field Descriptions

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.

27.5.2.44 Timer 2 Configuration Register 0 (FR_TI2CR0)

Figure 472. 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 27.6.17.2, "Absolute / Relative Timer T2"*.

Table 381. 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.

Table 381. FR_TI2CR0 Field Descriptions

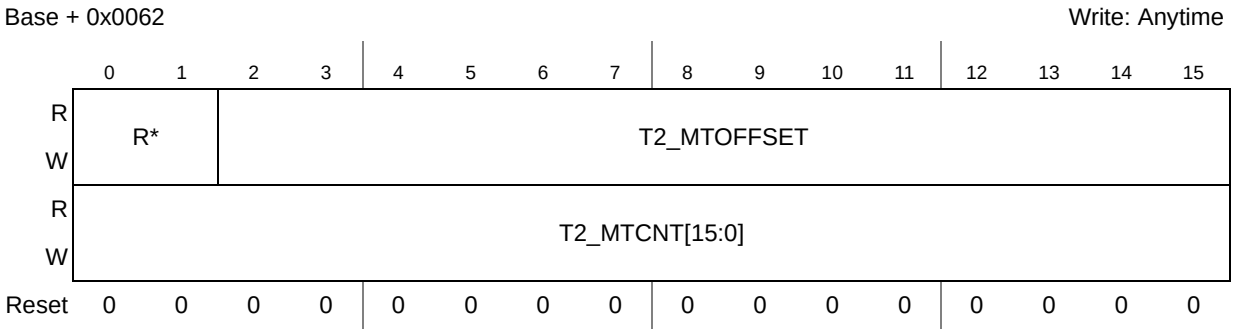
Field	Description
Fields for relative timer T2 (FR_TICCR[T2_CFG = 1)	
T2_MTCNT[31:16]	**Timer T2 Macrotick High Word** — This field defines the high word of the macrotick count for timer T2.

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.

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.

27.5.2.45 Timer 2 Configuration Register 1 (FR_TI2CR1)

Figure 473. 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 27.6.17.2, "Absolute / Relative Timer T2"*.

Table 382. 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.

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.

27.5.2.46 Slot Status Selection Register (FR_SSSR)

Figure 474. 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

This register is used to access the four internal non memory-mapped slot status selection registers FR_SSSR0 to FR_SSSR3. Each internal registers 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 384*. For a detailed description of slot status monitoring, refer to *Section 27.6.18, "Slot Status Monitoring*".

Table 383. 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 384. 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]

27.5.2.47 Slot Status Counter Condition Register (FR_SSCCR)

Figure 475. 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

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 386*. For a detailed description of slot status counters, refer to *Section 27.6.18.4, "Slot Status Counter Registers*".

Table 385. 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.

Table 385. FR_SSCCR Field Descriptions (continued)

Field	Description
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 386. Mapping between internal FR_SSCCRn and FR_SSCRn

Condition Register	Condition Defined for Register
FR_SSCCR0	FR_SSCR0
FR_SSCCR1	FR_SSCR1
FR_SSCCR2	FR_SSCR2
FR_SSCCR3	FR_SSCR3

27.5.2.48 Slot Status Registers (FR_SSR0–FR_SSR7)

Figure 476. 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

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 586*. The register bits are directly related to the protocol variables and described in more detail in *Section 27.6.18, "Slot Status Monitoring*".

Note: Slot status information of the message buffers should not be used when any one of the the error flags FR_CHIERFR[SBCF_EF] or FR_CHIERFR[ILSA_EF] is set.

Table 387. 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
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

Table 387. FR_SSR0–FR_SSR7 Field Descriptions (continued)

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
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1

27.5.2.49 Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)

Figure 477. Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)

Base + 0x0078 (FR_SSCR0)
Base + 0x007A (FR_SSCR1) Additional Reset: RUN Command
Base + 0x007C (FR_SSCR2)
Base + 0x007E (FR_SSCR3)

	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

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 27.6.18.4, "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 388. FR_SSCR0–FR_SSCR3 Field Descriptions

Field	Description
SLOTSTATUSCNT	**Slot Status Counter** — This field provides the current value of the Slot Status Counter.

27.5.2.50 MTS A Configuration Register (FR_MTSACFR)

Figure 478. 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

This register controls the transmission of the Media Access Test Symbol MTS on channel A. For more details, see *Section 27.6.13, "MTS Generation"*.

Table 389. 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.

27.5.2.51 MTS B Configuration Register (MTSBCFR)

Figure 479. 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

This register controls the transmission of the Media Access Test Symbol MTS on channel B. For more details, see *Section 27.6.13, "MTS Generation"*.

Table 390. 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.

27.5.2.52 Receive Shadow Buffer Index Register (FR_RSBIR)

Figure 480. 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	0	RSBIDX						
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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 27.6.6.3.5, "Receive Shadow Buffers Concept"*.

Table 391. 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	**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.

27.5.2.53 Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)

Figure 481. Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR)

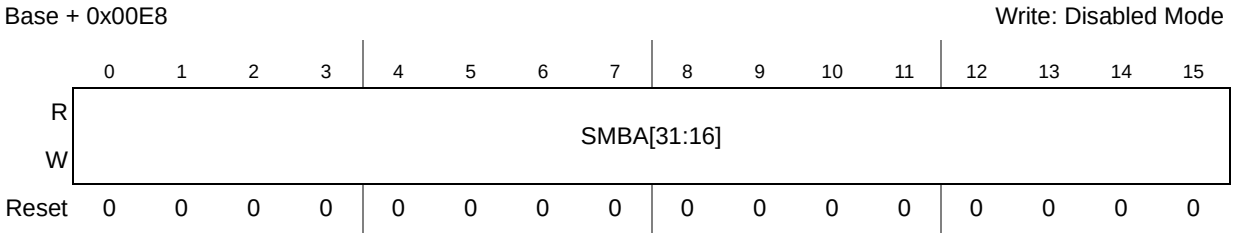
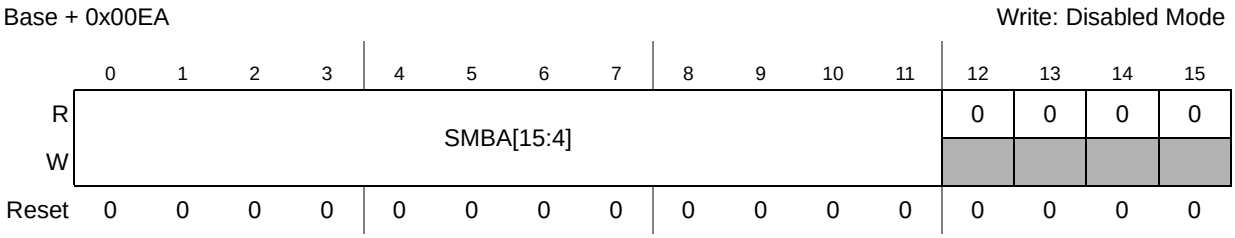


Figure 482. Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR)



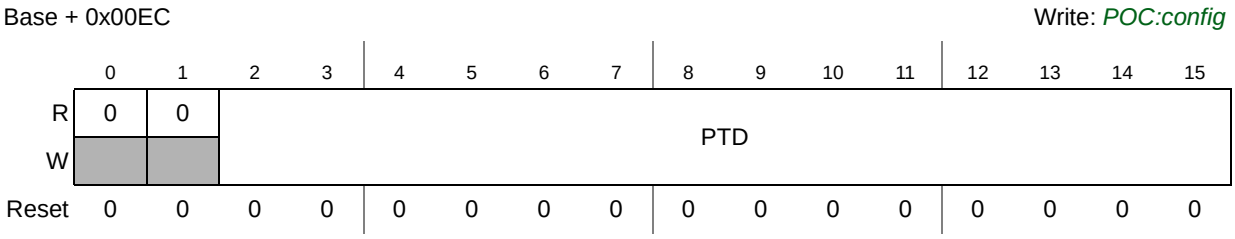
These registers define the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. The system memory base address is used by the BMIF to calculate the physical memory address for system memory accesses for the FIFOs.

Table 392. 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.

27.5.2.54 Receive FIFO Periodic Timer Register (FR_RFPTR)

Figure 483. 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 27.6.9.3, "FIFO Periodic Timer"*).

Table 393. 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

27.5.2.55 Receive FIFO Watermark and Selection Register (FR_RFWMSR)

Figure 484. 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

This register is used to

- select a receiver FIFO for subsequent programming access through the receiver FIFO configuration registers summarized in *Table 394*.
- to define the watermark for the selected FIFO.

Table 394. 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 395. 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

27.5.2.56 Receive FIFO Start Index Register (FR_RFSIR)

Figure 485. 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

This register defines the message buffer header index of the first message buffer of the selected FIFO.

Table 396. 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.

27.5.2.57 Receive FIFO Depth and Size Register (RFDSR)

Figure 486. 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

This register defines the structure of the selected FIFO, i.e. the number of entries and the size of each entry.

Table 397. 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.
$ENTRY_SIZE_A$ $ENTRY_SIZE_B$	**Entry Size** — This field defines the size of the frame data sections for the selected FIFO in 2 byte entities.

27.5.2.58 Receive FIFO A Read Index Register (FR_RFARIR)

Figure 487. 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

This register provides the message buffer header index of the next available FIFO A entry that the application can read.

Table 398. 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. If the old style FIFO mode is configured (FR_MCR[FIMD]=0), the CC updates this index by 1 entry, when the application writes to the FAFAIF flag in the *Global Interrupt Flag and Enable Register (FR_GIFER)*. If the new style FIFO mode is configured (FR_MCR[FIMD]=1), the CC updates this index by PCA entries, when the application writes to the *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)*.

Note: If the FIFO is empty, the RDIDX field points to an physical message buffer with invalid content.

27.5.2.59 Receive FIFO B Read Index Register (FR_RFBRIR)

Figure 488. 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

This register provides the message buffer header index of the next available FIFO B entry that the application can read.

Table 399. 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.

27.5.2.60 Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

Figure 489. 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

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 400. 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.

27.5.2.61 Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)

Figure 490. 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

This register defines the filter value for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see *Section 27.6.9.9, "FIFO Filtering"*.

Table 401. FR_RFMIDAFVR Field Descriptions

Field	Description
$MIDAFVAL_A$ $MIDAFVAL_B$	**Message ID Acceptance Filter Value** — Filter value for the message ID acceptance filter.

27.5.2.62 Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)

Figure 491. 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	MIDAFMSK$_A$/MIDAFMSK$_B$															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines the filter mask for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see *Section 27.6.9.9, "FIFO Filtering"*.

Table 402. FR_RFMIDAFMR Field Descriptions

Field	Description
MIDAFMSK$_A$ MIDAFMSK$_B$	**Message ID Acceptance Filter Mask** — Filter mask for the message ID acceptance filter.

27.5.2.63 Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)

Figure 492. 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

This register defines the filter value for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see *Section 27.6.9.9, "FIFO Filtering"*.

Table 403. FR_RFFIDRFVR Field Descriptions

Field	Description
FIDRFVAL$_A$ FIDRFVAL$_B$	**Frame ID Rejection Filter Value** — Filter value for the frame ID rejection filter.

27.5.2.64 Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)

Figure 493. 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

This register defines the filter mask for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see *Section 27.6.9.9, "FIFO Filtering"*.

Table 404. FR_RFFIDRFMR Field Descriptions

Field	Description
FIDRFMSK	**Frame ID Rejection Filter Mask** — Filter mask for the frame ID rejection filter.

27.5.2.65 Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

Figure 494. 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

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 27.6.9.9, "FIFO Filtering"*.

Table 405. 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

Table 405. FR_RFRFCFR Field Descriptions

Field	Description
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.

27.5.2.66 Receive FIFO Range Filter Control Register (FR_RFRFCTR)

Figure 495. 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

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 406. 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

Table 406. FR_RFRFCTR Field Descriptions

Field	Description
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

27.5.2.67 Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)

Figure 496. 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

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 407. FR_LDTXSLAR Field Descriptions

Field	Description
LASTDYNTXSLOTA	**Last Dynamic Transmission Slot Channel A** — protocol related variable: *zLastDynTxSlot* channel A Number of the last transmission slot in the dynamic segment for channel A. If no frame was transmitted during the dynamic segment on channel A, the value of this field is set to 0.

27.5.2.68 Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)

Figure 497. 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

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 408. FR_LDTXSLBR Field Descriptions

Field	Description
LASTDYNTXSLOTB	**Last Dynamic Transmission Slot Channel B** — protocol related variable: *zLastDynTxSlot* channel B Number of the last transmission slot in the dynamic segment for channel B. If no frame was transmitted during the dynamic segment on channel B the value of this field is set to 0.

27.5.2.69 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 409*. 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 409. 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
symbol_window_exists	*gdSymbolWindow*!=0	0	1	bool	9
symbol_window_after_action_point	*gdSymbolWindow* - *gdActionPointOffset* - 1			MT	6
tss_transmitter	gdTSSTransmitter			gdBit	5
wakeup_symbol_rx_idle	gdWakeupSymbolRxIdle			gdBit	5
wakeup_symbol_rx_low	gdWakeupSymbolRxLow			gdBit	3
wakeup_symbol_rx_window	gdWakeupSymbolRxWindow			gdBit	4
wakeup_symbol_tx_idle	gdWakeupSymbolTxIdle			gdBit	8
wakeup_symbol_tx_low	gdWakeupSymbolTxLow			gdBit	5
noise_listen_timeout	(*gListenNoise* * *pdListenTimeout*) - 1			µT	16/17
macro_initial_offset_a	pMacroInitialOffset[A]			MT	6
macro_initial_offset_b	pMacroInitialOffset[B]			MT	16
macro_per_cycle	gMacroPerCycle			MT	10
macro_after_first_static_slot	*gMacroPerCycle* - *gdStaticSlot*			MT	1
macro_after_offset_correction	*gMacroPerCycle* - *gOffsetCorrectionStart*			MT	28

Table 409. Protocol Configuration Register Fields (continued)

Name	Description[1]	Min	Max	Unit	FR_PCR
max_without_clock_correction_fatal	gMaxWithoutClockCorrectionFatal			cyclepairs	8
max_without_clock_correction_passive	gMaxWithoutClockCorrectionPassive			cyclepairs	8
minislot_exists	*gNumberOfMinislots*!=0	0	1	bool	9
minislots_max	*gNumberOfMinislots* - 1			minislot	29
number_of_static_slots	gNumberOfStaticSlots			static slot	2
offset_correction_start	gOffsetCorrectionStart			MT	11
payload_length_static	gPayloadLengthStatic			2-bytes	19
max_payload_length_dynamic	pPayloadLengthDynMax			2-bytes	24
first_minislot_action_point_offset	max(*gdActionPointOffset*, *gdMinislotActionPointOffset*) - 1			MT	13
allow_halt_due_to_clock	pAllowHaltDueToClock			bool	26
allow_passive_to_active	pAllowPassiveToActive			cyclepairs	12
cluster_drift_damping	pClusterDriftDamping			µT	24
comp_accepted_startup_range_a	*pdAcceptedStartupRange* - *pDelayCompensation[A]*			µT	22
comp_accepted_startup_range_b	*pdAcceptedStartupRange* - *pDelayCompensation[B]*			µT	26
listen_timeout	*pdListenTimeout* - 1			µT	14/15
key_slot_id	pKeySlotId			number	18
key_slot_used_for_startup	pKeySlotUsedForStartup			bool	11
key_slot_used_for_sync	pKeySlotUsedForSync			bool	11
latest_tx	*gNumberOfMinislots* - *pLatestTx*			minislot	21
sync_node_max	gSyncNodeMax			number	30
micro_initial_offset_a	pMicroInitialOffset[A]			µT	20
micro_initial_offset_b	pMicroInitialOffset[B]			µT	20
micro_per_cycle	pMicroPerCycle			µT	22/23
micro_per_cycle_min	pMicroPerCycle - pdMaxDrift			µT	24/25
micro_per_cycle_max	pMicroPerCycle + pdMaxDrift			µT	26/27
micro_per_macro_nom_half	round(*pMicroPerMacroNom* / 2)			µT	7
offset_correction_out	pOffsetCorrectionOut			µT	9
rate_correction_out	pRateCorrectionOut			µT	14
single_slot_enabled	pSingleSlotEnabled			bool	10
wakeup_channel	pWakeupChannel	see *Table 410*			10
wakeup_pattern	pWakeupPattern			number	18
decoding_correction_a	*pDecodingCorrection* + *pDelayCompensation[A]* + 2			µT	19

Table 409. Protocol Configuration Register Fields (continued)

Name	Description[1]	Min	Max	Unit	FR_PCR
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 410. Wakeup Channel Selection

wakeup_channel	Wakeup Channel
0	A
1	B

27.5.2.69.1 Protocol Configuration Register 0 (FR_PCR0)

Figure 498. Protocol Configuration Register 0 (FR_PCR0)

Base + 0x00A0 Write: *POC:config*

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

27.5.2.69.2 Protocol Configuration Register 1 (FR_PCR1)

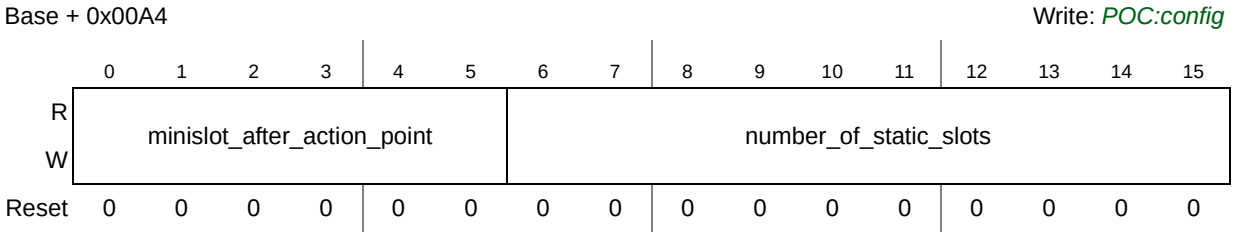
Figure 499. Protocol Configuration Register 1 (FR_PCR1)

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

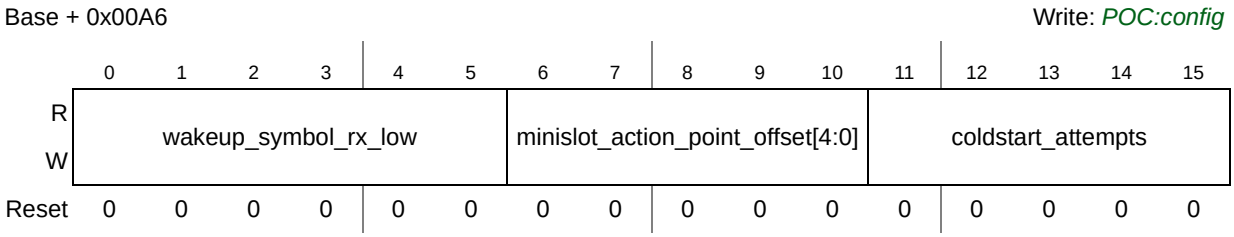
27.5.2.69.3 Protocol Configuration Register 2 (FR_PCR2)

Figure 500. Protocol Configuration Register 2 (FR_PCR2)



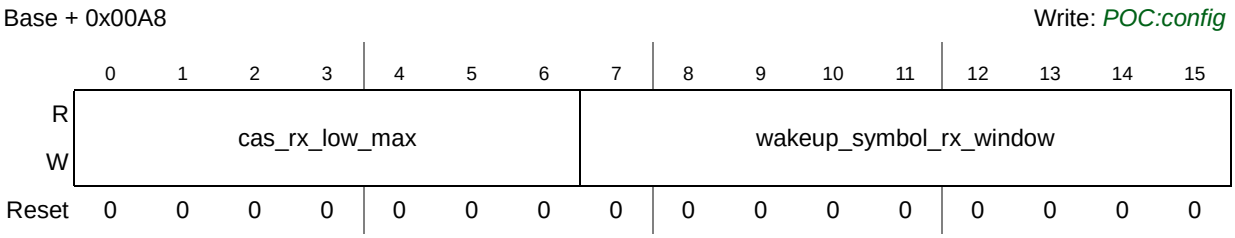
27.5.2.69.4 Protocol Configuration Register 3 (FR_PCR3)

Figure 501. Protocol Configuration Register 3 (FR_PCR3)



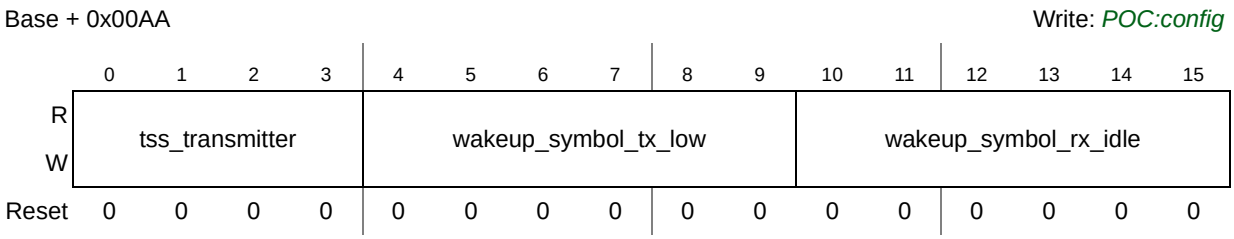
27.5.2.69.5 Protocol Configuration Register 4 (FR_PCR4)

Figure 502. Protocol Configuration Register 4 (FR_PCR4)



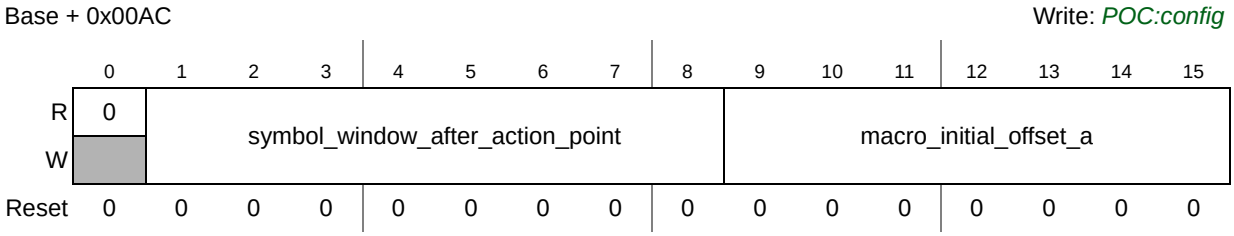
27.5.2.69.6 Protocol Configuration Register 5 (FR_PCR5)

Figure 503. Protocol Configuration Register 5 (FR_PCR5)



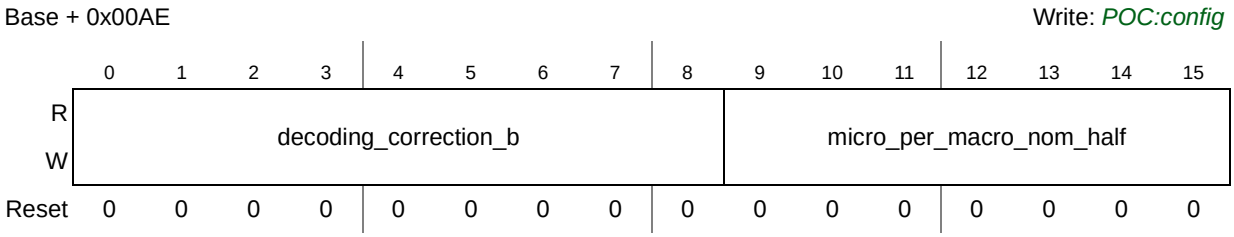
27.5.2.69.7 Protocol Configuration Register 6 (FR_PCR6)

Figure 504. Protocol Configuration Register 6 (FR_PCR6)



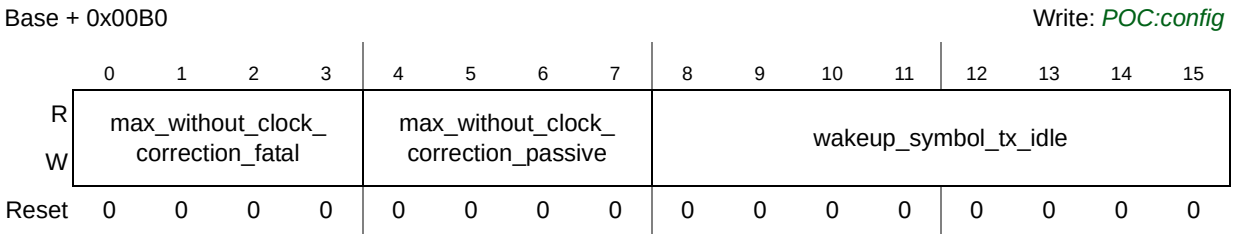
27.5.2.69.8 Protocol Configuration Register 7 (FR_PCR7)

Figure 505. Protocol Configuration Register 7 (FR_PCR7)



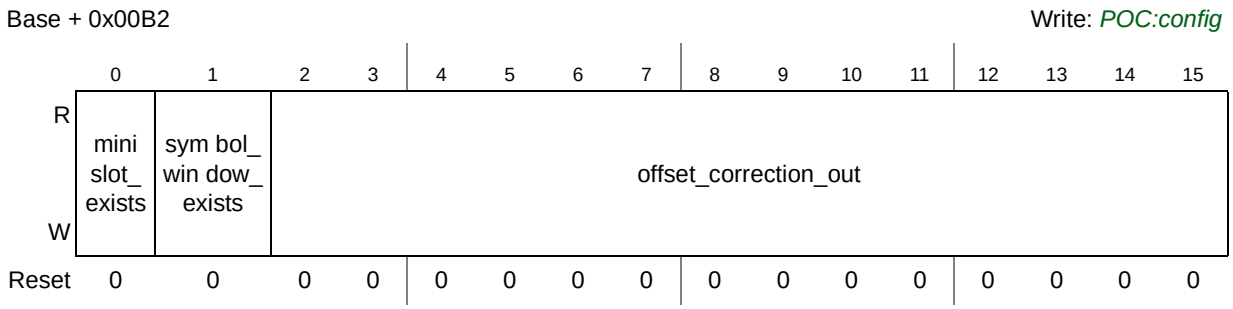
27.5.2.69.9 Protocol Configuration Register 8 (FR_PCR8)

Figure 506. Protocol Configuration Register 8 (FR_PCR8)



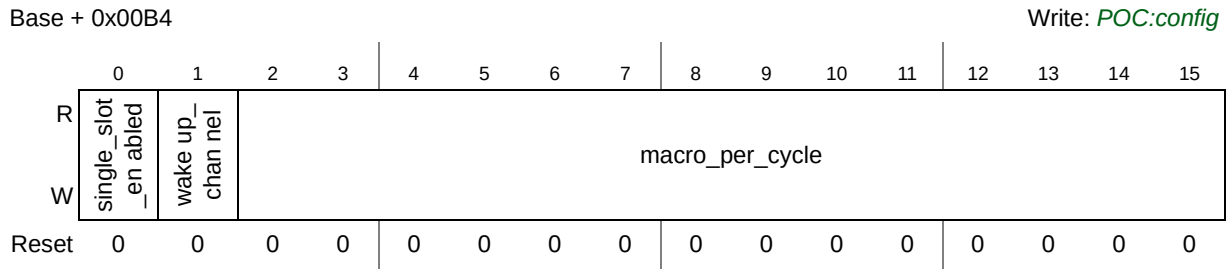
27.5.2.69.10 Protocol Configuration Register 9 (FR_PCR9)

Figure 507. Protocol Configuration Register 9 (FR_PCR9)



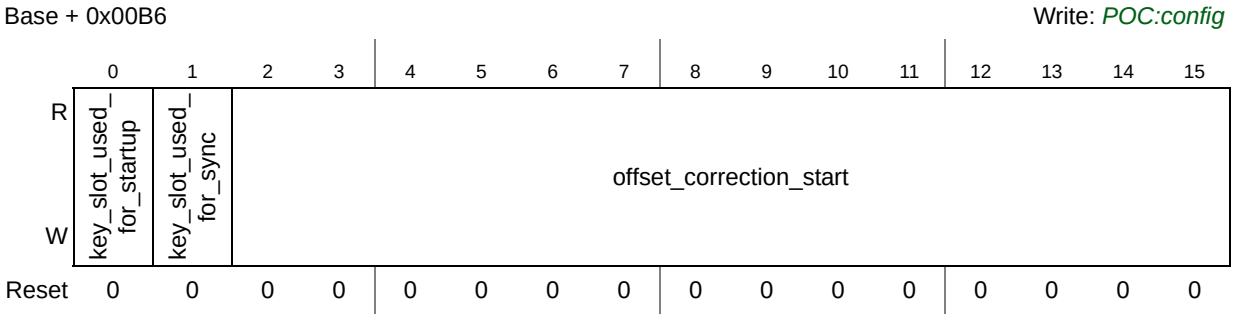
27.5.2.69.11 Protocol Configuration Register 10 (FR_PCR10)

Figure 508. Protocol Configuration Register 10 (FR_PCR10)



27.5.2.69.12 Protocol Configuration Register 11 (FR_PCR11)

Figure 509. Protocol Configuration Register 11 (FR_PCR11)



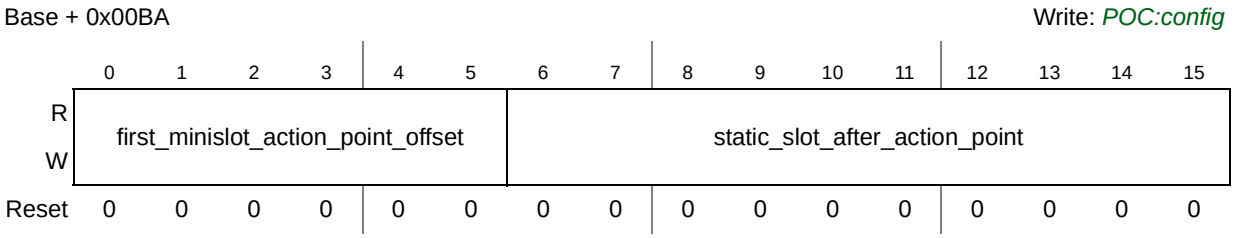
27.5.2.69.13 Protocol Configuration Register 12 (FR_PCR12)

Figure 510. Protocol Configuration Register 12 (FR_PCR12)



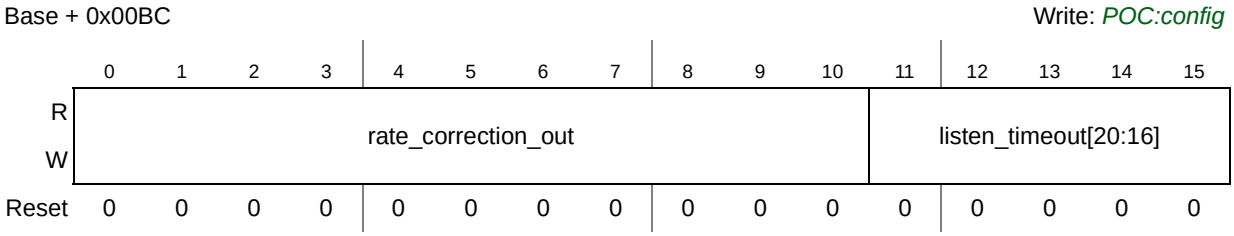
27.5.2.69.14 Protocol Configuration Register 13 (FR_PCR13)

Figure 511. Protocol Configuration Register 13 (FR_PCR13)



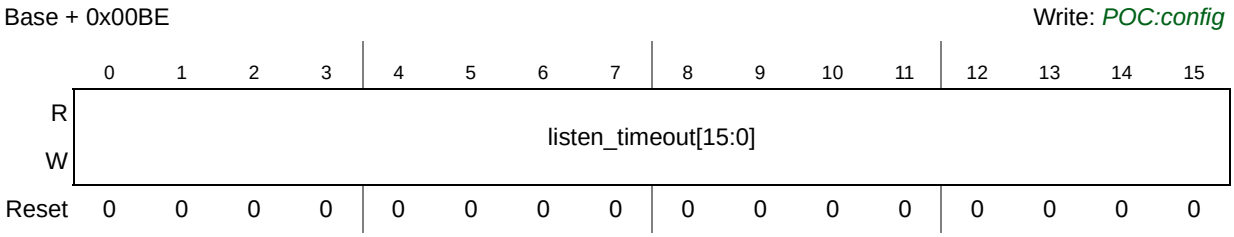
27.5.2.69.15 Protocol Configuration Register 14 (FR_PCR14)

Figure 512. Protocol Configuration Register 14 (FR_PCR14)



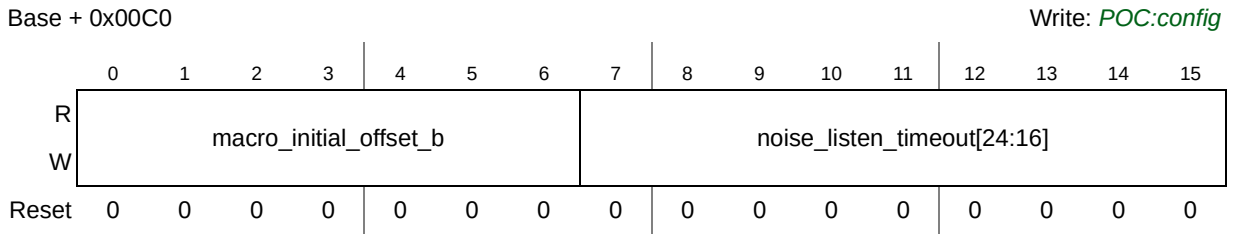
27.5.2.69.16 Protocol Configuration Register 15 (FR_PCR15)

Figure 513. Protocol Configuration Register 15 (FR_PCR15)



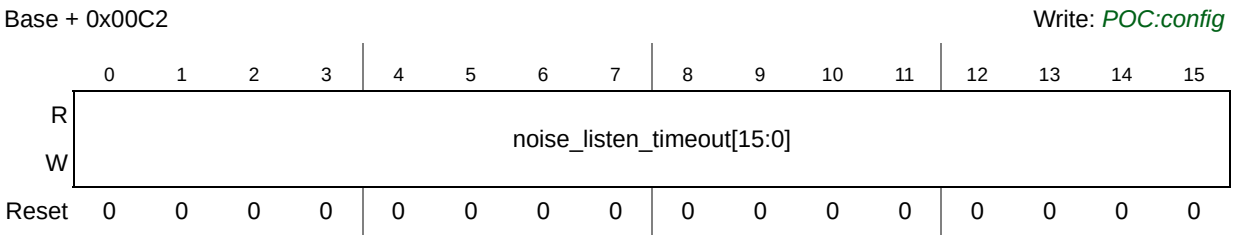
27.5.2.69.17 Protocol Configuration Register 16 (FR_PCR16)

Figure 514. Protocol Configuration Register 16 (FR_PCR16)



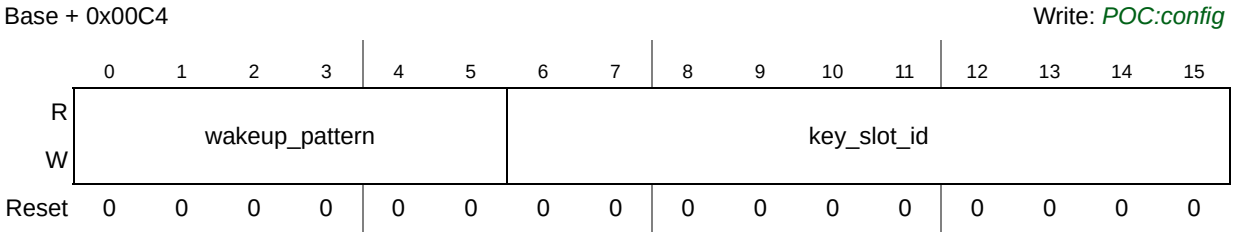
27.5.2.69.18 Protocol Configuration Register 17 (FR_PCR17)

Figure 515. Protocol Configuration Register 17 (FR_PCR17)



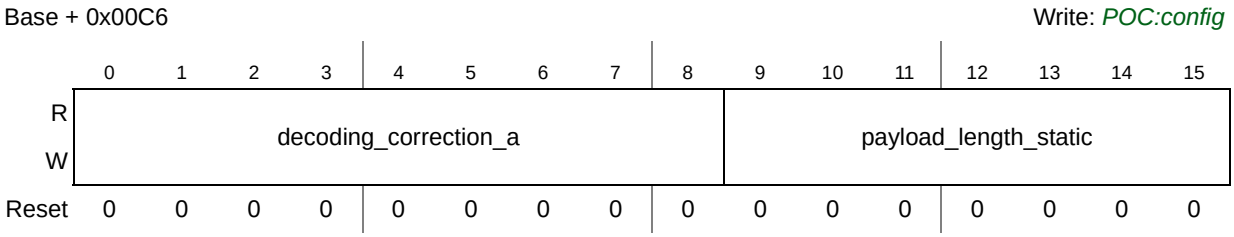
27.5.2.69.19 Protocol Configuration Register 18 (FR_PCR18)

Figure 516. Protocol Configuration Register 18 (FR_PCR18)



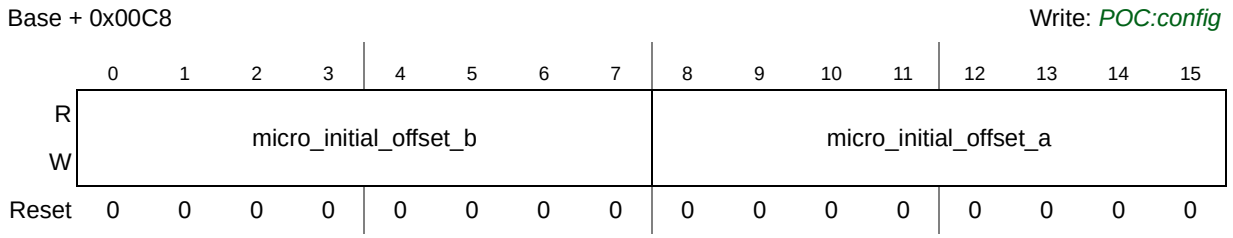
27.5.2.69.20 Protocol Configuration Register 19 (FR_PCR19)

Figure 517. Protocol Configuration Register 19 (FR_PCR19)



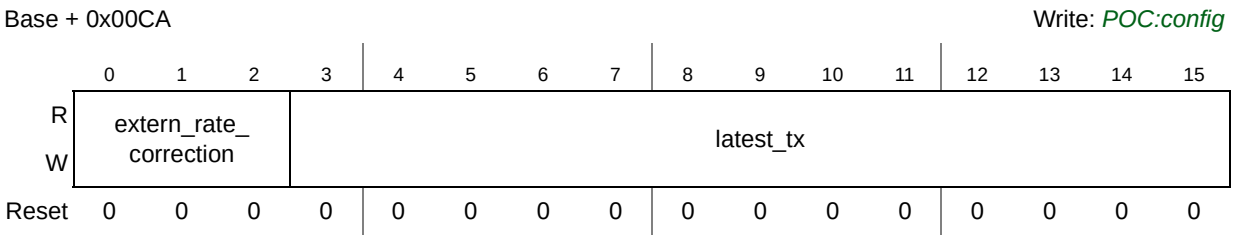
27.5.2.69.21 Protocol Configuration Register 20 (FR_PCR20)

Figure 518. Protocol Configuration Register 20 (FR_PCR20)



27.5.2.69.22 Protocol Configuration Register 21 (FR_PCR21)

Figure 519. Protocol Configuration Register 21 (FR_PCR21)



27.5.2.69.23 Protocol Configuration Register 22 (FR_PCR22)

Figure 520. 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

27.5.2.69.24 Protocol Configuration Register 23 (FR_PCR23)

Figure 521. 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

27.5.2.69.25 Protocol Configuration Register 24 (FR_PCR24)

Figure 522. 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

27.5.2.69.26 Protocol Configuration Register 25 (FR_PCR25)

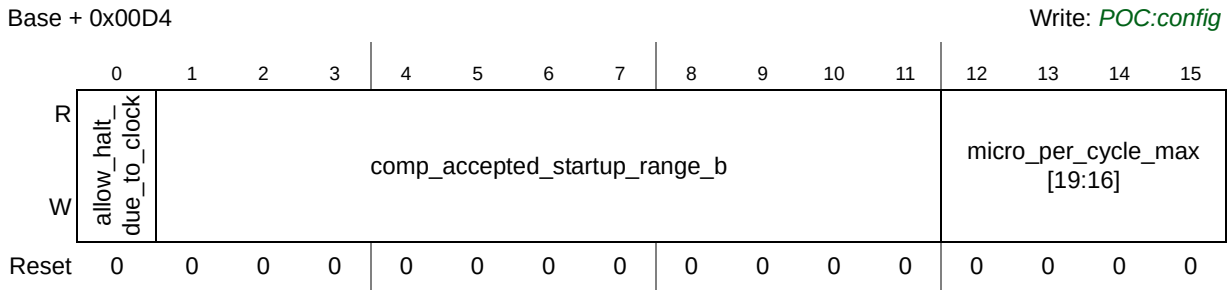
Figure 523. Protocol Configuration Register 25 (FR_PCR25)

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

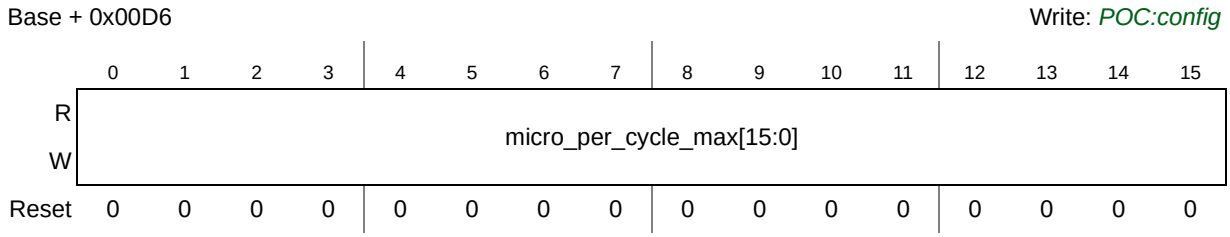
27.5.2.69.27 Protocol Configuration Register 26 (FR_PCR26)

Figure 524. Protocol Configuration Register 26 (FR_PCR26)



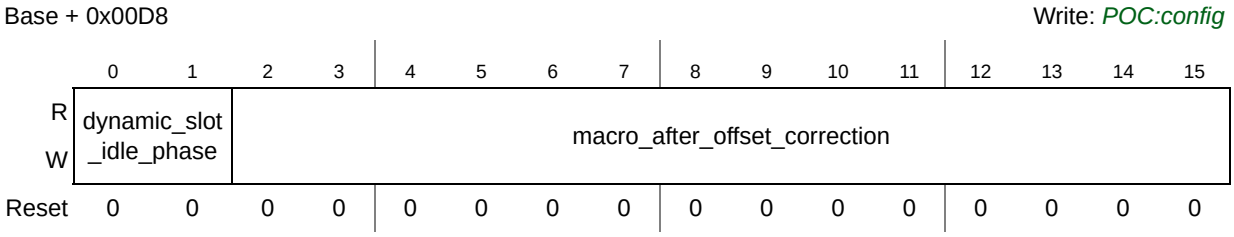
27.5.2.69.28 Protocol Configuration Register 27 (FR_PCR27)

Figure 525. Protocol Configuration Register 27 (FR_PCR27)



27.5.2.69.29 Protocol Configuration Register 28 (FR_PCR28)

Figure 526. Protocol Configuration Register 28 (FR_PCR28)



27.5.2.69.30 Protocol Configuration Register 29 (FR_PCR29)

Figure 527. Protocol Configuration Register 29 (FR_PCR29)



27.5.2.69.31 Protocol Configuration Register 30 (FR_PCR30)

Figure 528. 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

27.5.2.70 ECC Error Interrupt Flag and Enable Register (FR_EEIFER)

Figure 529. 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

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 27.6.24.2, "Memory Error Reporting"*.

Table 411. 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

Table 411. FR_EEIFER Field Descriptions (continued)

Field	Description
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
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

27.5.2.71 ECC Error Report and Injection Control Register (FR_EERICR)

Figure 530. 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

This register configures the error injection and error reporting and provides the selector for the content of the report registers.

Table 412. 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 27.5.2.2, "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 *Note: When the ECC functionality is required to be disabled (i.e.value of the FR_MCR[ECCE] is 0), Error Injection Enable bit FR_EERICR[EIE] should not be set to 1*

27.5.2.72 ECC Error Report Address Register (FR_EERAR)

Figure 531. 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the memory identifier, bank, and address for which the memory error is reported.

Table 413. 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. 000 reset value, updated to a valid value with the first ECC Error. For MID=0: 000 BANK0: PE DRAM [7:0] 001 BANK1: PE DRAM [15:8] others - not used For MID=1: 000 BANK0: FR_MBCCFR(2n) 001 BANK1: FR_MBFIDR(2n) 010 BANK2: FR_MBIDXR(2n) 011 BANK3: FR_MBCCFR(2n+1) 100 BANK4: FR_MBFIDR(2n+1) 101 BANK5: FR_MBIDXR(2n+1) others - not used
ADDR	**Memory Address** — This field provides the address of the failing memory location.

27.5.2.73 ECC Error Report Data Register (FR_EERDR)

Figure 532. ECC Error Report Data Register (FR_EERDR)

Base + 0x00F6

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

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 415.

Table 414. 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 415. 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	1						FR_MBFIDR(2n)										
CHI LRAM	2										FR_MBIDXR(2n)						
CHI LRAM	3	FR_MBCCFR(2n+1)															
CHI LRAM	4						FR_MBFIDR(2n+1)										
CHI LRAM	5										FR_MBIDXR(2n+1)						

27.5.2.74 ECC Error Report Code Register (FR_EERCR)

Figure 533. 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

This register provides the ecc related information of the reported memory read access.

Table 416. 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 27.6.24.2.2, "PE DRAM Syndrome*" The coding of the CHI LRAM syndrome is shown in *Section 27.6.24.2.4, "CHI LRAM Syndrome*". ERM=1: Checkbits. Shows the ecc checkbits read from the memory.

27.5.2.75 ECC Error Injection Address Register (FR_EEIAR)

Figure 534. 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

This register defines the memory module, bank, and address where the ECC error has to be injected.

Table 417. 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: 000 BANK0: FR_MBCCFR(2n) 001 BANK1: FR_MBFIDR(2n) 010 BANK2: FR_MBIDXR(2n) 011 BANK3: FR_MBCCFR(2n+1) 100 BANK4: FR_MBFIDR(2n+1) 101 BANK5: FR_MBIDXR(2n+1) others reserved
ADDR	**Memory Address** — This flag defines the memory address for ECC error injection.

27.5.2.76 ECC Error Injection Data Register (FR_EEIDR)

Figure 535. 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

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 415*.

Table 418. 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.

27.5.2.77 ECC Error Injection Code Register (FR_EEICR)

Figure 536. 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

This register defines the ecc code distortion pattern for the error injection write.

Table 419. 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.

27.5.2.78 Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)

Figure 537. Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)

Base + 0x0100 (FR_MBCCSR0)
Base + 0x0108 (FR_MBCCSR1)
...
Base + 0x02F8 (FR_MBCCSR63)

Write: MCM, MBT, MTD: *POC:config* or MB_DIS
CMT: MB_LCK or MB_DIS
EDT, LCKT, MBIE, MBIF: Normal Mode

Additional Reset: CMT, DUP, DVAL, MBIF: Message Buffer Disable

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MCM	MBT	MTD	CMT	0	0	MBIE	0	0	0	DUP	DVAL	EDS	LCKS	MBIF
W					rwm	EDT	LCKT									w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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 27.6.6, "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 420. FR_MBCCSRn Field Descriptions

Field	Description
Message Buffer Configuration	
MCM	**Message Buffer Commit Mode** — This bit configures the commit mode of a double buffered message buffer. 0 Streaming commit mode 1 Immediate commit mode
MBT	**Message Buffer Type** — This bit configures the buffering type of a transmit message buffer. 0 Single buffered message buffer 1 Double buffered message buffer
MTD	**Message Buffer Transfer Direction** — This bit configures the transfer direction of a message buffer. 0 Receive message buffer 1 Transmit message buffer
Message Buffer Control	
CMT	**Commit for Transmission** — This bit indicates if the transmit message buffer data are ready for transmission. 0 Message buffer data not ready for transmission 1 Message buffer data ready for transmission
EDT	**Enable/Disable Trigger** — If the application writes 1 to this bit, a message buffer enable or disable is triggered, depending on the current value EDS status bit is 0. 0 No effect 1 Message buffer enable or disable is triggered
LCKT	**Lock/Unlock Trigger** — If the application writes 1 to this bit, a message buffer lock or unlock is triggered, depending on the current value of the LCKS status bit. 0 No effect 1 Message buffer lock or unlock is triggered
MBIE	**Message Buffer Interrupt Enable** — This control bit defines whether the message buffer will generate an interrupt request when its MBIF flag is set. 0 Interrupt request generation disabled 1 Interrupt request generation enabled

Table 420. FR_MBCCSRn Field Descriptions (continued)

Field	Description
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

27.5.2.79 Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)

Figure 538. Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)

Base + 0x0102 (FR_MBCCFR0)
Base + 0x010A (FR_MBCCFR1)
...
Base + 0x02FA (FR_MBCCFR63)

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	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

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 27.6.7.1, "Message Buffer Cycle Counter Filtering"*.

Table 421. 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 422***.
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 422. 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 (Function not guaranteed)	store first valid frame received on either channel A or channel B	Reserved (Function 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.

27.5.2.80 Message Buffer Frame ID Registers (FR_MBFIDRn)

Figure 539. Message Buffer Frame ID Registers (FR_MBFIDRn)

Base + 0x0104 (FR_MBFIDR0)
Base + 0x010C (FR_MBFIDR1)
...
Base + 0x02FC (FR_MBFIDR63)

16-bit write access required

Write: *POC:config* or MB_DIS



Table 423. 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.

27.5.2.81 Message Buffer Index Registers (FR_MBIDXRn)

Figure 540. Message Buffer Index Registers (FR_MBIDXRn)

Base + 0x0106 (FR_MBIDXR0)
Base + 0x010E (FR_MBIDXR1)
...
Base + 0x02FE (FR_MBIDXR63)

16-bit write access required

Write: *POC:config* or MB_DIS

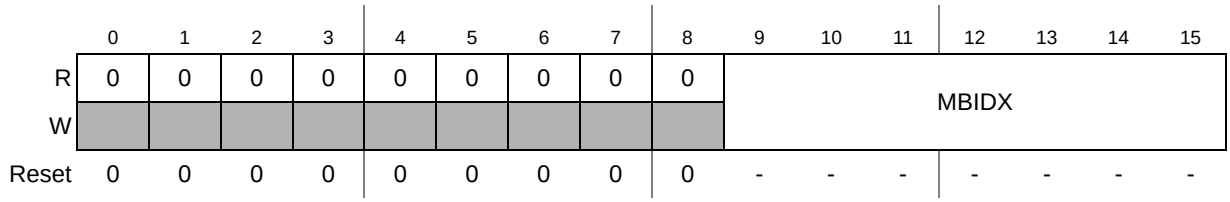


Table 424. FR_MBIDXRn Field Descriptions

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.

27.6 Functional Description

This section provides a detailed description of the functionality implemented in the CC.

27.6.1 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 27.6.3, "Message Buffer Types*". The physical message buffer is located in the FlexRay memory area and is described in *Section 27.6.2, "Physical Message Buffer*".

27.6.2 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 541*.

A physical message buffer consists of two fields, the *message buffer header field* and the *message buffer data field*. The message buffer header field contains the *frame header*, the *data field offset*, and the *slot status*.The message buffer data field contains the *frame data*.

The connection between the two fields is established by the *data field offset*.

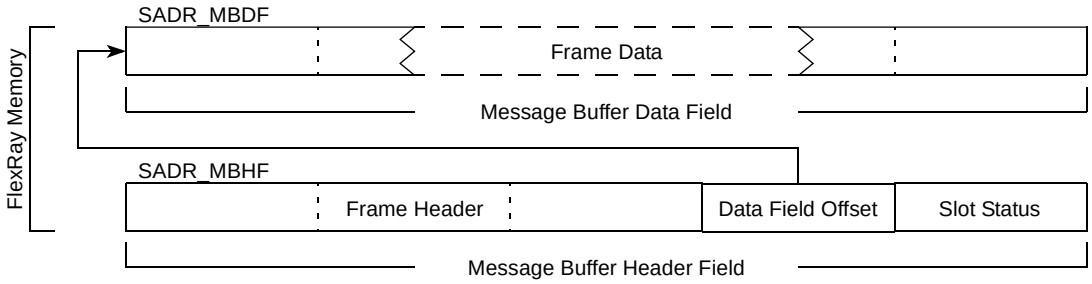


Figure 541. Physical Message Buffer Structure

27.6.2.1 Message Buffer Header Field

The message buffer header field is a contiguous region in the FlexRay memory area and occupies ten bytes. It contains the frame header, the data field offset, and the slot status. Its structure is shown in *Figure 541*. The physical start address *SADR_MBHF* of the message buffer header field must be 16-bit aligned.

27.6.2.1.1 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 27.6.5.2.1, "Frame Header Description"*.

27.6.2.1.2 Data Field Offset

The data field offset follows the frame header in the message buffer data field and occupies two bytes. It contains the offset of the corresponding message buffer data field with respect

to the 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 16*.

Equation 16 SADR_MBDF = [Data Field Offset] + SMBA

27.6.2.1.3 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 27.6.5.2.3, "Slot Status Description"*.

27.6.2.2 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 27.6.3, "Message Buffer Types*".

27.6.3 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.

27.6.3.1 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 27.6.6, "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 542*.

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 17*.

Equation 17 SADR_MBHF = (FR_MBIDXRn[MBIDX] * 10) + SMBA

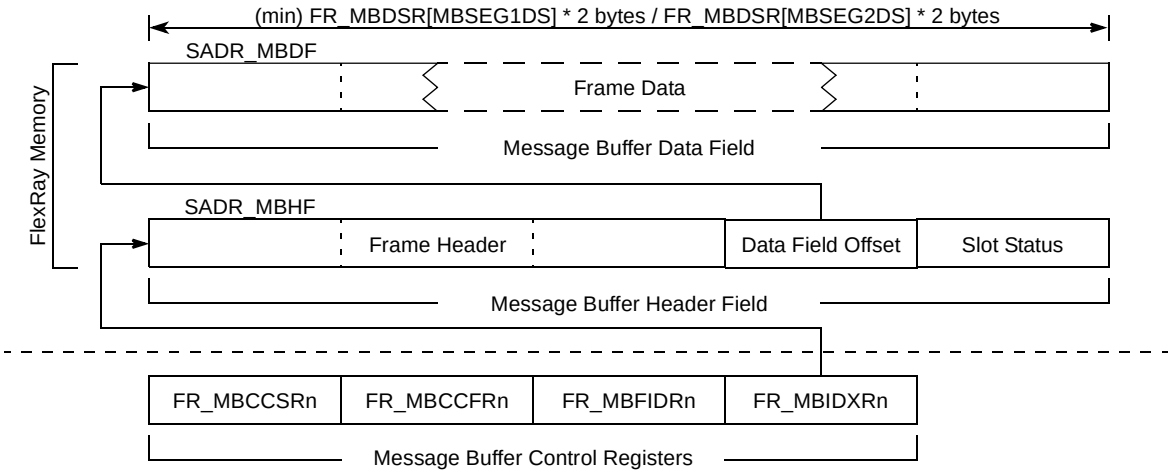


Figure 542. Individual Message Buffer Structure

27.6.3.1.1 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.

27.6.3.2 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 543*. 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 18*.

Equation 18 SADR_MBHF = (FR_RSBIR[RSBIDX] * 10) + 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)*.

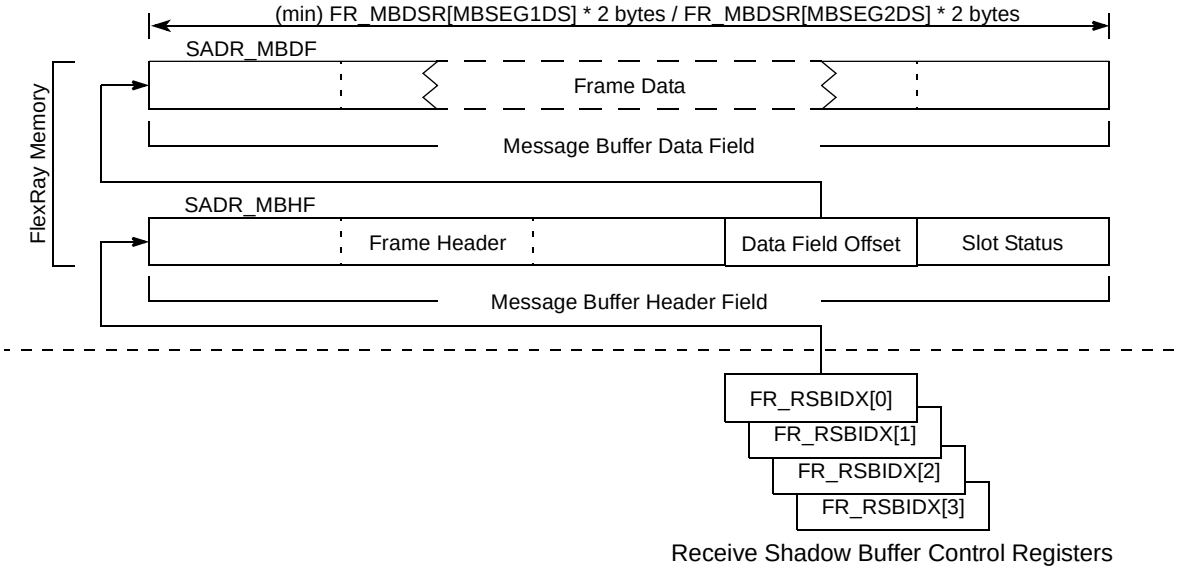


Figure 543. Receive Shadow Buffer Structure

27.6.3.3 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 544*.

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 is defined by the system memory base address register selected by the FIFO address mode bit FR_MCR[FAM].

The start byte address SADR_MBHF[1] of the first message buffer header field that belongs to the receive FIFO in the FlexRay memory area is determined according to *Equation 19*.

Equation 19 SADR_MBHF[1] = (10 * 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 20*.

Equation 20 SADR_MBHF[n] = (10 * (FR_RFSIR[SIDX] + RFDSR[FIFO_DEPTH])) + SMBA

Note: All message buffer header fields assigned to a receive FIFO must be a contiguous region.

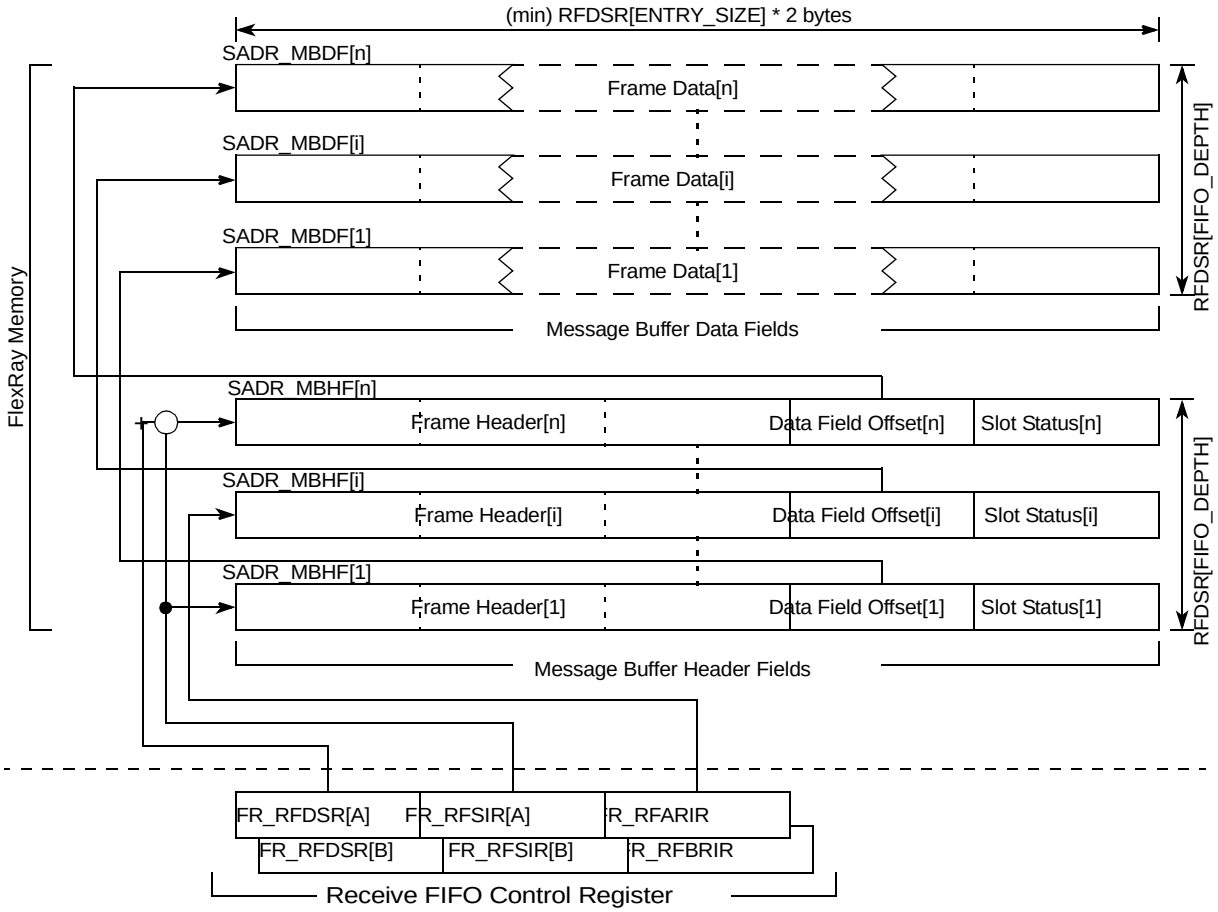


Figure 544. Receive FIFO Structure

27.6.3.4 Message Buffer Configuration and Control Data

This section describes the configuration and control data for each message buffer type.

27.6.3.4.1 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.

27.6.3.4.1.1 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 27.6.3.1.1, "Individual Message Buffer Segments*"

27.6.3.4.1.2 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 MCM, MBT, MTD bits configure 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.

27.6.3.5 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)*.

27.6.3.6 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.

27.6.3.7 Receive FIFO Control and Configuration Data

This section describes the configuration and control data for the two receive FIFOs.

27.6.3.7.1 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 System Memory Base Address Register (FR_RFSYMBADR)*
- *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 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)*

27.6.3.7.2 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)*

27.6.3.7.3 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)*

27.6.4 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].

27.6.4.1 FlexRay Memory Area Layout (FR_MCR[FAM] = 0)

Figure 545 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*.

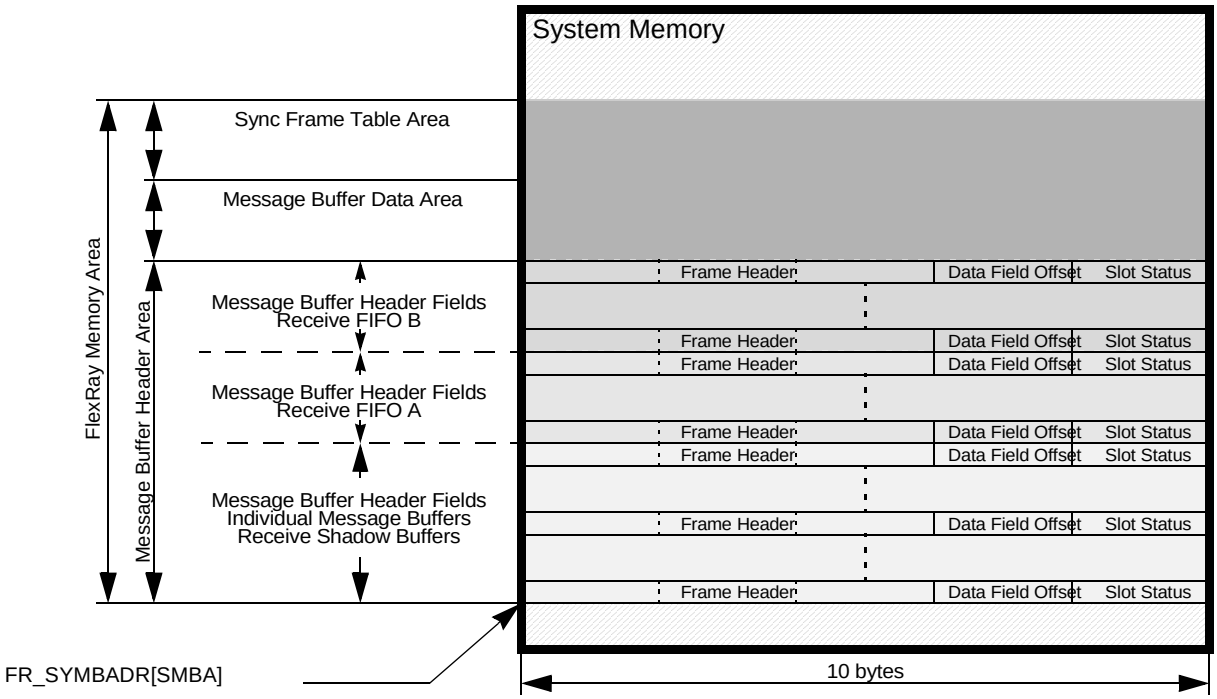


Figure 545. Example of FlexRay Memory Area Layout (FR_MCR[FAM] = 0)

27.6.4.2 FlexRay Memory Area Layout (FR_MCR[FAM] = 1)

Figure 546 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.

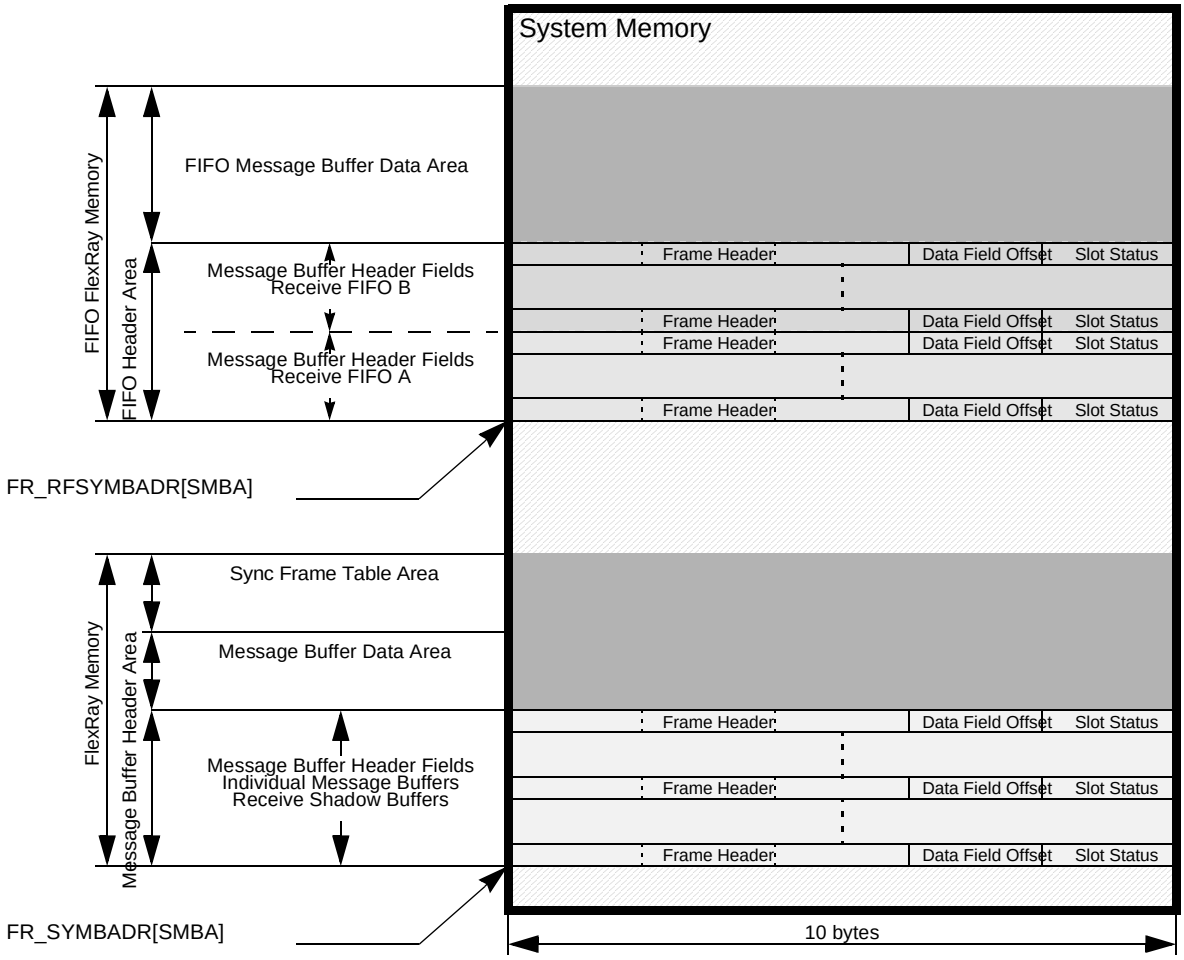


Figure 546. Example of FlexRay Memory Area Layout (FR_MCR[FAM] = 1)

27.6.4.3 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 21*.

Equation 21 **SADR_MBHF = (i * 10) + FR_SYMBADR[SMBA]; (0 <= i < 128)**

2. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill *Equation 22*.

Equation 22 **SADR_MBHF = (i * 10) + FR_SYMBADR[SMBA]; (0 <= i < 1024)**

Equation 23 **SADR_MBHF = (i * 10) + FR_SYMBADR[SMBA]; (0 <= i < 1024)**

3. The message buffer header fields for each FIFO have to be a contiguous area.

27.6.4.4 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 24*.

Equation 24 SADR_MBHF = (i * 10) + FR_SYMBADR[SMBA]; (0 <= i < 128)

27.6.4.5 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 25*.

Equation 25 SADR_MBHF = (i * 10) + FR_RFSYMBADR[SMBA]; (0 <= i < 1024)

2. The message buffer header fields for each FIFO have to be a contiguous area.

27.6.4.6 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.

27.6.4.7 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 27.6.12, "Sync Frame ID and Sync Frame Deviation Tables"* for the description of the sync frame table area.

27.6.5 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.

27.6.5.1 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.

27.6.5.2 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 27.6.2.1, "Message Buffer Header Field"*. Each message buffer header field consists of three sections: the frame header section, the data field offset, and the slot status

section. For a detailed description of the Data Field Offset, see *Section 27.6.2.1.2, "Data Field Offset"*.

27.6.5.2.1 Frame Header Description

27.6.5.2.1.1 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 547*. A detailed description is given in *Table 426*.

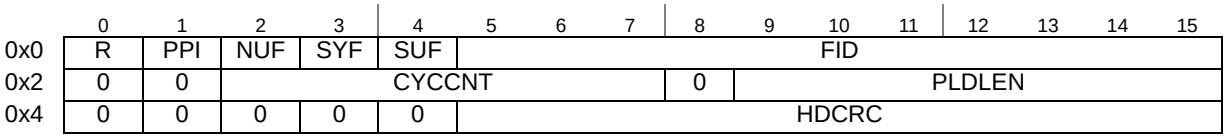


Figure 547. Frame Header Structure (Receive Message Buffer and Receive FIFO)

The structure of the frame header in the message buffer header field for transmit message buffers is given in *Figure 548*. A detailed description is given in *Table 427*. The checks that will be performed are described in *Frame Header Checks*.

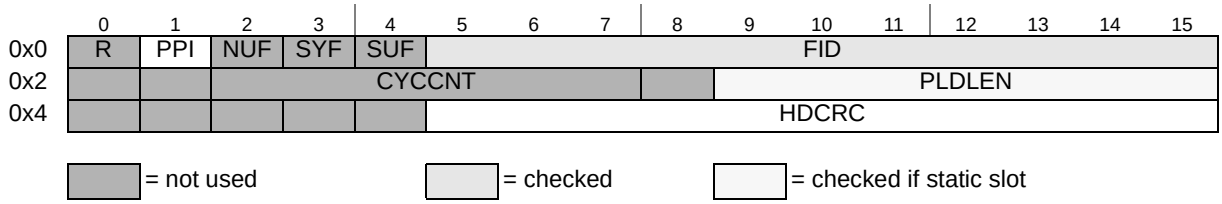


Figure 548. Frame Header Structure (Transmit Message Buffer)

The structure of the frame header in the message buffer header field for transmit message buffers assigned to key slot is given in *Figure 549*.

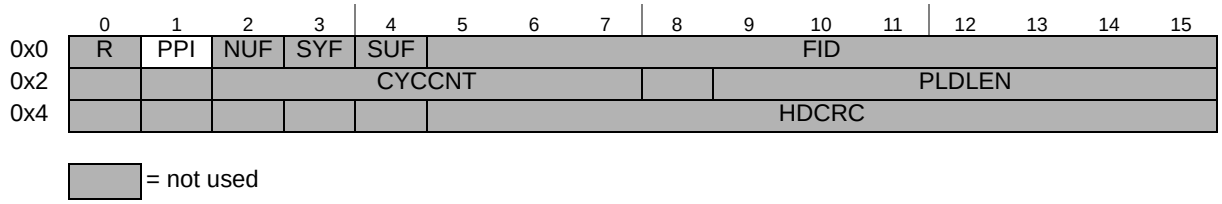


Figure 549. Frame Header Structure (Transmit Message Buffer for Key Slot)

27.6.5.2.1.2 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 425*. This table shows the condition under which the application can write to the frame header entries without corrupting the FlexRay message transmission.

Table 425. Frame Header Write Access Constraints (Transmit Message Buffer)

Field	Single Buffered		Double Buffered			
	Static Segment	Dynamic Segment	Static Segment		Dynamic Segment	
			Commit Side	Transmit Side	Commit Side	Transmit Side
FID	*POC:config* or MB_DIS					
PPI, PLDLEN, HDCRC	*POC:config* or MB_DIS or					
		MB_LCK			MB_LCK	

27.6.5.2.1.3 Frame Header Checks

As shown in *Figure 548* and *Figure 549* 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 426. 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 427. 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.

27.6.5.2.2 Data Field Offset Description

27.6.5.2.2.1 Data Field Offset Content

For a detailed description of the Data Field Offset, see *Section 27.6.2.1.2, "Data Field Offset"*.

27.6.5.2.2.2 Data Field Offset Access

The application shall program the Data Field Offset when configuring the message buffers either in the *POC:config* state or when the message buffer is disabled.

27.6.5.2.3 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.

27.6.5.2.3.1 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 428*.

Table 428. 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 550*
Individual Receive Message Buffer assigned to channel A FR_MBCCFRn[CHA]=1 and FR_MBCCFRn[CHB[=0	see *Figure 551*
Individual Receive Message Buffer assigned to channel B FR_MBCCFRn[CHA]=0 and FR_MBCCFRn[CHB]=1	see *Figure 552*
Receive FIFO Channel A Message Buffer	see *Figure 551*
Receive FIFO Channel B Message Buffer	see *Figure 552*

The meaning of the bits in the slot status structure is explained in *Table 429*.

	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 550. 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 551. 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 552. Receive Message Buffer Slot Status Structure (ChB)

Table 429. Receive Message Buffer Slot Status Field Descriptions (Continued)

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 0 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 429. Receive Message Buffer Slot Status Field Descriptions (Continued)

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

27.6.5.2.3.2 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 430*.

Table 430. 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 553*
Individual Transmit Message Buffer assigned to channel A FR_MBCCFRn[CHA]=1 and FR_MBCCFRn[CHB]=0	see *Figure 554*
Individual Transmit Message Buffer assigned to channel B FR_MBCCFRn[CHA]=0 and FR_MBCCFRn[CHB]=1	see *Figure 555*

The meaning of the bits in the slot status structure is described in *Table 429*.

	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 553. 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 554. 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 555. Transmit Message Buffer Slot Status Structure (ChB)

Table 431. 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 431. 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

27.6.5.3 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 432*. The structure of the message buffer data field is given in *Figure 556*.

Table 432. 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 432.

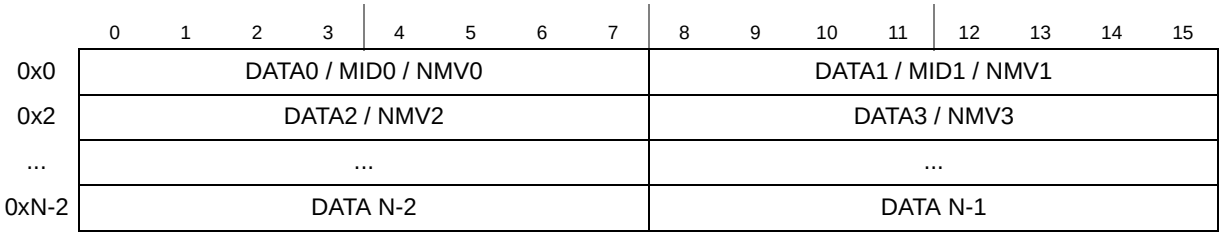


Figure 556. Message Buffer Data Field Structure

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.

27.6.5.3.1 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.

27.6.5.3.2 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 433*.

Table 433. Frame Data Write Access Constraints

Field	single buffered	double buffered	
		commit side	transmit side
DATA, MID, NMV	*POC:config* or MB_DIS or MB_LCK	*POC:config* or MB_DIS or MB_LCK	*POC:config* or MB_DIS

Table 434. 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.*

27.6.6 Individual Message Buffer Functional Description

The CC provides three basic types of individual message buffers:

1. Single Transmit Message Buffers
2. Double Transmit Message Buffers
3. 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 27.6.3.4.1, "Individual Message Buffer Configuration Data*".

27.6.6.1 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.

27.6.6.1.1 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)*.

27.6.6.1.2 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 435*.

Table 435. Individual Message Buffer Types

FR_MBCCSRn		Individual Message Buffer Description
MTD	MBT	
0	0	Receive Message Buffer
0	1	Reserved
1	0	Single Transmit Message Buffer
1	1	Double Transmit Message Buffer

The message buffer specific configuration data are

1. MCM, MBT, MTD bits in *Message Buffer Configuration, Control, Status Registers (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 27.6.6.2, "Single Transmit Message Buffers"*, *Section 27.6.6.3, "Receive Message Buffers"*, and *Section 27.6.6.4, "Double Transmit Message Buffer"*.

27.6.6.2 Single Transmit Message Buffers

The section provides a detailed description of the functionality of single buffered transmit message buffers.

A single transmit message buffer is used by the application to provide message data to the 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 single transmit message buffer by the following settings:

- FR_MBCCSRn[MBT] = 0 (single buffered message buffer)
- FR_MBCCSRn[MTD] = 1 (transmit message buffer)

27.6.6.2.1 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 single transmit message buffers are depicted in *Figure 557*. A description of the regions is given in *Table 436*. If an region is active as indicated in *Table 437*, the access scheme given for that region applies to the message buffer.

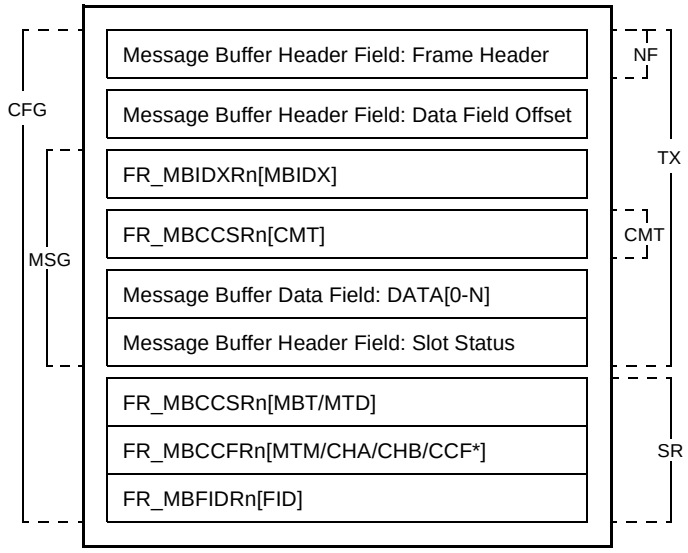


Figure 557. Single Transmit Message Buffer Access Regions

Table 436. Single Transmit Message Buffer Access Regions Description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	—	Message Buffer Configuration
MSG	read/write	—	Message Data and Slot Status Access
NF	—	read-only	Message Header Access for Null Frame Transmission
TX	—	read/write	Message Transmission and Slot Status Update
CM	—	read-only	Message Buffer Validation
SR	—	read-only	Message Buffer Search

The trigger bits 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 558*. A description of the states is given in *Table 437*, 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.

27.6.6.2.2 Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

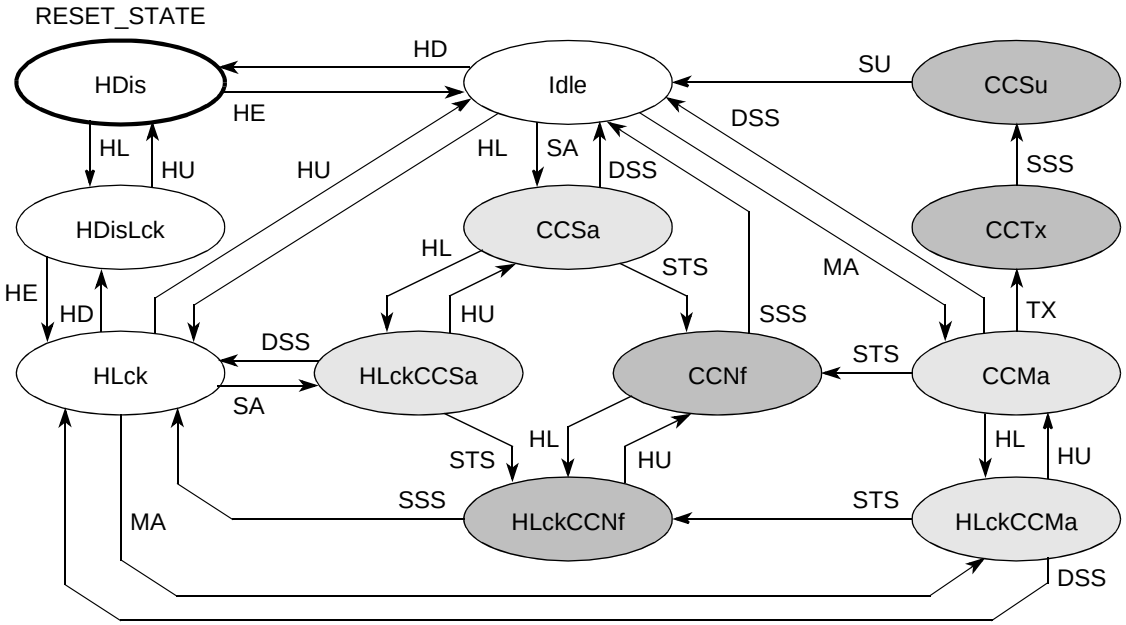


Figure 558. Single Transmit Message Buffer States

Table 437. Single 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.

Table 437. Single Transmit Message Buffer State Description (continued)

State	FR_MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
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.

27.6.6.2.3 Message Buffer Transitions

27.6.6.2.3.1 Application Transitions

The application transitions can be triggered by the application using the commands described in *Table 438*. 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.If the communication controller is started as a non-coldstart node and configured and enabled message buffers in the POC config state for Slot 1, then the message buffer CAN NOT be disabled in the INTEGRATION_LISTEN state by directly writing '1' to the EDT bit. To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. This should enable the message buffer disable during the LISTEN states.

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 438. Single 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.

27.6.6.2.3.2 Module Transitions

The module transitions that can be triggered by the CC are described in *Table 439*. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 439. Single Transmit Message Buffer Module Transitions

Transition	Condition	Description
SA	slot match and static slot	Slot Assigned - Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and 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.

27.6.6.2.3.3 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 440*, 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 440*.

Table 440. Single Transmit Message Buffer Transition Priorities

State	Priorities	Description
module vs. application		
Idle, HLck	SA > HD MA > HD	Slot Assigned > Message Buffer Disable Message Available > Message Buffer Disable
CCMa	TX > HL	Transmission Start > Message Buffer Lock
module internal		
Idle, HLck	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

27.6.6.2.4 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 27.6.3.1, "Individual Message Buffers*".

As indicated by *Table 437*, 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 438*. 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.

27.6.6.2.5 Message Transmission

As a result of the message buffer search described in *Section 27.6.7, "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 559*. 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.

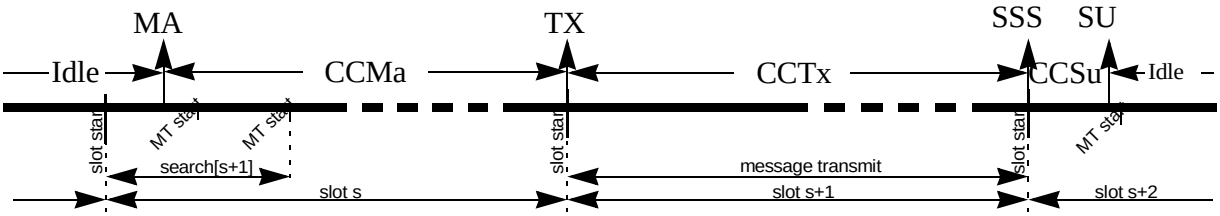


Figure 559. Message Transmission Timing

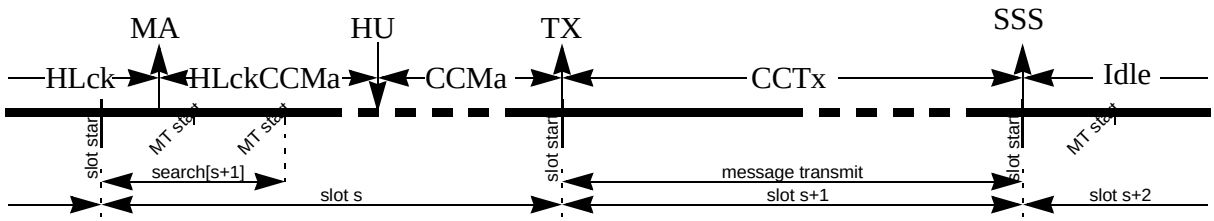


Figure 560. Message Transmission from HLck state with unlock

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 27.6.5.2.1, "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.

27.6.6.2.6 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 27.6.7, "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 561*.

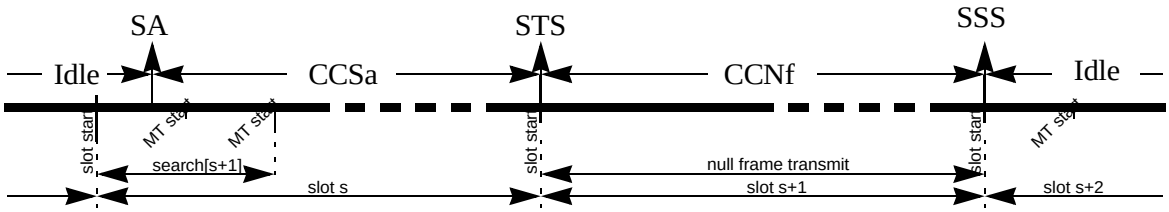


Figure 561. Null Frame Transmission from Idle state

A message buffer timing and state change diagram for null frame transmission from HLck state is given in *Figure 562*.

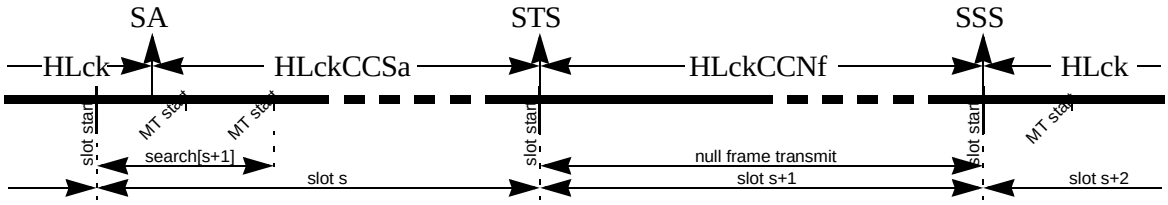


Figure 562. Null Frame Transmission from HLck state

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 563*.

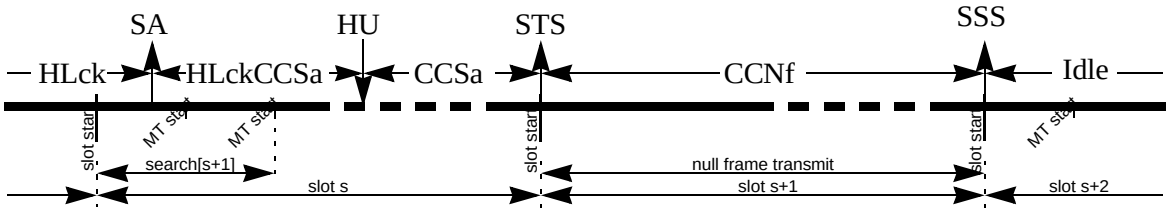


Figure 563. Null Frame Transmission from HLck state with unlock

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 564*.

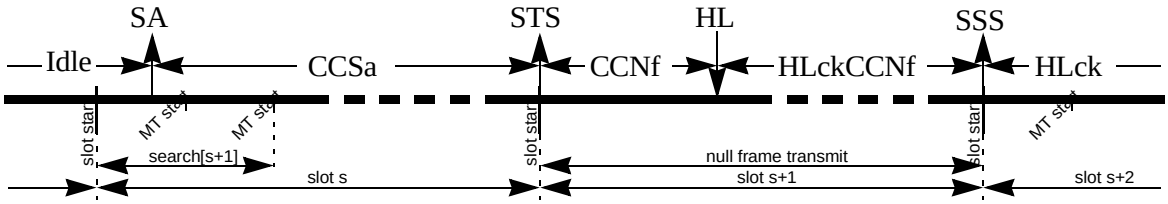


Figure 564. Null Frame Transmission from Idle state with locking

27.6.6.2.7 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.

27.6.6.2.7.1 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.

27.6.6.2.7.2 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.

27.6.6.2.7.3 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.

27.6.6.3 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 27.6.3.2, "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[MBT] = 0 (single buffered message buffer)
- 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 565*. A description of the regions is given in *Table 441*. If an region is active as indicated in *Table 442*, the access scheme given for that region applies to the message buffer.

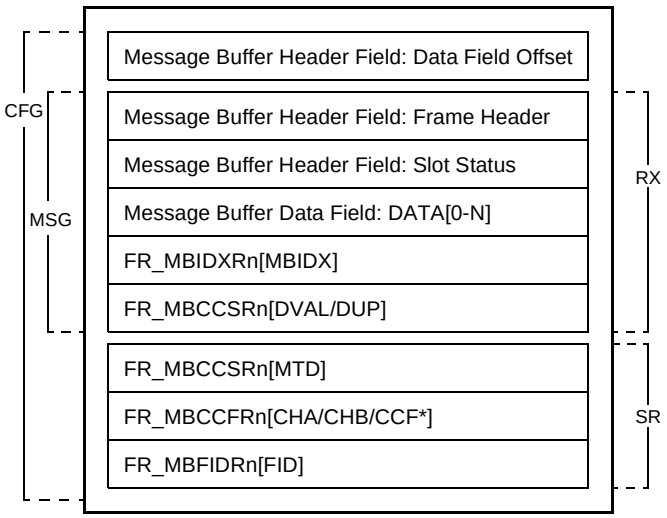


Figure 565. Receive Message Buffer Access Regions

Table 441. 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 566*. A description of the message buffer states is given in *Table 437*, 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.

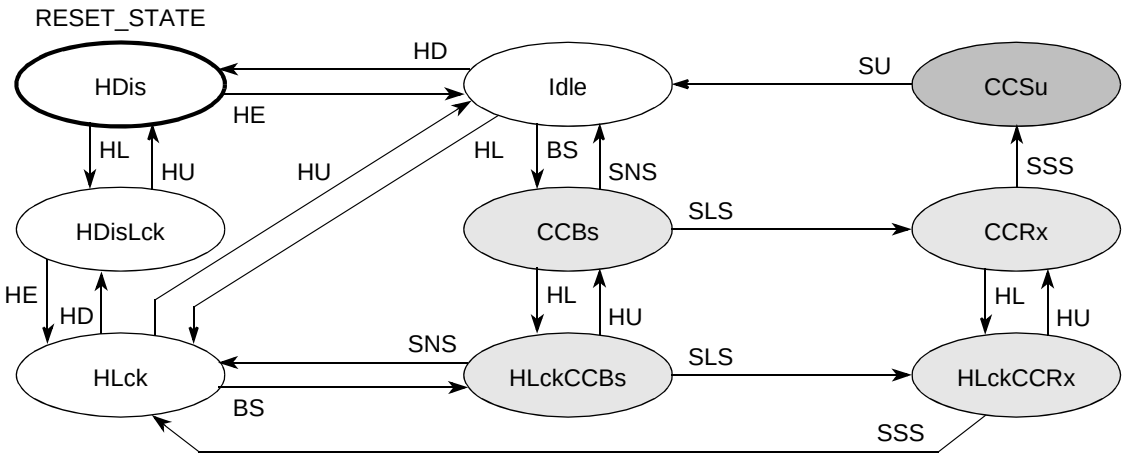


Figure 566. Receive Message Buffer States

Table 442. 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.
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.

27.6.6.3.1 Message Buffer Transitions

27.6.6.3.1.1 Application Transitions

The application transitions that can be triggered by the application using the commands described in *Table 443*. 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.If the communication controller is started as a non-coldstart node and configured and enabled message buffers in the POC config state for Slot 1, then the message buffer CAN NOT be disabled in the INTEGRATION_LISTEN state by directly writing '1' to the EDT bit. To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. This should enable the message buffer disable during the LISTEN states.

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 443. 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.

27.6.6.3.1.2 Module Transitions

The module transitions that can be triggered by the CC are described in *Table 444*. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 444. 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.

27.6.6.3.1.3 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 445*, 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 445. 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

27.6.6.3.2 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 27.6.6.3.5, "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.

27.6.6.3.3 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 446*.

Table 446. Receive Message Buffer Update (Continued)

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
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 567*.

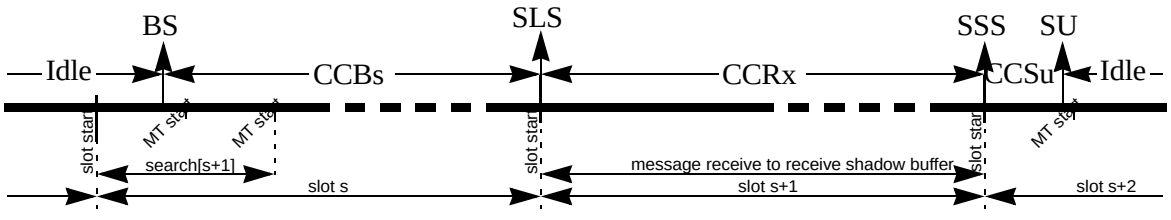


Figure 567. Message Reception Timing

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.

27.6.6.3.4 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 27.6.3.1, "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.

27.6.6.3.5 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 27.6.3.2, "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 446*), 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)*.

27.6.6.4 Double Transmit Message Buffer

The section provides a detailed description of the functionality of the double transmit message buffers.

Double transmit message buffers are used by the application to provide the CC with the message data to be transmitted over the FlexRay Bus. The CC uses this message buffer to provide information to the application about the transmission process, and status information about the slot in which message data was transmitted.

In contrast to the single transmit message buffers, the application can provide new transmission data while the transmission of the previously provided message data is running. This scheme is called double buffering and can be considered as a FIFO of depth 2.

Double transmit message buffers are implemented by combining two individual message buffers that form the two sides of an double transmit message buffer. One side is called the *commit side* and will be accessed by the application to provide the message data. The other side is called the *transmit side* and is used by the CC to transmit the message data to the FlexRay bus. The two sides are located in adjacent individual message buffers. The message buffer that implements the commit side has an even message buffer number 2n. The transmit side message buffer follows the commit side message buffer and has the message buffer number 2n+1. The basic structure and data flow of a double transmit message buffer is given in *Figure 568*.

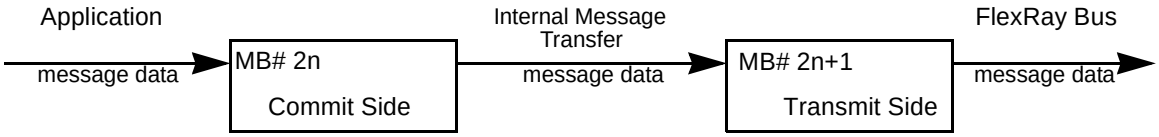


Figure 568. Double Transmit Buffer Structure and Data Flow

Note: Both the commit and the transmit side must be configured with identical values except for the Message Buffer Index Registers (FR_MBIDXRn).

27.6.6.4.1 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 controls the exclusive access to the data, control, and status bits of the message buffer.

The access scheme for double transmit message buffers is depicted in *Figure 569*. The given regions represent fields that can be accessed from both the application and the CC and, thus, require access restrictions. A description of the regions is given in *Table 447*.

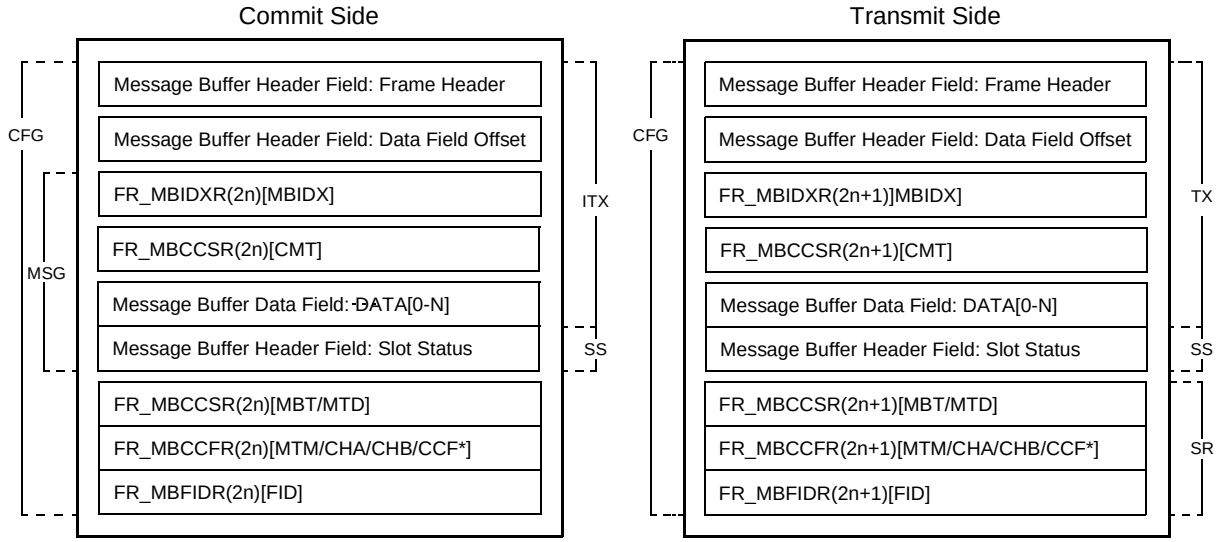


Figure 569. Double Transmit Message Buffer Access Regions Layout

Table 447. Double Transmit Message Buffer Access Regions Description

Region	Access		Description
	Type		
	Application	Module	
Commit Side			
CFG	read/write	—	Message Buffer Configuration
MSG	read/write	—	Message Buffer Data and Control access
ITX	—	read/write	Internal Message Transfer.
SS	—	write-only	Slot Status Update
Transmit Side			
CFG	read/write	—	Message Buffer Configuration
SR	—	read-only	Message Buffer Search
TX	—	read-only	Internal Message Transfer, Message Transmission
SS	—	write-only	Slot Status Update

The trigger bits 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 corresponding part of the double transmit message buffer. The application must adhere to

these restrictions in order to ensure data consistency. The states for the commit side of a double transmit message buffer are given in *Figure 570*. A description of the states is given in *Table 449*. The states for the transmit side of a double transmit message buffer are given in *Figure 571*. A description of the states is given in *Table 449*. The description tables also provide 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.

27.6.6.4.2 Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

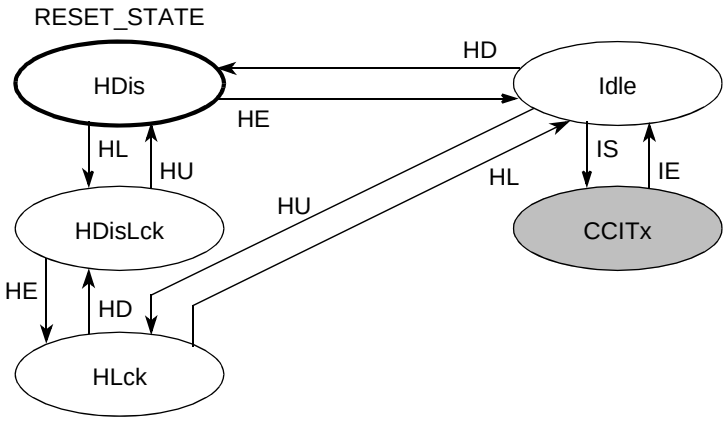


Figure 570. Double Transmit Message Buffer State Diagram (Commit Side)

A description of the states of the commit side of a double transmit message buffer is given in *Table 448*.

Table 448. Double Transmit Message Buffer State Description (Commit Side)

State	FR_MBCCSR(2n)		Access Region		Description
	EDS	LCKS	Appl.	Module	
common states					
HDis	0	0	CFG	–	**Disabled** - Message Buffer under configuration. Commit Side can *not* be used for internal message transfer.
CCITx	1	0	–	ITX	**Internal Message Transfer** - Message Buffer Data transferred from commit side to transmit side.
commit side specific states					
Idle	1	0	–	ITX, SS	**Idle** - Message Buffer Commit Side is idle. Commit Side can be used for internal message transfer.
HDisLck	0	1	CFG	SS	**Disabled and Locked** - Message Buffer under configuration. Commit Side can *not* be used for internal message transfer.
HLck	1	1	MSG	SS	Locked - Applications access to data, control, and status. Commit Side can *not* be used for internal message transfer.

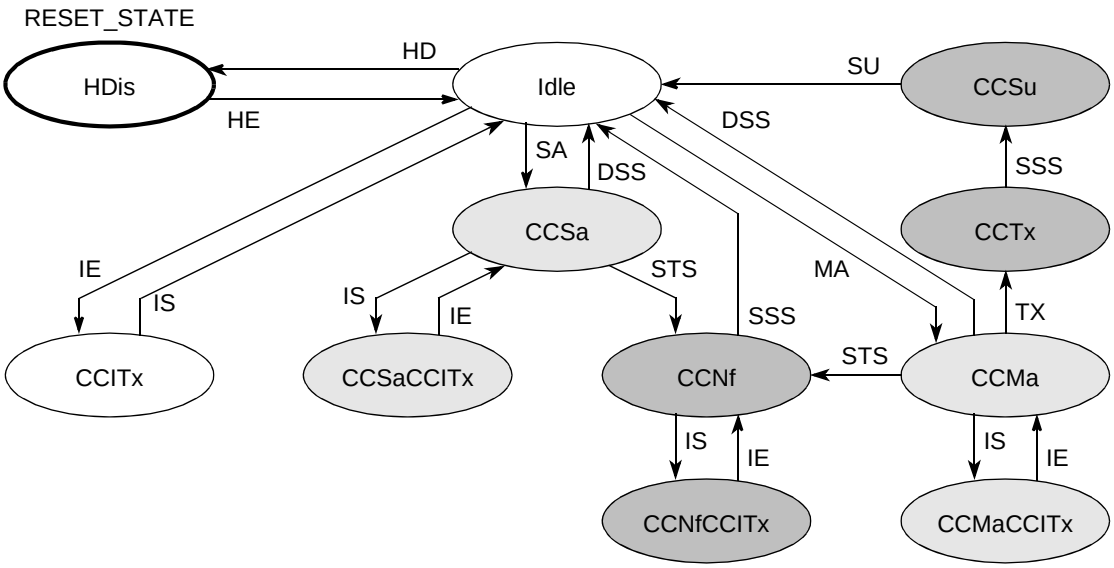


Figure 571. Double Transmit Message Buffer State Diagram (Transmit Side)

A description of the states of the transmit side of a double transmit message buffer is given in *Table 449*.

Table 449. Double Transmit Message Buffer State Description (Transmit Side)

State	FR_MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
common states					
HDis	0	0	CFG	–	**Disabled** - Message Buffer under configuration. Excluded from message buffer search.
CCITx	1	0	–	TX	**Internal Message Transfer** - Message Buffer Data transferred from commit side to transmit side.
transmit side specific states					
Idle	1	0	–	SR	**Idle** - Message Buffer Transmit Side is idle. Transmit Side is included in message buffer search.
CCSa	1	0	–	–	Slot Assigned - Message buffer assigned to next static slot. Ready for Null Frame transmission.
CCSaCCITx	1	0	–	TX	Slot Assigned and Internal Message Transfer - Message buffer assigned to next static slot and Message Buffer Data transferred from commit side to transmit side.
CCNf	1	0	–	TX	Null Frame Transmission Header is used for null frame transmission.
CCNfCCITx	1	0	–	TX	Null Frame Transmission and Internal Message Transfer - Header is used for null frame transmission and Message Buffer Data transferred from commit side to transmit side.
CCMa	1	0	–	–	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.

Table 449. Double Transmit Message Buffer State Description (Transmit Side) (continued)

State	FR_MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
CCMaCCITx	1	0	–	–	Message Available and Internal Message Transfer - Message buffer is assigned to next slot and cycle counter filter matches and Message Buffer Data transferred from commit side to transmit side.
CCTx	1	0	–	TX	Message Transmission - Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	–	SS	Status Update - Message buffer status update. Update of status flags, the slot status field, and the header index. Note: The slot status field of the commit side is updated too, even if the application has locked the commit side.

27.6.6.4.3 Message Buffer Transitions

27.6.6.4.3.1 Application Transitions

The application transitions that can be triggered by the application using the commands described in *Table 450*. 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 can be issued on the transmit side only. Any enable or disable command issued on the commit side will be ignored without notification. The transitions that will be triggered depends on the value of the EDS bit. The enable and disable commands will affect both the commit side and the transmit side at the same time. If the application triggers the disable transition HD while the transmit side is in one of the states CCSa, CCSaCCITx, CCNf, CCNfCCITx, CCMa, CCMaCCITx, CCTx, or CCSu, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

Message Buffer Lock and Unlock

The lock and unlock commands can be issued on the commit side only. Any lock or unlock command issued on the transmit side will be ignored and the double transmit buffer lock error flag DBL_EF in the *CHI Error Flag Register (FR_CHIERFR)* will be set. The transitions that will be triggered depends on the current value of the LCKS bit. The lock and unlock commands will only affect the commit side. If the application triggers the lock transition HL while the commit side is in the state CCITx, the message buffer state will not be changed and the message buffer lock error flag LCK_EF in the *CHI Error Flag Register (FR_CHIERFR)* will be set.

Table 450. Double Transmit Message Buffer Host Transitions

Transition	Host Command	Condition	Description
HE	FR_MBCCSR(2n+1)[EDT]:=1	FR_MBCCSR(2n+1)[EDS] = 0	Application triggers message buffer enable.
HD		FR_MBCCSR(2n+1)[EDS] = 1	Application triggers message buffer disable.
HL	FR_MBCCSR(2n)[LCKT]:=1	FR_MBCCSR(2n)[LCKS] = 0	Application triggers message buffer lock.
HU		FR_MBCCSR(2n)[LCKS] = 1	Application triggers message buffer unlock.

27.6.6.4.3.2 Module Transitions

The module transitions that can be triggered by the CC are described in *Table 451*. The transitions C1 and C2 apply to both sides of the message buffer and are applied at the same time. All other CC transitions apply to the transmit side only.

Table 451. Double Transmit Message Buffer Module Transitions

Transition	Condition	Description
common transitions		
IS	see *Section 27.6.6.4.5, "Internal Message Transfer*	Internal Message Transfer Start - Start transfer of message data from commit side to transmit side.
IE		Internal Message Transfer End - Stop transfer of message data from commit side to transmit side. Note: The internal message transfer is stopped before the slot or segment start.
transmit side specific transitions		
SA	slot match and static slot	Slot Assigned - Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and FR_MBCCSR(2n+1)[CMT]=1	Transmission Slot Start - Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated - Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start - Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start. - Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of static slot or dynamic slot or symbol window or NIT.

27.6.6.4.3.3 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 452*, the module transitions have a higher priority than the application transitions. The priorities among the CC transitions and the related states are given in the second part of *Table 452*. These priorities apply only to the transmit side. The internal message transmit start transition IS has tho lowest priority.

Table 452. Double Transmit Message Buffer Transition Priorities

State	Priority	Description
module vs. application		
Idle	IS > HD IS > HL	Internal Message Transfer Start > Message Buffer Disable Internal Message Transfer Start > Message Buffer Lock
module internal		
Idle	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

27.6.6.4.4 Message Preparation

The application provides the message data through the commit side. The transmission itself is executed from the transmit side. The transfer of the message data from the commit side to the transmit side is done by the *Internal Message Transfer*, which is described in *Section 27.6.6.4.5, "Internal Message Transfer*

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field of the commit side and sets the commit bit CMT in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. The physical access to the message buffer data field is described in *Section 27.6.3.1, "Individual Message Buffers*".

As indicated by *Table 448*, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, or HLck. The application can change the state of a message buffer if it issues the appropriate commands shown in *Table 450*. The state change is indicated through the FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] status bits.

27.6.6.4.5 Internal Message Transfer

The internal message transfer transfers the message data from the commit side to the transmit side. The internal message transfer is implemented as the swapping of the content of the *Message Buffer Index Registers (FR_MBIDXRn)* of the commit side and the transmit side. After the swapping, the commit side CMT bit is cleared, the commit side interrupt flag MBIF is set, the transmit side CMT bit is set, and the transmit side DVAL bit is cleared.

The conditions and the point in time when the internal message transfer is started are controlled by the message buffer commit mode bit MCM in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. The MCM bit configures the message buffer for either the streaming commit mode or the immediate commit mode. A detailed description is given in *Streaming Commit Mode* and *Immediate Commit Mode*. The

Internal Message Transfer is triggered with the transition IS. Both sides of the message buffer enter one of the CCITx states. The internal message transfer is finished with the transition IE.

27.6.6.4.5.1 Streaming Commit Mode

The intention of the streaming commit mode is to ensure that each committed message is transmitted *at least once*. The CC will not start the Internal Message Transfer for a message buffer as long as the message data on the transmit side is not transmitted at least once.

The streaming commit mode is configured by clearing the message buffer commit mode bit MCM in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*.

In this mode, the internal message transfer from the commit side to the transmit side is started for a double transmit message buffer when all of the following conditions are fulfilled

1. the commit side is in the Idle state
2. the commit site message data are valid, i.e. FR_MBCCSR(2n)[CMT] = 1
3. the transmit side is in one of the states Idle, CCSa, or CCMa
4. the transmit side contains either no valid message data, i.e. FR_MBCCSR(2n+1)[CMT] = 0 or
   the message data were transmitted at least once, i.e. FR_MBCCSR(2n+1)[DVAL] = 1

An example of a streaming commit mode state change diagram is given in *Figure 572*. In this example, both the commit and the transmit side do not contain valid message data and the application provides two messages. The message buffer does not match the next slot.

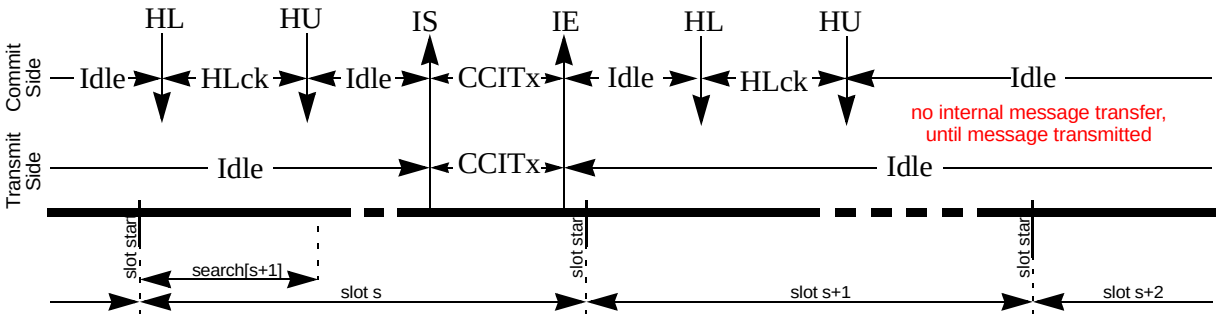


Figure 572. Internal Message Transfer in Streaming Commit Mode

27.6.6.4.5.2 Immediate Commit Mode

The intention of the immediate commit mode is to transmit the *latest* data provided by the application. This implies that it is not guaranteed that each provided message will be transmitted at least once.

The immediate commit mode is configured by setting the message buffer commit mode bit MCM in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*.

In this mode, the internal message transfer from the commit side to the transmit side is started for one double transmit message buffer when all of the following conditions are fulfilled

1. the commit side is in the Idle state
2. the commit site message data are valid, i.e. FR_MBCCSR(2n)[CMT] = 1
3. the transmit side is in one of the states Idle, CCSa, or CCMa

It is not checked whether the transmit side contains no valid message data or valid message data were transmitted at least once. If message data are valid and not transmitted, they may be overwritten.

An example of a streaming commit mode state change diagram is given in *Figure 573*. In this example, both the commit and the transmit side do not contain valid message data, and the application provides two messages and the first message is gets overwritten. The message buffer does not match the next slot.

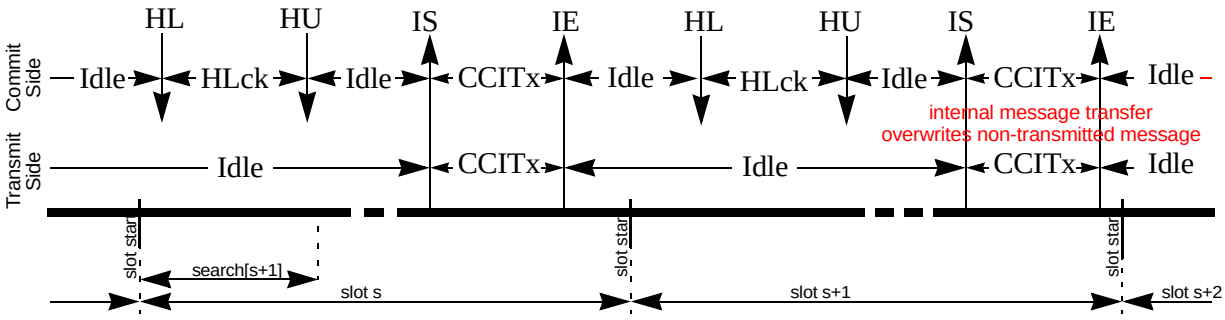


Figure 573. Internal Message Transfer in Immediate Commit Mode

27.6.6.4.6 Message Transmission

For double transmit message buffers, the message buffer search checks only the transmit side part. The internal scheduling ensures, that the internal message transfer is stopped on the message buffer search start. Thus, the transmit side of message buffer, that is not in its transmission or status update slot, is always in the Idle state.

The message transmit behavior and transmission state changes of the transmit side of a double transmit message buffer are the same as for single buffered transmit buffers, except that the transmit side of double buffers can not be locked by the application, i.e. the HU and HL transition do not exist. Therefore, refer to *Section 27.6.6.2.5, "Message Transmission".*

27.6.6.4.7 Message Buffer Status Update

The message buffer status update behavior of the transmit side of a double transmit message buffer is the same as for single transmit message buffers which is described in *Section 27.6.6.2.7, "Message Buffer Status Update".*

Additionally, the slot status field of the commit side is update after the update of the slot status field of the transmit side, even if the commit side is locked by the application. This is implemented to provide the slot status of the most recent transmission slot.

27.6.7 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
3. are the transmit side buffer in case of a double transmit message buffer.

On top of that, for the static segment only those message buffers are considered, that match the condition of at least one row of *Table 453*. For the dynamic segment only those message buffers are considered, that match the condition of at least one row of *Table 454*. 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 453* for the static segment and according to *Table 454* for the dynamic segment are selected.

Table 453. 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 27.6.7.1, "Message Buffer Cycle Counter Filtering"*

Table 454. 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

Table 454. Message Buffer Search Priority (dynamic segment)

Priority	MTD	LCKS	CMT	CCFM(1)	Description	Transition
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 27.6.7.1, "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 27.6.7.2, "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 27.6.3.1, "Individual Message Buffers*".

27.6.7.1 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 26 MBCCFRn[CCFE] = 0

Equation 27

CYCCNT & MBCCFRn[CCFMSK] = MBCCFRn[CCFVAL] & MBCCFRn[CCFMSK]

27.6.7.2 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 574*.

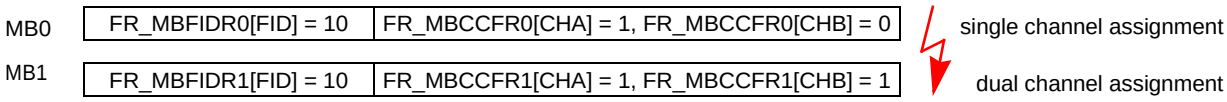


Figure 574. Inconsistent Channel Assignment

27.6.7.3 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 454* 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.

27.6.7.4 Message Buffer Search Error

If the message buffer search is running while the next message buffer search start event appears, the message buffer search is stopped and the Message Buffer Search Error Flag MSB_EF is set in the *CHI Error Flag Register (FR_CHIERFR)*. This appears only if the CHI frequency is too low to search through all message buffers within the NIT or a minislot. The message buffer result is not defined in this case. For more details see *Section 27.7.5, "Number of Usable Message Buffers".*

27.6.8 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.

27.6.8.1 Reconfiguration Schemes

Depending on the target and destination basic state of the message buffer that is to be reconfigured, there are three reconfiguration schemes.

27.6.8.1.1 Basic Type Not Changed (RC1)

A reconfiguration will not change the basic type of the individual message buffer, if both the message buffer transfer direction bit FR_MBCCSRn[MTD] and the message buffer type bit FR_MBCCSRn[MBT] are not changed. This type of reconfiguration is denoted by RC1 in *Figure 575*. Single transmit and receive message buffers can be RC1-reconfigured when in the HDis or HDisLck state. Double transmit message buffers can be RC1-reconfigured if both the transmit side and the commit side are in the HDis state.

27.6.8.1.2 Buffer Type Not Changed (RC2)

A reconfiguration will not change the buffer type of the individual message buffer if the message buffer buffer type bit FR_MBCCSRn[MBT] is not changed. This type of reconfiguration is denoted by RC2 in *Figure 575*. It applies only to single transmit and receive message buffers. Single transmit and receive message buffers can be RC2-reconfigured when in the HDis or HDisLck state.

27.6.8.1.3 Buffer Type Changed (RC3)

A reconfiguration will change the buffer type of the individual message buffer if the message buffer type bit FR_MBCCSRn[MBT] is changed. This type of reconfiguration is denoted by RC3 in *Figure 575*. The RC3 reconfiguration splits one double buffer into two single buffers or combines two single buffer into one double buffer. In the later case, the two single message buffers must have consecutive message buffer numbers and the smaller one must be even. Message Buffers can be RC3 reconfigured if they are in the HDis state.

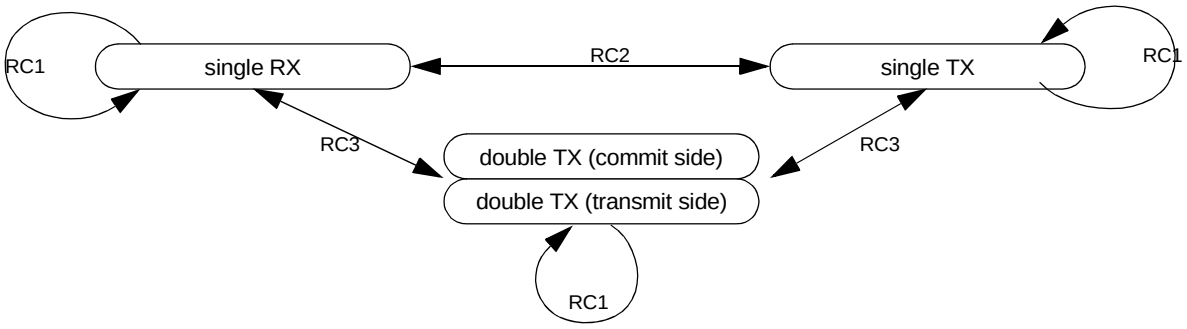


Figure 575. Message Buffer Reconfiguration Scheme

27.6.9 Receive FIFOs

This section provides the functional description of the two receive FIFOs.

27.6.9.1 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 27.6.3.3, "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 number of FIFO entries and the length of each FIFO entry as given by *Receive FIFO Depth and Size Register (RFDSR)*

27.6.9.2 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)*.

27.6.9.2.1 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)*.

27.6.9.2.2 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 27.6.3.7, "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 memory for the FlexRay memory area. This includes the allocation of the message buffer header area and the allocation of the message buffer data fields. For more details see *Section 27.6.4, "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 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

27.6.9.3 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.

27.6.9.4 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 576* 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.

27.6.9.5 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.

27.6.9.6 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 27.6.9.4, "FIFO Reception*" are fulfilled, then the fifo overflow error flag FR_CHIERFR[FOVA_EF] (FR_CHIERFR[FOVB_EF]) is asserted.

27.6.9.7 FIFO Message Access

The FIFOA (FIFOB) contains valid messages if the FIFO fill level FLA (FLB) is greater than 0. The *Receive FIFO A Read Index Register (FR_RFARIR)* (*Receive FIFO B Read Index Register (FR_RFBRIR)*) pointing to a message buffer with valid content and the oldest frames stored in the FIFO.

If the FIFO fill level FLA (FLB) is 0, than the FIFOA (FIFOB) contains no valid messages and the *Receive FIFO A Read Index Register (FR_RFARIR)* (*Receive FIFO B Read Index Register (FR_RFBRIR)*) pointing to a message buffer with invalid content. In this case the application must not read data from the 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 application can access the message data as described in *Section 27.6.3.3, "Receive FIFO".* When the application has read the message buffer data and status information, it can update the FIFO as described in *Section 27.6.9.8, "FIFO Update*".

27.6.9.8 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.

27.6.9.8.1 FIFO Interrupt Flag Update

Th 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.

27.6.9.9 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 576*.

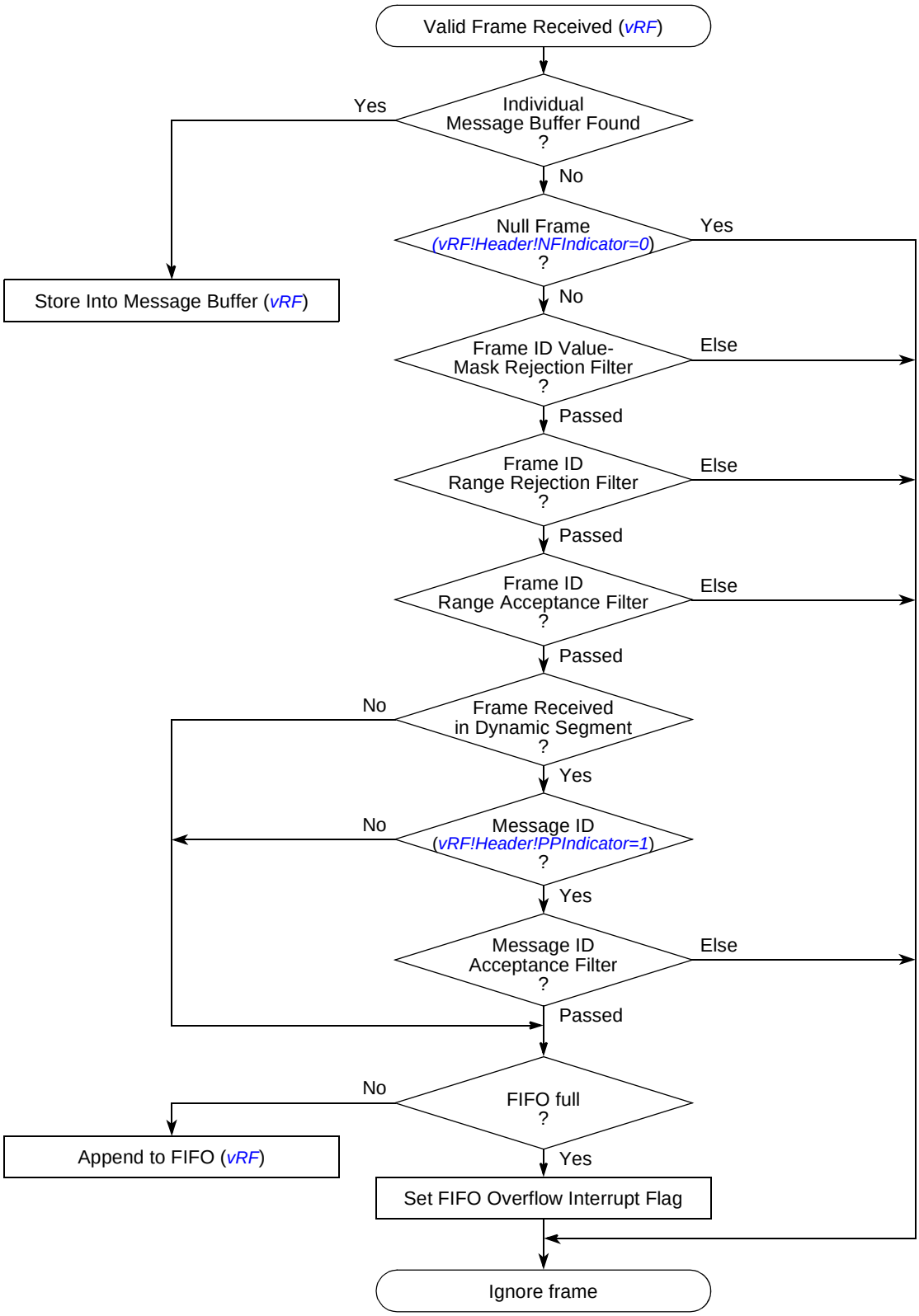


Figure 576. Received Frame FIFO Filter Path

A received frame passes the FIFO filtering if it has passed all three type of filter.

27.6.9.9.1 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 28* is fulfilled.

Equation 28

$$\begin{aligned} &FID \,\&\, FR_RFFIDRFMR[FIDRFMSK] \neq \\ &\neq FR_RFFIDRFVR[FIDRFVAL] \,\&\, FR_RFFIDRFMR[FIDRFMSK] \end{aligned}$$

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 28* 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.

27.6.9.9.2 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 29* is fulfilled.

Equation 29

$$(FID < FR_RFRFCFR_{SEL}[SID_{IBD=0}]) \text{ or } (FR_RFRFCFR_{SEL}[SID_{IBD=1}] < FID)$$

Consequently, all frames with a frame ID that fulfills *Equation 30* for at least one of the enabled rejection filters will be rejected and thus not pass.

Equation 30

$$FR_RFRFCFR_{SEL}[SID_{IBD=0}] \leq FID \leq FR_RFRFCFR_{SEL}[SID_{IBD=1}]$$

27.6.9.9.3 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 31* is fulfilled.

Equation 31

$$FR_RFRFCFR_{SEL}[SID_{IBD=0}] \leq FID \leq FR_RFRFCFR_{SEL}[SID_{IBD=1}]$$

27.6.9.9.4 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 32* is fulfilled.

Equation 32

$$MID \,\&\, FR_RFMIDAFMR[MIDAFMSK] = FR_RFMIDAFMR[MIDAFVAL] \,\&\, FR_RFMIDAFMR[MIDAFMSK]$$

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 32* is always fulfilled and all frames will pass.

27.6.10 Channel Device Modes

This section describes the two FlexRay channel device modes that are supported by the CC.

27.6.10.1 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 CA_RX, CA_TX, and CA_TR_EN is connected to the physical bus channel A and the FlexRay port consisting of CB_RX, CB_TX, and CB_TR_EN is connected to the physical bus channel B. The dual channel system is shown in *Figure 577*.

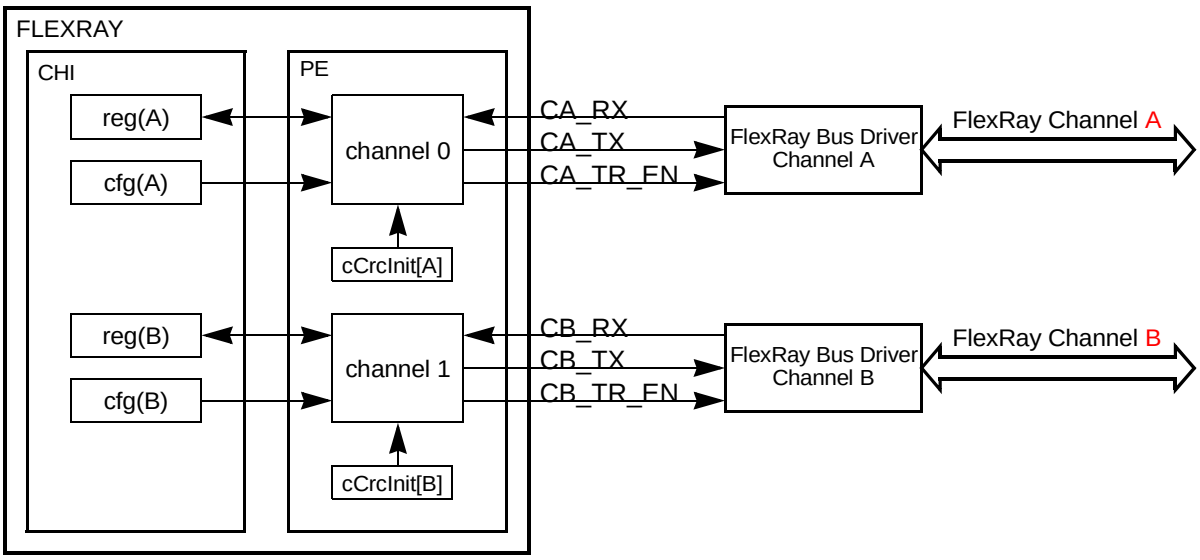


Figure 577. Dual Channel Device Mode

27.6.10.2 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 CA_RX, CA_TX, and CA_TR_EN and can be connected to either the physical bus channel A (shown in *Figure 578*) or the physical bus channel B (shown in *Figure 579*).

If the device is configured as a single channel device by setting FR_MCR.SCD 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.

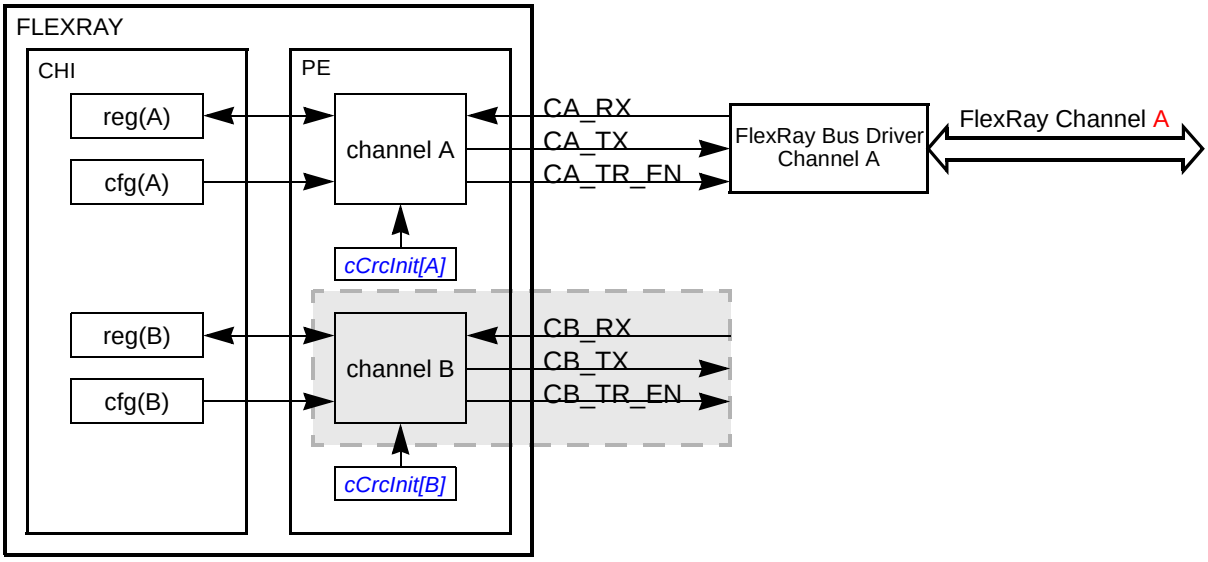


Figure 578. Single Channel Device Mode (Channel A)

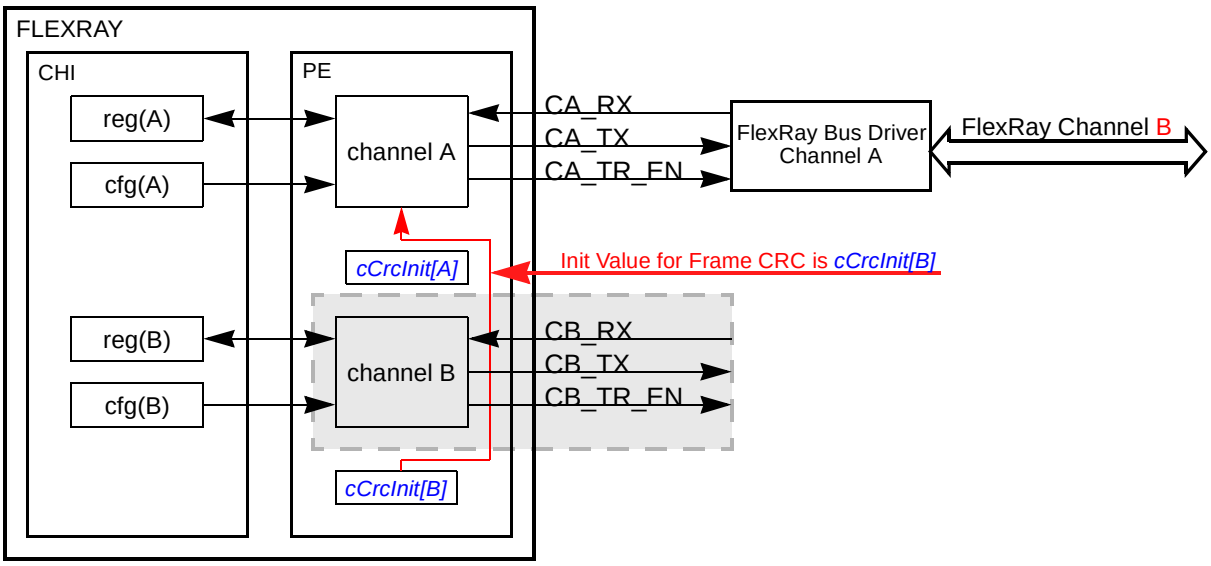


Figure 579. Single Channel Device Mode (Channel B)

27.6.11 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 580* and *Figure 581*.

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 580* for timing details.

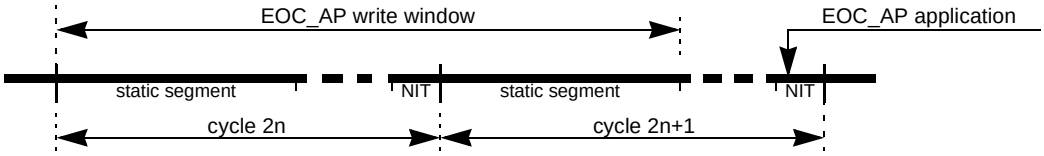


Figure 580. External Offset Correction Write and Application Timing

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 581* for details.

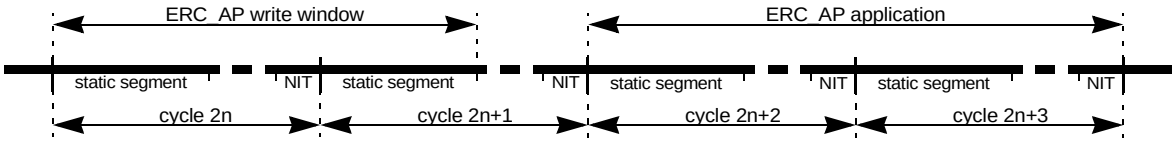


Figure 581. External Rate Correction Write and Application Timing

27.6.12 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.

27.6.12.1 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.

27.6.12.2 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.

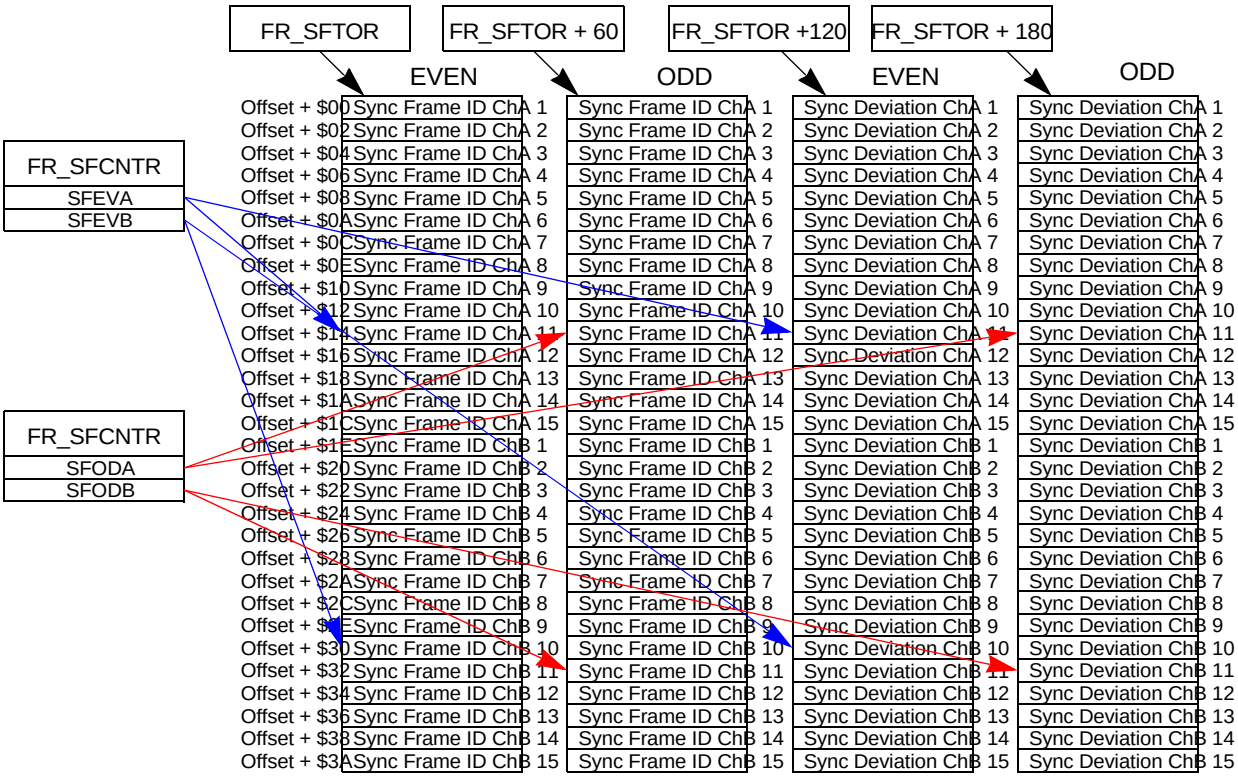


Figure 582. Sync Table Memory Layout

27.6.12.3 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 582*. 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)*.

27.6.12.4 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 455*.

Table 455. 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 582*.

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)*.

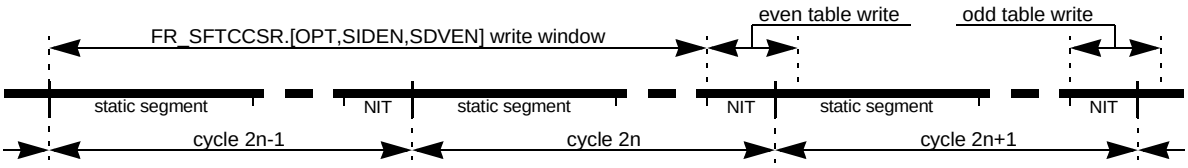


Figure 583. Sync Frame Table Trigger and Generation Timing

27.6.12.5 Sync Frame Table Access

The sync frame tables will be transferred into the FlexRay memory area during the table write windows shown in *Figure 583*. 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.

27.6.12.5.1 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.

27.6.13 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 34*, *Equation 35*, and *Equation 36* are fulfilled.

Equation 33 FR_PSR0[PROTSTATE] = *POC:normal active*

Equation 34 FR_MTSACRF[MTE] = 1

Equation 35

$$CYCCNT \& FR_MTSACFR[CYCCNTMSK] = FR_MTSACFR[CYCCNTVAL] \& FR_MTSACFR[CYCCNTMSK]$$

The MTS is transmitted over channel B in the communication cycle with number CYCCNT, if *Equation 33*, *Equation 36*, and *Equation 37* are fulfilled.

Equation 36 FR_MTSBCRF[MTE] = 1

Equation 37

$$CYCCNT \& FR_MTSBCFR[CYCCNTMSK] = FR_MTSBCFR[CYCCNTVAL] \& FR_MTSBCFR[CYCCNTMSK]$$

27.6.14 Key Slot Transmission

27.6.14.1 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.

27.6.14.2 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*.

27.6.14.3 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 456* 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 27.6.6.2.5, "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 27.6.6.2.6, "Null Frame Transmission"*).

Table 456. 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.

27.6.15 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.

27.6.15.1 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 38* or *Equation 39*evaluates to true.

Equation 38 FR_MCR[SFFE] = 0

Equation 39

FID & FR_SFIDAFMR[FMSK] = FR_SFIDAFVR[FVAL] & FR_SFIDAFMR[FMSK]

Note: Sync frames are transmitted in the static segment only. Thus FID <= 1023.

27.6.15.2 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 40* or *Equation 41* evaluates to true.

Equation 40 FR_MCR[SFFE] = 0

Equation 41 FID ≠ FR_SFIDRFR[SYNFRID]

Note: Sync frames are transmitted in the static segment only. Thus FID <= 1023.

27.6.16 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 343*.

27.6.16.1 Strobe Signal Assignment

Each of the strobe signals listed in *Table 343* 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.

27.6.16.2 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 343* with a negative clock offset. An example waveform is given in *Figure 584*.

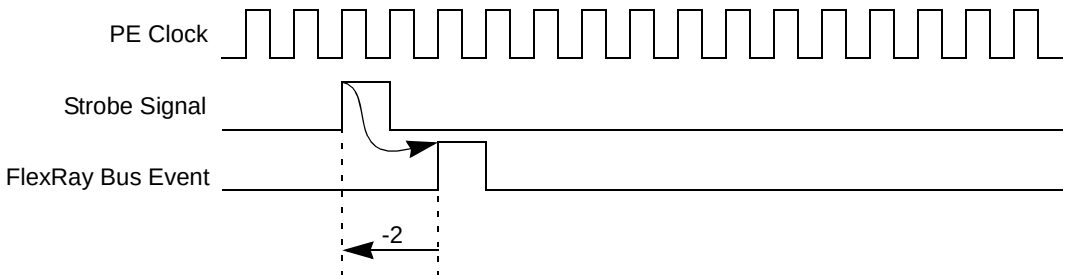
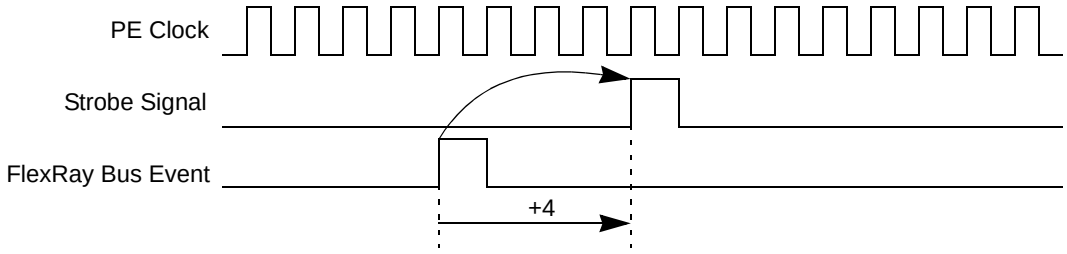


Figure 584. Strobe Signal Timing (type = pulse, clk_offset = -2)

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 343* with a positive clock offset. An example waveform is given in *Figure 585*.

Figure 585. Strobe Signal Timing (type = pulse, clk_offset = +4)



27.6.17 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.

27.6.17.1 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 42* and *Equation 43* are fulfilled

Equation 42

$$\text{CYCTR[CTCCNT]} \,\&\, \text{FR_TI1CYSR[T1_CYC_MSK]} = \\ = \text{FR_TI1CYSR[T1_CYC_VAL]} \,\&\, \text{FR_TI1CYSR[T1_CYC_MSK]}$$

Equation 43 $$\text{FR_MTCTR[MTCT]} = \text{FR_TI1MTOR[T1_MTOFFSET]}$$

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.

27.6.17.2 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.

27.6.17.2.1 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.

27.6.17.2.2 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.

27.6.18 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 457*. The PE provides the slot status vector within the first macrotick after the end of the related slot/window/NIT, as shown in *Figure 586*.

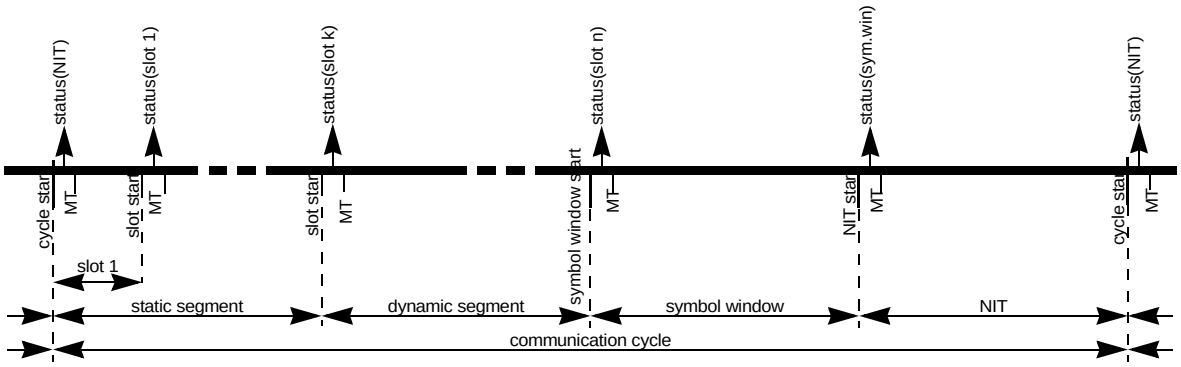


Figure 586. Slot Status Vector Update

Note: The slot status for the NIT of cycle n is provided after the start of cycle n+1.

Table 457. Slot Status Content

	Status Content

static / dynamic Slot	slot related status
	vSS!ValidFrame - valid frame received
	vSS!SyntaxError - syntax error occurred while receiving
	vSS!ContentError - content error occurred while receiving
	vSS!BViolation - boundary violation while receiving
	for slots in which the module transmits:
	vSS!TxConflict - reception ongoing while transmission starts
	for slots in which the module does not transmit:
	vSS!TxConflict - reception ongoing while transmission starts
	first valid - channel that has received the first valid frame
	received frame related status
	extracted from
	a) header of valid frame, if vSS!ValidFrame = 1
	b) last received header, if vSS!ValidFrame = 0
	c) set to 0, if nothing was received
	vRF!Header!NFIndicator - Null Frame Indicator (0 for null frame)
	vRF!Header!SuFIndicator - Startup Frame Indicator
	vRF!Header!SyFIndicator - Sync Frame Indicator

Symbol Window	window related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - reception ongoing while transmission starts received symbol related status *vSS!ValidMTS* - valid Media Test Access Symbol received received frame related status see static/dynamic slot
NIT	NIT related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - always 0 received frame related status see static/dynamic slot

27.6.18.1 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.

27.6.18.2 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.

27.6.18.3 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 457*, except of the *first valid* indicator for non-transmission slots.

27.6.18.4 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 587*.

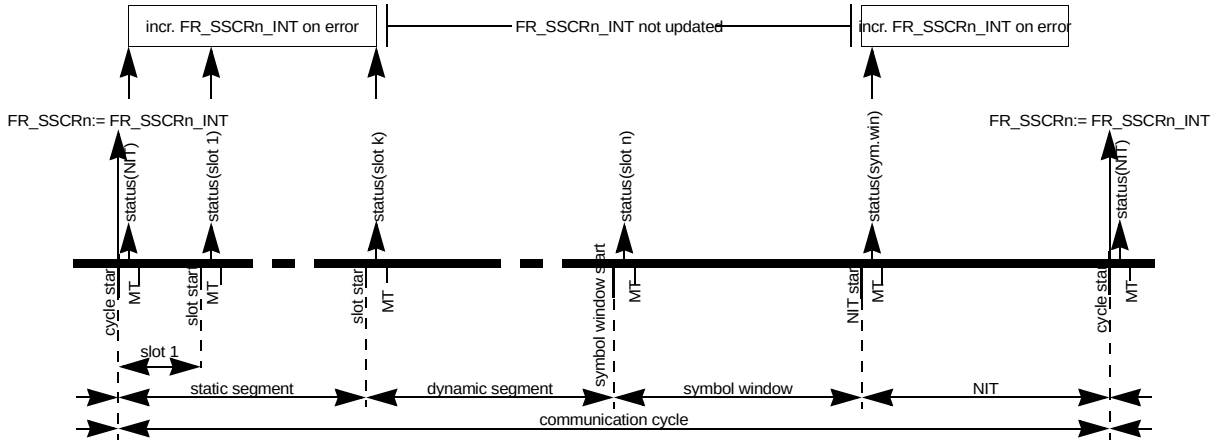


Figure 587. Slot Status Counting and FR_SSCRn Update

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.

27.6.18.5 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 457* 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 27.6.6, "Individual Message Buffer Functional Description*".

27.6.19 System Bus Access

This section provides a description of the system bus accesses performed by the CC.

All FlexRay memory area data located in the system memory are accessed via the system bus. There are two types of failures that can occur during the system bus access, the system bus illegal address access and the system bus access timeout.

The behavior of the CC after the occurrence of a system bus failure is defined by the SBFF bit in the *Module Configuration Register (FR_MCR)*.

27.6.19.1 System Bus Illegal Address Access

If the system bus detects an CC access to an illegal address, the CC receives a notification from the system bus about this event and sets the ILSA_EF flag in the *CHI Error Flag Register (FR_CHIERFR)*.

27.6.19.2 System Bus Access Timeout

The CC starts a timer when it has send an access request to the system bus. This timer expires after 2 * FR_SYMATOR[TIMEOUT] + 2 system bus clock cycles. If the access is not finished within this amount of time, the SBCF_EF flag in the *CHI Error Flag Register (FR_CHIERFR)* is set.

Note: The value of the TIMEOUT field should be set to greater than 1. For the value 1 and 0, a system bus access timeout error will occur in any case.

27.6.19.3 Continue after System Bus 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.

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.

27.6.19.4 Freeze after System Bus 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.

27.6.20 Interrupt Support

The CC provides 108 individual interrupt sources and five combined interrupt sources.

27.6.20.1 Individual Interrupt Sources

27.6.20.1.1 Message Buffer Interrupts

The CC provides 64 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.

27.6.20.1.2 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.

27.6.20.1.3 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.

27.6.20.1.4 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.

27.6.20.1.5 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.

27.6.20.2 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.

27.6.20.2.1 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.

27.6.20.2.2 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.

27.6.20.2.3 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.

27.6.20.2.4 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.

27.6.20.2.5 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.

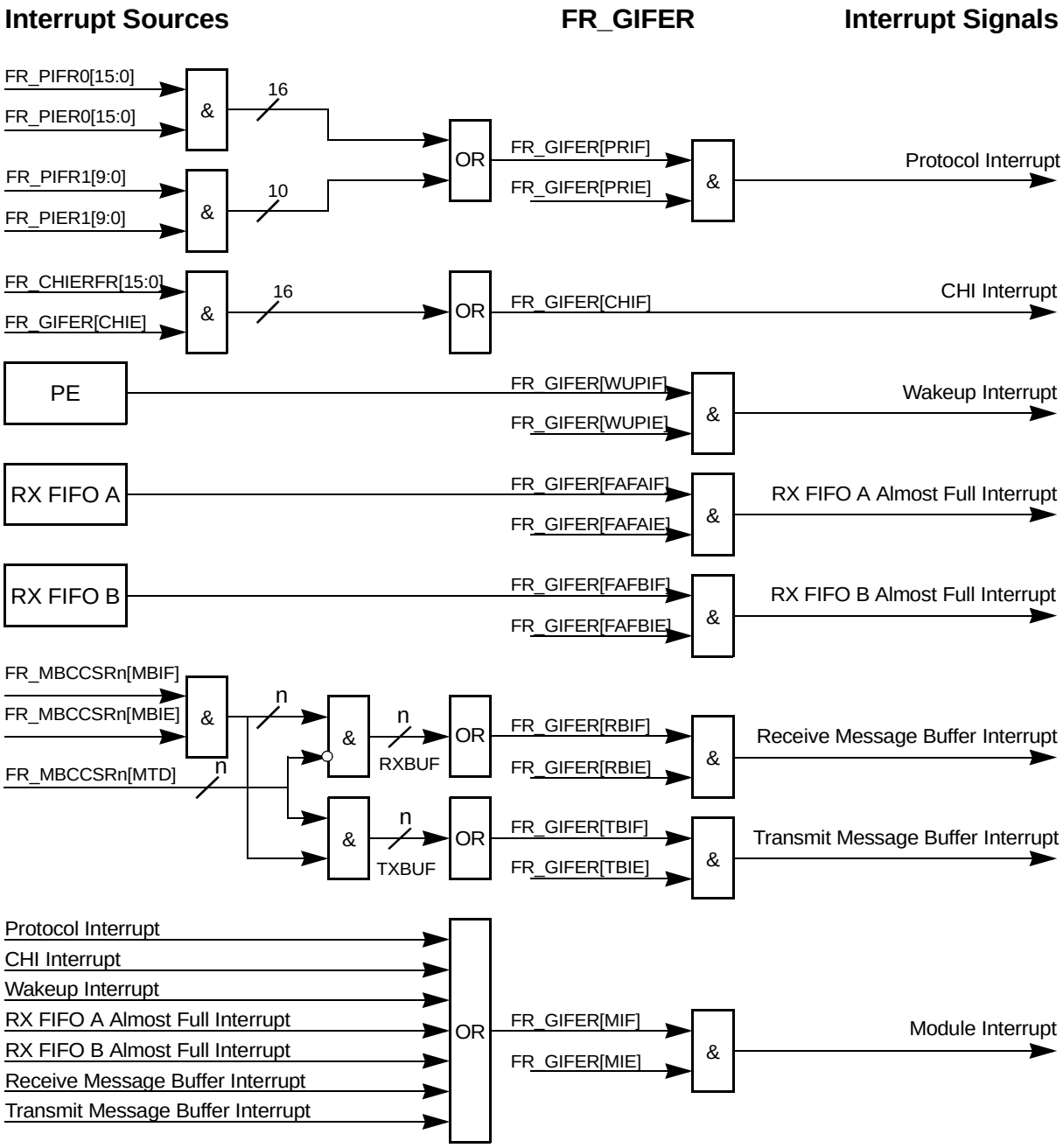


Figure 588. Scheme of FR_GIFER interrupt signal generation

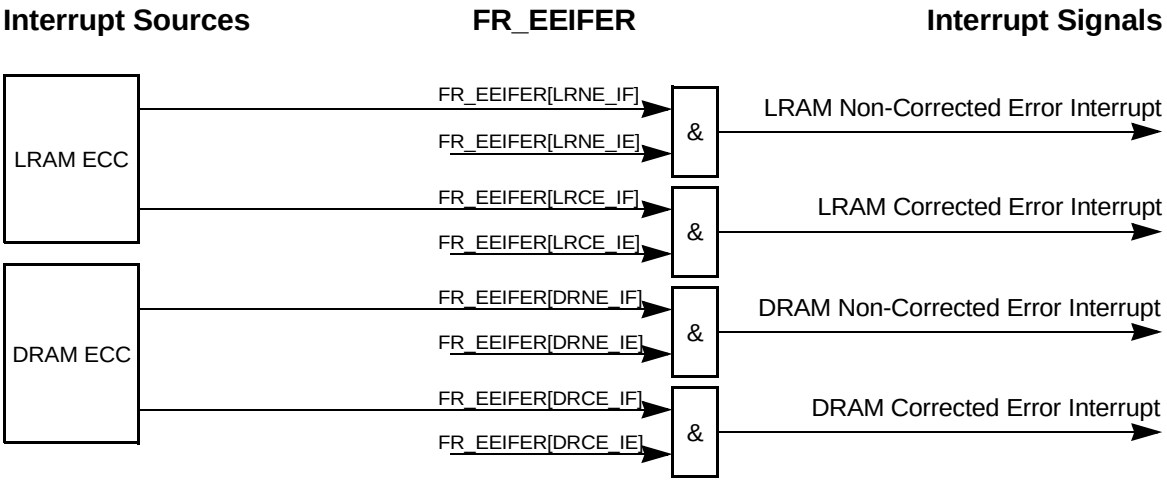


Figure 589. Scheme of FR_EEIFER interrupt signal generation

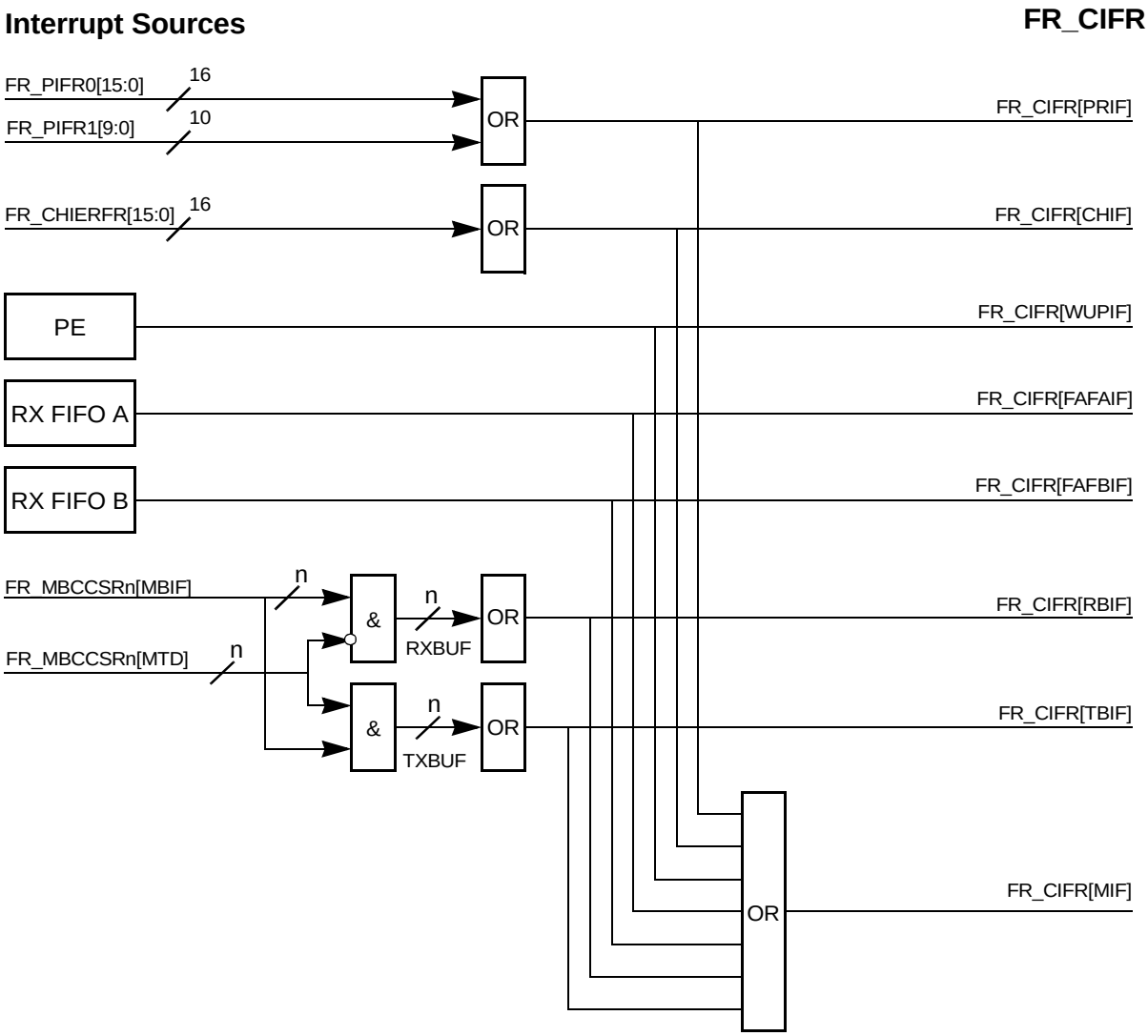


Figure 590. Scheme of FR_CIFR flags generation

27.6.21 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 458*.

Table 458. FlexRay Channel Bit Rate Control

FlexRay Channel Bit Rate [Mbit/s]	FR_MCR.BIT RATE	*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.

27.6.22 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 459. 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.

27.6.22.1 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:= 0x00E0; // INST=0x0; ADDR=070
2. wait until FR_PEDRAR[DAD] == 1; // wait for end of PE DRAM access
3. val = FR_PEDRDR[DATA]; // get 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).

27.6.22.2 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_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

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 27.6.22.3, "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)*.

27.6.22.3 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.

27.6.23 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. The configuration data for two message buffers are contained in one memory row. The CHI LRAM is divided into 6 memory BANKs.

Table 460. 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
...						
0x0F	FR_MBIDXR63	FR_MBFIDR63	MBCCF63	FR_MBIDXR62	MBFID62	FR_MBCCFR62

The CHI LRAM is accessed by the application via regular register read and write accesses.

27.6.24 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.

27.6.24.1 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 461*. 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 461. 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 27.6.24.2.3, "CHI LRAM Checkbits"* and *Section 27.6.24.2.3, "CHI LRAM Checkbits"*
PE DRAM			one out of {No error, Corrected Error, Non-Corrected Error}, defined by coding given in *Section 27.6.24.2.1, "PE DRAM Checkbits"* and *Section 27.6.24.2.2, "PE DRAM Syndrome"*

27.6.24.2 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.

27.6.24.2.1 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 463*. 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 462. 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.

27.6.24.2.2 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 463*.

Table 463. 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

27.6.24.2.3 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 464*. This table shows the implemented Hamming code. If the error injection was applied to distort the checkbits, then the distorted checkbits are reported.

Table 464. 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.

27.6.24.2.4 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 465*.

Table 465. 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

27.6.24.3 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.

27.6.24.3.1 CHI LRAM Memory Error Response after Module Read

The FlexRay module reads the message buffer configuration buffer data located in the CHI LRAM for each each message buffer one time in each slot and in the NIT.

If a non-corrected memory error is detected during this module read access, 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.

27.6.24.3.2 CHI LRAM Memory Error Response after Application Read

The application can read the content of the CHI LRAM via reading the FR_MBCCFRn, FR_MBFIDRn, and FR_MBIDXRn 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.

27.6.24.3.3 PE DRAM Error Response after Module Read

If the module detects an non-corrected memory error during 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.

27.6.24.3.4 PE DRAM Error Response after Application Read in *POC:default config* state

If the module 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.

27.6.24.3.5 PE DRAM Error Response after Application Read out of *POC:default config*

If the module 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.

27.6.25 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.

27.6.25.1 CHI LRAM Error Injection

The following sequence describes an error injection sequence for the CHI LRAM. This sequence includes 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 and impacts of error injection for the application. For error injection out of *POC:default config* see *Section 27.7.2, "CHI LRAM 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;
   - define the bank for error injection; I_BANK = {0,1,2,3,4,5}
5. FR_EEIAR[ADDR]:= I_ADDR;
   - define the address for error injection; 0<= I_ADDR <= 0x0F
6. FR_EEIDR[DATA]:= D_DIST;
   - define the data distortion pattern
7. FR_EEICR[CODE]:= C_DIST;
   - define the checkbit distortion pattern
8. FR_EERICE[EIE]:=1;
   - enable error injection

Application Write Access:

1. If (I_BANK==0) -> FR_MBCCFR(2*I_ADDR):= DATA;
   If (I_BANK==1) -> FR_MBFIDR(2*I_ADDR):= DATA;
   If (I_BANK==2) -> FR_MBIDXR(2*I_ADDR):= DATA;

If (I_BANK==3) -> FR_MBCCFR(2*I_ADDR+1):= DATA;
If (I_BANK==4) -> FR_MBFIDR(2*I_ADDR+1):= DATA;
If (I_BANK==5) -> FR_MBIDXR(2*I_ADDR+1):= DATA;
- write DATA to the defined injection bank and injection address

27.6.25.2 PE DRAM Error Injection

The following sequence describes an error injection sequence for the PE DRAM. This sequence includes the setup of 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 and impacts of error injection for the application. For error injection out of *POC:default config* see *Section 27.7.3, "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 the bank for error injection; I_BANK = {0,1}
5. FR_EEIAR[ADDR]:= I_ADDR;
   - define the address for error injection; 0<= I_ADDR <= 0x7F
6. FR_EEIDR[DATA]:= D_DIST;
   - define the data distortion pattern
7. FR_EEICR[CODE]:= C_DIST;
   - define the checkbit distortion pattern
8. FR_EERICE[EIE]:=1;
   - enable error injection

Application Write Access (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 read from PE DRAM in the next cycle, which triggers the detection of the distorted data.

27.7 Application Information

27.7.1 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.

27.7.1.1 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 27.7.1.2, "Protocol Initialization*".

27.7.1.2 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.

27.7.1.3 CHI LRAM Initialization

The module will start reading CHI LRAM data if it has entered the start up state, thus, all ecc bits have to set correctly. To fulfill this requirement, the application must write initial values into all message buffer configuration registers FR_MBCCFRn, FR_MBFIDRn, and FR_MBIDXRn during the protocol config state, even if the message buffers are not used.

27.7.1.4 PE DRAM Initialization

The PE DRAM initialization is performed by the module 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*.

27.7.2 CHI LRAM Error Injection out of *POC:default config*

When the FlexRay module is out of the *POC:default config* state, it reads the configuration data of all utilized message buffers in every slot. If the module reads the CHI LRAM address that was used for error injection, an memory error is detected and the message buffer is not used for transmission or reception. This section describes how to inject errors on the CHI LRAM without disturbing the running application.

- Set injection address to FR_EEIDR[ADDR] = 0x0F- only the last two message buffers are affected by error injection
- Utilize less than 63 message buffers; FR_MBSSUTR[LAST_MB_UTIL]<=62
  - the last two message buffers are not used and configuration data are not read by the module

27.7.3 PE DRAM Error Injection out of *POC:default config*

When the FlexRay module is out of the *POC:default config* state, only the PE DRAM address 0x70 is writable by the application. This location is not used by the FlexRay module.

27.7.4 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)*

27.7.5 Number of Usable Message Buffers

This section describes the relationship between the number of message buffers that can be utilized and the required minimum CHI clock frequency. Additional constraints for the minimum CHI clock frequency are given in *Section 27.3, "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 must be finished within one FlexRay slot. The shortest FlexRay slot is an empty dynamic slot. An empty dynamic slot is a minislot and consists of *gdMinislot* macroticks with a nominal duration of *gdMacrotick*. The minimum duration of a corrected macrotick is $gdMacrotick_{min}$ = 39 µµT. This results in a minimum slot length of

Equation 44 $\Delta_{slotmin} = 39 \cdot pdMicrotick \cdot gdMinislot$

The search engine is located in the CHI and runs on the CHI clock. It evaluates one individual message buffer per CHI clock cycle. For internal status update and double buffer

commit operations, and as a result of the clock domain crossing jitter, an additional amount of 10 CHI clock cycles is required to ensure correct operation. For a given number of message buffers and for a given CHI clock frequency f_{chi}, this results in a search duration of

Equation 45 $$\Delta_{search} = \frac{1}{f_{chi}} \cdot (\#\ \text{MessageBuffers} + 10)$$

The message buffer search must be finished within one slot which requires that *Equation 46* must be fulfilled

Equation 46 $$\Delta_{search} \leq \Delta_{slotmin}$$

This results in the formula given in *Equation 47* which determines the required minimum CHI frequency for a given number of message buffers that are utilized.

Equation 47 $$f_{chi} \geq \frac{\#\ \text{MessageBuffers} + 10}{39 \cdot pdMicrotick \cdot gdMinislot}$$

The minimum CHI frequency for a selected set of relevant protocol parameters is given in *Table 466*.

Table 466. Minimum f_{chi} [MHz] examples (64 message buffers)

pdMicrotick [ns]	*gdMinislot*					
	2	3	4	5	6	7
25.0	37.94	25.30	18.98	15.18	12.65	10.84
50.0	18.98	12.65	9.45	7.59	6.33	5.43

27.7.6 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 347* 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 467*. 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 467*. 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 467. 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

27.7.7 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.

27.7.7.1 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 468. Transmit Buffer Configuration

Register	Field	Value	Description
FR_MBCCSR*t*	MCM	-	used only for double buffers
	MBT	0	single transmit buffer
	MTD	1	transmit buffer

Table 468. Transmit Buffer Configuration

Register	Field	Value	Description
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 469. Receive Buffer Configuration

Register	Field	Value	Description
FR_MBCCSR*r*	MCM	-	n/a
	MBT	-	n/a
	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.

The cycle set {4n} = {0,4,8,12,...} is assigned to both buffers.

27.7.7.2 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.

27.7.7.3 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.

27.7.7.3.1 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 591*

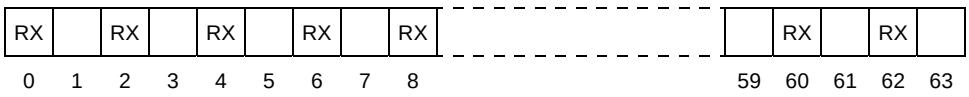


Figure 591. Transmit Data Not Available

27.7.7.3.2 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 591*

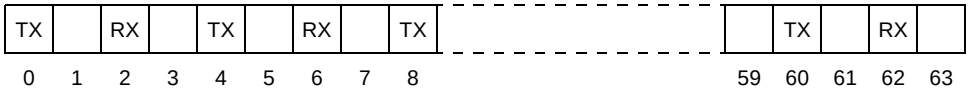


Figure 592. Transmit Data Not Available

28 Frequency-Modulated Phase-Locked Loop (FMPLL)

28.1 Introduction

This section describes the features and functions of the two independent FMPLL modules implemented in SPC56ELxx.

28.2 Overview

The FMPLLs allow the user to generate high speed system clocks from a common 4–40 MHz input clock. Further, the FMPLLs support programmable frequency modulation of the system clock. The FMPLL multiplication factor and the output clock divider ratio are software-configurable.

SPC56ELxx has two FMPLLs: one for the system clock using frequency modulation (FM) and one that can be used for motor control peripherals.

The FMPLL's block diagram is shown in *Figure 593*.

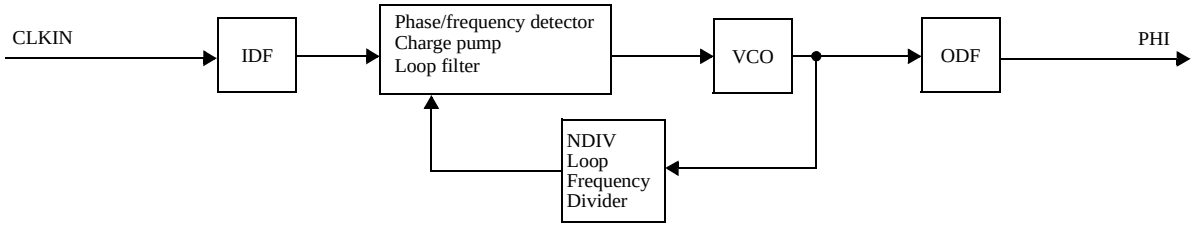


Figure 593. FMPLL block diagram

28.3 Features

The FMPLL has the following major features:

- Input clock frequency from 4–40 MHz
- Voltage controlled oscillator (VCO) range from 256–512 MHz (see *Section 28.7, Requirements*)
- Output divider (ODF) for reduced frequency operation without forcing the FMPLL to relock
- Frequency modulated phase-locked loop (FMPLL)
  - Modulation enabled/disabled through software
  - Triangle wave modulation
- Programmable modulation depth
  - ±0.25% to ±4% deviation from center spread frequency
  - –0.5% to –8% deviation from down spread frequency
  - Programmable modulation frequency dependent on reference frequency
- Self-clocked mode (SCM) operation
- Four operating modes
  - Normal mode
  - Progressive clock switching (only on FMPLL_0 for the system clock)
  - Normal Mode with Spread Spectrum Clock Generation (SSCG)
  - Powerdown mode

28.4 Memory map

The FMPLLs are mapped through the MC_CGM register slot.

Table 470 shows the memory map locations. Addresses are given as offsets of the module base address.

Table 470. FMPLL Memory Map

Address	Register	Access
Base: 0xC3FE_00A0 (FMPLL_0) 0xC3FE_00C0 (FMPLL_1)		
0x0000	Control Register - CR	R/W
0x0004	Modulation register	Special

28.5 Register descriptions

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

28.5.1 Control Register (CR)

Figure 594. Control Register (CR)

Offset 0x0000

Access: User read, 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	1	0	1	0	1	0	0	0	0	0	0

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

Table 471. CR Field Descriptions

Field	Description
2-5 IDF	The value of this field sets the FMPLL Input division factor as described in *Table 472.* The reset value is set during integration.
6-7 ODF	The value of this field sets the FMPLL Output division factor as described in *Table 473.* The reset value is set during integration.
9-15 NDIV	The value of this field sets the FMPLL Loop division factor as described in *Table 474.*The reset value is set during integration.
23 en_pll_sw	This bit is used to enable progressive clock switching. 0 Progressive clock switching disabled 1 Progressive clock switching enabled
24-25	Reserved
27 i_lock	This bit is set by hardware whenever there is a lock/unlock event.It is cleared by software, writing 1.
28 s_lock	This bit is an indication of whether the FMPLL has acquired lock. 0 FMPLL unlocked 1 FMPLL locked

Table 471. CR Field Descriptions (continued)

Field	Description
29 pll_fail_mask	This bit is used to mask the pll_fail output. 0 pll_fail not masked 1 pll_fail masked
30 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 472. Input divide ratios

IDF[3:0]	*Input division factor (idf)*
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 473. Output Divide Ratios

ODF[1:0]	*Output division factor (odf)*
00	Divide by 2
01	Divide by 4
10	Divide by 8
11	Divide by 16

Table 474. Loop Divide Ratios

NDIV[6:0]	*Loop divide factor (ldf)*
0000000-0011111	NA
0100000	Divide by 32
0100001	Divide by 33
0100010	Divide by 34
...	...
1011111	Divide by 95
1100000	Divide by 96
1100001-1111111	NA

28.5.2 Modulation Register (MR)

Figure 595. Modulation Register (MR)

Offset 0x0004

Access: User read, supervisor read/write

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

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

Table 475. MR Field Descriptions

Field	Description
0 STRB_BYPASS	Strobe bypass The STRB_BYPASS signal is used to bypass the STRB signal used inside FMPLL to latch the correct values for control bits (INC_STEP, MOD_PERIOD and SPRD_SEL). 0 = STRB is used to latch FMPLL modulation control bits 1 = STRB is bypassed. In this case control bits need to be static. The control bits must be changed only when FMPLL is in power down mode.
2 SPRD_SEL	Spread type selection The SPRD_SEL control the spread type in Frequency Modulation mode. 0 = Center SPREAD 1 = Down SPREAD
3-15 MOD_PERIOD	Modulation period The MOD_PERIOD field is the binary equivalent of the value modperiod derived from following formula: $modperiod = \frac{f_{ref}}{4 \times f_{mod}}$ where: *fref:* represents the output frequency of the feedback divider (NDIV Loop Frequency Divider) *fmod*: represents the modulation frequency
16 SSCG_EN	Frequency Modulation Enable The SSCG_EN enables the frequency modulation. 0 = Frequency Modulation disabled 1= Frequency Modulation enabled
17-31 INC_STEP	Increment step The INC_STEP field is the binary equivalent of the value incstep derived from following formula: $incstep = round\left(\frac{(2^{15} - 1) \times md \times MDF}{100 \times 5 \times MODPERIOD}\right)$ where: *md*: represents the peak modulation depth in percentage (Center spread -- pk-pk=+/-md, Downspread -- pk-pk=-2*md) *MDF*: represents the nominal value of loop divider (NDIV in FMPLL Control Register)

28.6 Functional description

28.6.1 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 (XOSC) through this relation:

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

where the value of *idf*, *ldf* and *odf* are set in CR and can be derived from *Table 472*, *Table 473*, and *Table 474*.

See also *Section 28.7, Requirements*.

28.6.2 Progressive clock switching

Progressive clock switching is expected to be used on system clock. When changing the system clock to run on a PLL frequency, the PLL locks at a divided frequency, then gradually decreases the division until it is divided by 1. (See illustration below.) The effect is to gradually increase current consumption instead of a single large increase.

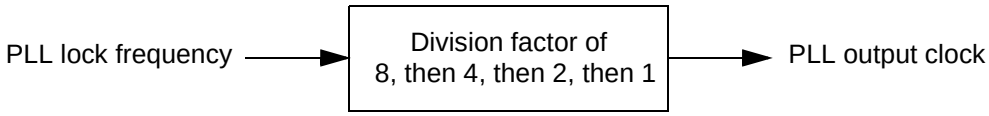


Figure 596. Illustration of progressive clock switching

The gradual transition is further illustrated in *Table 476*.

Table 476. Progressive clock switching for 120 MHz FMPLL operation

FMPLL lock frequency	Division factor	FMPLL output clock frequency	Number of FMPLL output clock cycles during division	Number of FMPLL lock frequency cycles during division
120 MHz	8	15 MHz	15	120
120 MHz	4	30 MHz	30	120
120 MHz	2	60 MHz	60	120
120 MHz	1	120 MHz	onward	—

Progressive clock switching is enabled using the CR[en_pll_sw] field.

Progressive clock switching is a software induced hardware mechanism that can be used as an optional measure to aid in dampening the effects of the large transients that are seen when switching the system clock frequency from very small to very large values or vice versa. Using the progressive clock switching feature alone should be sufficient given the board and power supply are designed to handle the rapid and large changes in current demand that are inherent when switching core frequencies. If issues are encountered, then the change of the system clock can be spread over a larger time period by using application software to manually increase or decrease the PLL frequency in smaller steps.

This can be done by repeatedly decrementing or incrementing the ODF field by one, number of times until the desired frequency is obtained, with the duration of each step controlled by a small software delay loop.

28.6.3 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 modulation is activated using these steps:

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

To latch these values inside the FMPLL, two ways are used depending on the value of STRB_BYPASS register bit in MR.

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

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

The modulation depth in % is

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

Note: You must ensure that the product of INCTEP and MODPERIOD is less than (2^{15}-1) and MODPERIOD is less than or equal to 2^{12}.

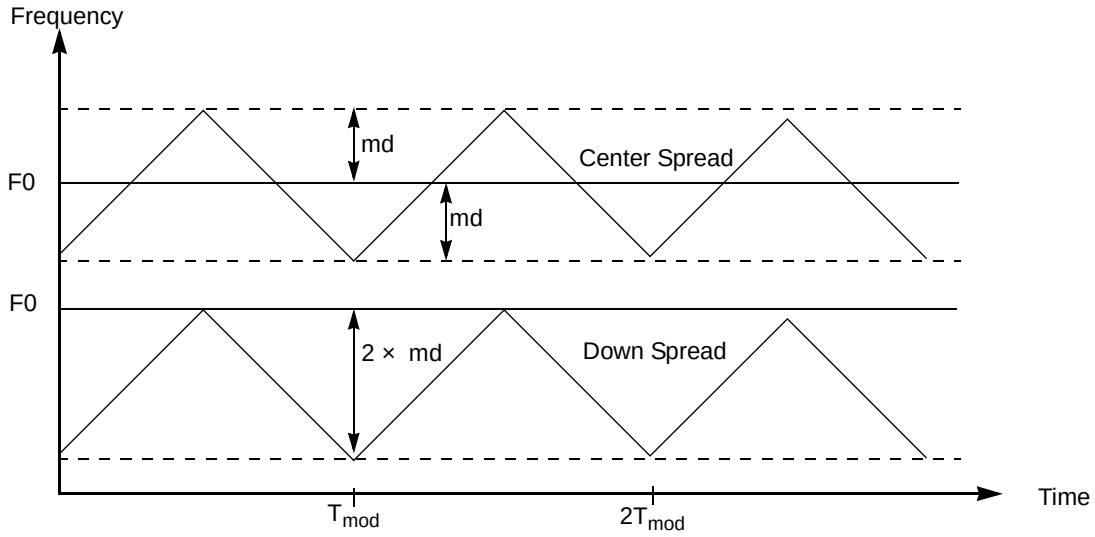


Figure 597. Frequency modulation

28.6.4 Powerdown mode

The FMPLL can be switched off when not required to achieve lower consumption by programming the ME_x_MC registers in the MC_ME module.

28.7 Requirements

The FMPLL VCO frequency f_{VCO}, defined in *Equation 48*, must be in the range 256–512 MHz.

Equation 48 $$f_{VCO} = \frac{CLKIN}{IDF} \cdot NDIV$$

Care is required when programming the multiplication and division factors to respect this requirement.

As a result, the maximum system frequency cannot be reached with all supported (4 MHz to 40 MHz) XOSC (External oscillator) and IRCOSC (Internal RC oscillator) clock frequencies. Before selecting the XOSC clock frequency, you must verify which system clock frequencies will be supported by the available settings for NDIV, IDF, ODF, and other parameters.

28.8 Recommendations

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

- You must change the multiplication, division factors only when the FMPLL output clock is not selected as the source for an active clock on the chip. MOD_PERIOD, INC_STEP, SPREAD_SEL bits should be modified before activating the FM modulated mode. Then strobe has to be generated to enable the new settings. If STRB_BYP is set

to 1 then MOD_PERIOD, INC_STEP and SPREAD_SEL can be modified only when FMPLL is in power down mode.

- Use progressive clock switching for the system clock.

29 Interrupt Controller (INTC)

29.1 Introduction

29.1.1 Module overview

The interrupt controller (INTC) provides priority-based preemptive scheduling of interrupt requests. This scheduling scheme is suitable for statically scheduled hard real-time systems. The INTC supports 256 interrupt requests. It is targeted to work with a Power Architecture processor and automotive powertrain applications where the ISRs nest to multiple levels, but it also can be used with other processors and applications.

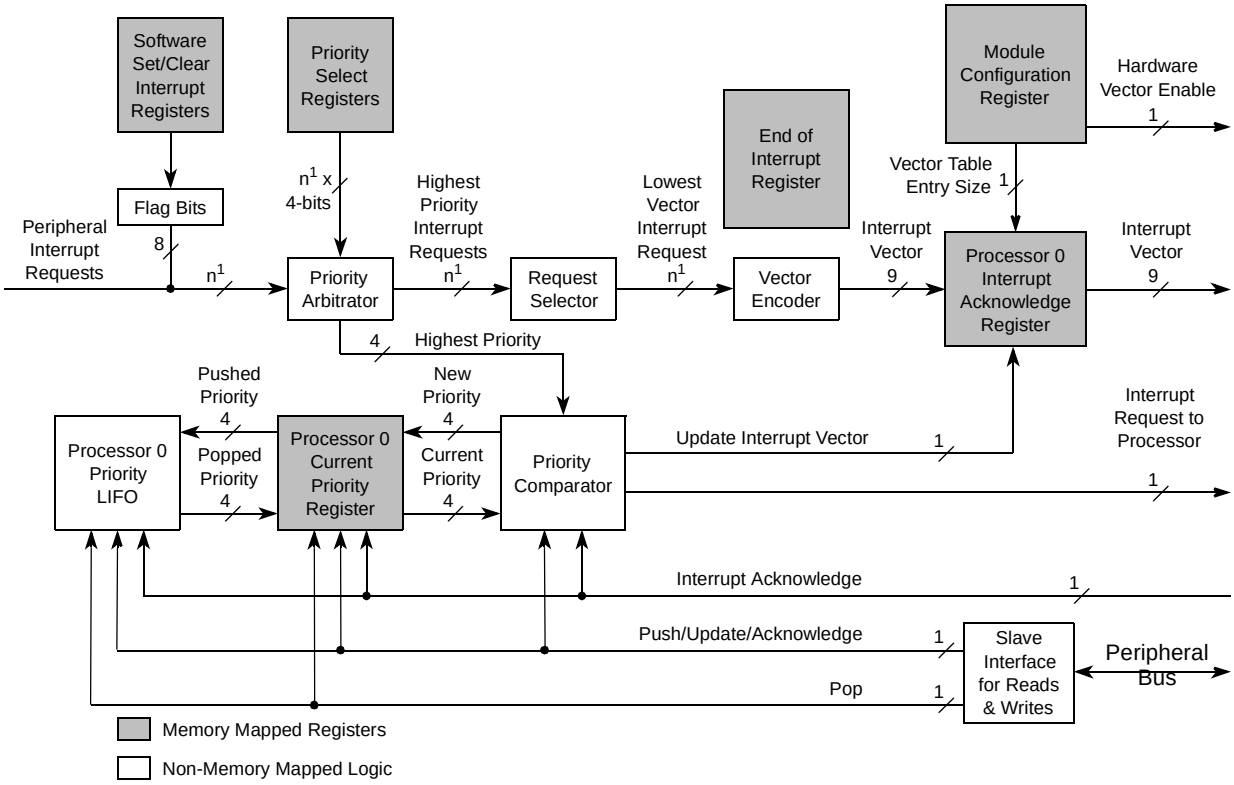
For high priority interrupt requests in these target applications, the time from the assertion of the interrupt request from the peripheral to when the processor is performing useful work to service the interrupt request needs to be minimized. The INTC supports this goal by providing a unique vector for each interrupt request source. It also provides 16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. Since each individual application will have different priorities for each source of interrupt request, the priority of each interrupt request is configurable.

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

Multiple processors can assert interrupt requests to each other through software settable interrupt requests. 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. Therefore these software settable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

29.1.2 Block diagram

Figure 598 shows a block diagram of the interrupt controller (INTC).



1. The total number of interrupt sources is 256.

Figure 598. INTC block diagram

29.1.3 Features

- Supports 248 peripheral interrupt and 8 software-configurable interrupt request sources
- 9-bit vector
  - Unique vector for each interrupt request source.
  - Hardware connection to processor or read from register.
- 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 - three clocks from receipt of interrupt request from peripheral to interrupt request to processor.

29.2 Modes of Operation

29.2.1 Normal Mode

In normal mode, the INTC has two handshaking modes with the processor: software vector mode and hardware vector mode.

29.2.1.1 Software Vector Mode

In software vector mode, software, that is the interrupt exception handler, must read a register in the INTC to obtain the vector associated with the interrupt request to the processor. The INTC will use software vector mode for a given processor when its associated INTC_BCR[HVEN_PRC0] bit is cleared. The hardware vector enable signal to the processor is driven as negated when its associated HVEN_PRC0 bit is negated. The vector is read from the INTC_IACKR_PRC0 register. Reading the INTC_IACKR_PRC0 negates the interrupt request to the associated processor. Even if a higher priority interrupt request arrived while waiting for this interrupt acknowledge, the interrupt request to the processor will negate for at least one clock. The reading also pushes the PRI value in the INTC_CPR register onto the associated LIFO and updates PRI in the associated INTC_CPR_PRC0 with the new priority.

Furthermore, the interrupt vector to the processor is driven as all ‘0’s. The interrupt acknowledge signal from the associated processor is ignored.

29.2.1.2 Hardware Vector Mode

In hardware vector mode, the hardware is the interrupt vector signal from the INTC in conjunction with a processor with the capability to use that vector. In hardware vector mode, this hardware causes the first instruction to be executed in handling the interrupt request to the processor to be specific to that vector. Therefore the interrupt exception handler is specific to a peripheral or software settable interrupt request rather than being common to all of them. The INTC will use hardware vector mode for a given processor when its associated HVEN_PRC0 bit in the INTC_BCR is asserted. The hardware vector enable signal to the associated processor is driven as asserted. When the interrupt request to the associated processor asserts, the interrupt vector signal is updated. The value of that interrupt vector is the unique vector associated with the preempting peripheral or software settable interrupt request. The vector value matches the value of the INTVEC_PRC0 field in the INTC_IACKR_PRC0.

The processor negates the interrupt request to the processor driven by the INTC by asserting the interrupt acknowledge signal for one clock. Even if a higher priority interrupt request arrived while waiting for this interrupt acknowledge, the interrupt request to the processor will negate for at least one clock.

The assertion of the interrupt acknowledge signal for a given processor pushes the associated PRI value in the associated INTC_CPR_PRC0 register onto the associated LIFO and updates the associated PRI in the associated INTC_CPR_PRC0 register with the new priority. This pushing of the PRI value onto the associated LIFO and updating PRI in the associated INTC_CPR_PRC0 does not occur when the associated interrupt acknowledge signal asserts and the INTC_SSCIR register is written at a time such that the PRI value in the associated INTC_CPR_PRC0 register would need to be pushed and the previously last pushed PRI value would need to be popped simultaneously. In this case, PRI in the associated INTC_CPR_PRC0 is updated with the new priority, and the associated LIFO is neither pushed or popped.

29.2.2 Debug Mode

The INTC operation in debug mode is identical to its operation in normal mode.

29.2.3 Stop Mode

The INTC supports the stop mode mechanism. The INTC can have its clock input disabled at any time by the clock driver on the SoC. While its clocks are disabled, the INTC registers are not accessible.

Some SoC applications require that any peripheral interrupt request source be able to awaken a portion or all of the SoC from stop mode. Since the INTC requires clocking in order for a peripheral interrupt request to generate an interrupt request to the processor, it does not support that requirement if it is not clocked.

29.2.4 Factory Test Mode

All INTC registers are accessible in factory test mode.

29.3 External Signal Description

The INTC has no external MCU signals.

29.4 Memory map/register definition

29.4.1 Memory map

Table 477 is the INTC memory map.

Table 477. INTC memory map

Offset from INTC_BASE	Register	Access(1)	Reset value(2)	Location
0x0000	INTC Block Configuration Register (INTC_BCR)	R/W	0x0000_0000	*on page 943*
0x0004	Reserved			
0x0008	INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)	R/W	0x0000_000F	*on page 944*
0x000C	Reserved			
0x0010	INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0)	R/W	0x0000_0000	*on page 945*
0x0014	Reserved			
0x0018	INTC End Of Interrupt Register for Processor 0 (INTC_EOIR_PRC0)	R/W	0x0000_0000	*on page 946*
0x001C	Reserved			
0x0020	INTC Software Set/Clear Interrupt Register 0 - 3 (INTC_SSCIR0_3)	R/W	0x0000_0000	*on page 947*

Table 477. INTC memory map (continued)

Offset from INTC_BASE	Register	Access(1)	Reset value(2)	Location
0x0024	INTC Software Set/Clear Interrupt Register 4 - 7 (INTC_SSCIR4_7)	R/W	0x0000_0000	*on page 947*
0x0028–0x003C	Reserved			
0x0040–0x013C	INTC Priority Select Register 0 - 3 (INTC_PSR0_3) - INTC Priority Select Register 252 - 255 (INTC_PSR252_255)(3)	R/W	0x0000_0000	*on page 948*
0x0140–0x3FFF	Reserved			

1. In this column, R/W = Read/Write, R = Read-only, and W = Write-only.
2. In this column, the symbol “U” indicates one or more bits in a byte are undefined at reset. See the associated description for more information.
3. The PRI fields are 'reserved' for peripheral interrupt requests whose vectors are labeled as Not Used in *Table 480*

29.4.2 Register Information

With exception of the INTC_SSCIRn and INTC_PSRn, all registers are 32 bits in width. Any combination of accessing the four bytes of a register with a single access is supported, provided that the access does not cross a register boundary. These supported accesses include types and sizes of 8 bits, aligned 16 bits, misaligned 16 bits to the middle 2 bytes, and aligned 32 bits.

Although INTC_SSCIRn and INTC_PSRn 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 are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to INTC_SSCIR*n* or INTC_EOIR_PRC0 does not affect on the operation of the write.

29.4.3 INTC Block Configuration Register (INTC_BCR)

INTC_BASE

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	VTES_PRC0	0	0	0	0	HVEN_PRC0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 599. INTC Block Configuration Register (INTC_BCR)

The Block Configuration Register is used to configure options of the INTC.

VTES_PRC0 — Vector Table Entry Size

The VTES_PRC0 bit controls the number of '0's to the right of INTC_IACKR_PRC0[INTVEC_PRC0] in. If the contents of INTC_IACKR_PRC0 are used as an address of an entry in a vector table, then the number of rightmost '0's will determine the size of each vector table entry. See *Section 29.6.3, Code Compression's Impact on Vector Table,* for a use of the VTES_PRC0 bit.

1 = 8 bytes
0 = 4 bytes

HVEN_PRC0 — Hardware Vector Enable

The HVEN bit controls whether the INTC is in hardware vector mode or software vector mode. Refer to *Section 29.2.1, Normal Mode,* for the details of the handshaking with the processor in each mode.

1 = Hardware vector mode
0 = Software vector mode

29.4.4 INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)

INTC_BASE+0x8

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

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

= Unimplemented or Reserved

Figure 600. INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)

The Current Priority Register 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_PRC0 from generating an interrupt request to Processor 0. When INTC_IACKR_PRC0 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 INTC_EOIR_PRC0 is written, the LIFO is popped into the INTC_CPR_PRC0's PRI field. An exception case in hardware vector mode to this behavior is described in *Section 29.2.1.2, Hardware Vector Mode.*

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to *Section 29.6.6, Priority Ceiling Protocol*.

Note: Depending on an SoC implementation's pipelining capabilities and bus architecture, 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 29.6.6.2, Ensuring coherency, for example code to ensure coherency.

PRI[0:3] — Priority.

PRI is the priority of the currently executing ISR according to the field values defined in *Table 478*.

Table 478. PRI Values

PRI	Meaning
1111	Priority 15 - highest priority
1110	Priority 14
1101	Priority 13
1100	Priority 12
1011	Priority 11
1010	Priority 10
1001	Priority 9
1000	Priority 8
0111	Priority 7
0110	Priority 6
0101	Priority 5
0100	Priority 4
0011	Priority 3
0010	Priority 2
0001	Priority 1
0000	Priority 0 - lowest priority

29.4.5 INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0)

Figure 601. INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0)

INTC_BASE+0x10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VTBA_PRC0 (most significant 16 bits)															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20(1)	21	22	23	24	25	26	27	28	29(1)	30	31
R W	VTBA_PRC0 (least significant 5 bits)					INTVEC_PRC0									0	0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. When the INTC_BCR[VTES_PRC0] bit is set, INTVEC_PRC0 is shifted to the left one bit. Bit 29 is read as a ‘0’. VTBA_PRC0 is narrowed to 20 bits in width.

The Interrupt Acknowledge Register for Processor 0 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.

Also, in software vector mode, the INTC_IACKR_PRC0 has side effects from reads. Therefore, it must not be read speculatively while in this mode. The side effects are the same regardless of the size of the read. Reading the INTC_IACKR_PRC0 does not have side effects in hardware vector mode.

VTBA_PRC0[0:20] — Vector Table Base Address for Processor 0.

VTBA_PRC0 can be the base address of a vector table of addresses of ISRs for Processor 0. The VTBA_PRC0 only uses the leftmost 20 bits when the VTES_PRC0 bit in INTC_BCR is asserted.

INTVEC_PRC0[0:8] — Interrupt Vector for Processor 0.

INTVEC_PRC0 is the vector of the peripheral or software settable 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 is in software or hardware vector mode.

29.4.6 INTC End of Interrupt Register for Processor 0 (INTC_EOIR_PRC0)

INTC_BASE+0x18

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	See text															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	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	See text															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 602. INTC End Of Interrupt Register for Processor 0 (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 case in hardware vector mode to this behavior is described in *Section 29.2.1.2, Hardware Vector Mode.*

The values and size of data written to the INTC_EOIR_PRC0 are ignored. Those 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 ‘0’s to the INTC_EOIR_PRC0.

29.4.7 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3 - INTC_SSCIR4_7)

INTC_BASE+0x20

	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

= Unimplemented or Reserved

Figure 603. INTC Software Set/Clear Interrupt Register 0 - 3 (INTC_SSCIR0_3)

INTC_BASE+0x24

	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

= Unimplemented or Reserved

Figure 604. INTC Software Set/Clear Interrupt Register 4 - 7 (INTC_SSCIR4_7)

The Software Set/Clear Interrupt Registers support the setting or clearing of software settable interrupt request. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. With the exception of 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 just like a peripheral interrupt request. Writing a ‘1’ to SET*x* will leave SET*x* unchanged at ‘0’ but will set CLR*x*. Writing a ‘0’ to SET*x* will have no effect. CLR*x* is the flag bit. Writing a ‘1’ to CLR*x* will clear it. Writing a ‘0’ to CLR*x* will have no effect. If a ‘1’ is written to a pair SET*x* and CLR*x* bits at the same time, CLR*x* will be asserted, regardless of whether CLR*x* was asserted before the write.

SET0 - SET7 — Set Flag bits

Writing a ‘1’ will set the corresponding CLR*x* bit. Writing a ‘0’ will have no effect. Each SET*x* always will be read as a ‘0’.

CLR0 - CLR7 — Clear Flag bits

CLR*x* is the flag bit. Writing a ‘1’ to CLR*x* will clear it provided that a ‘1’ is not written simultaneously to its corresponding SET*x* bit. Writing a ‘0’ to CLR*x* will have no effect.

1 = Interrupt request pending within INTC.
0 = Interrupt request not pending within INTC.

29.4.8 INTC Priority Select Registers (INTC_PSR0_3 - INTC_PSR252_255)

INTC_BASE+0x40

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI0				0	0	0	0	PRI1			
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PRI2				0	0	0	0	PRI3			
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 605. INTC Priority Select Register 0 - 3 (INTC_PSR0_3)

INTC_BASE+0x13C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI252				0	0	0	0	PRI253			
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	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	PRI254				0	0	0	0	PRI255			
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 606. INTC Priority Select Register 252 - 255 (INTC_PSR252_255)

The Priority Select Registers support the selection of an individual priority for each source of interrupt request. 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 interrupts 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 8–255, and their priorities are configured in INTC_PSR8_11 through INTC_PSR252_255, respectively.

Note: The PRIx field of an INTC_PSRx_x must not be modified while its corresponding peripheral or software settable interrupt request is asserted.

PRI0[0:3] - PRI255[0:3] — Priority Select.

PRI*x* selects the priority for the interrupt requests. Refer to the field values in *Table 478*.

29.5 Functional Description

The functional description involves the areas of interrupt request sources, priority management, and handshaking with the processor. In addition, spaces in the memory map have been reserved for other possible implementations of the INTC.

29.5.1 Interrupt Request Sources

The INTC has two types of interrupt requests, peripheral and software settable. These interrupt requests can assert on any clock cycle.

The INTC has no spurious vector support. Therefore, if an asserted peripheral or software settable interrupt request, whose PRIx value in the INTC_PSR*n* is higher than the PRI value in INTC_CPR_PRC0, 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_PRC0 will be updated with the corresponding PRIx value in INTC_PSRx_x.

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.

29.5.1.1 Peripheral Interrupt Requests

An interrupt event in a peripheral's hardware sets a flag bit which resides in that 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 cycles.

Interrupt requests from devices external to the INTC are classified as peripheral interrupt requests in this document. An anticipated way that the INTC will support an external interrupt request is by having a flag bit in a peripheral which records an external interrupt request. This flag bit then drives a peripheral interrupt request. It is the responsibility of the interrupt handler to clear this flag bit before the INTC_EOIR_PRC0 can be written.

29.5.1.2 Software settable Interrupt Requests

An interrupt request is triggered by software by writing a '1' to a SET*x* bit in the INTC_SSCIR*n* registers. This write sets the corresponding CLR*x* bit, which is a flag bit, 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.

29.5.1.3 Unique Vector for Each Interrupt Request Source

Each peripheral and software settable interrupt request is assigned a hardwired unique 9-bit vector. Software settable interrupts 0 - 7 are assigned vectors 0 - 7 respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to cover all of the peripheral interrupt requests.

29.5.2 Priority Management

The asserted interrupt requests are compared to each other based on their PRI*x* values set in INTC_PSR*n*. The result of that comparison also is compared to PRI in the associated INTC_CPR_PRC0 register. The results of those comparisons are used to manage the priority of the ISR being executed by the associated processor. The associated LIFO also assists in managing that priority.

29.5.2.1 Current Priority and Preemption

The priority arbitrator, selector, encoder, and comparator subblocks shown in *Figure 598* are used to compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software settable interrupt request is higher than the current priority for a given processor, then the interrupt request to the processor is asserted. Also, a unique vector for the preempting peripheral or software settable interrupt request is generated for the associated INTC_IACKR_PRC0 register, and if in hardware vector mode, for the interrupt vector provided to the processor.

29.5.2.1.1 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 settable. 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.

29.5.2.1.2 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, then 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 settable interrupt requests.

29.5.2.1.3 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.

29.5.2.1.4 Priority Comparator Subblock

The priority comparator subblock compares the highest priority output from the associated priority arbitrator subblock with PRI in the associated INTC_CPR_PRC0. 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 the associated INTC_CPR_PRC0, or the PRI value in the associated INTC_CPR_PRC0 is lowered below this highest priority. This highest priority then becomes the new priority which will be written to PRI in the associated INTC_CPR_PRC0 when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI*x* in INTC_PSR*x*_*x* are zero will not cause a preemption because their PRI*x* will not be higher than PRI in the associated INTC_CPR_PRC0.

29.5.2.2 LIFO

The LIFO stores the preempted PRI values from the associated INTC_CPR_PRC0. 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 associated INTC_CPR_PRC0 does not need to be loaded from the associated INTC_CPR_PRC0 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 associated INTC_CPR_PRC0.

The PRI value in the associated INTC_CPR_PRC0 is pushed onto the LIFO when the associated INTC_IACKR_PRC0 is read in software vector mode or the interrupt acknowledge signal from the associated processor is asserted in hardware vector mode. The priority is popped into PRI in the associated INTC_CPR_PRC0 whenever the associated INTC_EOIR_PRC0 is written. An exception case in hardware vector mode to this behavior is described in *Section 29.2.1.2, Hardware Vector Mode*.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR_PRC0 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.

29.5.3 Handshaking with processor

29.5.3.1 Software vector mode handshaking

29.5.3.1.1 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 607*. The INTC examines the peripheral and software settable interrupt requests. When it finds an asserted peripheral or software settable interrupt request with a higher priority than PRI in the associated *Section 29.4.4, INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)*, it asserts the interrupt request to the associated processor. The INTVEC field in the associated *Section 29.4.5, INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0)* 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 29.2.1.1, Software Vector Mode*.

29.5.3.1.2 End of Interrupt Exception Handler

Before the interrupt exception handling completes, *Section 29.4.6, INTC End of Interrupt Register for Processor 0 (INTC_EOIR_PRC0)*, must be written. When it is written, the associated LIFO is popped so that the preempted priority is restored into PRI of the associated INTC_CPR_PRC0. Before it is written, the peripheral or software settable flag bit must be cleared so that the peripheral or software settable interrupt request is negated.

Note: A store to clear the peripheral or software settable interrupt flag bit which closely precedes the store to the INTC_EOIR_PRC0 can result in that peripheral or software settable interrupt request being serviced again. To prevent this, execute a Power Architecture ISYNC, MSYNC, or MBAR instruction before the store to the INTC_EOIR_PRC0 as shown in Section 29.6.2.1, Software vector mode.

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 the associated INTC_CPR_PRC0 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.

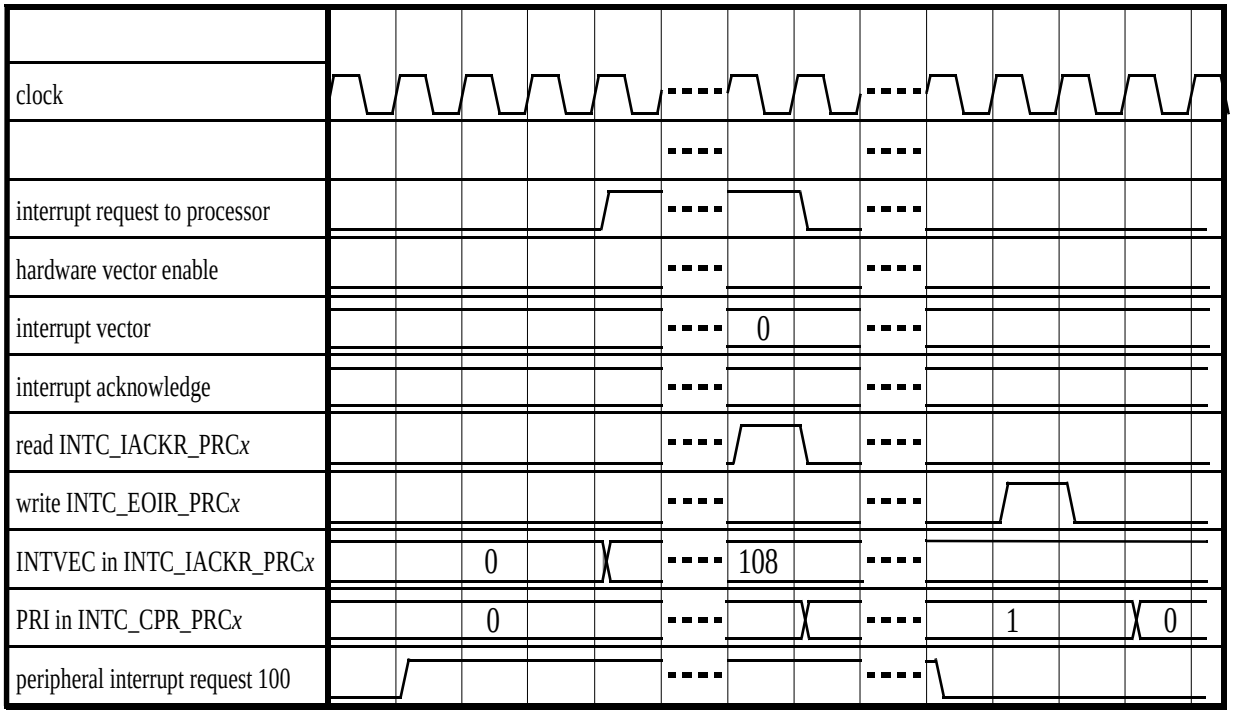


Figure 607. Software Vector Mode Handshaking Timing Diagram

29.5.3.2 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 608*. 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 the associated INTC_CPR_PRC0, it asserts the interrupt request to the associated processor. The INTVEC field in the associated INTC_IACKR_PRC0 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 associated processor is asserted. In addition, the value of the interrupt vector to the associated processor matches the value of the INTVEC field in the

associated INTC_IACKR_PRC0. The rest of the handshaking is described in *Section 29.2.1.2, Hardware Vector Mode*.

The handshaking near the end of the interrupt exception handler, that is the writing to the associated INTC_EOIR_PRC0, is the same as in software vector mode. Refer to *Section 29.5.3.1.2, End of Interrupt Exception Handler*.

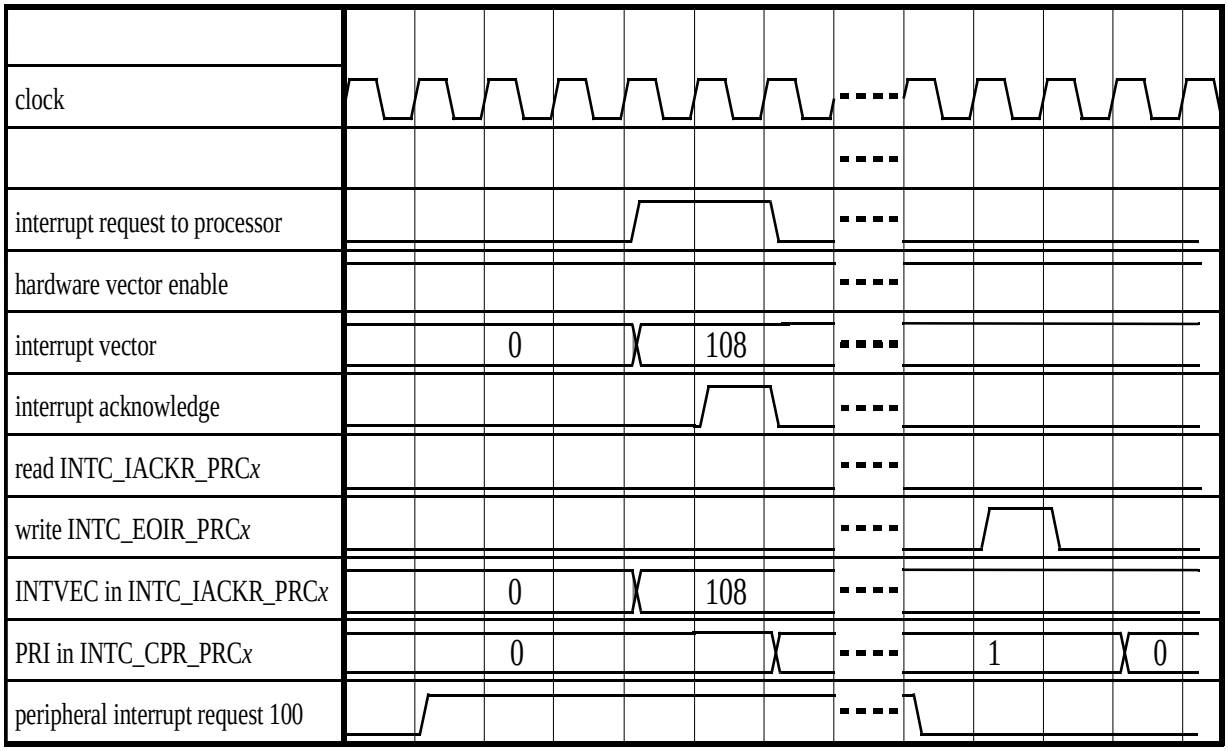


Figure 608. Hardware Vector Mode Handshaking Timing Diagram

29.6 Initialization/application information

29.6.1 Initialization flow

After exiting reset, all of the PRI*x* and PRC_SEL*x* fields in *Section 29.4.8, INTC Priority Select Registers (INTC_PSR0_3 - INTC_PSR252_255)*, will be zero, and PRI in both *Section 29.4.4, INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)*, will be 15. These reset values will prevent the INTC from asserting the interrupt request to the processors. Furthermore, the Indigo-Line section of the IPI specification states that the peripherals must have a bit to enable or mask peripheral interrupt request signals. An initialization sequence for allowing the peripheral and software settable interrupt requests to cause an interrupt request to the processor is:

```
interrupt_request_initialization:
configure VTES_PRC0 and HVEN_PRC0 in INTC_BCR
configure VTBA_PRC0 in INTC_IACKR_PRC0
raise the PRIx fields to the desired processor in INTC_PSRx_x
```

```
set the enable bits or clear the mask bits for the peripheral interrupt
requests
lower PRI in INTC_CPR_PRC0 to zero
enable processor recognition of interrupts
```

29.6.2 Interrupt exception handler

These example interrupt exception handlers use Power Architecture assembly code.

29.6.2.1 Software vector mode

```
interrupt_exception_handler:
code to save SRR0 and SRR1

lisr3,hi(INTC_IACKR_PRC0)# form INTC_IACKR_PRC0 address
orir3,r3,lo(INTC_IACKR_PRC0)
lwzr3,0x0(r3)# load INTC_IACKR_PRC0, 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_PRC0 contents into link register
blrl       # branch to ISR; link register updated with epilog
           # address

epilog:
lisr3,hi(INTC_EOIR_PRC0)# form INTC_EOIR_PRC0 address
orir3,r3,lo(INTC_EOIR_PRC0)
li r4,0x0  # form 0 to write to INTC_EOIR_PRC0
stwr4,0x0(r3)# store to INTC_EOIR_PRC0, 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
```

29.6.2.2 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 save SRR0 and SRR1
code to enable processor recognition of interrupts and save context
required by EABI

bl ISRx    # branch to ISR for interrupt with vector x


epilog:
lisr3,hi(INTC_EOIR_PRC0)# form INTC_EOIR_PRC0 address
orir3,r3,lo(INTC_EOIR_PRC0)
li r4,0x0  # form 0 to write to INTC_EOIR_PRC0
stwr4,0x0(r3)# store to INTC_EOIR_PRC0, 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


ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr       # branch to epilog
```

29.6.3 Code Compression's Impact on Vector Table

The entries in the vector table in the interrupt exception handler example in *Section 29.6.2.1, Software vector mode*, are addresses of ISRs. A vector table also can be written whose entries are branches to ISRs. The instruction flow branches to the entry in the vector table corresponding to the vector of the peripheral or software settable interrupt request, and then the instruction flow branches to the corresponding ISR.

While a vector table of addresses of ISRs is not affected by code compression, a vector table of branches to ISRs can be affected. Code compression techniques can produce for some cases instructions that are wider than 32 bits but less than 64 bits. If the vector table is compressed and some instructions are wider than 32 bits, the VTES_PRC0 bit in *Section 29.4.3, INTC Block Configuration Register (INTC_BCR)*, can be set. Each entry in the vector table then can occupy 64 bits.

29.6.4 ISR, RTOS, and Task Hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in *Section 29.4.4, INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)*, having a value of 0. The RTOS will execute the tasks according to whatever priority scheme that it may have, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC_CPR_PRC0 priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR_PRC0 priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR_PRC0 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_PRC0 while the shared resource is being accessed.

An ISR whose PRI*x* in *Section 29.4.8, INTC Priority Select Registers (INTC_PSR0_3 - INTC_PSR252_255)*, has a value of 0 will 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 will cause it to remain negated, which consequently also will not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR will not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

29.6.5 Order of Execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software settable interrupt requests. However, if multiple peripheral or software settable 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 settable interrupt requests asserted.

The example in *Table 479* shows the order of execution of both ISRs with different priorities and the same priority.

Table 479. Order of ISR execution example

		Code executing at end of step						
Step#	**Step Description**	**RTOS**	**ISR108[1]**	**ISR208**	**ISR308**	**ISR408**	**interrupt exception handler**	**PRI in INTC_CPR at end of step**
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 is asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 is asserts.					X		4
5	Peripheral interrupt request 200 at priority 3 is asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR_PRC0.						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_PRC0.						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR_PRC0.						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR_PRC0.						X	0
12	RTOS continues execution.	X						0

1. ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software settable interrupt requests.

29.6.6 Priority Ceiling Protocol

29.6.6.1 Elevating Priority

The PRI field in *Section 29.4.4, INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)*, is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol therefore 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*x* to 3, the ceiling of all of the ISR priorities. After they release the resource, they must lower the PRI value in INTC_CPR_PRC*x* to prevent further priority inversion. If they do not raise their priority, then 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 can not 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 reduces the priority inversion time when accessing a shared resource. For example, while ISR3 can not preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

29.6.6.2 Ensuring coherency

A scenario can exist on some SoC implementations that can cause non-coherent accesses to the shared resource. As an example, ISR1 and ISR2 both share a resource. ISR1 has a lower priority than ISR2. ISR1 is executing, and it writes to the INTC_CPR_PRC0. The instruction following this store is a store to a value in a shared coherent data block. Either just before or at the same time as the first store, the INTC asserts the interrupt request to the processor because the peripheral interrupt request for ISR2 has asserted. As the processor is responding to the interrupt request from the INTC, and as it is aborting transactions and flushing its pipeline, it is possible in some SoC implementations that both of these stores will be executed. ISR2 thereby thinks that it can access the data block coherently, but the data block has been corrupted.

OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent this corruption of a coherent data block, modifications to PRI in INTC_CPR_PRC0 can be made by those system services with the code sequence:

```
disable processor recognition of interrupts
PRI modification
enable processor recognition of interrupts
```

29.6.7 Selecting Priorities According to Request Rates and Deadlines

The selection of the priorities for the ISRs can be made using Rate Monotonic Scheduling or a superset of it, Deadline Monotonic Scheduling. 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 could be much 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 covered, regardless of the number of ISRs.

Reducing the number of priorities does cause some priority inversion which 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 can be placed at the same priority without any further priority inversion, and they do not need to use the PCP to access the shared resource.

29.6.8 Software settable Interrupt Requests

The software settable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and for processors to interrupt other processors in a multiple processor system.

29.6.8.1 Scheduling a Lower Priority Portion of an ISR

A portion of an ISR needs to be executed at the PRI*x* value in *Section 29.4.8, INTC Priority Select Registers (INTC_PSR0_3 - INTC_PSR252_255)*,, which becomes the PRI value in either *Section 29.4.4, INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)*, 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 which does not need to be executed at this higher priority can block the execution of ISRs which do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This priority inversion reduces the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a SET*x* bit in *Section 29.4.7, INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3 - INTC_SSCIR4_7)*. Writing a '1' to SET*x* causes a software settable interrupt request. This software settable interrupt request, which usually will have a lower PRI*x* value in the INTC_PSR*x*_*x*, therefore will not cause priority inversion.

29.6.8.2 Scheduling an ISR on Another Processor

Since 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 simply 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 processor executing the software settable ISR has not completed the work before asking it to again execute that 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. The procedure is that the first processor writes a '1' to a SET*x* bit on the second processor. The second processor, after accessing the block of data, clears the corresponding CLR*x* bit and then writes '1' to a SET*x* bit on the first processor, informing it that it now can access the block of data.

29.6.9 Lowering Priority Within an ISR

In SoC implementations without the software settable interrupt requests in *Section 29.4.7, INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3 - INTC_SSCIR4_7)*, the only other way besides scheduling a task through an RTOS to not have priority inversion with an ISR whose work spans multiple priorities as described in *Section 29.6.8.1, Scheduling a Lower Priority Portion of an ISR*, is to lower the current priority. However, the INTC has a LIFO whose depth is determined by the number of priorities.

Note: Lowering the PRI value in either Section 29.4.4, INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0), within an ISR to below the ISR's corresponding PRI value in Section 29.4.8, INTC Priority Select Registers (INTC_PSR0_3 - INTC_PSR252_255), allows more preemptions than the depth of the LIFO can support.

Therefore, the INTC does not support lowering the current priority within an ISR as a way to avoid priority inversion.

29.6.10 Negating an Interrupt Request Outside of its ISR

29.6.10.1 Negating an Interrupt Request as a Side Effect of an ISR

Some peripherals have flag bits which 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 consequently their corresponding interrupt requests too. 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.

29.6.10.2 Negating Multiple Interrupt Requests in One ISR

An ISR can clear other flag bits besides its own flag bit. One reason that an ISR clears multiple flag bits is because it serviced those other flag bits, and therefore the ISRs for these other flag bits do not need to be executed.

29.6.10.3 Proper Setting of Interrupt Request Priority

Whether an interrupt request negates outside of 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 settable interrupt requests for these other flag bits must be selected properly. Their PRI*x* values in *Section 29.4.8, INTC Priority Select Registers (INTC_PSR0_3 - INTC_PSR252_255)*, must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits still 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 *Section 29.4.7, INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3 - INTC_SSCIR4_7)*, as the clearing of the flag bit that caused the present ISR to be executed. Refer to *Section 29.5.3.1.2, End of Interrupt Exception Handler*.

A flag bit whose enable bit or mask bit is negating 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*.

29.6.11 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 should want to read the contents, such as in debug mode, they are not memory mapped. The contents still can be read by popping the LIFO and reading the PRI field in either *Section 29.4.4, INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)*. The code sequence is:

```
pop_lifo:
store to INTC_EOIR_PRC0
load INTC_CPR_PRC0, 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_PRCx
load INTC_IACKR_PRC0
if stacked PRI values are not depleted, branch to push_lifo
```

29.7 Interrupt sources

Table 480 defines the interrupt sources for interrupt controller INTC_0 and INTC_1. Interrupts generated by on-platform peripherals are routed to their own interrupt controller only. Interrupts generated by off-plaform peripherals are routed to both interrupt controllers.

In Decoupled Parallel Mode the interrupt load can be distributed to both cores by assigning different interrupt levels (interrupt level 0 to effectively disable an IRQ) to the same source at each interrupt controller.

Table 480. Interrupt sources for INTC_0 and INTC_1

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
Section B (On-Platform Peripherals)					
0	0x0000	16	Software setable flag 0	INTC_0 (Software)	INTC_1 (Software)
1	0x0010	16	Software setable flag 1	INTC_0 (Software)	INTC_1 (Software)
2	0x0020	16	Software setable flag 2	INTC_0 (Software)	INTC_1 (Software)
3	0x0030	16	Software setable flag 3	INTC_0 (Software)	INTC_1 (Software)
4	0x0040	16	Software setable flag 4	INTC_0 (Software)	INTC_1 (Software)
5	0x0050	16	Software setable flag 5	INTC_0 (Software)	INTC_1 (Software)
6	0x0060	16	Software setable flag 6	INTC_0 (Software)	INTC_1 (Software)
7	0x0070	16	Software setable flag 7	INTC_0 (Software)	INTC_1 (Software)
8	0x0080	16	Not used		

Table 480. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
9	0x0090	16	Platform Flash Bank 0 Abort | Platform Flash Bank 0 Stall | Platform Flash Bank 1 Abort | Platform Flash Bank 1 Stall | Platform Flash Bank 2 Abort | Platform Flash Bank 2 Stall | Platform Flash Bank 3 Abort | Platform Flash Bank 3 Stall	ECSM_0	ECSM_1(1)
10	0x00A0	16	Combined Error	DMA_0	DMA_1
11	0x00B0	16	Channel 0	DMA_0	DMA_1
12	0x00C0	16	Channel 1	DMA_0	DMA_1
13	0x00D0	16	Channel 2	DMA_0	DMA_1
14	0x00E0	16	Channel 3	DMA_0	DMA_1
15	0x00F0	16	Channel 4	DMA_0	DMA_1
16	0x0100	16	Channel 5	DMA_0	DMA_1
17	0x0110	16	Channel 6	DMA_0	DMA_1
18	0x0120	16	Channel 7	DMA_0	DMA_1
19	0x0130	16	Channel 8	DMA_0	DMA_1
20	0x0140	16	Channel 9	DMA_0	DMA_1
21	0x0150	16	Channel 10	DMA_0	DMA_1
22	0x0160	16	Channel 11	DMA_0	DMA_1
23	0x0170	16	Channel 12	DMA_0	DMA_1
24	0x0180	16	Channel 13	DMA_0	DMA_1
25	0x0190	16	Channel 14	DMA_0	DMA_1
26	0x01A0	16	Channel 15	DMA_0	DMA_1
27	0x01B0	16	Reserved		
28	0x01C0	16	SWT Timeout(2)	SWT_0	SWT_0
29	0x01D0	16	SWT Timeout(2)	SWT_1	SWT_1
30	0x01E0	16	Match on channel 0	STM_0	STM_1
31	0x01F0	16	Match on channel 1	STM_0	STM_1
32	0x0200	16	Match on channel 2	STM_0	STM_1
33	0x0210	16	Match on channel 3	STM_0	STM_1
34	0x0220	16	Not used		
35	0x0230	16	ECC_PlatformFlash_ noncorrectable error| ECC_DBD_PlatformRAM_ noncorrectable error	ECSM_0	ECSM_1(1)

Table 480. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
36	0x0240	16	ECC_PlatformFlash_1bit_correction| ECC_PlatformRAM_1bit_correction	ECSM_0	ECSM_1(1)
37	0x0250	16	Not used	Not allocated	Not allocated
Section C (Off-Platform Peripherals)					
38	0x0260	16	Not used		
39	0x0270	16	Not used		
40	0x0280	16	Not used		
41	0x0290	16	EIRQ[0:7]	SIUL	
42	0x02A0	16	EIRQ[8:15]	SIUL	
43	0x02B0	16	EIRQ[16:23]	SIUL	
44	0x02C0	16	EIRQ[24:31]	SIUL	
45	0x02D0	16	Not used		
46	0x02E0	16	Not used		
47	0x02F0	16	Not used		
48	0x0300	16	Not used		
49	0x0310	16	Not used		
50	0x0320	16	Not used		
51	0x0330	16	Safe Mode Interrupt	MC_ME	MC_ME
52	0x0340	16	Mode Transition Interrupt	MC_ME	MC_ME
53	0x0350	16	Invalid Mode Interrupt	MC_ME	MC_ME
54	0x0360	16	Invalid Mode Config	MC_ME	MC_ME
55	0x0370	16	Not used		
56	0x0380	16	'Functional' Reset Alternate Event Interrupt	MC_RGM	MC_RGM
57	0x0390	16	XOSC counter expired	XOSC	XOSC
58	0x03A0	16	Not used		
59	0x03B0	16	PITimer Channel 0	PIT	PIT
60	0x03C0	16	PITimer Channel 1	PIT	PIT
61	0x03D0	16	PITimer Channel 2	PIT	PIT
62	0x03E0	16	ADC_EOC	ADC_0	ADC_0
63	0x03F0	16	ADC_ER	ADC_0	ADC_0
64	0x0400	16	ADC_WD	ADC_0	ADC_0
65	0x0410	16	FLEXCAN_ESR[ERR_INT]	FlexCAN_0	FlexCAN_0

Table 480. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
66	0x0420	16	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_0	FlexCAN_0
67	0x0430	16	Reserved	Reserved	Reserved
68	0x0440	16	FLEXCAN_BUF_00_03	FlexCAN_0	FlexCAN_0
69	0x0450	16	FLEXCAN_BUF_04_07	FlexCAN_0	FlexCAN_0
70	0x0460	16	FLEXCAN_BUF_08_11	FlexCAN_0	FlexCAN_0
71	0x0470	16	FLEXCAN_BUF_12_15	FlexCAN_0	FlexCAN_0
72	0x0480	16	FLEXCAN_BUF_16_31	FlexCAN_0	FlexCAN_0
73	0x0490	16	Not used		
74	0x04A0	16	DSPI_SR[TFUF] | DSPI_SR[RFOF]	DSPI_0	DSPI_0
75	0x04B0	16	DSPI_SR[EOQF]	DSPI_0	DSPI_0
76	0x04C0	16	DSPI_SR[TFFF]	DSPI_0	DSPI_0
77	0x04D0	16	DSPI_SR[TCF]	DSPI_0	DSPI_0
78	0x04E0	16	DSPI_SR[RFDF]	DSPI_0	DSPI_0
79	0x04F0	16	LINFlexD_RXI	LINFlexD_0	LINFlexD_0
80	0x0500	16	LINFlexD_TXI	LINFlexD_0	LINFlexD_0
81	0x0510	16	LINFlexD_ERR	LINFlexD_0	LINFlexD_0
82	0x0520	16	ADC_EOC	ADC_1	ADC_1
83	0x0530	16	ADC_ER	ADC_1	ADC_1
84	0x0540	16	ADC_WD	ADC_1	ADC_1
85	0x0550	16	FLEXCAN_ESR[ERR_INT]	FlexCAN_1	FlexCAN_1
86	0x0560	16	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_1	FlexCAN_1
87	0x0570	16	Reserved		
88	0x0580	16	FLEXCAN_BUF_00_03	FlexCAN_1	FlexCAN_1
89	0x0590	16	FLEXCAN_BUF_04_07	FlexCAN_1	FlexCAN_1
90	0x05A0	16	FLEXCAN_BUF_08_11	FlexCAN_1	FlexCAN_1
91	0x05B0	16	FLEXCAN_BUF_12_15	FlexCAN_1	FlexCAN_1
92	0x05C0	16	FLEXCAN_BUF_16_31	FlexCAN_1	FlexCAN_1
93	0x05D0	16	Not used		
94	0x05E0	16	DSPI_SR[TFUF] | DSPI_SR[RFOF]	DSPI_1	DSPI_1

Table 480. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
95	0x05F0	16	DSPI_SR[EOQF]	DSPI_1	DSPI_1
96	0x0600	16	DSPI_SR[TFFF]	DSPI_1	DSPI_1
97	0x0610	16	DSPI_SR[TCF]	DSPI_1	DSPI_1
98	0x0620	16	DSPI_SR[RFDF]	DSPI_1	DSPI_1
99	0x0630	16	LINFlexD_RXI	LINFlexD_1	LINFlexD_1
100	0x0640	16	LINFlexD_TXI	LINFlexD_1	LINFlexD_1
101	0x0650	16	LINFlexD_ERR	LINFlexD_1	LINFlexD_1
102	0x0660	16	Not used		
103	0x0670	16	Not used		
104	0x0680	16	Not used		
105	0x0690	16	Not used		
106	0x06A0	16	Not used		
107	0x06B0	16	Not used		
108	0x06C0	16	Not used		
109	0x06D0	16	Not used		
110	0x06E0	16	Not used		
111	0x06F0	16	Not used		
112	0x0700	16	Not used		
113	0x0710	16	Not used		
114	0x0720	16	DSPI_SR[TFUF] | DSPI_SR[RFOF]	DSPI_2	DSPI_2
115	0x0730	16	DSPI_SR[EOQF]	DSPI_2	DSPI_2
116	0x0740	16	DSPI_SR[TFFF]	DSPI_2	DSPI_2
117	0x0750	16	DSPI_SR[TCF]	DSPI_2	DSPI_2
118	0x0760	16	DSPI_SR[RFDF]	DSPI_2	DSPI_2
119	0x0770	16	Not used		
120	0x0780	16	Not used		
121	0x0790	16	Not used		
122	0x07A0	16	Not used		
123	0x07B0	16	Not used		
124	0x07C0	16	Not used		
125	0x07D0	16	Not used		
126	0x07E0	16	Not used		
127	0x07F0	16	PITimer Channel 3	PIT	PIT

Table 480. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
128	0x0800	16	Not used		
129	0x0810	16	Not used		
130	0x0820	16	Not used		
131	0x0830	16	LRNEIF | DRNEIF	FlexRay	FlexRay
132	0x0840	16	LRCEIF | DRCEIF	FlexRay	FlexRay
133	0x0850	16	FAFAIF	FlexRay	FlexRay
134	0x0860	16	FAFVIF	FlexRay	FlexRay
135	0x0870	16	WUPIF	FlexRay	FlexRay
136	0x0880	16	PRIF	FlexRay	FlexRay
137	0x0890	16	CHIF	FlexRay	FlexRay
138	0x08A0	16	TBIF	FlexRay	FlexRay
139	0x08B0	16	RBIF	FlexRay	FlexRay
140	0x08C0	16	MIF	FlexRay	FlexRay
141	0x08D0	16	Reserved		
142	0x08E0	16	Reserved		
143	0x08F0	16	Reserved		
144	0x0900	16	Reserved		
145	0x0910	16	Reserved		
146	0x0920	16	Reserved		
147	0x0930	16	Reserved		
148	0x0940	16	Reserved		
149	0x0950	16	Reserved		
150	0x0960	16	Reserved		
151	0x0970	16	Reserved		
152	0x0980	16	Reserved		
153	0x0990	16	Reserved		
154	0x09A0	16	Reserved		
155	0x09B0	16	Reserved		
156	0x09C0	16	Reserved		
Section D (Off-Platform Peripherals)					
157	0x09D0	16	TC0IR	eTimer_0	eTimer_0
158	0x09E0	16	TC1IR	eTimer_0	eTimer_0
159	0x09F0	16	TC2IR	eTimer_0	eTimer_0
160	0x0A00	16	TC3IR	eTimer_0	eTimer_0

Table 480. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
161	0x0A10	16	TC4IR	eTimer_0	eTimer_0
162	0x0A20	16	TC5IR	eTimer_0	eTimer_0
163	0x0A30	16	Not used		
164	0x0A40	16	Not used		
165	0x0A50	16	WTIF	eTimer_0	eTimer_0
166	0x0A60	16	Not used		
167	0x0A70	16	RCF	eTimer_0	eTimer_0
168	0x0A80	16	TC0IR	eTimer_1	eTimer_1
169	0x0A90	16	TC1IR	eTimer_1	eTimer_1
170	0x0AA0	16	TC2IR	eTimer_1	eTimer_1
171	0x0AB0	16	TC3IR	eTimer_1	eTimer_1
172	0x0AC0	16	TC4IR	eTimer_1	eTimer_1
173	0x0AD0	16	TC5IR	eTimer_1	eTimer_1
174	0x0AE0	16	Not used		
175	0x0AF0	16	Not used		
176	0x0B00	16	Not used		
177	0x0B10	16	Not used		
178	0x0B20	16	RCF	eTimer_1	eTimer_1
179	0x0B30	16	RF0	FlexPWM_0	
180	0x0B40	16	COF0	FlexPWM_0	
181	0x0B50	16	CAF0	FlexPWM_0	
182	0x0B60	16	RF1	FlexPWM_0	
183	0x0B70	16	COF1	FlexPWM_0	
184	0x0B80	16	CAF1	FlexPWM_0	
185	0x0B90	16	RF2	FlexPWM_0	
186	0x0BA0	16	COF2	FlexPWM_0	
187	0x0BB0	16	CAF2	FlexPWM_0	
188	0x0BC0	16	RF3	FlexPWM_0	
189	0x0BD0	16	COF3	FlexPWM_0	
190	0x0BE0	16	CAF3	FlexPWM_0	
191	0x0BF0	16	FFLAG	FlexPWM_0	
192	0x0C00	16	REF	FlexPWM_0	
193	0x0C10	16	MRS_I	CTU_0	
194	0x0C20	16	T0_I	CTU_0	

Table 480. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
195	0x0C30	16	T1_I	CTU_0	
196	0x0C40	16	T2_I	CTU_0	
197	0x0C50	16	T3_I	CTU_0	
198	0x0C60	16	T4_I	CTU_0	
199	0x0C70	16	T5_I	CTU_0	
200	0x0C80	16	T6_I	CTU_0	
201	0x0C90	16	T7_I	CTU_0	
202	0x0CA0	16	FIFO0_I	CTU_0	
203	0x0CB0	16	FIFO1_I	CTU_0	
204	0x0CC0	16	FIFO2_I	CTU_0	
205	0x0CD0	16	FIFO3_I	CTU_0	
206	0x0CE0	16	ADC_I	CTU_0	
207	0x0CF0	16	ERR_I	CTU_0	
208	0x0D00	16	Not used		
209	0x0D10	16	Not used		
210	0x0D20	16	Not used		
211	0x0D30	16	Not used		
212	0x0D40	16	Not used		
213	0x0D50	16	Not used		
214	0x0D60	16	Not used		
215	0x0D70	16	Not used		
216	0x0D80	16	Not used		
217	0x0D90	16	Not used		
218	0x0DA0	16	Not used		
219	0x0DB0	16	Not used		
220	0x0DC0	16	Not used		
221	0x0DD0	16	Not used		
222	0x0DE0	16	TC0IR	eTimer_2	
223	0x0DF0	16	TC1IR	eTimer_2	
224	0x0E00	16	TC2IR	eTimer_2	
225	0x0E10	16	TC3IR	eTimer_2	
226	0x0E20	16	TC4IR	eTimer_2	
227	0x0E30	16	TC5IR	eTimer_2	
228	0x0E40	16	Not used		

Table 480. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
229	0x0E50	16	Not used		
230	0x0E60	16	Not used		
231	0x0E70	16	Not used		
232	0x0E80	16	RCF	eTimer_2	
233	0x0E90	16	RF0	FlexPWM_1	
234	0x0EA0	16	COF0	FlexPWM_1	
235	0x0EB0	16	CAF0	FlexPWM_1	
236	0x0EC0	16	RF1	FlexPWM_1	
237	0x0ED0	16	COF1	FlexPWM_1	
238	0x0EE0	16	CAF1	FlexPWM_1	
239	0x0EF0	16	RF2	FlexPWM_1	
240	0x0F00	16	COF2	FlexPWM_1	
241	0x0F10	16	CAF2	FlexPWM_1	
242	0x0F20	16	RF3	FlexPWM_1	
243	0x0F30	16	COF3	FlexPWM_1	
244	0x0F40	16	CAF3	FlexPWM_1	
245	0x0F50	16	FFLAG	FlexPWM_1	
246	0x0F60	16	REF	FlexPWM_1	
247	0x0F70	16	Core_0 Failed Lock Attempt(3)	SEMA4_0	-
			Core_1 Failed Lock Attempt(3)	-	SEMA4_0
248	0x0F80	16	Core_0 Failed Lock Attempt(3)	SEMA4_1	-
			Core_1 Failed Lock Attempt(3)	-	SEMA4_1
249	0x0F90	16	Not used		
250	0x0FA0	16	Alarm Interrupt (ALRM)	FCCU	
251	0x0FB0	16	Configuration Time-out (CFG_TO)	FCCU	
252	0x0FC0	16	Self Check RCC0 Failure	FCCU	
253	0x0FD0	16	Self Check RCC1 Failure	FCCU	
254	0x0FE0	16	Voltage Monitor Fault or High/Low Voltage Detector Error	PMU	
255	0x0FF0	16	SWG Error (SERR)	SWG	

1. As the PFLASHC_1 is not operational in DP mode, the ECSM_1 instantiation does not receive any flash memory ECC status or RWW information and therefore can not signal any RWW, ECC SBC or non-correctable ECC interrupt to the INTC_1. Therefore CORE_1 will NOT receive any status notification on FLASH RWW events and ECC errors in DP Mode.
2. The interrupt from SWT 0 goes to both INTCs as IRQ 28. The interrupt from SWT 1 goes to both INTCs as IRQ 29.
3. The SEMA4 unit is only available in DP Mode.

30 JTAG Controller (JTAGC)

30.1 Introduction

Figure 609 is a block diagram of the JTAG Controller (JTAGC) block.

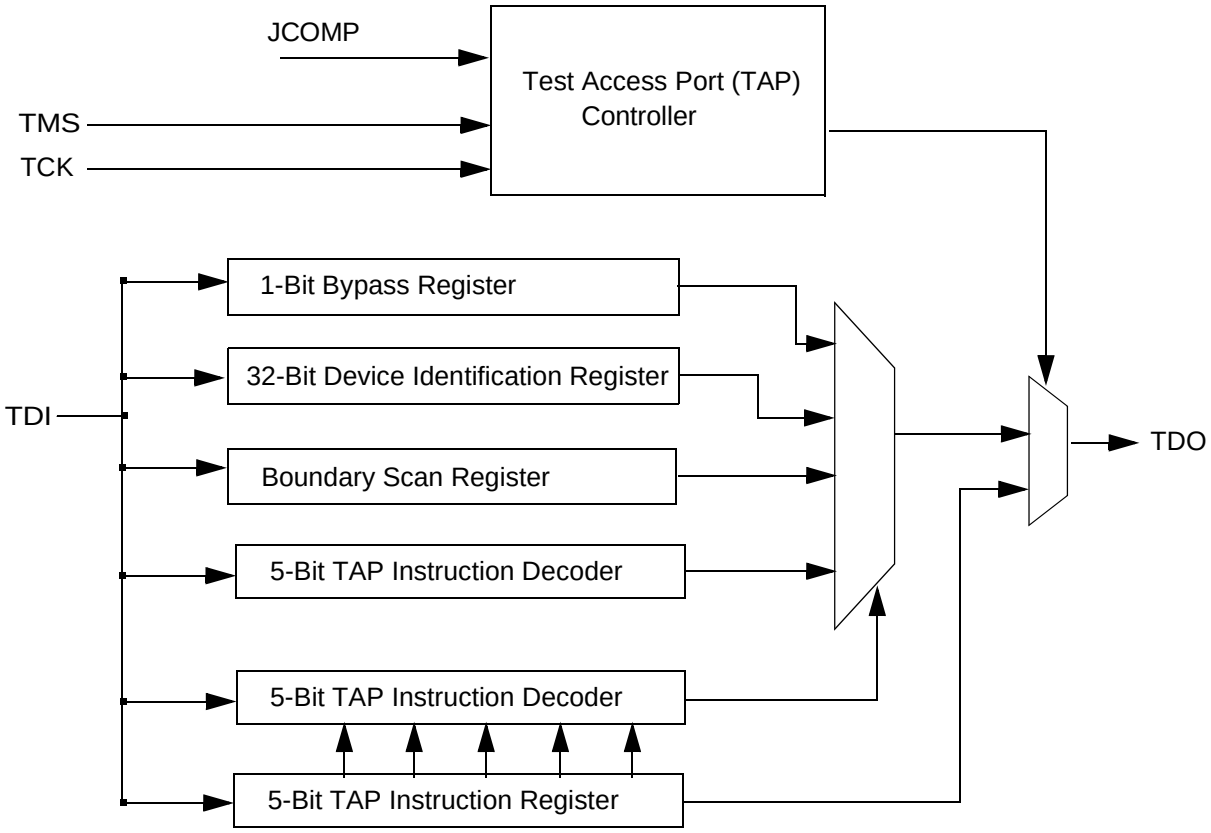


Figure 609. JTAG STL (IEEE 1149.1) block diagram

30.1.1 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.

30.1.2 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 JCOMP input that provides reset control and the ability to share the TAP.
- 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 483* for a list of supported instructions.
- Sharing of the TAP with other TAP controllers via ACCESS_AUX_TAP_x instructions.
- A TEST_CTRL register (for cut2/3 only)
- 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.

30.1.3 Modes of operation

The JTAGC block uses JCOMP and 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.

30.1.3.1 Reset

The JTAGC block is placed in reset when either power-on reset is asserted, JCOMP is negated, 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 or setting JCOMP to a value other than the value required to enable the JTAGC block 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

30.1.3.2 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, 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 functionality of each test mode is explained in more detail in *Section 30.4.4, JTAGC Block Instructions*.

30.1.3.3 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.

30.2 External signal description

30.2.1 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 481*.

Table 481. JTAG Signal Properties

Name	I/O	Function	Reset State	Pull(1)
TCK	Input	Test Clock	-	Up
TDI	Input	Test Data In	-	Up
TDO	Output	Test Data Out	High Z(2)	-
TMS	Input	Test Mode Select	-	Up
JCOMP	Input	JTAG Compliancy	-	Down

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.

30.2.2 Detailed Signal Descriptions

This section describes each of the signals listed in *Table 481* in more detail.

30.2.2.1 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.

30.2.2.2 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.

30.2.2.3 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 30.4.3, 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.

30.2.2.4 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.

30.2.2.5 JCOMP - JTAG Compliancy

The JCOMP signal provides IEEE 1149.1-2001 compatibility and provides the ability to share the TAP. The JTAGC TAP controller is enabled when JCOMP is set to the JTAGC enable encoding , otherwise the JTAGC TAP controller remains in reset.

30.3 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.

30.3.1 Register descriptions

The JTAGC block registers are described in this section.

30.3.1.1 Instruction Register

The JTAGC block uses a 5-bit instruction register as shown in *Table 610*. 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 610. 5-Bit Instruction Register

30.3.1.2 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS 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.

30.3.1.3 Device Identification Register

The device identification register, shown in *Figure 611*, 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.

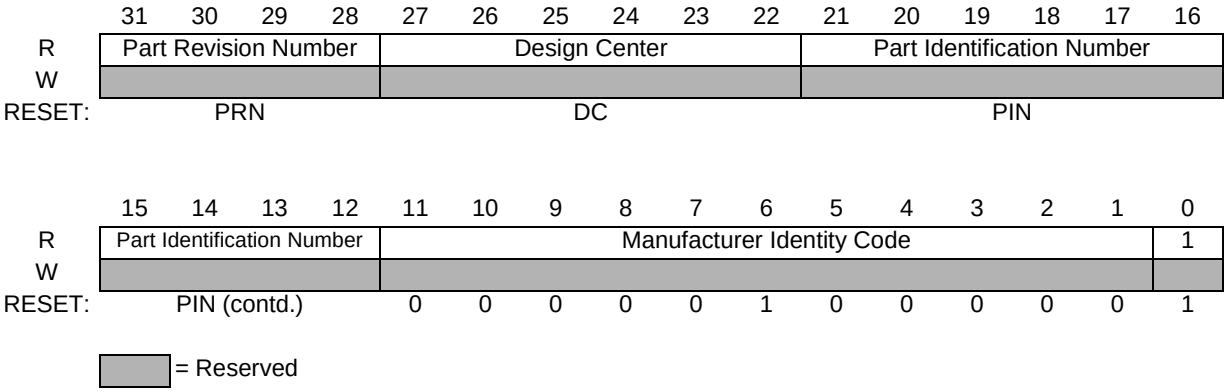


Figure 611. Device Identification Register

PRN — Part Revision Number

Bits [31:28] contain the revision number of the part.

cut1: 0x0

cut2: 0x0

cut3: 0x1

cut3.1: 0x2

DC — Design Center

Bits [27:22] indicate the design center. The default value is 0x2B.

PIN — Part Identification Number

Bits [21:12] contain the part number of the device.

cut1: 0x2A2

cut2: 0x2A3

cut3: 0x2A3

cut3.1: 0x2A3

MIC — Manufacturer Identity Code

Bits [11:1] contain the reduced Joint Electron Device Engineering Council (JEDEC) ID. The default value is 0x020.

Bit [0] — IDCODE Register ID

Bit [0] identifies this register as the device identification register and not the bypass register

30.3.1.4 TEST_CTRL register (cut2/3 only)

The TEST_CTRL register (supported on cut2/3 only) is a K-bit shift register path from TDI to TDO selected when the ENABLE_TEST_CTRL instruction is active. The size (K) of the TEST_CTRL register is defined by the TEST_CTRL_SIZE parameter. The TEST_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.

Figure 612. TEST_CTRL register (cut2/3 only)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
R	PSTAT_SEL	TDO_SEL	RDY_SEL	RDY_ENABLE	1(1)	0(1)	1(1)	1(1)	1(1)	0(1)	1(1)	0(1)	0(1)	0(1)	0(1)
W															
Reset	0	0	0	0	1	0	1	1	1	0	1	0	0	0	0

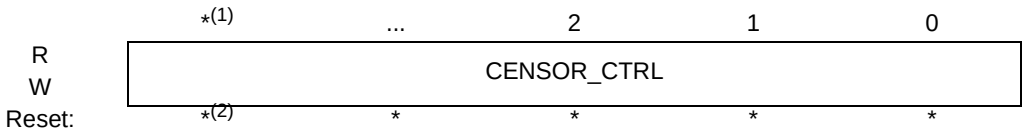
1. This bit is reserved. The defined value will be used for future compatibility.

Table 482. TEST_CTRL field descriptions

Field	Description
PSTAT_SEL	**Processor Status Select** Selects whether the Core_0 or the Core_1 processor status outputs will appear on the Nexus MDO pins when Processor Status Mode is enabled in the NPC Port Configuration Register. 0 Core_0 PSTAT outputs selected. 1 Core_1 PSTST outputs selected.
TDO_SEL	**TDO Select** Selects whether the Core_0 or Core_1 TDO output will be observable. 0 Core_0 TDO output selected. 1 Core_1 TDO output selected.
RDY_SEL	**$\overline{\text{RDY}}$ Select** Selects whether the Nexus AUX interface $\overline{\text{RDY}}$ pad output buffer is fed by Core_0 or by Core_1 RDY_B signal 0 Core_0 provides $\overline{\text{RDY}}$ signal 1 Core_1 provides $\overline{\text{RDY}}$ signal
RDY_ENABLE	**$\overline{\text{RDY}}$ Enable** Enables $\overline{\text{RDY}}$ functionality for corresponding GPIO pin. Setting this bit overrides SIUL output muxing configuration for the corresponding pad. 0 Disables $\overline{\text{RDY}}$ functionality (Output mux for the pad is controlled by SIUL) 1 Enables $\overline{\text{RDY}}$ functionality (Output mux for the pad is controlled by JTAGC)

30.3.1.5 CENSOR_CTRL Register

The CENSOR_CTRL register is a 65-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 65'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 65 bits .
2. The reset value of CENSOR_CTRL is 65'b0 .

Figure 613. CENSOR_CTRL Register

CENSOR_CTRL - Censorship Control

The CENSOR_CTRL bits are used to control chiptop censorship functions.

30.3.1.6 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 30.4.5, Boundary Scan.* The size of the boundary scan register and bit ordering is device-dependent and can be found in the device BSDL file.

30.4 Functional Description

30.4.1 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.

30.4.2 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. Ownership of the port is determined by the value of the JCOMP signal and the currently loaded instruction. For more detail on TAP sharing via JTAGC instructions refer to *Section 30.4.4.6, 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 614*. This applies for the instruction register, test data registers, and the bypass register.

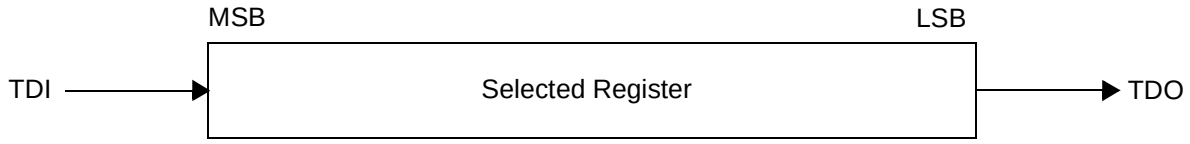
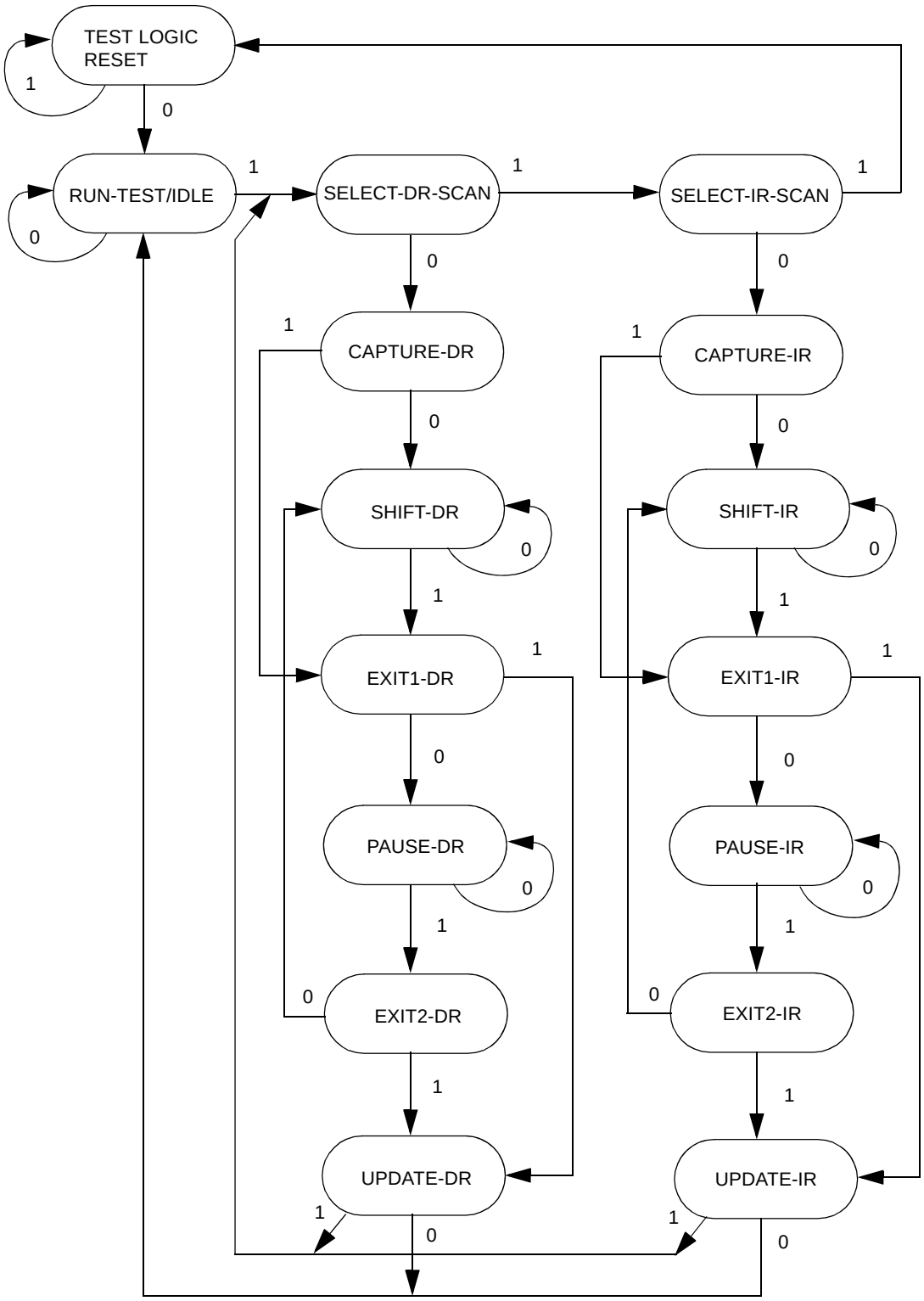


Figure 614. Shifting Data Through a Register

30.4.3 TAP Controller State Machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. *Figure 615* 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 615* 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 615. IEEE 1149.1-2001 TAP Controller Finite State Machine

30.4.3.1 Enabling the TAP Controller

The JTAGC TAP controller is enabled by setting JCOMP to a logic 1 value.

30.4.3.2 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.

30.4.4 JTAGC Block Instructions

The JTAGC block implements the IEEE 1149.1-2001 defined instructions listed in *Table 483*. 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 483. JTAG Instructions

Instruction	Code[4:0]	Instruction Summary
IDCODE	00001	Selects device identification register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_x	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
Factory debug reserved	00101, 00110, 01010, 00111	Intended for factory debug only
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller
ACCESS_AUX_TAP_CORE_0	10001	Enables access to Core_0 TAP controller only
ACCESS_AUX_TAP_NXSS_LSM	10110	Enables access to the NXSS modules in LSM (cut2/3 only)
ACCESS_AUX_TAP_NXSS_0	10111	Enables access to the NXSS_0 module (cut2/3 only)
ACCESS_AUX_TAP_NXSS_1	11000	Enables access to the NXSS_1 module (cut2/3 only)
ACCESS_AUX_TAP_CORE_1	11001	Enables access to Core_1 TAP controller only
ACCESS_AUX_TAP_LSM	11010	Enables access to Core TAP controllers in LS mode. Both cores receive TDI input data.

Table 483. JTAG Instructions (continued)

Instruction	Code[4:0]	Instruction Summary
ACCESS_AUX_TAP_MULTI	11100	Enables access to and serializes the Core_0 and Core_1 TAPs
BYPASS	11111	Selects bypass register for data operations
Reserved(1)	All other opcodes	Decoded to select bypass register

1. The manufacturer reserves the right to change the decoding of reserved instruction codes in the future

30.4.4.1 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.

30.4.4.2 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 instruction to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the Shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the Update-DR state. The data is applied to the external output pins by the EXTEST instruction. System operation is not affected.

30.4.4.3 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.

30.4.4.4 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.

30.4.4.5 ENABLE_CENSOR_CTRL Instruction

The ENABLE_CENSOR_CTRL instruction selects the CENSOR_CTRL register for connection as the shift path between TDI and TDO.

30.4.4.6 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.

30.4.4.7 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.

30.4.5 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.

30.5 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:

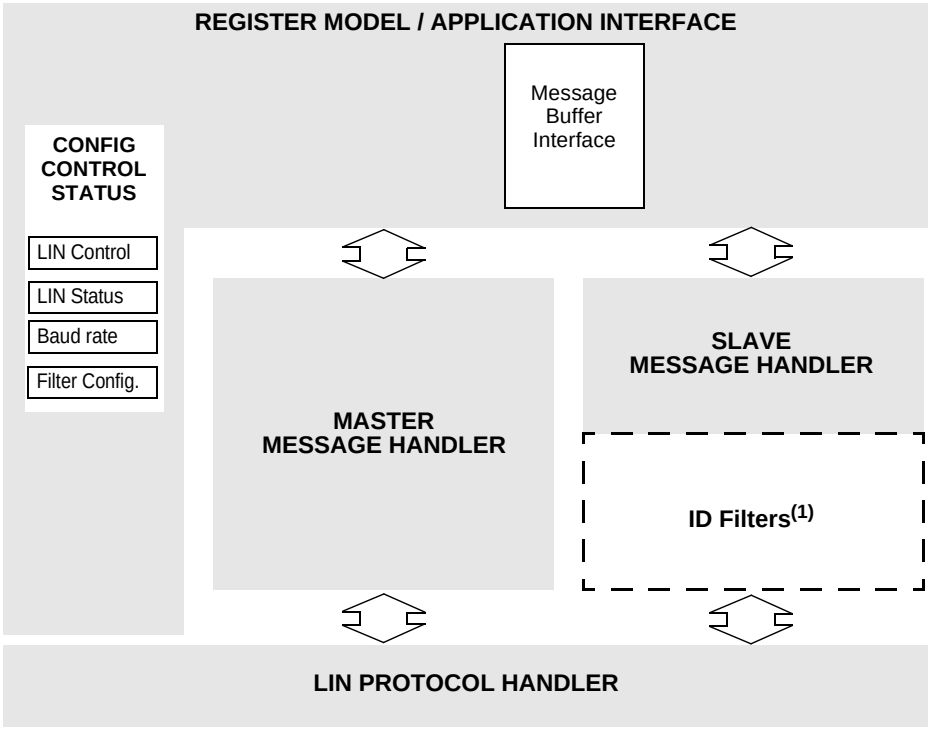
1. Set the JCOMP signal to the JTAGC enable value, thereby enabling the JTAGC TAP controller
2. Load the appropriate instruction for the test or action to be performed

31 LIN Controller (LINFlexD)

31.1 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 616 shows the LINFlexD block diagram.



1 Filter activation optional

Figure 616. LINFlexD block diagram

31.2 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

31.2.1 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 internal RC oscillator as clock source
- Identifier filters for autonomous message handling in Slave mode

31.2.2 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

31.3 The LIN protocol

The LIN (Local Interconnect Network) is a serial communication protocol. The topology of a LIN network is shown in *Figure 617*. A LIN network consists of:

- One master task
- Several slave tasks
- 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.

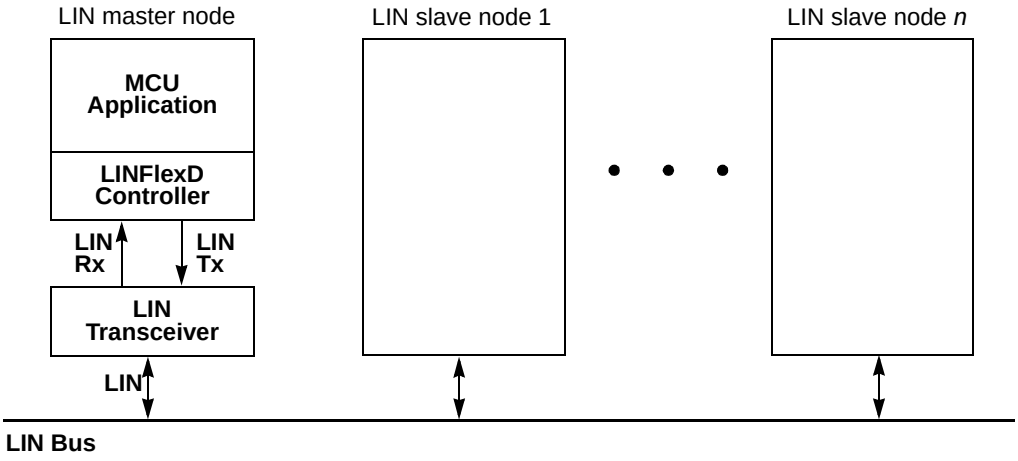


Figure 617. LIN network topology

31.3.1 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)

31.3.2 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 618*.

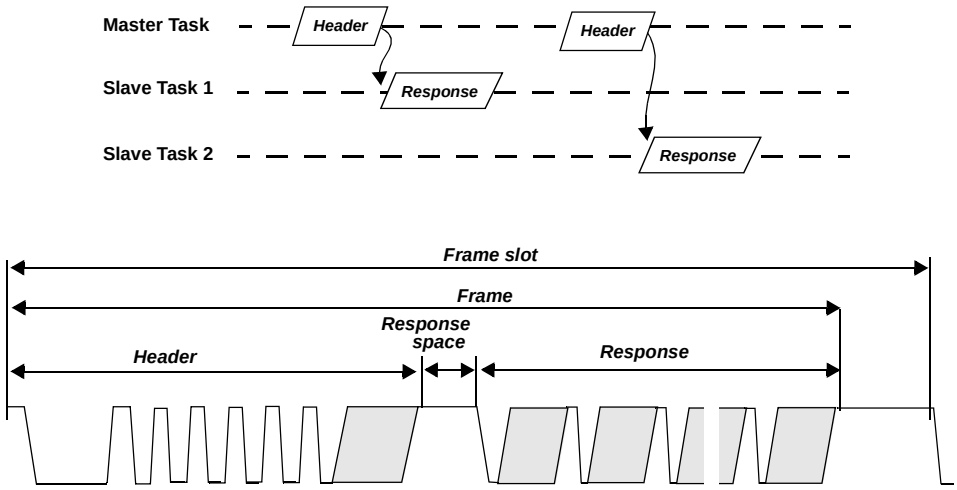


Figure 618. LIN frame structure

31.3.3 LIN header

The header consists of:

- A break field (described in *Section 31.3.3.1, Break field*)
- A sync pattern (described in *Section 31.3.3.2, Sync pattern*)
- An identifier (described in *Section 31.3.4.2, Identifier*)

The slave task associated with the identifier provides the response.

31.3.3.1 Break field

The break field, shown in *Figure 619*, 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 619. Break field

31.3.3.2 Sync pattern

The sync pattern is a byte consisting of alternating dominant and recessive bits as shown in *Figure 620*. It forms a data value of 0x55.

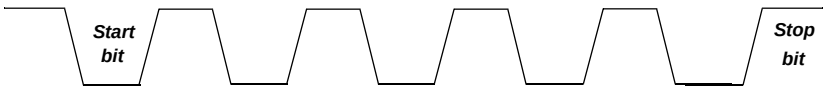


Figure 620. Sync pattern

31.3.4 Response

The response consists of:

- A data field (described in *Section 31.3.4.1, Data field*)
- A checksum (described in *Section 31.3.4.3, Checksum*)

The slave task interested in the data associated with the identifier receives the response and verifies the checksum.

31.3.4.1 Data field

The structure of the data field transmitted on the LIN bus is shown in *Figure 621*. 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.

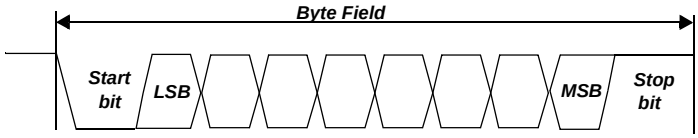


Figure 621. Structure of the data field

31.3.4.2 Identifier

The identifier, shown in *Figure 622*, 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)

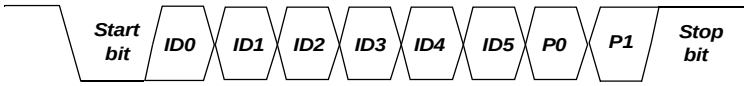


Figure 622. Identifier

31.3.4.3 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.

31.4 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.

31.5 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 623*. The software instructs LINFlexD to enter Initialization mode or Sleep mode by setting LINCR1[INIT] or LINCR1[SLEEP], respectively.

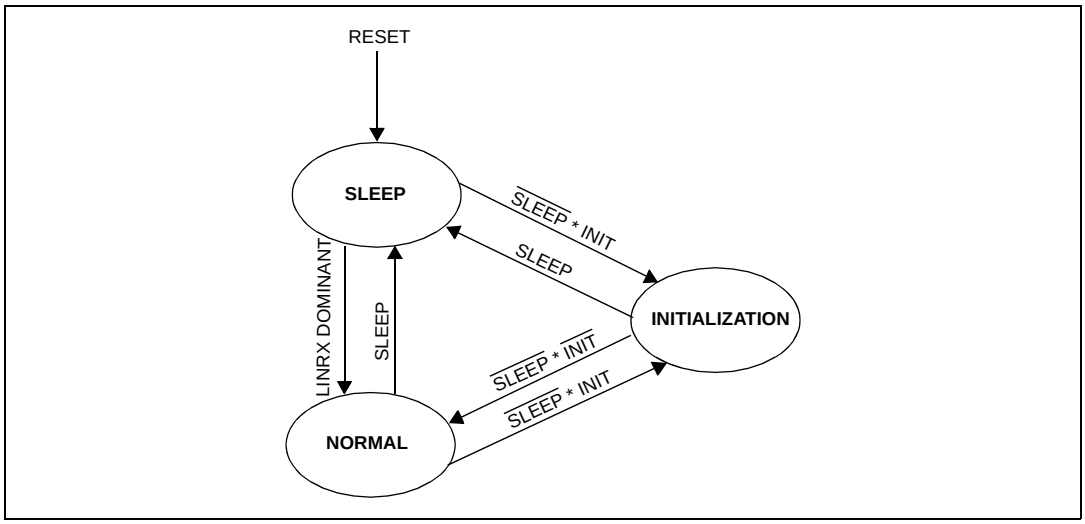


Figure 623. LINFlexD controller operating modes

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.

31.6 Controller-level operating modes

31.6.1 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

31.6.2 Normal mode

After initialization is complete, the software must clear LINCR1[INIT] to put the LINFlexD controller into Normal mode.

31.6.3 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.

31.7 LIN modes

31.7.1 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.

31.7.1.1 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].

31.7.1.2 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.

31.7.1.3 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.

31.7.1.4 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 484 lists the errors detected in Master mode and the LINFlexD controller's response to these errors.

Table 484. 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 31.12.1, 8-bit timeout counter,* for more details	

31.7.1.5 Overrun

Once the messag buffer is full (LINSR[RMB] = 1), the next valid message reception leads to an overrun and a 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.

31.7.2 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.

31.7.2.1 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 625*).

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 31.7.3, Slave mode with identifier filtering*) in order to manage several identifiers with one filter only.

31.7.2.2 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 setting 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 625*).

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 31.7.3, Slave mode with identifier filtering*) in order to manage several identifiers with one filter only.

31.7.2.3 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.

31.7.2.4 Error detection and handling

Table 485 lists the errors detected in Slave mode and the LINFlexD controller's response to these errors.

Table 485. 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

31.7.2.5 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.

31.7.2.6 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.

31.7.2.7 Overrun

Once the message buffer is full (LINSR[RMB] = 1), the next valid message reception leads to an overrun and a message is lost. The hardware signals the overrun condition by setting the BOF bit in the LINESR. Which message is lost depends on the buffer lock function control bit RBLM.

- If the buffer lock function control bit is cleared (LINCR1[RBLM] = 0) the old message in the buffer will be overwritten by the most recent message.
- If buffer lock function control bit is set (LINCR1[RBLM] = 1) the most recent message is discarded, and the oldest message is available in the buffer.
- If buffer is not released (LINSR[RMB] = 1) before reception of next Identifier and if RBLM is set then ID along with the data is discarded.

31.7.3 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.

31.7.3.1 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 486 describes the two available filter submodes.

Table 486. 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 624*.

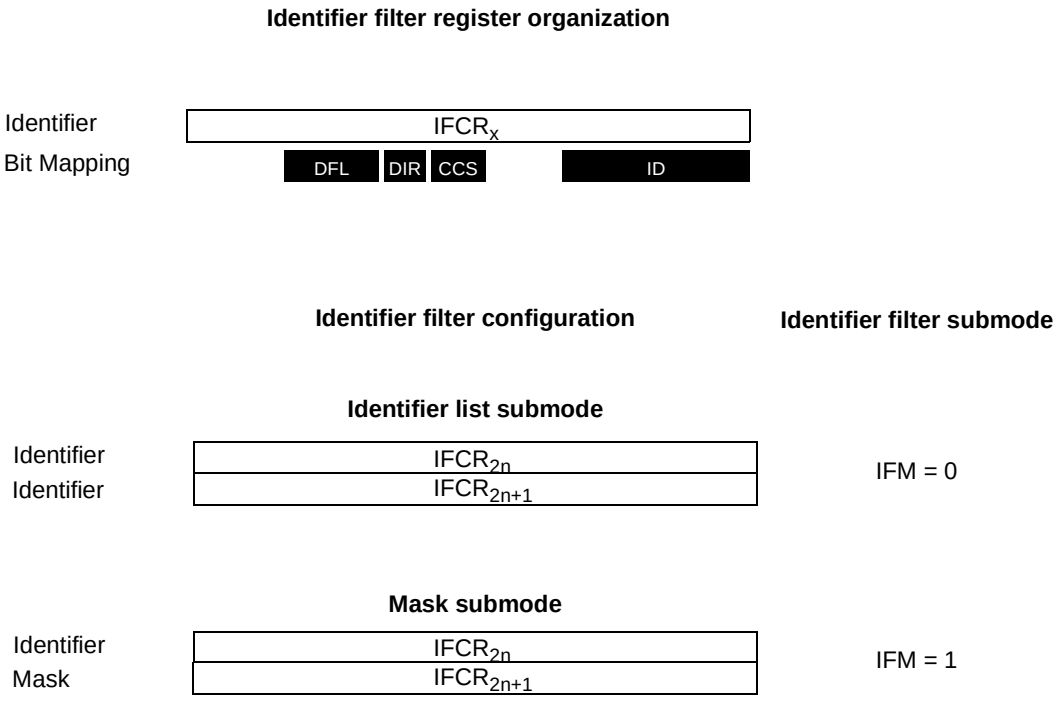


Figure 624. Filter configuration - register organization

31.7.3.2 Identifier filter submode configuration

The identifier filters are configured in the IFCR registers. To configure an identifier filter the filter must first be deactivated by clearing 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 487* and *Figure 625*.

Table 487. 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

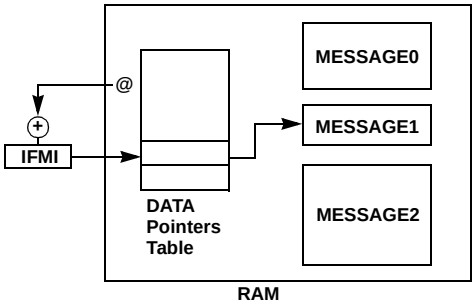


Figure 625. Identifier match index

31.7.4 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 $f_{ipg_clock_lin}$ deviation up to 14%, as specified in the LIN standard.

This mode is similar to Slave mode as described in *Section 31.7.2, 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.

31.7.4.1 Automatic resynchronization method

When automatic resynchronization is enabled, after each LIN break, the time duration between five falling edges on RDI is sampled on $f_{ipg_clock_lin}$ as shown in *Figure 626*. 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.

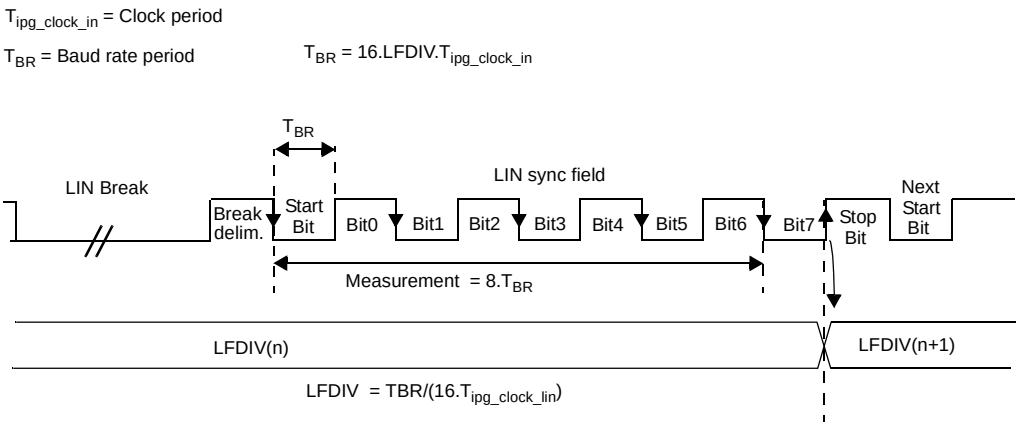


Figure 626. LIN sync field measurement

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.

31.7.4.2 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.

31.8 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.

31.8.1 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 627*.

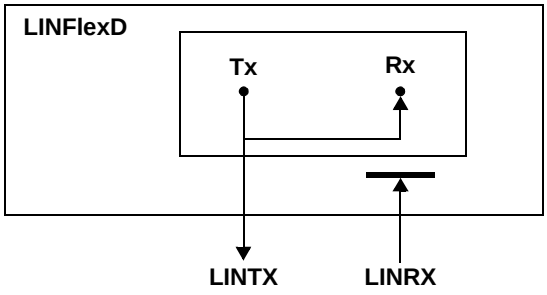


Figure 627. LINFlexD in Loop Back mode

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.

31.8.2 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 628*.

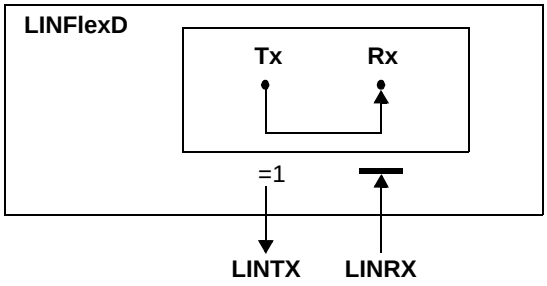


Figure 628. LINFlexD in Self Test mode

31.9 UART mode

The main features of UART mode are presented in *Section 31.2.2, UART mode features*.

31.9.1 Data frame structure

31.9.1.1 8-bit data frame

The 8-bit UART data frame is shown in *Figure 629*. 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.

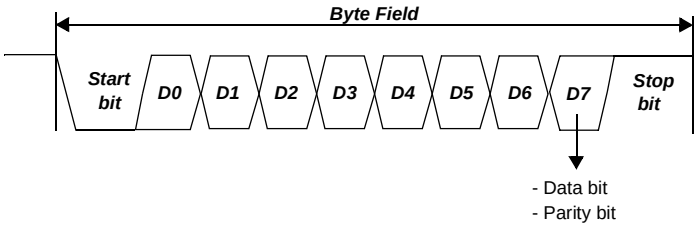


Figure 629. UART mode 8-bit data frame

31.9.1.2 9-bit data frame

The 9-bit UART data frame is shown in *Figure 630*. 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.

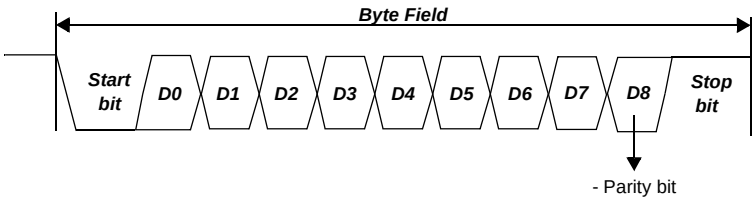


Figure 630. UART mode 9-bit data frame

31.9.1.3 16-bit data frame

The 16-bit UART data frame is shown in *Figure 631*. 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.

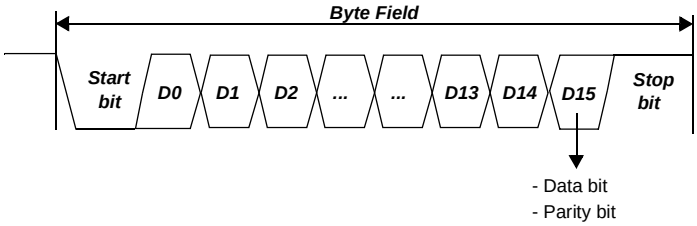


Figure 631. UART mode 16-bit data frame

31.9.1.4 17-bit data frame

The 17-bit UART data frame is shown in *Figure 632*. 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.

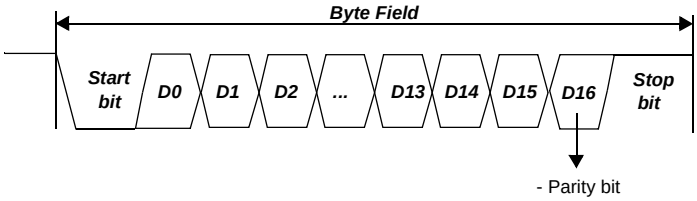


Figure 632. UART mode 17-bit data frame

31.9.2 Buffer

The 8-byte buffer is divided into two parts — one for receiver and one for transmitter — as shown in *Table 488*.

Table 488. 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.

31.9.3 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 500*).

The Transmit buffer size is as follows:

- 4 bytes when UARTCR[WL1] = 0
- 2 half-words when UARTCR[WL1] = 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 489*.

Table 489. 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 489. 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 WL1 and WL0 bits of the UARTCR register. In UART FIFO mode (UARTCR[TFBM] = 1),any read operation causes an IPS transfer error.

31.9.4 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 490*.

Table 490. 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 490. 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 491 lists some common scenarios, controller responses, and suggestions when the LINFlexD controller is acting as a UART receiver.

Table 491. 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.

Table 491. UART receiver scenarios (continued)

Scenario	Responses and suggestions
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 LINCR[RBLM]). – An interrupt is generated if LINIER[BOIE] is set.

31.10 Memory map and register description

Table 492 shows the LINFlexD memory/register map. See the device memory map for the base address.

Table 492. LINFlexD memory map

Address offset	Register description	Location
0x00	LIN control register 1 (LINCR1)	*on page 1004*
0x04	LIN interrupt enable register (LINIER)	*on page 1007*
0x08	LIN status register (LINSR)	*on page 1009*
0x0C	LIN error status register (LINESR)	*on page 1012*
0x10	UART mode control register (UARTCR)	*on page 1013*
0x14	UART mode status register (UARTSR)	*on page 1016*
0x18	LIN timeout control status register (LINTCSR)	*on page 1018*
0x1C	LIN output compare register (LINOCR)	*on page 1019*
0x20	LIN timeout control register (LINTOCR)	*on page 1020*
0x24	LIN fractional baud rate register (LINFBRR)	*on page 1021*
0x28	LIN integer baud rate register (LINIBRR)	*on page 1021*
0x2C	LIN checksum field register (LINCFR)	*on page 1022*
0x30	LIN control register 2 (LINCR2)	*on page 1023*
0x34	Buffer identifier register (BIDR)	*on page 1024*
0x38	Buffer data register least significant (BDRL)	*on page 1025*
0x3C	Buffer data register most significant (BDRM)	*on page 1026*
0x40	Identifier filter enable register (IFER)	*on page 1027*
0x44	Identifier filter match index (IFMI)	*on page 1027*
0x48	Identifier filter mode register (IFMR)	*on page 1028*
0x4C–0x88	Identifier filter control registers 0–15 (IFCR0–IFCR15)	*on page 1029*

Table 492. LINFlexD memory map (continued)

Address offset	Register description	Location
0x8C	Global control register (GCR)	*on page 1030*
0x90	UART preset timeout register (UARTPTO)	*on page 1032*
0x94	UART current timeout register (UARTCTO)	*on page 1032*
0x98	DMA Tx enable register (DMATXE)	*on page 1034*
0x9C	DMA Rx enable register (DMARXE)	*on page 1034*

31.10.1 LIN control register 1 (LINCR1)

Figure 633. 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	CCD (1)	CFD (1)	LASE(1)	AWUM(1)	MBL(1)				BF(1)	SFTM(1)	LBKM(1)	MME(1)	SBDT(1)	RBLM(1)	SLEEP	INIT
W																
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

Table 493. LINCR1 field descriptions

Field	Description
CCD	Checksum Calculation disable This bit is used to disable the checksum calculation (see *Table 494*). 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 494*). 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.*

Table 493. LINCR1 field descriptions (continued)

Field	Description
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 the SLEEP bit of the LINCR register. 1: The Sleep mode is exited automatically by hardware on RX dominant state detection. The SLEEP bit of the LINCR register is cleared by hardware whenever WUF bit in LINSR is set. *Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.*
MBL	LIN Master Break Length These bits indicate the Break length in Master mode (see *Table 495*). *Note: These bits can be written in Initialization mode only. They are read-only in Normal or Sleep mode.*
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.* *Note: 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 31.8.2, 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 31.8.1, 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.*

Table 493. LINCR1 field descriptions (continued)

Field	Description
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 the AWUM bit in LINCR1 and the WUF bit in LINSR are set (see *Table 496*).
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 496*).

Table 494. 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 495. 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 496. Operating mode selection

SLEEP	INIT	Operating mode
1	0	Sleep (reset value)
x	1	Initialization
0	0	Normal

31.10.2 LIN interrupt enable register (LINIER)

Figure 634. 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	SZIE	OCIE	BEIE	CEIE	HEIE	0	0	FEIE	BOIE	LSIE	WUIE	DBFIE	DBEIETOIE	DRIE	DTIE	HRIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 497. LINIER field descriptions

Field	Description
SZIE	Stuck at Zero Interrupt Enable 0: No interrupt when SZF bit in LINESR or UARTSR is set 1: Interrupt generated when SZF bit in LINESR or UARTSR is set
OCIE	Output Compare Interrupt Enable 0: No interrupt when OCF bit in LINESR or UARTSR is set 1: Interrupt generated when OCF bit in LINESR or UARTSR is set
BEIE	Bit Error Interrupt Enable 0: No interrupt when BEF bit in LINESR is set 1: Interrupt generated when BEF bit in LINESR is set
CEIE	Checksum Error Interrupt Enable 0: No interrupt on Checksum error 1: Interrupt generated when checksum error flag (CEF) is set in LINESR
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

Table 497. LINIER field descriptions (continued)

Field	Description
FEIE	Framing Error Interrupt Enable 0: No interrupt on Framing error 1: Interrupt generated on Framing error
BOIE	Buffer Overrun Interrupt Enable 0: No interrupt on Buffer overrun 1: Interrupt generated on Buffer overrun
LSIE	LIN State Interrupt Enable 0: No interrupt on LIN state change 1: Interrupt generated on LIN state change This interrupt can be used for debugging purposes. It has no status flag but is reset when writing '1111' into the LIN state bits in the LINSR register.
WUIE	Wake-up Interrupt Enable 0: No interrupt when WUF bit in LINSR or UARTSR is set 1: Interrupt generated when WUF bit in LINSR or UARTSR is set
DBFIE	Data Buffer Full Interrupt Enable 0: No interrupt when buffer data register is full 1: Interrupt generated when data buffer register is full
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
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
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

31.10.3 LIN status register (LINSR)

Figure 635. 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
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Table 498. 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.

Table 498. LINSR field descriptions (continued)

Field	Description
RBSY	Receiver Busy Flag 0: Receiver is Idle 1: Reception ongoing Note: In Slave mode, after header reception, if DIR bit in BIDR is reset and reception starts then this bit is set. In this case, user cannot set DTRQ bit in LINCR2.
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 WUIE bit in LINIER is set.
DBFF	Data Buffer Full Flag This bit is set by hardware and indicates the buffer is full. It is set only when receiving extended frames (DFL > 7). This bit must be cleared by software. It is reset by hardware in Initialization mode.
DBEF	Data Buffer Empty Flag This bit is set by hardware and indicates the buffer is empty. It is set only when transmitting extended frames (DFL > 7). This bit must be cleared by software, once buffer has been filled again, in order to start transmission. This bit is reset by hardware in Initialization mode.
DRF	Data Reception Completed Flag This bit is set by hardware and indicates the data reception is completed. This bit must be cleared by software. It is reset by hardware in Initialization mode. *Note: This flag is not set in case of bit error or framing error.*
DTF	Data Transmission Completed Flag This bit is set by hardware and indicates the data transmission is completed. This bit must be cleared by software. It is reset by hardware in Initialization mode. *Note: This flag is not set in case of bit error if IOBE bit is reset.*
HRF	Header Reception Flag This bit is set by hardware and indicates a valid header reception is completed. This bit must be cleared by software. This bit is reset by hardware in Initialization mode and at end of completed or aborted frame. *Note: If filters are enabled, this bit is set only when identifier software filtering is required, that is to say:* – all filters are inactive and BF bit in LINCR1 is set – no match in any filter and BF bit in LINCR1 is set – TX filter match

31.10.4 LIN error status register (LINESR)

Figure 636. 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

Table 499. 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).

Table 499. LINESR field descriptions (continued)

Field	Description
IDPEF	Identifier Parity Error Flag This bit is set by hardware and indicates that a Identifier Parity error occurred. *Note: Header interrupt is triggered when SFEF or BDEF or IDPEF bit is set and HEIE bit in LINIER is set.*
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that 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.

31.10.5 UART mode control register (UARTCR)

Figure 637. 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(2)	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).

Table 500. 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 500. 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	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	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.

31.10.6 UART mode status register (UARTSR)

Figure 638. 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
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 501. 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

Table 501. UARTSR field descriptions (continued)

Field	Description
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
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.

Table 501. UARTSR field descriptions (continued)

Field	Description
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.

31.10.7 LIN timeout control status register (LINTCSR)

Figure 639. 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).

Table 502. 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 not reset to Idle on timeout event 1: LIN state machine reset to Idle on timeout event This bit can be set/cleared in Initialization mode only.

Table 502. LINTCSR field descriptions (continued)

Name	Description
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 and software cannot modify it.
CNT	Counter Value These bits indicate the LIN Timeout counter value.

31.10.8 LIN output compare register (LINOCR)

Figure 640. 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)

Table 503. 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].

31.10.9 LIN timeout control register (LINTOCR)

Figure 641. 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.

Table 504. 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.

31.10.10 LIN fractional baud rate register (LINFBRR)

Figure 642. LIN timeout control register (LINTOCR)

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.

Table 505. 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.

31.10.11 LIN integer baud rate register (LINIBRR)

Figure 643. 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).

Table 506. LINIBRR field descriptions

Field	Description
DIV_M	LFDIV mantissa These bits define the LINFlexD divider (LFDIV) mantissa value (see *Table 507*). This register can be written in Initialization mode only.

Table 507. Integer baud rate selection

DIV_M	Mantissa
0x0	LIN clock disabled
0x1	1
...	...
0xFFFFE	1048574
0xFFFFF	1048575

31.10.12 LIN checksum field register (LINCFR)

Figure 644. 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

Table 508. 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 494*.

31.10.13 LIN control register 2 (LINCR2)

Figure 645. 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)	WURQ3	DDRQ3	DTRQ3	ABRQ3	HTRQ3	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

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 509. 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.

Table 509. LINCR2 field descriptions (continued)

Field	Description
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.
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.

31.10.14 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.

Figure 646. Buffer identifier register (BIDR)

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

Table 510. 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.
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.

31.10.15 Buffer data register least significant (BDRL)

Figure 647. 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

Table 511. 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

Table 511. BDRL field descriptions (continued)

Field	Description
DATA1	Data Byte 1 Data byte 1 of the data field
DATA0	Data Byte 0 Data byte 0 of the data field

31.10.16 Buffer data register most significant (BDRM)

Figure 648. Buffer data register most significant (BDRM)

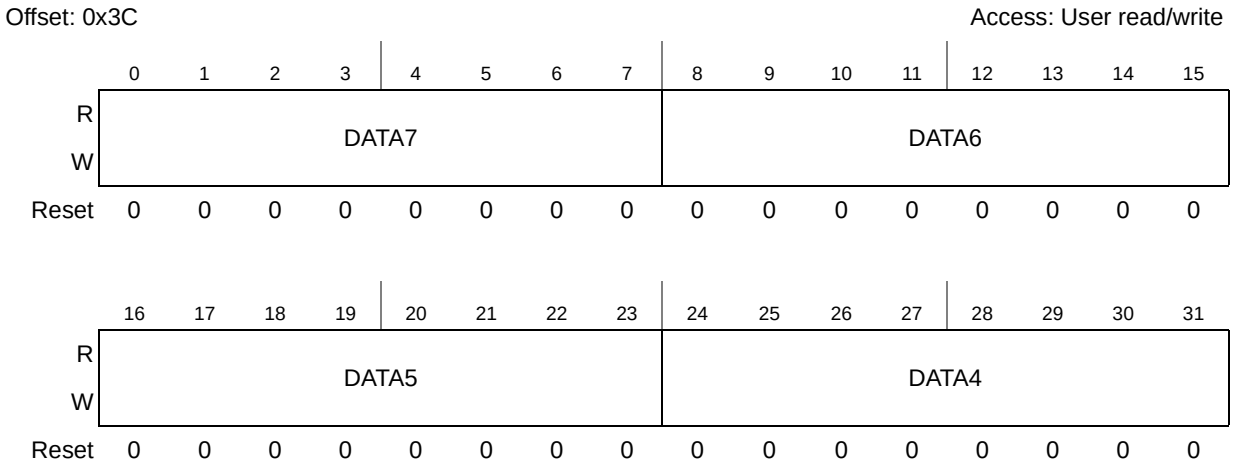


Table 512. BDRM field descriptions

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

31.10.17 Identifier filter enable register (IFER)

Figure 649. 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	FACT(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).

Table 513. 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.

31.10.18 Identifier filter match index (IFMI)

Figure 650. 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

Table 514. 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 31.7.2, Slave mode,* for more details). When no filter matches, IFMI = 0. When Filter n is matching, IFMI = n + 1.

31.10.19 Identifier filter mode register (IFMR)

Figure 651. 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(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).

Table 515. 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*).

Table 516. 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).

Table 516. IFMR[IFM] configuration (continued)

Bit	Value	Result
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).

31.10.20 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 517*.

Table 517. 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].

Figure 652. Identifier filter control registers (IFCR0–IFCR15)

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).

Table 518. 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.

31.10.21 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.

Figure 653. Global control register (GCR)

Offset: 0x8C 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).

Table 519. 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.
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 always reads “0”.

31.10.22 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 31.10.23, UART current timeout register (UARTCTO)*.

Figure 654. UART preset timeout register (UARTPTO)

Offset: 0x90 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

Table 520. UARTPTO field descriptions

Field	Description
PTO	Preset value of the timeout counter Do not set PTO = 0 (otherwise, UARTSR[TO] would immediately be set).

31.10.23 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 31.10.22, 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

Figure 655. UART current timeout register (UARTCTO)

Offset: 0x94 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

Table 521. 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.

31.10.24 DMA Tx enable register (DMATXE)

This register enables the DMA Tx interface.

Figure 656. DMA Tx enable register (DMATXE)

Offset: 0x98 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	DTE 15	DTE 14	DTE 13	DTE 12	DTE 11	DTE 10	DTE 9	DTE 8	DTE 7	DTE 6	DTE 5	DTE 4	DTE 3	DTE 2	DTE 1	DTE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 522. 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.*

31.10.25 DMA Rx enable register (DMARXE)

This register enables the DMA Rx interface.

Figure 657. DMA Rx enable register (DMARXE)

Offset: 0x9C 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	DRE 15	DRE 14	DRE 13	DRE 12	DRE 11	DRE 10	DRE 9	DTE 8	DRE 7	DRE 6	DRE 5	DRE 4	DRE 3	DRE 2	DRE 1	DRE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 523. 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.*

31.11 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.

31.11.1 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 658*.

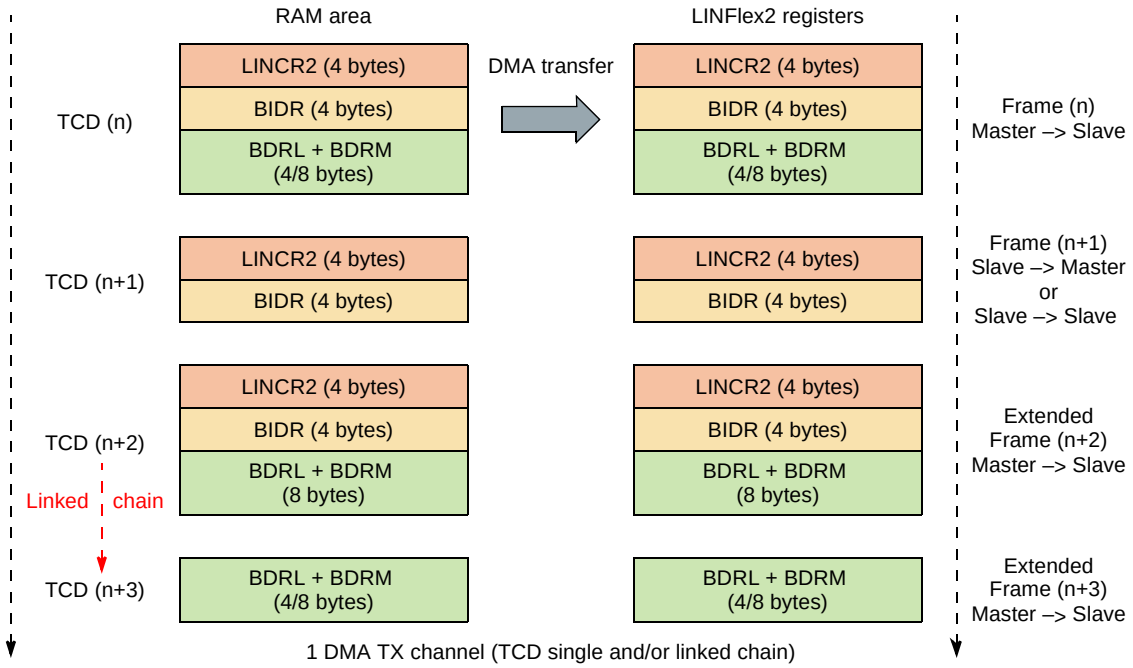


Figure 658. TCD chain memory map (master node, TX mode)

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 524*.

Table 524. 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 659*. The DMA TX FSM will move to IDLE state immediately at next clock edge if DMATXE[0] = 0.

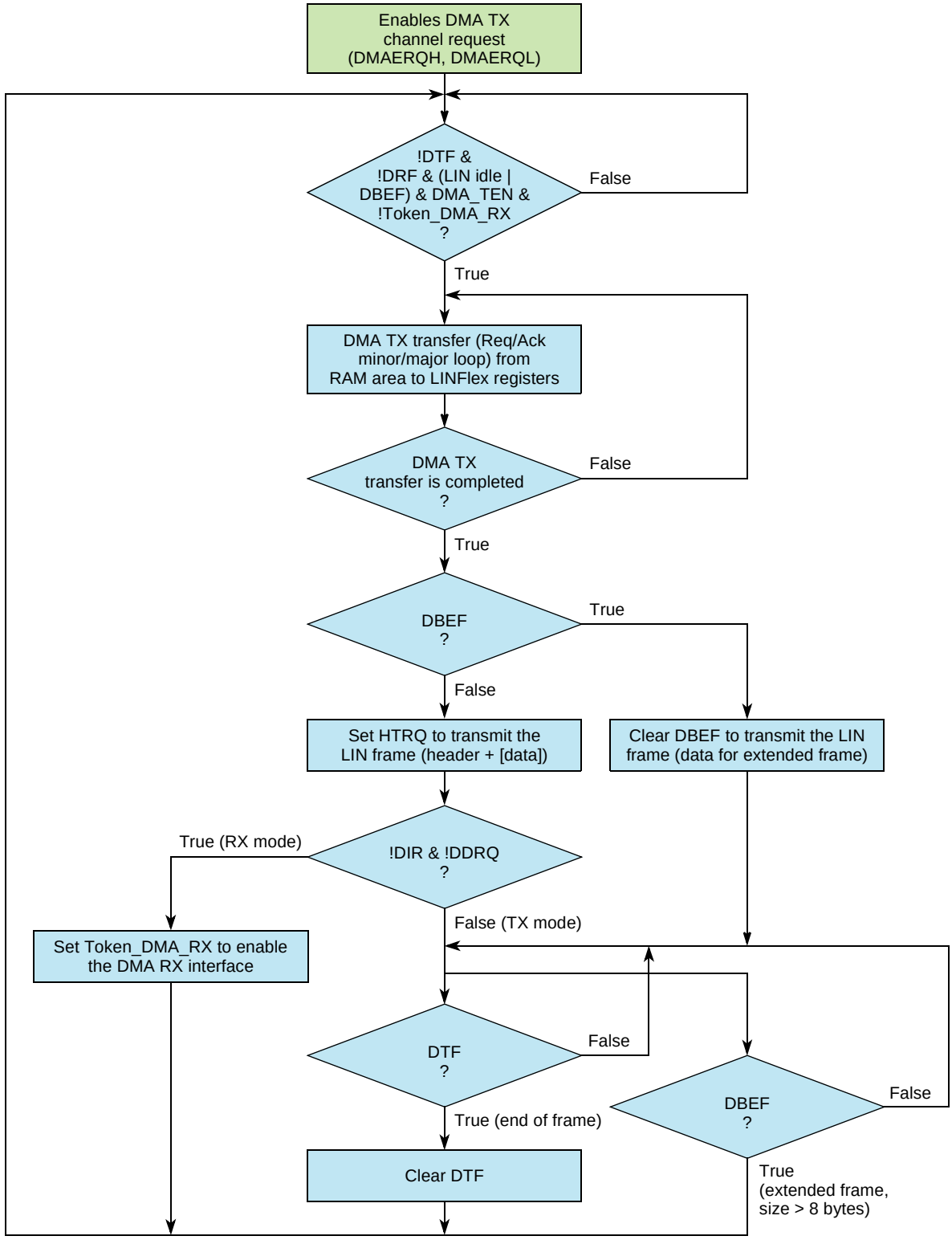


Figure 659. FSM to control the DMA TX interface (master node)

The TCD settings (word transfer) are shown in *Table 525*. All other TCD fields are equal to 0. TCD settings based on half-word or byte transfers are allowed.

Table 525. 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

31.11.2 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 660*.

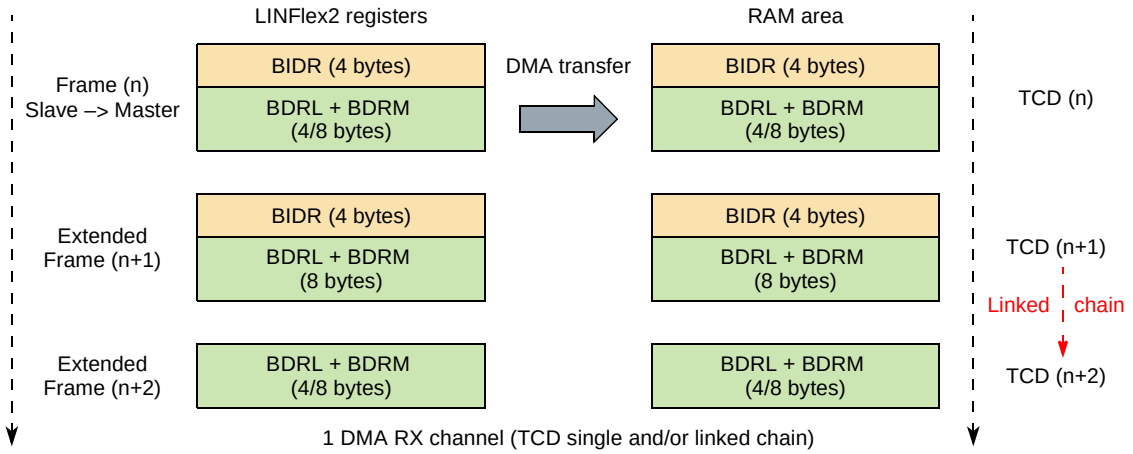


Figure 660. TCD chain memory map (master node, RX mode)

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 661*. The DMA RX FSM will move to IDLE state immediately at next clock edge if DMARXE[0]=0.

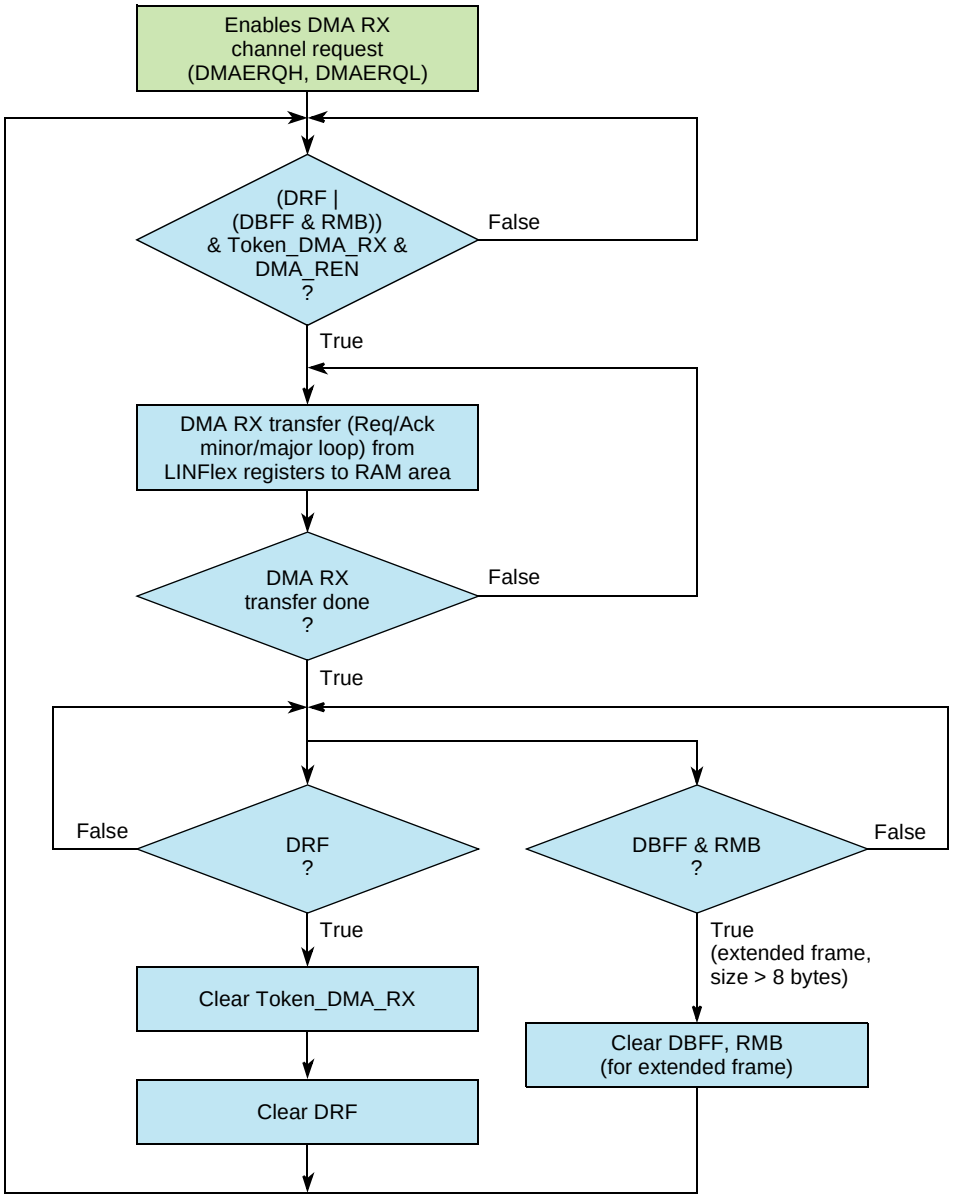


Figure 661. FSM to control the DMA RX interface (master node)

The TCD settings (word transfer) are shown in *Table 526*. All other TCD fields are equal to 0. TCD settings based on half-word or byte transfer are allowed.

Table 526. 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

31.11.3 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 662*.

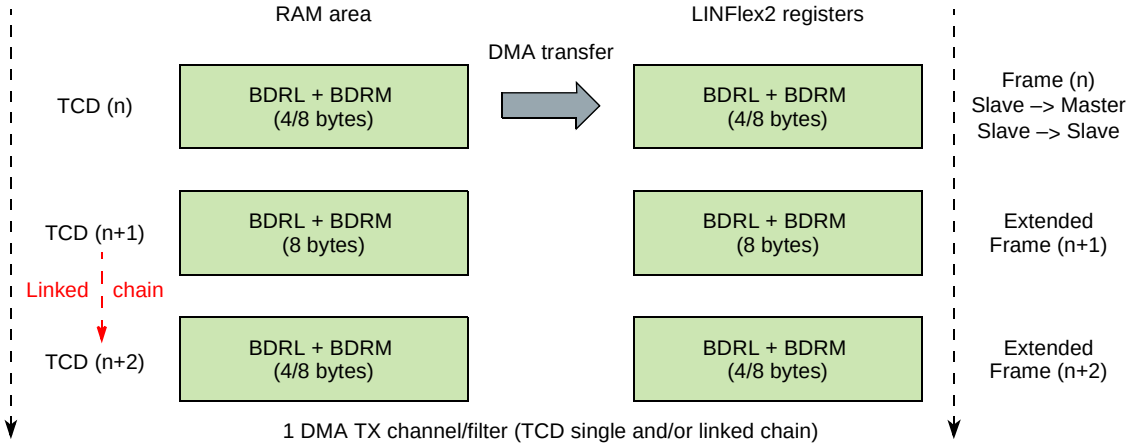


Figure 662. TCD chain memory map (slave node, TX mode)

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 527*.

Table 527. 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 663*. DMA TX FSM will move to idle state if DMATXE[*x*] = 0, where *x* = IFMI – 1.

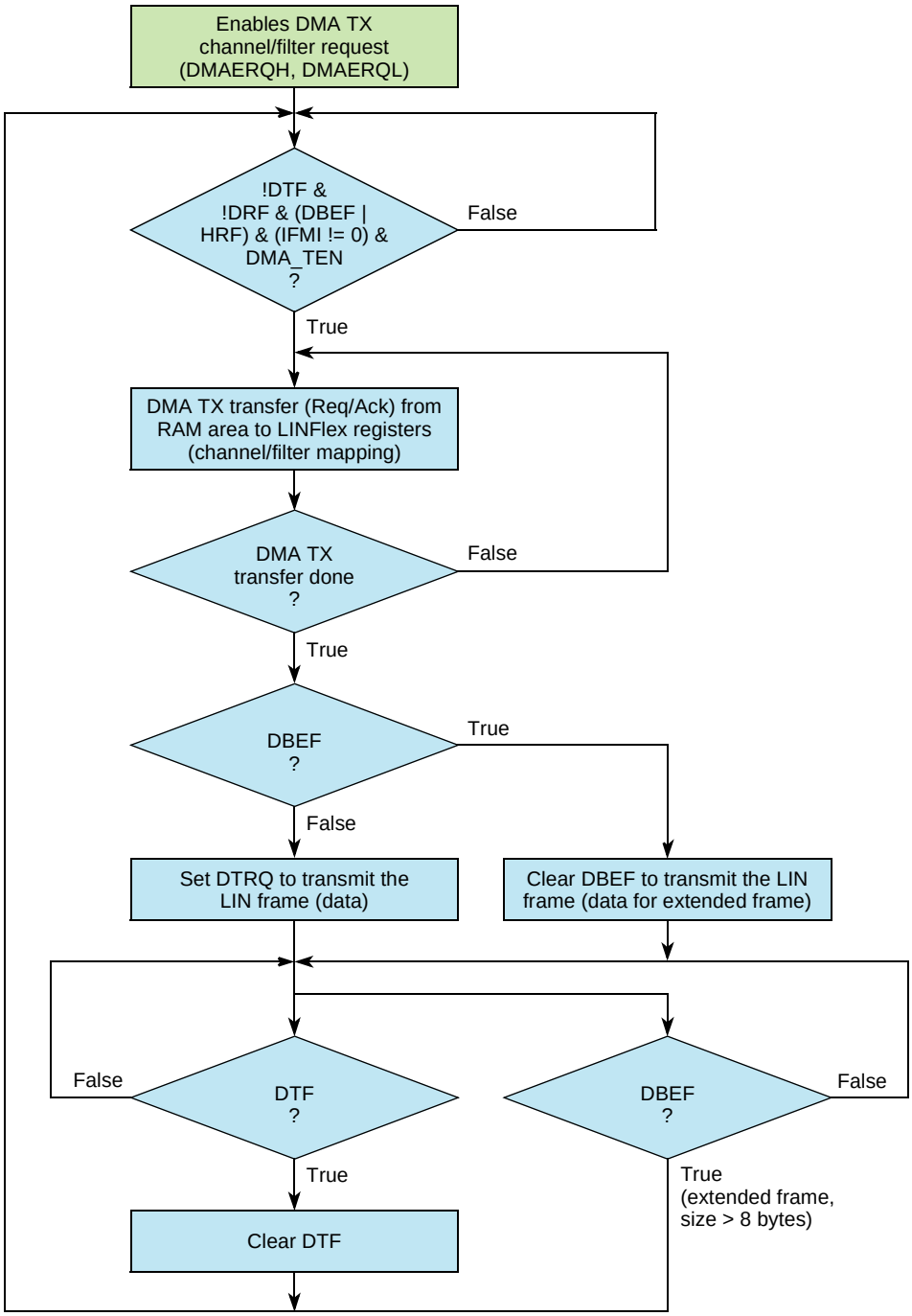


Figure 663. FSM to control the DMA TX interface (slave node)

The TCD settings (word transfer) are shown in *Table 528*. All other TCD fields are equal to 0. TCD settings based on half-word or byte transfer are allowed.

Table 528. 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

31.11.4 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 664*.

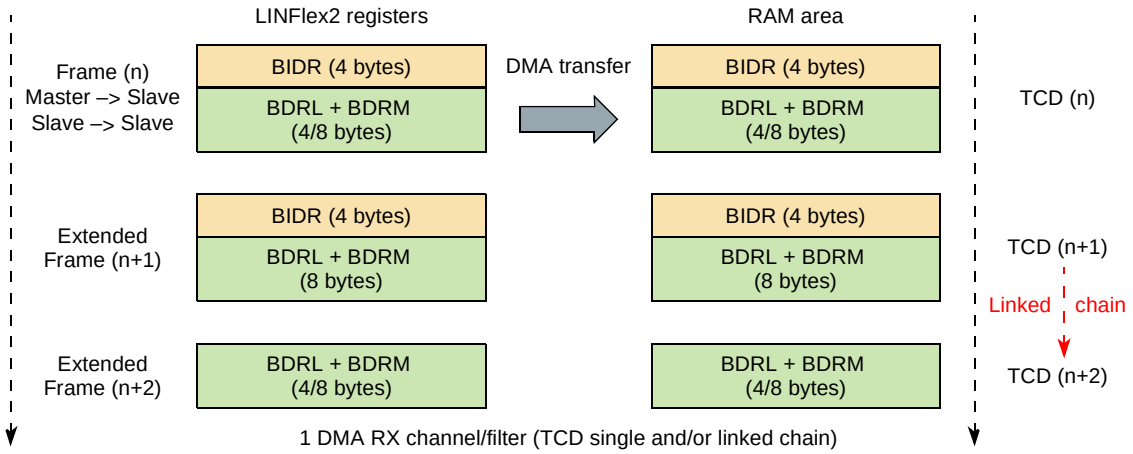


Figure 664. TCD chain memory map (slave node, RX mode)

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 529*.

Table 529. 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 665*. DMA RX FSM will move to idle state if DMARXE[*x*] = 0 where *x* = IFMI - 1.

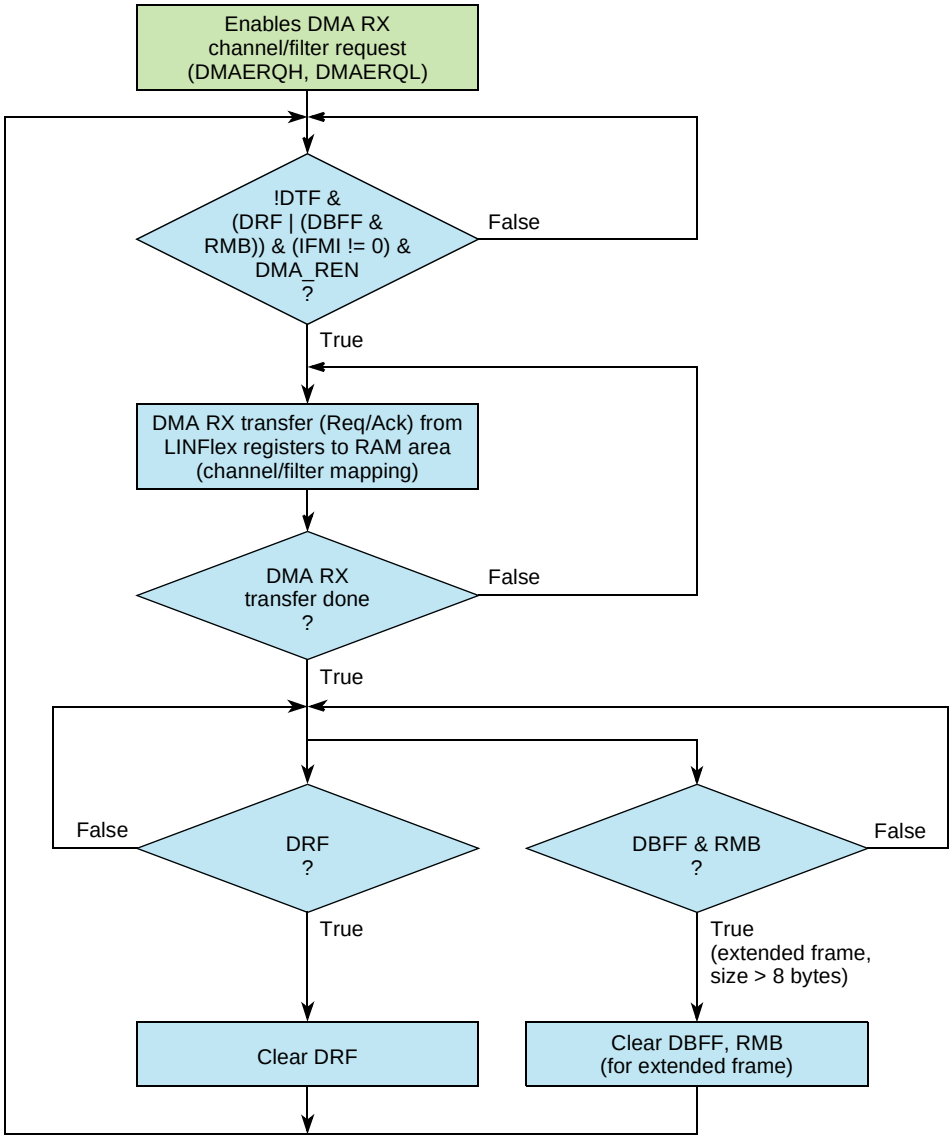


Figure 665. FSM to control the DMA RX interface (slave node)

The TCD settings (word transfer) are shown in *Table 530*. All other TCD fields = 0. TCD settings based on half-word or byte transfer are allowed.

Table 530. 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 530. 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

31.11.5 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 666*.

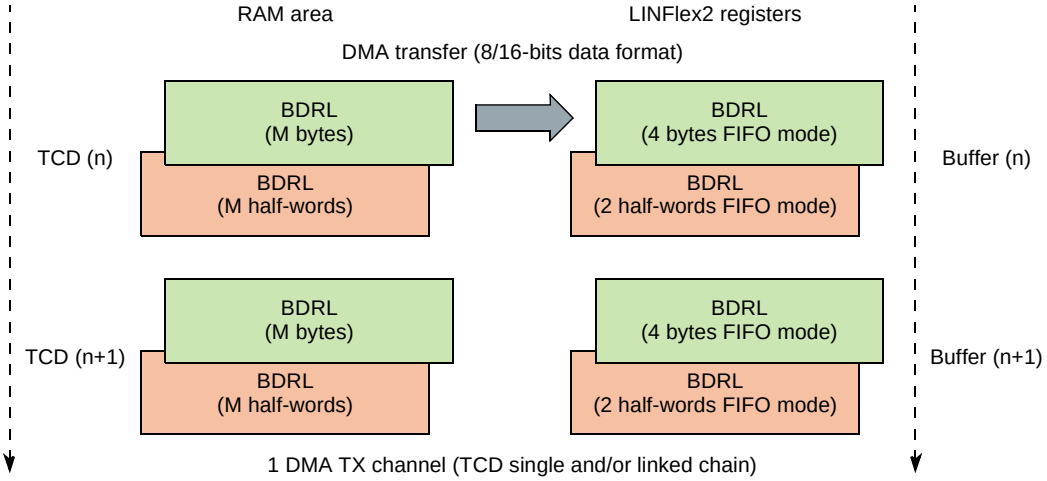


Figure 666. TCD chain memory map (UART node, TX mode)

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 667*. DMA TX FSM will move to idle state if DMATXE[0] = 0.

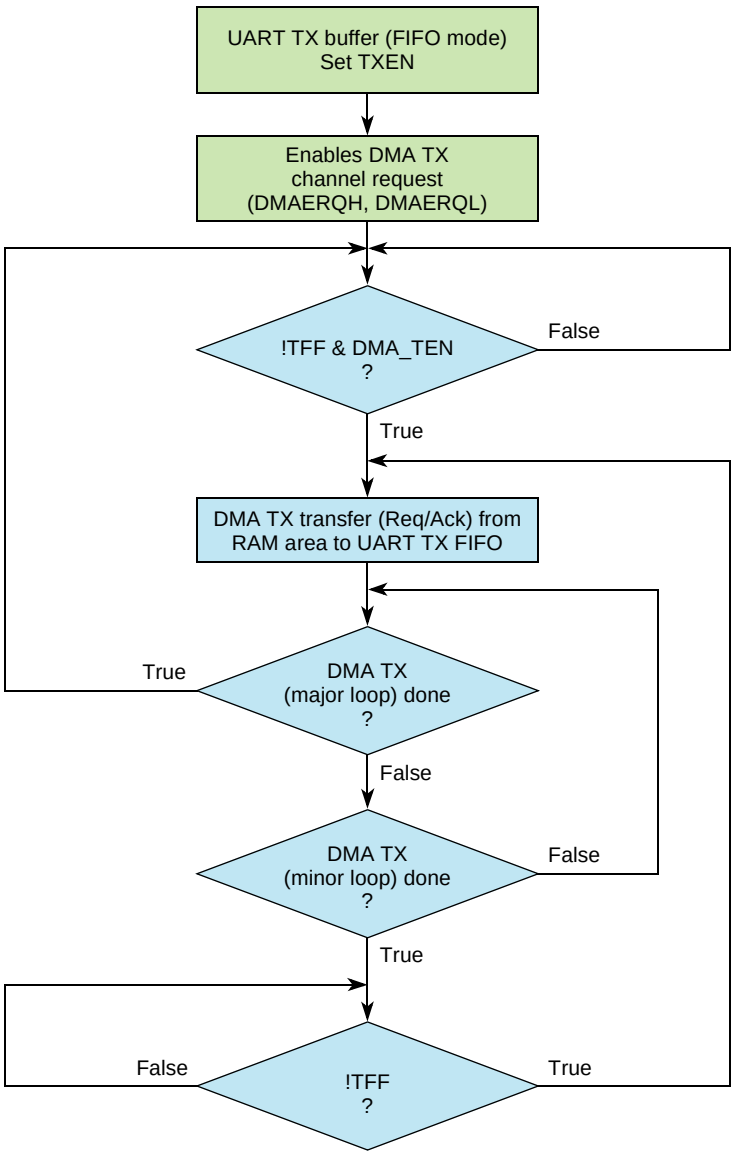


Figure 667. FSM to control the DMA TX interface (UART node)

The TCD settings (typical case) are shown in *Table 531*. 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 531. 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

31.11.6 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 668*.

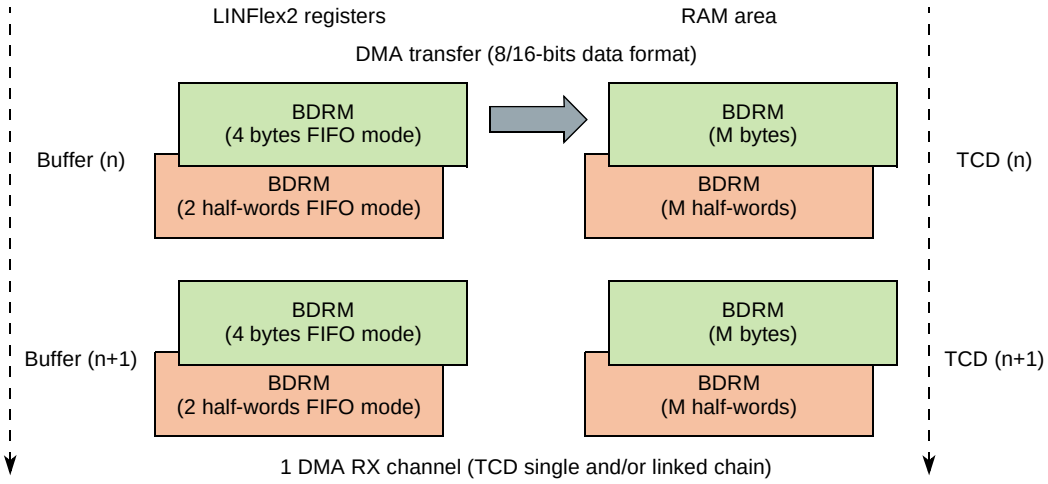


Figure 668. TCD chain memory map (UART node, RX mode)

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 669*. DMA Rx FSM will move to idle state if DMARXE[0] = 0.

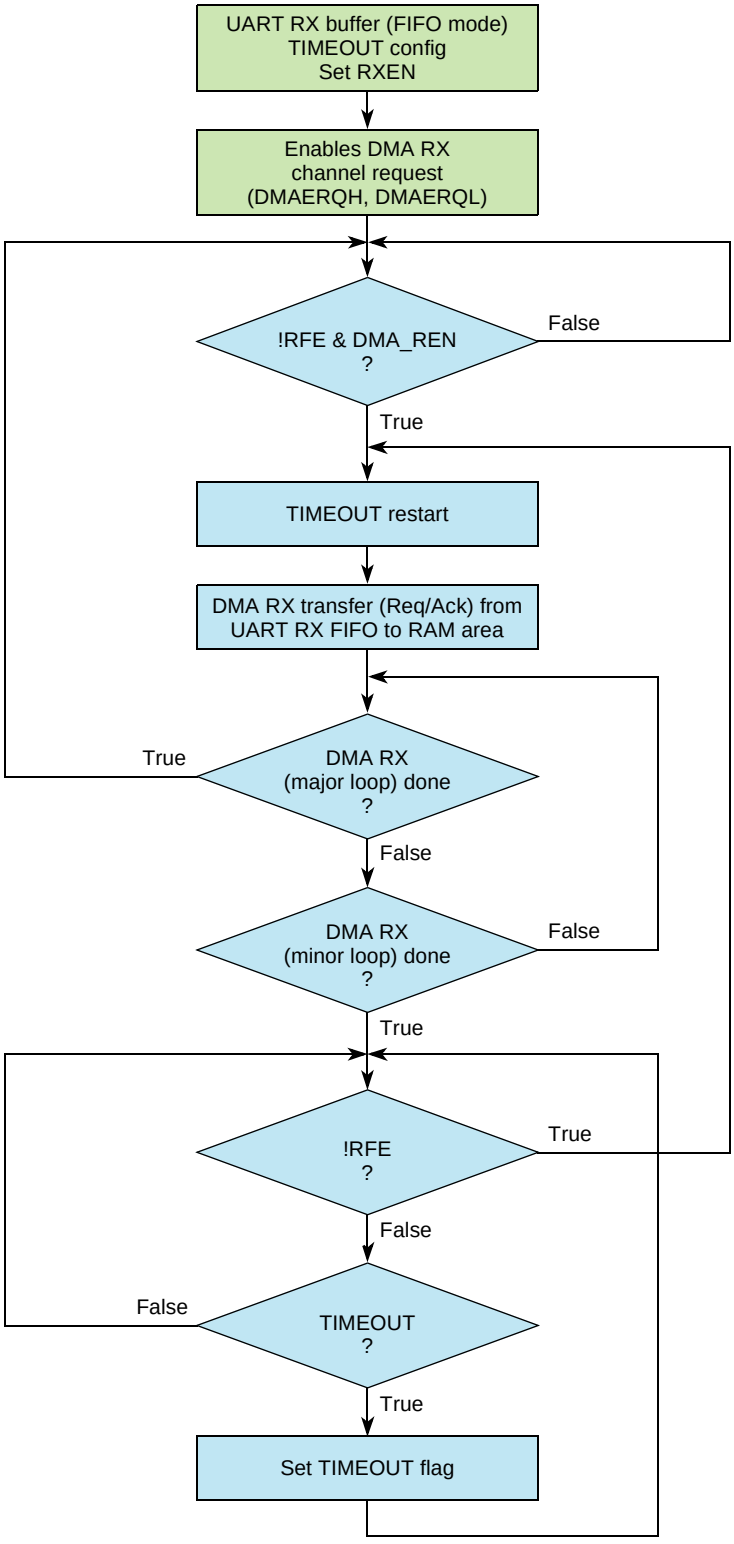


Figure 669. FSM to control the DMA RX interface (UART node)

The TCD settings (typical case) are shown in *Table 532*. 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 532. 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

31.11.7 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.

31.12 Functional description

31.12.1 8-bit timeout counter

31.12.1.1 LIN timeout mode

The timeout counter can be used to monitor LIN frames timing as well as problems on the LIN bus such as:

- Bus stuck at dominant state
- Header timeout
- No bus activity
- Slave not responding error (in Master mode)
- Wake-up timeout

The header timeout and response timeout values can be tuned in the LINTOCR register. The LINTOCR reset value corresponds to T_{Header_max}, $T_{Response_max}$ and T_{Frame_max} defined in the LIN Standard.

If the LINTCSR[LTOM] = 0, OC1 and OC2 output compare values in the LINOCR register are automatically updated.(o)

OC1 is used to check T_{Header} and $T_{Response}$ and OC2 is used to check T_{Frame} (refer to *Figure 670*).

When LINFlexD moves from Break state to Break Delimiter state (refer to LINSR register):

- 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 x 9 (frame timeout value for an 8-byte frame)),
- The TOCE bit is set.

On the start bit of the first data byte of the response (and if no error occurred during the header reception), OC1 is updated with the value of $OC_{Response}$ ($OC_{Response}$ = CNT + RTO x 9 (response timeout value for an 8-byte frame)).

When BIDR[DFL] is updated (and if no error occurred during the header reception), 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 cleared.

This configuration is used to detect header timeout and response timeout.

o. The LINOCR register is read-only.

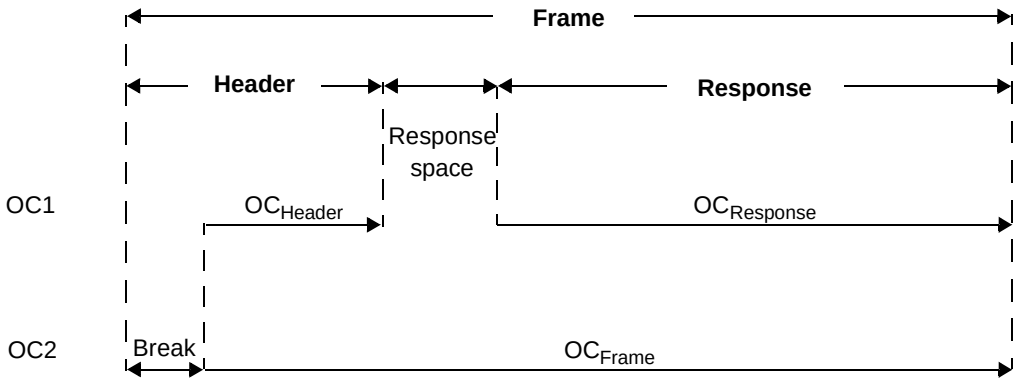


Figure 670. Header and response timeout

31.12.1.2 Output compare mode

The timeout counter can also be used as an Output Compare function. In this case, the LTOM bit must be set and output compare values can be updated in LINTOCR register by software.

31.12.2 Interrupts

Table 533. 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
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
UART Timeout Error	TO	TOIE	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.

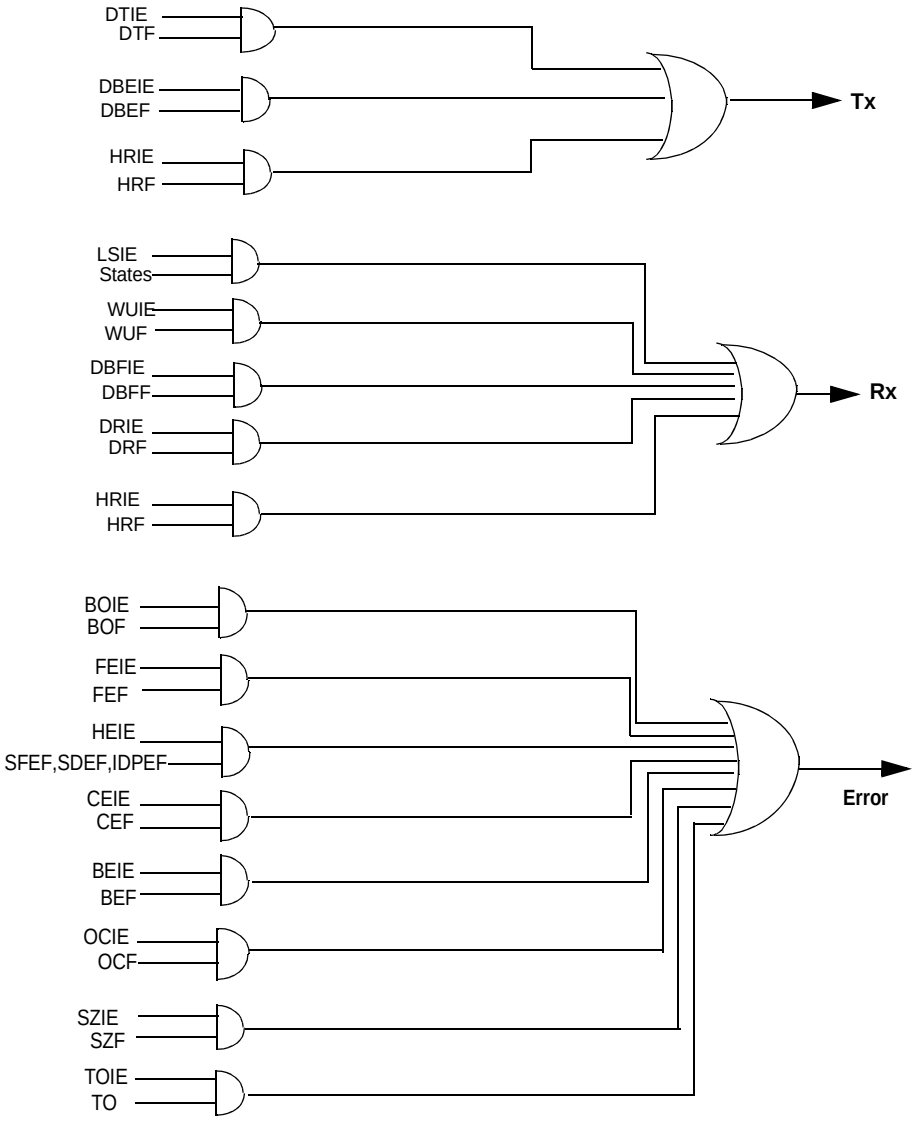


Figure 671. Interrupt diagram

31.12.3 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 7

If LINIBRR = 27d and LINFBRR = 12d, then

Mantissa (LFDIV) = 27d

Fraction (LFDIV) = 12/16 = 0.75d

Therefore LFDIV = 27.75d

Example 8

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 534. Error calculation for programmed baud rates

Baud rate	$f_{ipg_clock_lin}$ = 64 MHz				$f_{ipg_clock_lin}$ = 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	383	16	-0.003	10416.7	95	16	-0.003
19200	19201.9	208	5	0.01	19207.7	52	1	0.04
57600	57605.8	69	7	0.01	57554	17	6	-0.08
115200	115108	34	12	-0.08	115108	8	11	-0.08
230400	230216	17	6	-0.08	231884	4	5	0.644
460800	460432	8	11	-0.08	457143	2	3	-0.794
921600	927536	4	5	0.644	941176	1	1	2.124

31.13 Programming considerations

This section describes the various configurations in which the LINFlexD can be used.

31.13.1 Master node

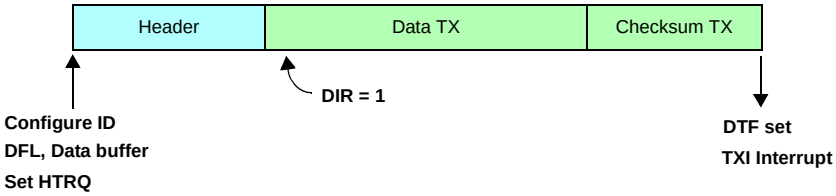


Figure 672. Programming consideration: master node, transmitter

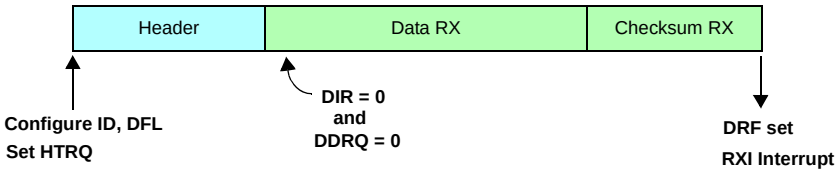


Figure 673. Programming consideration: master node, receiver

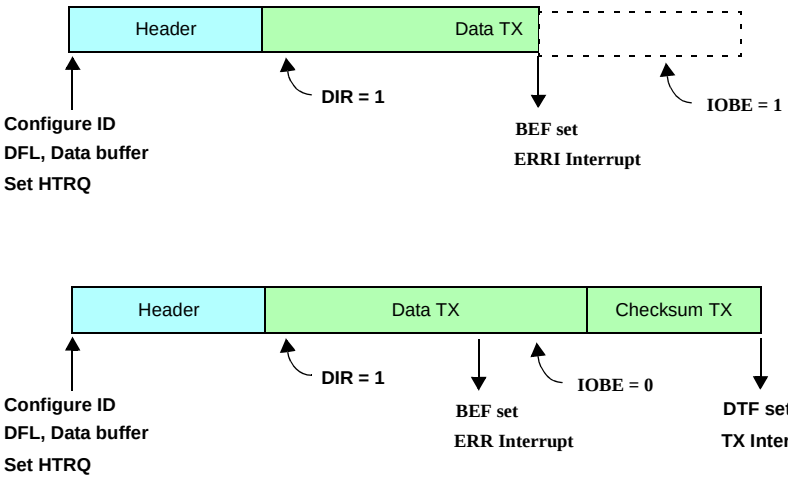


Figure 674. Programming consideration: master node, transmitter, bit error

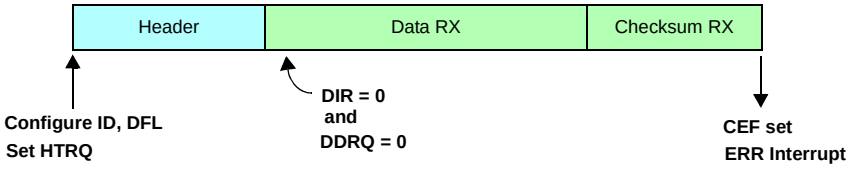


Figure 675. Programming consideration: master node, receiver, checksum error

31.13.2 Slave node

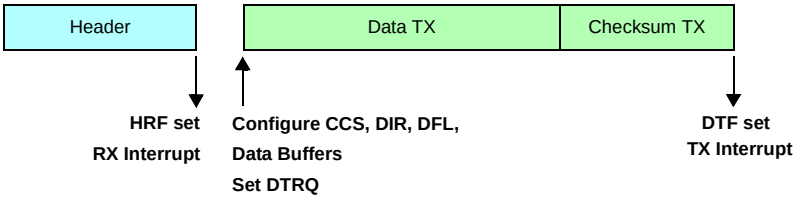


Figure 676. Programming consideration: slave node, transmitter, no filters

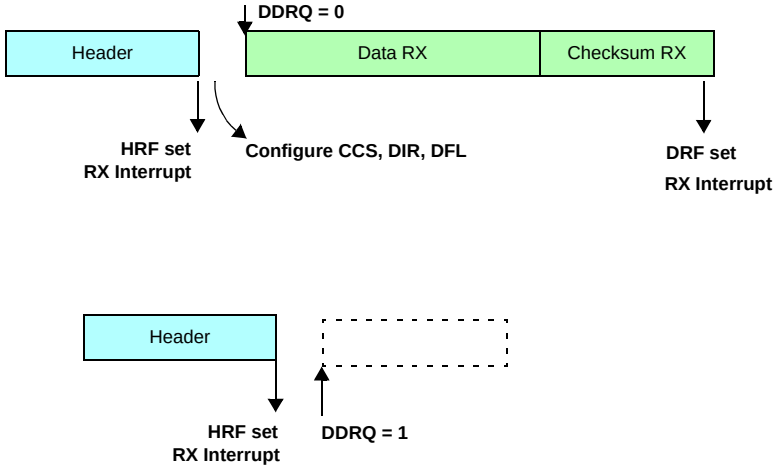


Figure 677. Programming consideration: slave node, receiver, no filters

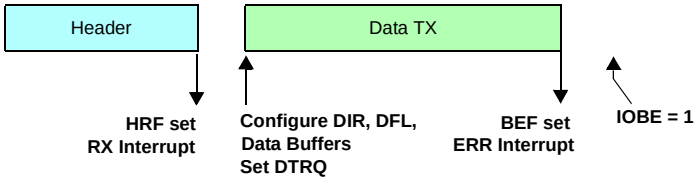


Figure 678. Programming consideration: slave node, transmitter, no filters, bit error

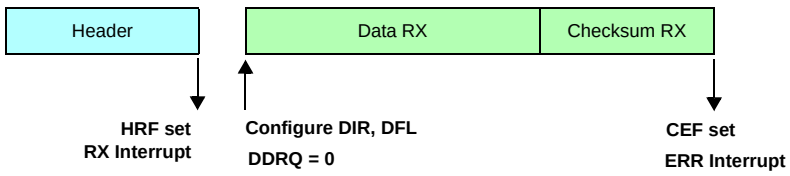
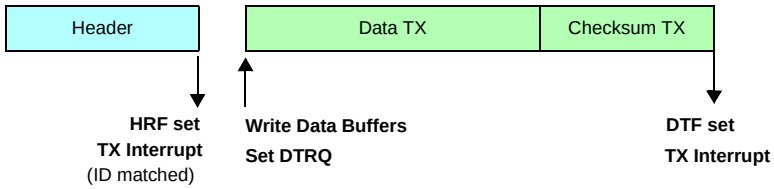


Figure 679. Programming consideration: slave node, receiver, no filters, checksum error



Note: This configuration can be used in case the slave never receives data (for example, as with a sensor).

Figure 680. Programming consideration: slave node, at least one TX filter, BF is reset, ID matches filter

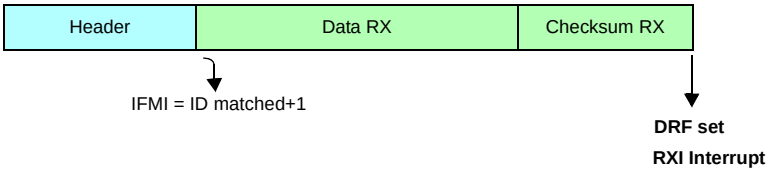


Figure 681. Programming consideration: slave node, at least one RX filter, BF is reset, ID matches filter

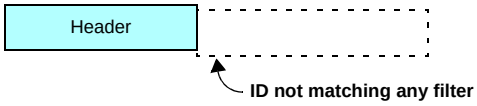
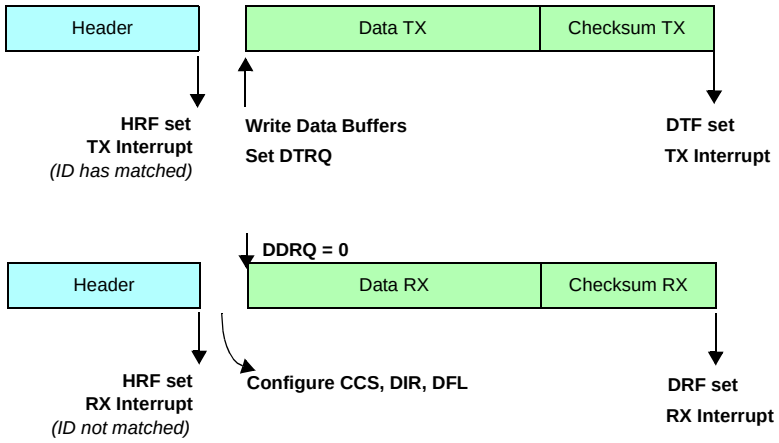


Figure 682. Programming consideration: slave node, RX only, TX only, RX and TX filters, ID not matching filter, BF is reset



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 683. Programming consideration: slave node, TX filter, BF is set

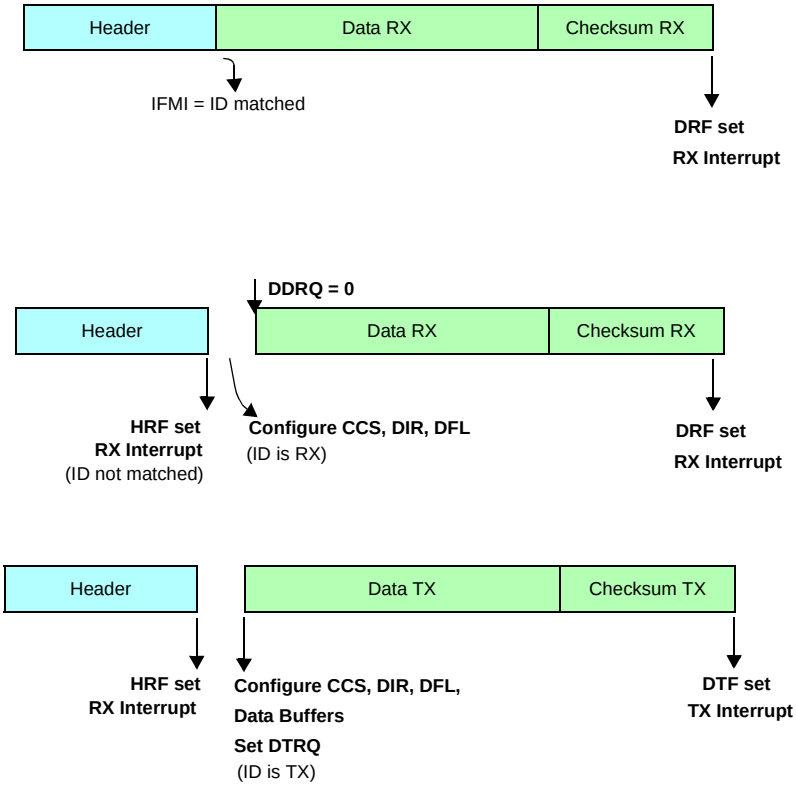
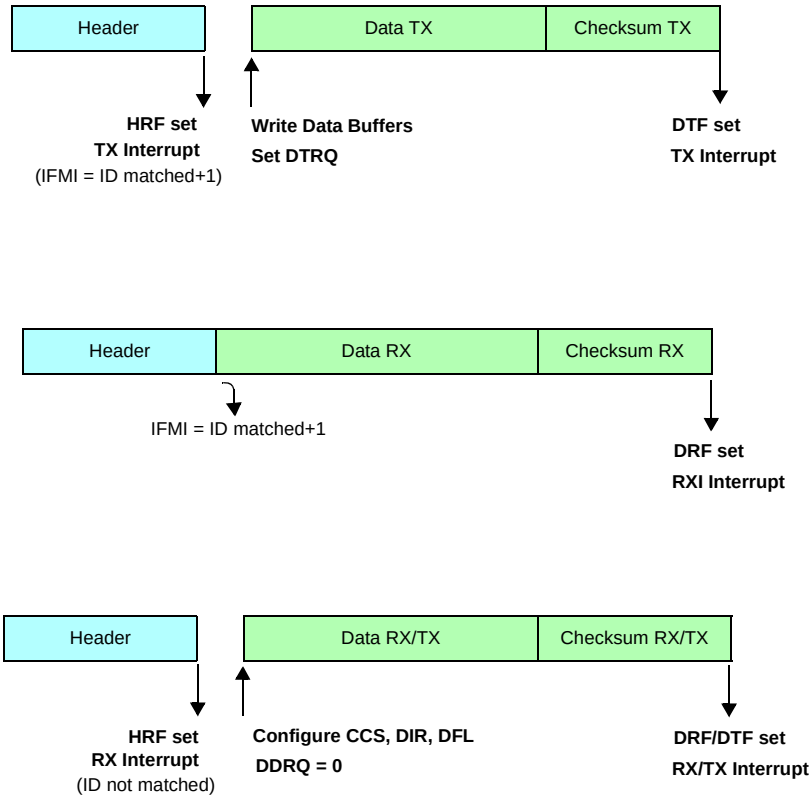


Figure 684. Programming consideration: slave node, RX filter, BF is set



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 685. Programming consideration: slave node, TX filter, RX filter, BF is set

31.13.3 Extended frames

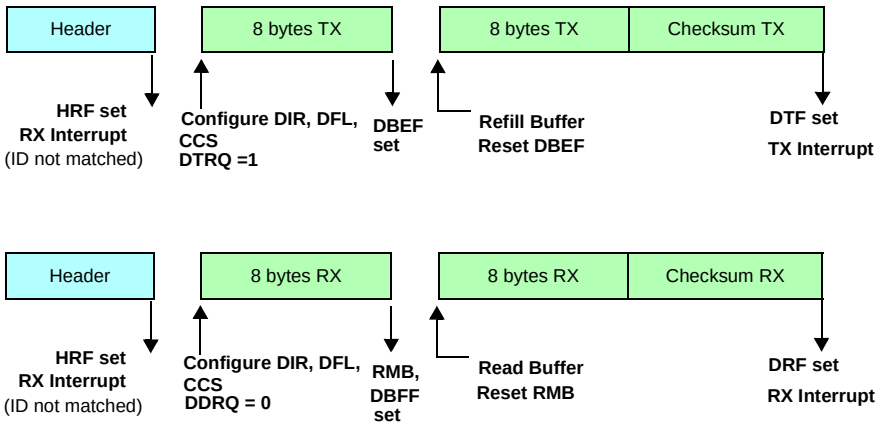


Figure 686. Programming consideration: extended frames

31.13.4 Timeout

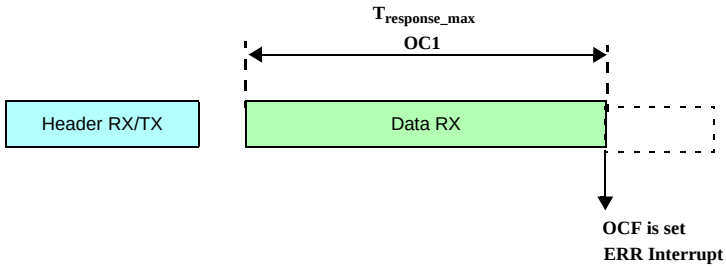


Figure 687. Programming consideration: response timeout

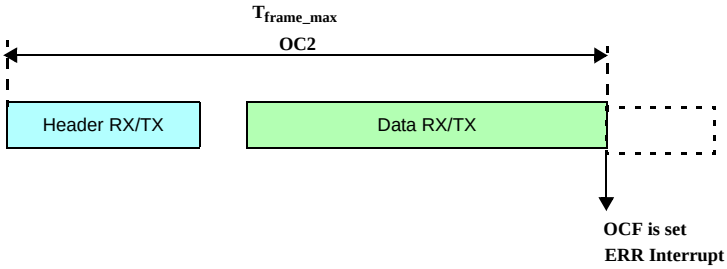


Figure 688. Programming consideration: frame timeout

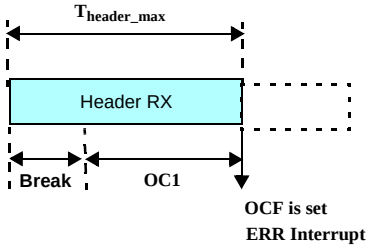


Figure 689. Programming consideration: header timeout

31.13.5 UART mode

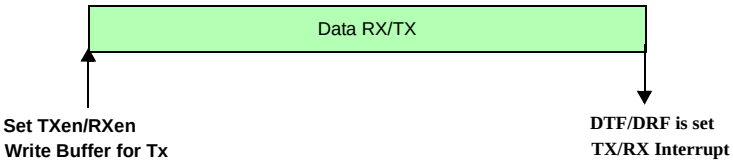


Figure 690. Programming consideration: UART mode

32 Memory Protection Unit (MPU)

32.1 Introduction

The Memory Protection Unit (MPU) provides hardware access control for all memory references generated in a device. Using region descriptors which define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The MPU implements a set of program-visible region descriptors which are used to monitor all system bus addresses. The result is a hardware structure with a two-dimensional connection matrix, where the region descriptors represent one dimension and the individual system bus addresses and attributes are the second dimension.

32.2 Block diagram

A simplified block diagram of the MPU module is shown in *Figure 691*. The hardware's two-dimensional connection matrix is clearly visible with the basic access evaluation "macro" shown as the replicated submodule block. The AHB[(p)] bus slave ports (s{0,1,2,3}_h*) are shown on the left side of the diagram, the region descriptor registers in the middle and the IPS bus interface (ips_*) on the right side. The evaluation macro contains the two magnitude comparators connected to the start and end address registers from each region descriptor (rgd*n*) as well as the combinational logic blocks to determine the region hit and the access protection error. For information on the details of the access evaluation macro, see *Section 32.7.1, Access evaluation macro*.

p. ARM's Advanced High-performance Bus

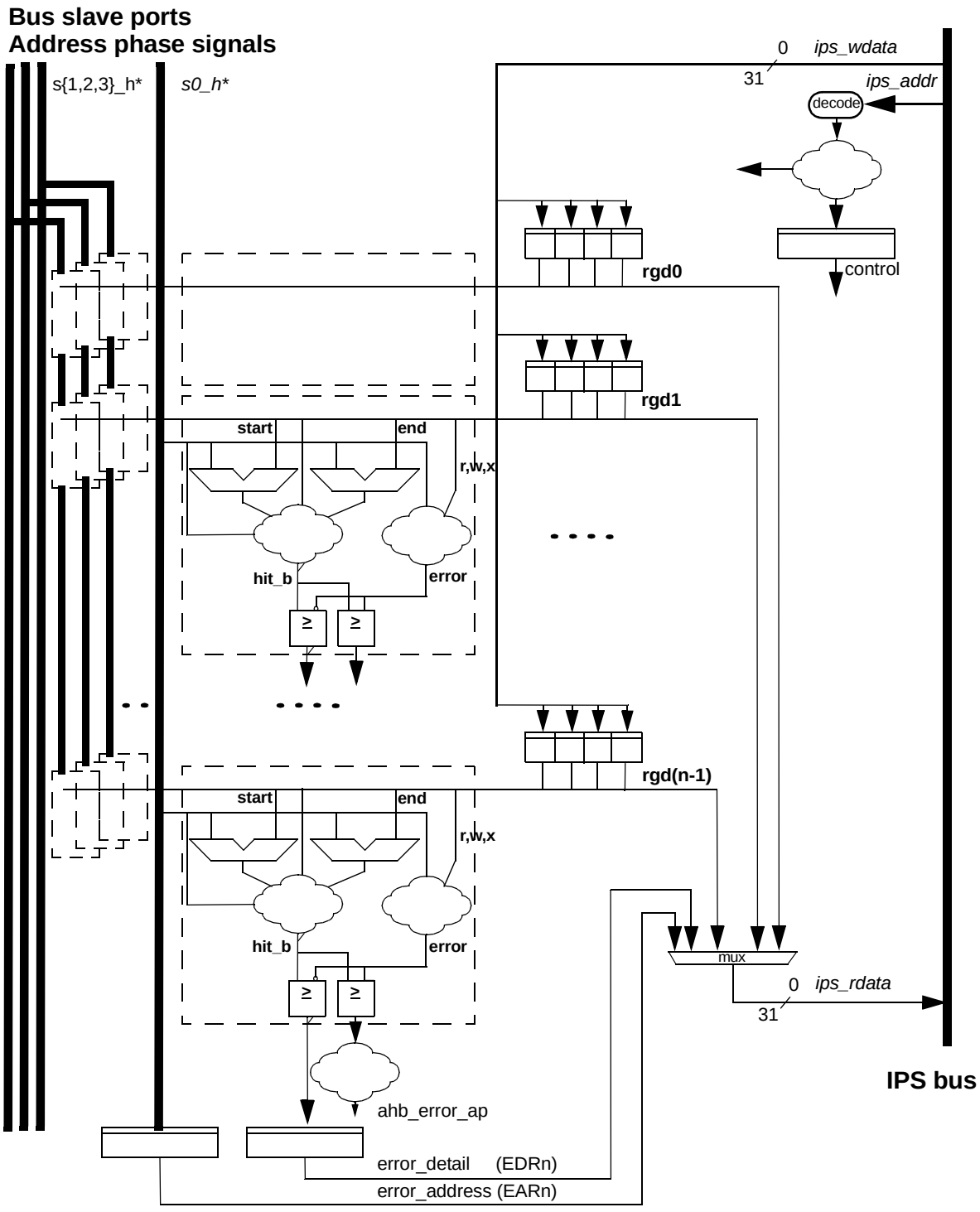


Figure 691. MPU block diagram

32.3 Features

The MPU implements a two-dimensional hardware array of memory region descriptors and the crossbar slave AHB ports to continuously monitor the legality of every memory reference generated by each bus master in the system. The feature set includes:

- Support for 16 memory region descriptors
  - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB
  - 4 bus masters that support the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses.
  - Automatic hardware maintenance of the region descriptor valid bit removes issues associated with maintaining a coherent image of the descriptor
  - Alternate memory view of the access control word for each descriptor provides an efficient mechanism to dynamically alter only the access rights of a descriptor
  - For overlapping region descriptors, priority is given to permission granting over access denying as this approach provides more flexibility to system software. See *Section 32.7.2, Putting it all together and AHB error terminations*, for details and *Section 32.9, Application information*, for an example.
- Support for 3 AHB slave port connections
  - MPU hardware continuously monitors every AHB slave port access using the pre-programmed memory region descriptors
  - An access protection error is detected if a memory reference does not hit in any memory region or the reference is flagged as illegal in *all* memory regions where it does hit. In the event of an access error, the AHB reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device. This results in a Machine Check Exception.
  - 64-bit error registers, one for each AHB slave port, capture the last faulting address, attributes and “detail” information
- Global MPU enable/disable control bit provides a mechanism to easily load region descriptors during system startup or allow complete access rights during debug with the module disabled

On this device, the two instantiations of the MPU are referred to as MPU_0 (attached to the slave side of XBAR_0) and MPU_1 (attached to the slave side of XBAR_1). Both instantiations are identical and do not differ in LS Mode or DP Mode.

The MPU port allocation is shown in *Table 535*.

Table 535. MPU port allocation

AXBS slave port	MPU_0	MPU_1
Flash memory	Port 0	Port 0
SRAM	Port 1	Port 1
Peripheral bridge	Port 2	Port 2

32.4 Modes of operation

The MPU module does not support any special modes of operation. As a memory-mapped device located on the platform’s high-speed system bus, it responds based strictly on the

memory addresses of the connected system buses. The IPS bus is used to access the MPU's programming model and the memory protection functions are evaluated on a reference-by-reference basis using the addresses from the AHB system bus port(s).

Power dissipation is minimized when the MPU's global enable/disable bit is cleared (MPU_CESR[VLD] = 0).

32.5 External signal description

The MPU module does not include any external interface.

32.6 Memory map and register definition

The MPU module provides an IPS programming model mapped to an SPP-standard on-platform 16 KB space. The programming model is partitioned into three groups:

- Control/status registers
- The data structure containing the region descriptors
- The alternate view of the region descriptor access control values

The programming model can only be referenced using 32-bit (word) accesses. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (for example, a write to a read-only register or a read of a write-only register) generate an IPS error termination.

The MPU programming model map is shown in *Table 536*.

Table 536. MPU memory map

Offset address	Register name	Register description	Size (bits)	Location
0x0000	MPU_CESR	MPU Control/Error Status Register	32	*on page 1066*
0x0004–0x000F	Reserved			
0x0010	MPU_EAR0	MPU Error Address Register, Slave Port 0	32	*on page 1067*
0x0014	MPU_EDR0	MPU Error Detail Register, Slave Port 0	32	*on page 1068*
0x0018	MPU_EAR1	MPU Error Address Register, Slave Port 1	32	*on page 1067*
0x001C	MPU_EDR1	MPU Error Detail Register, Slave Port 1	32	*on page 1068*
0x0020	MPU_EAR2	MPU Error Address Register, Slave Port 2	32	*on page 1067*
0x0024	MPU_EDR2	MPU Error Detail Register, Slave Port 2	32	*on page 1068*
0x0028–0x03FF	Reserved			
0x0400	MPU_RGD0	MPU Region Descriptor 0	128	*on page 1069*
0x0410	MPU_RGD1	MPU Region Descriptor 1	128	*on page 1069*
0x0420	MPU_RGD2	MPU Region Descriptor 2	128	*on page 1069*
0x0430	MPU_RGD3	MPU Region Descriptor 3	128	*on page 1069*
0x0440	MPU_RGD4	MPU Region Descriptor 4	128	*on page 1069*

Table 536. MPU memory map (continued)

Offset address	Register name	Register description	Size (bits)	Location
0x0450	MPU_RGD5	MPU Region Descriptor 5	128	on page 1069
0x0460	MPU_RGD6	MPU Region Descriptor 6	128	on page 1069
0x0470	MPU_RGD7	MPU Region Descriptor 7	128	on page 1069
0x0480	MPU_RGD8	MPU Region Descriptor 8	128	on page 1069
0x0490	MPU_RGD9	MPU Region Descriptor 9	128	on page 1069
0x04A0	MPU_RGD10	MPU Region Descriptor 10	128	on page 1069
0x04B0	MPU_RGD11	MPU Region Descriptor 11	128	on page 1069
0x04C0	MPU_RGD12	MPU Region Descriptor 12	128	on page 1069
0x04D0	MPU_RGD13	MPU Region Descriptor 13	128	on page 1069
0x04E0	MPU_RGD14	MPU Region Descriptor 14	128	on page 1069
0x04F0	MPU_RGD15	MPU Region Descriptor 15	128	on page 1069
0x0500-0x07FF	Reserved			
0x0800	MPU_RGDAAC0	MPU RGD Alternate Access Control 0	32	on page 1073
0x0804	MPU_RGDAAC1	MPU RGD Alternate Access Control 1	32	on page 1073
0x0808	MPU_RGDAAC2	MPU RGD Alternate Access Control 2	32	on page 1073
0x080C	MPU_RGDAAC3	MPU RGD Alternate Access Control 3	32	on page 1073
0x0810	MPU_RGDAAC4	MPU RGD Alternate Access Control 4	32	on page 1073
0x0814	MPU_RGDAAC5	MPU RGD Alternate Access Control 5	32	on page 1073
0x0818	MPU_RGDAAC6	MPU RGD Alternate Access Control 6	32	on page 1073
0x081C	MPU_RGDAAC7	MPU RGD Alternate Access Control 7	32	on page 1073
0x0820	MPU_RGDAAC8	MPU RGD Alternate Access Control 8	32	on page 1073
0x0824	MPU_RGDAAC9	MPU RGD Alternate Access Control 9	32	on page 1073
0x0828	MPU_RGDAAC10	MPU RGD Alternate Access Control 10	32	on page 1073
0x082C	MPU_RGDAAC11	MPU RGD Alternate Access Control 11	32	on page 1073
0x0830	MPU_RGDAAC12	MPU RGD Alternate Access Control 12	32	on page 1073
0x0834	MPU_RGDAAC13	MPU RGD Alternate Access Control 13	32	on page 1073
0x0838	MPU_RGDAAC14	MPU RGD Alternate Access Control 14	32	on page 1073
0x083C	MPU_RGDAAC15	MPU RGD Alternate Access Control 15	32	on page 1073
0x0840-0x3FFF	Reserved			

32.6.1 MPU Control/Error Status Register (MPU_CESR)

The MPU_CESR provides one byte of error status plus three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Offset MPU_Base + 0x000 Access: Read/Partial Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SPERR			0	0	0	0	0	1	0	0	0	HRL				NSP				NRGD				0	0	0	0	0	0	0	VLD
W	w1c																															
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0

Figure 692. MPU Control/Error Status Register (MPU_CESR)

Table 537. MPU_CESR field descriptions

Field	Description
SPERR	Slave Port n Error, where the slave port number matches the bit number. Each bit in this field represents a flag maintained by the MPU for signaling the presence of a captured error contained in the MPU_EARn and MPU_EDRn registers. The individual bit is set when the hardware detects an error and records the faulting address and attributes. It is cleared when the corresponding bit is written as a logical one. If another error is captured at the exact same cycle as a write of a logical one, this flag remains set. A “find first one” instruction (or equivalent) can be used to detect the presence of a captured error. 0 The corresponding MPU_EARn/MPU_EDRn registers do not contain a captured error. 1 The corresponding MPU_EARn/MPU_EDRn registers do contain a captured error.
HRL	Hardware Revision Level. This 4-bit read-only field specifies the MPU’s hardware and definition revision level. It can be read by software to determine the functional definition of the module.
NSP	Number of Slave Ports. This 4-bit read-only field specifies the number of slave ports connected to the MPU. For this device, NSP = 3.
NRGD	Number of Region Descriptors. This 4-bit read-only field specifies the number of region descriptors implemented in the MPU. The defined encodings include: 0b00 8 region descriptors 0b01 12 region descriptors 0b10 16 region descriptors (correct value for this device)
VLD	Valid. This bit provides a global enable/disable for the MPU. 0 The MPU is disabled. 1 The MPU is enabled. While the MPU is disabled, all accesses from all bus masters are allowed.

32.6.2 MPU Error Address Register, Slave Port n (MPU_EARn)

When the MPU detects an access error on slave port n, the 32-bit reference address is captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field set. Additional information about the faulting access is captured in the corresponding MPU_EDRn register at the same time. This register and the corresponding MPU_EDRn register contain the most recent access error; there are no hardware interlocks with the MPU_CESR[SPERR] field as the error registers are always loaded upon the occurrence of each protection violation.

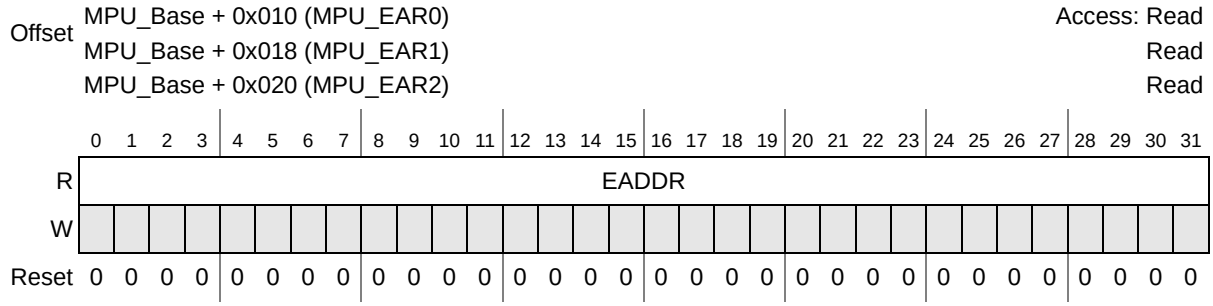


Figure 693. MPU Error Address Register, Slave Port n (MPU_EARn)

Table 538. MPU_EARn field descriptions

Field	Description
0–31 EADDR	Error Address. This read-only field is the reference address from slave port n that generated the access error.

32.6.3 MPU Error Detail Register, Slave Port n (MPU_EDRn)

When the MPU detects an access error on slave port n, 32 bits of error detail are captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field set. Information on the faulting address is captured in the corresponding MPU_EARn register at the same time. Note this register and the corresponding MPU_EARn register contain the most recent access error; there are no hardware interlocks with the MPU_CESR[SPERR] field as the error registers are always loaded upon the occurrence of each protection violation.

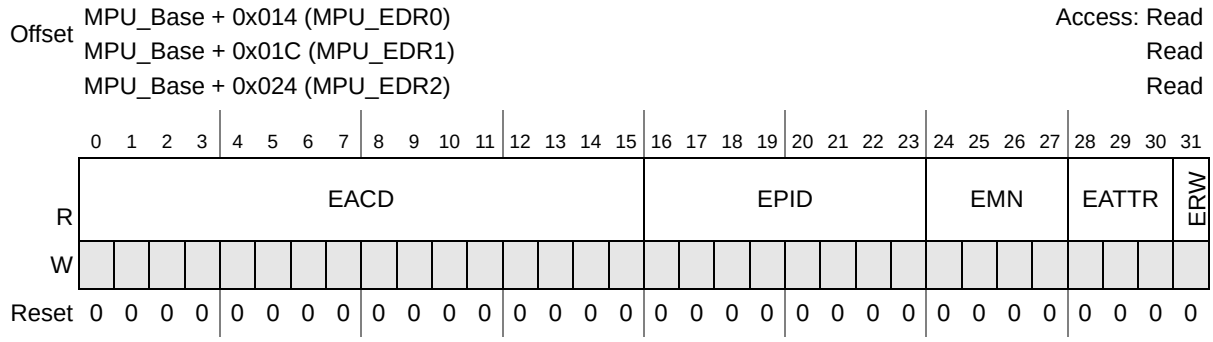


Figure 694. MPU Error Detail Register, Slave Port n (MPU_EDRn)

Table 539. MPU_EDRn field descriptions

Field	Description
0–15 EACD	Error Access Control Detail. This 16-bit read-only field implements one bit per region descriptor and is an indication of the region descriptor hit where the error occurred. The MPU performs a reference-by-reference evaluation to determine the presence/absence of an access error. When an error is detected, the hit-qualified access control vector is captured in this field. If the MPU_EDRn register contains a captured error and the EACD field is all zeroes, this signals an access that did not hit in any region descriptor. All non-zero EACD values signal references that hit in a region descriptor(s), but failed due to a protection error as defined by the specific set bits. If only a single EACD bit is set, then the protection error was caused by a single non-overlapping region descriptor. If two or more EACD bits are set, then the protection error was caused in an overlapping set of region descriptors.
16–23 EPID	Error Process Identification. This 8-bit read-only field records the process identifier of the faulting reference. The process identifier is typically driven only by processor cores; for other bus masters, this field is cleared.
24–27 EMN	Error Master Number. This 4-bit read-only field records the logical master number of the faulting reference. This field is used to determine the bus master that generated the access error.
28–30 EATTR	Error Attributes. This 3-bit read-only field records attribute information about the faulting reference. The supported encodings are defined as: 0b00 User mode, instruction access 0b01 User mode, data access 0b10 Supervisor mode, instruction access 0b11 Supervisor mode, data access All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (0b011).
31 ERW	Error Read/Write. This 1-bit read-only field signals the access type (read, write) of the faulting reference. 0 Read 1 Write

32.6.4 MPU Region Descriptor n (MPU_RGDn)

Each 128-bit (four 32-bit word) region descriptor specifies a given memory space and the access attributes associated with that space. The descriptor definition is the very essence of the operation of the Memory Protection Unit.

The region descriptors are organized sequentially in the MPU's programming model and each of the four 32-bit words are detailed in the subsequent sections.

32.6.4.1 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 32.6.4.4, MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3),* for more information).

Offset MPU_Base + 0x400 + (16*n) + 0x0 (MPU_RGDn.Word0) Access: R/W

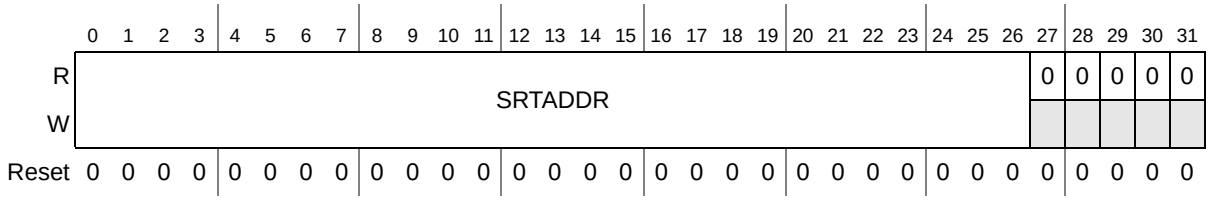


Figure 695. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0)

Table 540. MPU_RGDn.Word0 field descriptions

Field	Description
0–26 SRTADDR	Start Address. This field defines the most significant bits of the 0-modulo-32 byte start address of the memory region.

32.6.4.2 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 32.6.4.4, MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3),* for more information).

Offset MPU_Base + 0x400 + (16*n) + 0x4 (MPU_RGDn.Word1) Access: R/W

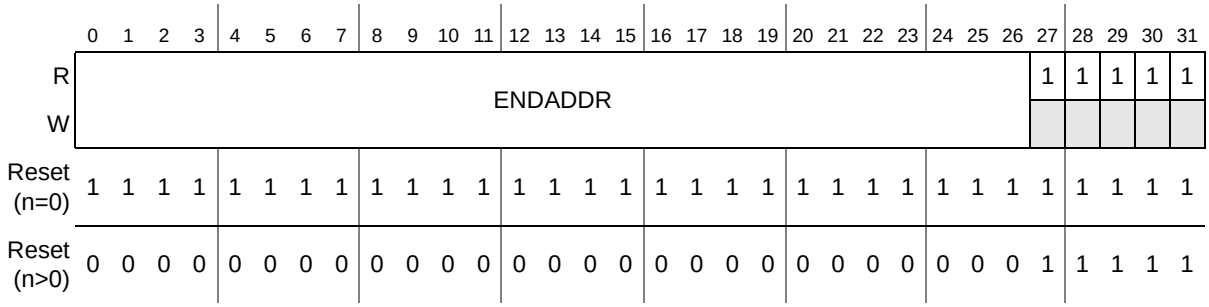


Figure 696. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1)

Table 541. MPU_RGDn.Word1 field descriptions

Field	Description
0–26 ENDADDR	End Address. This field defines the most significant bits of the 31-modulo-32 byte end address of the memory region. There are no hardware checks to verify that ENDADDR >= SRTADDR; it is software's responsibility to properly load these region descriptor fields.

32.6.4.3 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. Bus masters 0-3 have a 6-bit field defining separate privilege rights for user and supervisor mode accesses as well as the optional inclusion of a process identification field within the definition. For these fields, the bus master number refers to the logical master number as specified in *Section 15.1.4, Logical master IDs.* (Nexus controllers have IDs 8

and 9, but only the last 3 bits of the ID are used for this purposes, so they share privileges with cores 0 and 1.)

For the processor privilege rights, there are three flags associated with this function: {read, write, execute}. In this context, these flags follow the traditional definition:

- Read (r) permission refers to the ability to access the referenced memory address using an operand (data) fetch.
- Write (w) permission refers to the ability to update the referenced memory address using a store (data) instruction.
- Execute (x) permission refers to the ability to read the referenced memory address using an instruction fetch.

Writes to this word clear the region descriptor's valid bit (see *Section 32.6.4.4, 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 MPU_Base + 0x400 + (16*n) + 0x8 (MPU_RGDn.Word2) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	M3PE	M3SM		M3UM			M2PE	M2SM		M2UM			M1PE	M1SM		M1UM			M0PE	M0SM		M0UM		
W											r	w	x				r	w	x				r	w	x						r	w	x
Reset (n=0)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Reset (n>0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 697. MPU Region Descriptor, Word 2 Register (MPU_RGDn.Word2)

Table 542. MPU_RGDn.Word2 field descriptions

Field	Description
M3PE	Bus master 3 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M3SM	Bus master 3 supervisor mode access control. This 2-bit field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, –, x = read and execute allowed, but no write 0b10 r, w, – = read and write allowed, but no execute 0b11 Same access controls as that defined by M3UM for user mode

Table 542. MPU_RGDn.Word2 field descriptions (continued)

Field	Description
M3UM	Bus master 3 user mode access control. This 3-bit field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M2PE	Bus master 2 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M2SM	Bus master 2 supervisor mode access control. This 2-bit field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, –, x = read and execute allowed, but no write 0b10 r, w, – = read and write allowed, but no execute 0b11 Same access controls as that defined by M2UM for user mode
M2UM	Bus master 2 user mode access control. This 3-bit field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M1PE	Bus master 1 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M1SM	Bus master 1 supervisor mode access control. This 2-bit field defines the access controls for bus master 1 when operating in supervisor mode. The M1SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, –, x = read and execute allowed, but no write 0b10 r, w, – = read and write allowed, but no execute 0b11 Same access controls as that defined by M1UM for user mode
M1UM	Bus master 1 user mode access control. This 3-bit field defines the access controls for bus master 1 when operating in user mode. The M1UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M0PE	Bus master 0 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M0SM	Bus master 0 supervisor mode access control. This 2-bit field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, –, x = read and execute allowed, but no write 0b10 r, w, – = read and write allowed, but no execute 0b11 Same access controls as that defined by M0UM for user mode
M0UM	Bus master 0 user mode access control. This 3-bit field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

32.6.4.4 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 4-word entity, there are potential coherency issues as this structure is being updated since multiple writes are required to update the entire descriptor. Accordingly, the MPU hardware assists in the operation of the descriptor valid bit to prevent incoherent region descriptors from generating spurious access errors. In particular, it is expected that a complete update of a region descriptor is typically done with sequential writes to MPU_RGDn.Word0, then MPU_RGDn.Word1,... and finally MPU_RGDn.Word3. The MPU hardware automatically clears the valid bit on any writes to words {0,1,2} of the descriptor. Writes to this word set/clear the valid bit in a normal manner.

Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU_RGDn.Word2) as different tasks execute, an alternate programming view of MPU_RGDn.Word2 is provided. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAACn (Alternate Access Control n) as stores to these locations do *not* affect the descriptor's valid bit.

Offset MPU_Base + 0x400 + (16*n) + 0xc (MPU_RGDn.Word3) Access: R/W

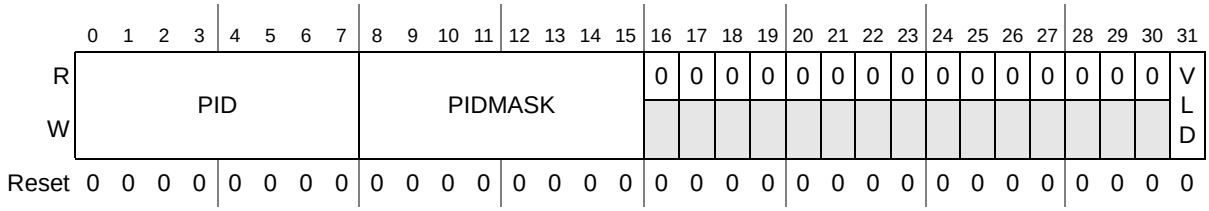


Figure 698. MPU Region Descriptor, Word 3 Register (MPU_RGDn.Word3)

Table 543. MPU_RGDn.Word3 field descriptions

Field	Description
0–7 PID	Process Identifier. This 8-bit field specifies that the optional process identifier is to be included in the determination of whether the current access hits in the region descriptor. This field is combined with the PIDMASK and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set.
8–15 PIDMASK	Process Identifier Mask. This 8-bit field provides a masking capability so that multiple process identifiers can be included as part of the region hit determination. If a bit in the PIDMASK is set, then the corresponding bit of the PID is ignored in the comparison. This field is combined with the PID and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set. For more information on the handling of the PID and PIDMASK, see *Section 32.7.1.1, Access evaluation – hit determination*.
31 VLD	Valid. This bit signals the region descriptor is valid. Any write to MPU_RGDn.Word{0,1,2} clears this bit, while a write to MPU_RGDn.Word3 sets or clears this bit depending on bit 31 of the write operand. 0 Region descriptor is invalid 1 Region descriptor is valid

32.6.5 MPU Region Descriptor Alternate Access Control n (MPU_RGDAACn)

As noted in *Section 32.6.4.3, 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 MPU_Base + 0x800 + (4*n) (MPU_RGDAACn) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	M3PE	M3SM			M3UM				M2PE	M2SM			M2UM				M1PE	M1SM			M1UM				M0PE	M0SM			M0UM		
W													r	w	x						r	w	x						r	w	x						r	w	x
Reset (n=0)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Reset (n>0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 699. 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 544* are identical to those presented in *Table 542*.

Table 544. MPU_RGDAACn field descriptions

Field	Description
M3PE	Bus master 3 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M3SM	Bus master 3 supervisor mode access control. This 2-bit field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as: 0b00 `r, w, x` = read, write and execute allowed 0b01 `r, –, x` = read and execute allowed, but no write 0b10 `r, w, –` = read and write allowed, but no execute 0b11 Same access controls as that defined by M3UM for user mode
M3UM	Bus master 3 user mode access control. This 3-bit field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M2PE	Bus master 2 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.

Table 544. MPU_RGDAACn field descriptions (continued)

Field	Description
M2SM	Bus master 2 supervisor mode access control. This 2-bit field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as: 0b00 `r, w, x` = read, write and execute allowed 0b01 `r, –, x` = read and execute allowed, but no write 0b10 `r, w, –` = read and write allowed, but no execute 0b11 Same access controls as that defined by M2UM for user mode
M2UM	Bus master 2 user mode access control. This 3-bit field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M1PE	Bus master 1 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M1SM	Bus master 1 supervisor mode access control. This 2-bit field defines the access controls for bus master 1 when operating in supervisor mode. The M1SM field is defined as: 0b00 `r, w, x` = read, write and execute allowed 0b01 `r, –, x` = read and execute allowed, but no write 0b10 `r, w, –` = read and write allowed, but no execute 0b11 Same access controls as that defined by M1UM for user mode
M1UM	Bus master 1 user mode access control. This 3-bit field defines the access controls for bus master 1 when operating in user mode. The M1UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M0PE	Bus master 0 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M0SM	Bus master 0 supervisor mode access control. This 2-bit field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as: 0b00 `r, w, x` = read, write and execute allowed 0b01 `r, –, x` = read and execute allowed, but no write 0b10 `r, w, –` = read and write allowed, but no execute 0b11 Same access controls as that defined by M0UM for user mode
M0UM	Bus master 0 user mode access control. This 3-bit field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

32.7 Functional description

In this section, the functional operation of the MPU is detailed. In particular, subsequent sections discuss the operation of the access evaluation macro as well as the handling of error-terminated AHB bus cycles.

32.7.1 Access evaluation macro

As previously discussed, the basic operation of the MPU is performed in the access evaluation macro, a hardware structure replicated in the two-dimensional connection matrix. As shown in *Figure 700*, 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*).

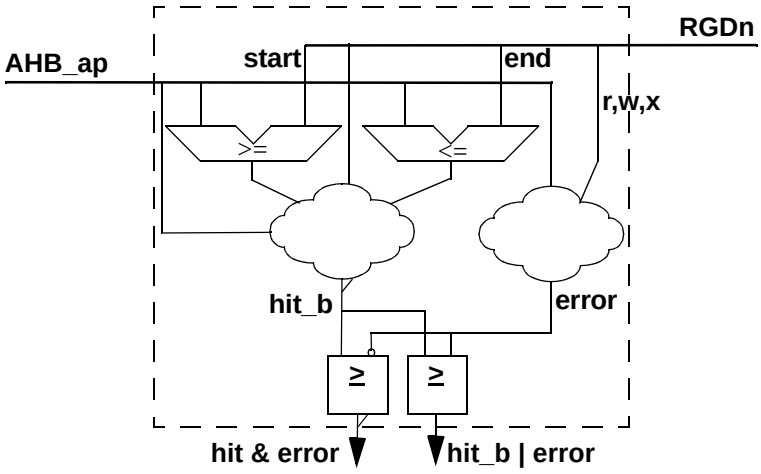


Figure 700. MPU access evaluation macro

Figure 700 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.

32.7.1.1 Access evaluation – hit determination

To determine if the current AHB reference hits in the given region, two magnitude comparators are used in conjunction with the region's start and end addresses. The boolean equation for this portion of the hit determination is defined as:

Equation 49 region_hit =

((haddr[0:26] >= rgdn.srtaddr[0:26]) & (haddr[0:26] <=

rgdn.endaddr[0:26])) & rgdn.vld

where `haddr[*]` is the current AHB reference address, `rgdn.srtaddr[*]` and `rgdn.endaddr[*]` are the start and end addresses, and `rgdn.vld` is the valid bit, all from region descriptor n. Recall there are *no hardware checks* to verify that `rgdn.endaddr` $\geq$ `rgdn.srtaddr`, and it is software's responsibility to properly load appropriate values into these fields of the region descriptor.

In addition to the algebraic comparison of the AHB reference address versus the region descriptor's start and end addresses, the optional process identifier is examined against the region descriptor's PID and PIDMASK fields. Using the `hmaster[*]` number to select the appropriate MxPE field from the region descriptor, a process identifier hit term is formed as:

Equation 50 pid_hit

= ~rgdn.mxpe | ((current_pid[0:7] | rgdn.pidmask[0:7]) ==

(rgdn.pid[0:7] | rgdn.pidmask[0:7]))

where the `current_pid[*]` is the selected process identifier from the current bus master, and `rgdn.pid[*]` and `rgdn.pidmask[*]` are the appropriate process identifier fields from the region descriptor n. For AHB bus masters that do *not* output a process identifier, the MPU forces the `pid_hit` term to be asserted.

As shown in *Figure 700*, the access evaluation macro actually forms the logical complement (*hit_b*) of the combined region_hit and pid_hit boolean equations.

32.7.1.2 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. Using the AHB `hmaster[*]` and `hprot[1]` (supervisor/user mode) signals, a set of effective permissions (`eff_rgd[r,w,x]`) is generated from the appropriate fields in the region descriptor. The protection violation logic then evaluates the access against the effective permissions using the specification shown in *Table 545*.

Table 545. Protection violation definition

Description	Inputs					Output
	hwrite	hprot[0]	eff_rgd[r]	eff_rgd[w]	eff_rgd[x]	Protection Violation?
inst fetch read	0	0	—	—	0	yes, no x permission
inst fetch read	0	0	—	—	1	no, access is allowed
data read	0	1	0	—	—	yes, no r permission
data read	0	1	1	—	—	no, access is allowed
data write	1	—	—	0	—	yes, no w permission
data write	1	—	—	1	—	no, access is allowed

The resulting boolean equation for the processor protection violations is:

Equation 51 cpu_protection_violation

= ~hwrite & ~hprot[0] & ~eff_rgdn[x]// ifetch & no x

| ~hwrite & hprot[0] & ~eff_rgdn[r]// data_read & no r

| hwrite & ~eff_rgdn[w] // data_write& no w

The resulting boolean equation for the non-processor protection violations is:

Equation 52 protection_violation

= ~hwrite & ~eff_rgdn[r]// data_read & no r

| hwrite & ~eff_rgdn[w] // data_write& no w

As shown in *Figure 700*, the output of the protection violation logic is the *error* signal, that is, *error* = protection_violation.

The access evaluation macro then uses the *hit_b* and *error* signals to form two outputs. The combined *(hit_b | error)* signal is used to signal the current access is not allowed and *(~hit_b & error)* is used as the input to MPU_EDRn (error detail register) in the event of an error.

The critical timing arc through the access evaluation macro involves the delay from the arrival of haddr[*] through the two magnitude comparators and through the *hit_b* generation as shown in *Figure 701*.

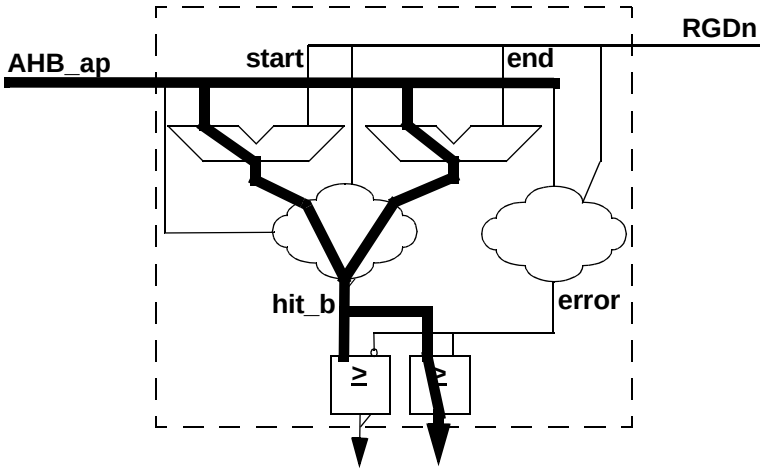


Figure 701. Access evaluation macro critical timing path

32.7.2 Putting it all together and AHB error terminations

For each AHB slave port being monitored, the MPU performs a **reduction-AND** of all the individual *(hit_b | error)* terms from each access evaluation macro. This expression then terminates the bus cycle with an error and reports a protection error for three conditions:

1. If the access does not hit in *any* region descriptor, a protection error is reported.
2. If the access hits in a single region descriptor and that region signals a protection violation, then a protection error is reported.
3. If the access hits in multiple (overlapping) regions and *all* regions signal protection violations, then a protection error is reported.

The third condition reflects that priority is given to *permission granting over access denying* for overlapping regions as this approach provides more flexibility to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see *Section 32.9, Application information*.

The handling of the AHB bus cycle with a protection error requires two distinct actions. First, recall the protection error logic resides within the AHB address phase pipeline stage. In the event of a protection error, the reference must be inhibited from entering the AHB data phase for the targeted slave device. Stated differently, the reference's address phase must be aborted as viewed by the targeted slave device. This is accomplished by dynamically revising the htrans[*] signal sent to the slave device. The htrans[*] attribute signals

an idle or valid reference (as a non-sequential or sequential access), and this signal is “intercepted” by the MPU. If the access is allowed, then the MPU simply passes the original `htrans[*]` value to the slave device. However if a protection error is detected, then the MPU forces `htrans[*]` sent to the slave to the IDLE encoding. This effectively cancels the transaction before it is seen by the slave device.

While forcing `htrans[*] = IDLE` effectively prevents the bus cycle from being transmitted to the slave device, the AHB transaction has already been “committed” in other portions of the platform, namely in the initiating bus master and the crossbar switch. To properly terminate the bus cycle for these modules, it is necessary to post the standard 2-cycle AHB error response. For this response, the `hresp[*]` signal is forced to the ERROR encoding for 2 cycles, the first with `hready` negated and the second with `hready` asserted. To perform these termination functions, the MPU “intercepts” the `hready` and `hresp[*]` signals from the slave devices, performs the required logic functions to force the response on the bus cycle with a protection error, and then outputs the adjusted values to the crossbar switch.

32.8 Initialization information

The reset state of MPU_CESR[VLD] disables the entire module. Recall while the MPU is disabled, all accesses from all bus masters are allowed. This state also minimizes the power dissipation of the MPU. The power dissipation of each access evaluation macro is minimized when the associated region descriptor is marked as invalid or when MPU_CESR[VLD] = 0.

Typically the appropriate number of region descriptors (MPU_RGDn) are loaded at system startup, including the setting of the MPU_RGDn.Word3[VLD] bits, *before* MPU_CESR[VLD] is set, enabling the module. This approach allows all the loaded region descriptors to be enabled simultaneously. Recall if a memory reference does not hit in *any* region descriptor, the attempted access is terminated with an error.

32.9 Application information

In an operational system, interfacing with the MPU can generally be classified into the following activities:

1. Creation of a new memory region requires loading the appropriate region descriptor into an available register location. When a new descriptor is loaded into a RGDn, it would typically be performed using four 32-bit word writes. As discussed in *Section 32.6.4.4, 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.
5. When the MPU detects an access error, the current AHB bus cycle is terminated with an error response and information on the faulting reference captured in the MPU_EARn and MPU_EDRn registers. The error-terminated AHB bus cycle typically initiates some type of error response in the originating bus master. For example, a processor core may respond with a bus error exception, while a data movement bus master may respond with an error interrupt. In any event, the processor can retrieve the captured error address and detail information simply be reading the MPU_E{A,D}Rn registers. Information on which error registers contain captured fault data is signaled by MPU_CESR[SPERR].
6. Finally, consider the use of overlapping region descriptors. Application of overlapping regions can often reduce the number of descriptors required for a given set of access controls. It is important to note that, in the overlapping memory space, the protection rights of the corresponding region descriptors are logically summed together (the boolean OR operator). In the following example of a dual-core system, there are four bus masters: the two processors (CP0, CP1) and two DMA engines (DMA1, a traditional data movement engine transferring data between RAM and peripherals and DMA2, a second engine transferring data to/from the RAM only).
   Consider the region descriptor assignments in *Table 546*. In this example, there are 8 descriptors used to span 9 regions in the three main spaces of the system memory map (flash, RAM and IPS peripheral space). Each region indicates the specific permissions for each of the four bus masters and this definition provides an appropriate set of shared, private and executable memory spaces.

   Of particular interest are the two overlapping spaces: region descriptors 2 & 3 and 3 & 4.

   The space defined by RGD2 with no overlap is a private data and stack area that provides read/write access to CP0 only. The overlapping space between RGD2 and RGD3 defines a shared data space for passing data from CP0 to CP1 and the access controls are defined by the logical OR of the two region descriptors. Thus, CP0 has (`rw- | r--`) = (`rw-`) permissions, while CP1 has (`--- | r--`) = (`r--`) permission in this space. Both DMA engines are excluded from this shared processor data region.
   The overlapping spaces between RGD3 and RGD4 defines another shared data space, this one for passing data from CP1 to CP0. For this overlapping space, CP0 has (`r-- | ---`) = (`r--`) permission, while CP1 has (`rw- | r--`) = (`rw-`) permission. The non-overlapped space of RGD4 defines a private data and stack area for CP1 only.

   The space defined by RGD5 is a shared data region, accessible by all four bus masters. Finally, the slave peripheral space mapped onto the IPS bus is partitioned into two regions: one containing the MPU's programming model accessible only to the two processor cores and the remaining peripheral region accessible to both processors and the traditional DMA1 master.

   This simple example is intended to show one possible application of the capabilities of the Memory Protection Unit in a typical system.

Table 546. Overlapping region descriptor example

Region description	RGDn	CP0	CP1	DMA1	DMA2	System space
CP0 code	0	rwx	r--	--	--	Flash memory
CP1 code	1	r--	rwx	--	--	
CP0 data and stack	2	rw-	---	--	--	SRAM
CP0 → CP1 shared data	3	r--	r--	--	--	
CP1 → CP0 shared data						
CP1 data and stack	4	---	rw-	--	--	
Shared DMA data	5	rw-	rw-	rw	rw	
MPU	6	rw-	rw-	--	--	IPS
Peripherals	7	rw-	rw-	rw	--	

33 Mode Entry Module (MC_ME)

33.1 Introduction

33.1.1 Overview

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

Figure 702 depicts the MC_ME block diagram.

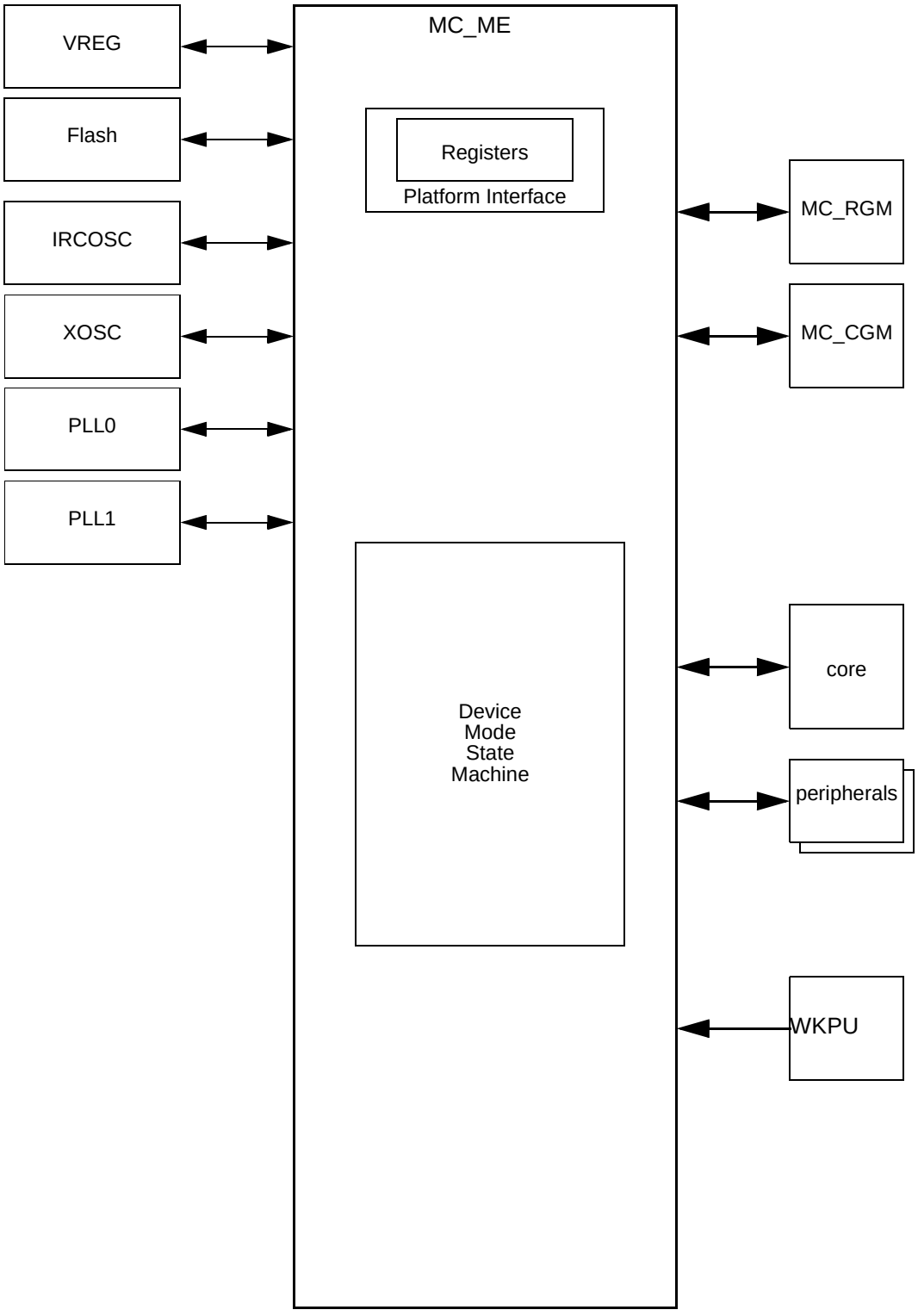


Figure 702. MC_ME block diagram

33.1.2 Features

The MC_ME includes the following features:

- control of the available modes by the **ME_ME** register
- definition of various device mode configurations by the **ME_<*mode*>_MC** registers
- control of the actual device mode by the **ME_MCTL** register
- capture of the current mode and various resource status within the contents of the **ME_GS** register
- optional generation of various mode transition interrupts
- status bits for each cause of invalid mode transitions
- peripheral clock gating control based on the **ME_RUN_PC0…7**, **ME_LP_PC0…7**, and **ME_PCTL0…143** registers
- capture of current peripheral clock gated/enabled status

33.1.3 Modes of operation

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

Table 547 describes the MC_ME modes.

Table 547. MC_ME mode descriptions

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

Table 547. MC_ME mode descriptions (continued)

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

33.2 External signal description

The MC_ME has no connections to any external pins.

33.3 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

33.3.1 Memory map

Table 548. 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 1093*
0xC3FD_C004	ME_MCTL	Mode Control	word	read	read/write	read/write	*on page 1095*

Table 548. MC_ME register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C008	ME_ME	Mode Enable	word	read	read/write	read/write	on page 1096
0xC3FD_C00C	ME_IS	Interrupt Status	word	read	read/write	read/write	on page 1098
0xC3FD_C010	ME_IM	Interrupt Mask	word	read	read/write	read/write	on page 1099
0xC3FD_C014	ME_IMTS	Invalid Mode Transition Status	word	read	read/write	read/write	on page 1100
0xC3FD_C018	ME_DMTS	Debug Mode Transition Status	word	read	read	read	on page 1101
0xC3FD_C020	ME_RESET_MC	***RESET*** Mode Configuration	word	read	read	read	on page 1104
0xC3FD_C024	ME_TEST_MC	***TEST*** Mode Configuration	word	read	read/write	read/write	on page 1105
0xC3FD_C028	ME_SAFE_MC	***SAFE*** Mode Configuration	word	read	read/write	read/write	on page 1105
0xC3FD_C02C	ME_DRUN_MC	***DRUN*** Mode Configuration	word	read	read/write	read/write	on page 1106
0xC3FD_C030	ME_RUN0_MC	***RUN0*** Mode Configuration	word	read	read/write	read/write	on page 1106
0xC3FD_C034	ME_RUN1_MC	***RUN1*** Mode Configuration	word	read	read/write	read/write	on page 1106
0xC3FD_C038	ME_RUN2_MC	***RUN2*** Mode Configuration	word	read	read/write	read/write	on page 1106
0xC3FD_C03C	ME_RUN3_MC	***RUN3*** Mode Configuration	word	read	read/write	read/write	on page 1106
0xC3FD_C040	ME_HALT0_MC	***HALT0*** Mode Configuration	word	read	read/write	read/write	on page 1107
0xC3FD_C048	ME_STOP0_MC	***STOP0*** Mode Configuration	word	read	read/write	read/write	on page 1107
0xC3FD_C060	ME_PS0	Peripheral Status 0	word	read	read	read	on page 1109
0xC3FD_C064	ME_PS1	Peripheral Status 1	word	read	read	read	on page 1109
0xC3FD_C068	ME_PS2	Peripheral Status 2	word	read	read	read	on page 1110
0xC3FD_C080	ME_RUN_PC0	Run Peripheral Configuration 0	word	read	read/write	read/write	on page 1110
0xC3FD_C084	ME_RUN_PC1	Run Peripheral Configuration 1	word	read	read/write	read/write	on page 1110

Table 548. MC_ME register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
...							
0xC3FD_C09C	ME_RUN_PC7	Run Peripheral Configuration 7	word	read	read/write	read/write	on page 1110
0xC3FD_C0A0	ME_LP_PC0	Low-Power Peripheral Configuration 0	word	read	read/write	read/write	on page 1111
0xC3FD_C0A4	ME_LP_PC1	Low-Power Peripheral Configuration 1	word	read	read/write	read/write	on page 1111
...							
0xC3FD_C0BC	ME_LP_PC7	Low-Power Peripheral Configuration 7	word	read	read/write	read/write	on page 1111
0xC3FD_C0C4	ME_PCTL4	DSPI0 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0C5	ME_PCTL5	DSPI1 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0C6	ME_PCTL6	DSPI2 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0D0	ME_PCTL16	FlexCAN0 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0D1	ME_PCTL17	FlexCAN1 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0D8	ME_PCTL24	FlexRay Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0E0	ME_PCTL32	ADC0 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0E1	ME_PCTL33	ADC1 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0E3	ME_PCTL35	CTU Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0E6	ME_PCTL38	eTimer0 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0E7	ME_PCTL39	eTimer1 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0E8	ME_PCTL40	eTimer2 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0E9	ME_PCTL41	FlexPWM0 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0EA	ME_PCTL42	FlexPWM1 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0F0	ME_PCTL48	LIN_FLEX0 Control	byte	read	read/write	read/write	on page 1112

Table 548. MC_ME register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C0F1	ME_PCTL49	LIN_FLEX1 Control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0FA	ME_PCTL58	CRC control	byte	read	read/write	read/write	on page 1112
0xC3FD_C0FE	ME_PCTL62	SWG control	byte	read	read/write	read/write	on page 1112
0xC3FD_C11C	ME_PCTL92	PIT control	byte	read	read/write	read/write	on page 1112

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 549. MC_ME memory map

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C000	ME_GS	R	S_CURRENT_MODE				S_MTRANS	1	0	0	S_PDO	0	0	S_MVR	reserved		S_FLA	
		W																
		R	0	0	0	0	0	0	0	0	S_PLL1	S_PLL0	S_XOSC	S_IRCOSC	S_SYSCLK			
		W																
0xC3FD_C004	ME_MCTL	R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1
		W	KEY															
0xC3FD_C008	ME_ME	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0 (cut1) RESET_DEST (cut2/3)	0	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET (cut1) RESET_FUNC (cut2/3)
		W																

Table 549. MC_ME memory map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C00C	ME_IS (cut1)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	I_ICONF	I_IMODE	I_SAFE	I_MTC
		W													w1c	w1c	w1c	w1c
	ME_IS (cut2/3)	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	0	M_ICONF	M_IMODE	M_SAFE	M_MTC
		W																
0xC3FD_C014	ME_IMTS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	S_MTI	S_MRI	S_DMA	S_NMA	S_SEA
		W												w1c	w1c	w1c	w1c	w1c
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	IRCOSC_SC	SCSRC_SC	SYSCLK_SW	reserved	FLASH_SC	CDP_PRPH_0_143	0	0			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	reserved		FLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																

Table 549. MC_ME memory map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C024	ME_TEST_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	reserved		FLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C028	ME_SAFE_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	reserved		FLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C02C	ME_DRUN_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	reserved		FLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C030 … 0xC3FD_C03C	ME_RUN0…3_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	reserved		FLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C040	ME_HALT0_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	reserved		FLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C044	reserved																	

Table 549. MC_ME memory map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C048	ME_STOP0_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	reserved		FLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C04C ... 0xC3FD_C05C	reserved																	
0xC3FD_C060	ME_PS0	R	0	0	0	0	0	0	0	S_FlexRay	0	0	0	0	0	0	S_FlexCAN1	S_FlexCAN0
		W																
		R	0	0	0	0	0	0	0	0	0	S_DSPI2	S_DSPI1	S_DSPI0	0	0	0	0
		W																
0xC3FD_C064	ME_PS1	R	0	S_SWG	0	0	0	S_CRC	0	0	0	0	0	0	0	0	S_LIN_FLEX1	S_LIN_FLEX0
		W																
		R	0	0	0	0	0	S_FlexPWM1	S_FlexPWM0	S_eTimer2	S_eTimer1	S_eTimer0	0	0	S_CTU	0	S_ADC1	S_ADC0
		W																
0xC3FD_C068	ME_PS2	R	0	0	0	S_PIT	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FD_C06C	reserved																	
0xC3FD_C070	reserved																	
0xC3FD_C074 ... 0xC3FD_C07C	reserved																	

Table 549. MC_ME memory map (continued)

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

1. There is space in the register map for 144 peripherals. Please see *Table 548* for the ME_PCTLn locations actually occupied. The unoccupied locations contain a read-only byte value of 0x00.

33.3.2 Register description

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

33.3.2.1 Global Status Register (ME_GS)

Figure 703. Global Status Register (ME_GS)

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

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R									S_PLL1	S_PLL0	S_XOSC	S_IRCOSC	S_SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register contains global mode status.

Table 550. Global Status Register (ME_GS) field descriptions

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

Table 550. Global Status Register (ME_GS) field descriptions (continued)

Field	Description
S_PDO	**Output power-down status** — This bit specifies output power-down status of I/Os. This bit is asserted whenever outputs of pads are forced to high impedance state or the pads power sequence driver is switched off. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled 1 In ***SAFE***/***TEST*** modes, outputs of pads are forced to high impedance state and the pads power sequence driver is disabled. The inputs are level unchanged. In ***STOP0*** mode, only the pad power sequence driver is disabled, but the state of the output remains functional.
S_MVR	**Main voltage regulator status** 0 Main voltage regulator is not ready 1 Main voltage regulator is ready for use
S_FLA	Flash **availability status** 00 Flash is not available 01 Flash is in power-down mode 10 Flash is in low-power mode 11 Flash is in normal mode and available for use
S_PLL1	**secondary FMPLL status** 0 secondary FMPLL is not stable 1 secondary FMPLL is providing a stable clock
S_PLL0	**system FMPLL status** 0 system FMPLL is not stable 1 system FMPLL is providing a stable clock
S_XOSC	**4-40 MHz crystal oscillator status** 0 4-40 MHz crystal oscillator is not stable 1 4-40 MHz crystal oscillator is providing a stable clock
S_IRCOSC	**16 MHz internal RC oscillator status** 0 16 MHz internal RC oscillator is not stable 1 16 MHz internal RC oscillator is providing a stable clock
S_SYSCLK	**System clock switch status** — These bits specify the system clock currently used by the system. 0000 16 MHz int. RC osc. 0001 reserved 0010 4–40 MHz crystal osc. 0011 reserved 0100 system 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

33.3.2.2 Mode Control Register (ME_MCTL)

Figure 704. Mode Control Register (ME_MCTL)

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

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

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

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

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

Table 551. Mode Control Register (ME_MCTL) Field Descriptions

Field	Description
TARGET_MODE	**Target device mode** — These bits provide the target device mode to be entered by software programming. The mechanism to enter into any mode by software requires the write operation twice: first time with key, and second time with inverted key. These bits are automatically updated by hardware while entering ***SAFE*** on hardware request. Also, while exiting from the ***HALT0*** and ***STOP0*** modes on hardware exit events, these are updated with the appropriate ***RUN0...3*** mode value. 0000 ***RESET*** 0001 ***TEST*** 0010 ***SAFE*** 0011 ***DRUN*** 0100 ***RUN0*** 0101 ***RUN1*** 0110 ***RUN2*** 0111 ***RUN3*** 1000 ***HALT0*** 1001 reserved 1010 ***STOP0*** 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 Cut1: Reserved cut2/3: ***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)

33.3.2.3 Mode Enable Register (ME_ME)

Figure 705. Mode Enable Register (ME_ME) for cut1

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

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET (cut1)
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 706. Mode Enable Register (ME_ME) for cut2/3

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	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET_FUNC (cut2/3)
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

This register allows a way to disable the device modes which are not required for a given device. ***RESET (for cut1), RESET_DEST & RESET_FUNC (for cut2/3)***, ***SAFE***, ***DRUN***, and ***RUN0*** modes are always enabled.

Table 552. Mode Enable Register (ME_ME) Field Descriptions

Field	Description
RESET_DEST (cut2/3 only)	***'Destructive' RESET* mode enable** 1 ***'Destructive' RESET*** mode is enabled
STOP0	***STOP0* mode enable** 0 ***STOP0*** mode is disabled 1 ***STOP0*** mode is enabled
HALT0	***HALT0* mode enable** 0 ***HALT0*** mode is disabled 1 ***HALT0*** mode is enabled
RUN3	***RUN3* mode enable** 0 ***RUN3*** mode is disabled 1 ***RUN3*** mode is enabled
RUN2	***RUN2* mode enable** 0 ***RUN2*** mode is disabled 1 ***RUN2*** mode is enabled
RUN1	***RUN1* mode enable** 0 ***RUN1*** mode is disabled 1 ***RUN1*** mode is enabled
RUN0	***RUN0* mode enable** 0 ***RUN0*** mode is disabled 1 ***RUN0*** mode is enabled
DRUN	***DRUN* mode enable** 0 ***DRUN*** mode is disabled 1 ***DRUN*** mode is enabled
SAFE	***SAFE* mode enable** 0 ***SAFE*** mode is disabled 1 ***SAFE*** mode is enabled
TEST	***TEST* mode enable** 0 ***TEST*** mode is disabled 1 ***TEST*** mode is enabled
RESET (cut1)	***RESET* mode enable** 0 ***RESET*** mode is disabled 1 ***RESET*** mode is enabled
RESET_FUNC (cut2/3)	***'Functional' RESET* mode enable** 1 ***'Functional' RESET*** mode is enabled

33.3.2.4 Interrupt Status Register (ME_IS)

Figure 707. Interrupt Status Register (ME_IS) for cut1

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	0	I_ICONF	I_IMODE	I_SAFE	I_MTC
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 708. Interrupt Status Register (ME_IS) for cut2/3

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

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

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

This register provides the current interrupt status.

Table 553. Interrupt Status Register (ME_IS) field descriptions

Field	Description
I_ICONF_CU (cut2/3 only)	**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

Table 553. Interrupt Status Register (ME_IS) field descriptions (continued)

Field	Description
I_IMODE	**Invalid mode interrupt** — This bit is set whenever an invalid mode transition is requested. It is cleared by writing a '1' to this bit. 0 No invalid mode interrupt occurred 1 Invalid mode interrupt is pending
I_SAFE	***SAFE* mode interrupt** — This bit is set whenever the device enters ***SAFE*** mode on hardware requests generated in the system. It is cleared by writing a '1' to this bit. 0 No ***SAFE*** mode interrupt occurred 1 ***SAFE*** mode interrupt is pending
I_MTC	**Mode transition complete interrupt** — This bit is set whenever the mode transition process completes (S_MTRANS transits from 1 to 0). It is cleared by writing a '1' to this bit. This mode transition interrupt bit will not be set while entering low-power modes ***HALT0***, or ***STOP0***. 0 No mode transition complete interrupt occurred 1 Mode transition complete interrupt is pending

33.3.2.5 Interrupt Mask Register (ME_IM)

Figure 709. Interrupt Mask Register (ME_IM) for cut1

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	0	M_ICONF	M_IMODE	M_SAFE	M_MTC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 710. Interrupt Mask Register (ME_IM) for cut2/3

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

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

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

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

Table 554. Interrupt Mask Register (ME_IM) field descriptions

Field	Description
M_CONF_CU (cut2/3 only)	**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

33.3.2.6 Invalid Mode Transition Status Register (ME_IMTS)

Figure 711. Invalid Mode Transition Status Register (ME_IMTS)

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

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

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

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

Table 555. 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 33.4.5, Mode Transition Interrupts* for the exceptions to this behavior. It is cleared by writing a '1' to this bit. 0 Mode transition requested is not illegal 1 Mode transition requested is illegal
S_MRI	**Mode Request Illegal status** — This bit is set whenever the target mode requested is not a valid mode with respect to current mode. It is cleared by writing a '1' to this bit. 0 Target mode requested is not illegal with respect to current mode 1 Target mode requested is illegal with respect to current mode
S_DMA	**Disabled Mode Access status** — This bit is set whenever the target mode requested is one of those disabled modes determined by ME_ME register. It is cleared by writing a '1' to this bit. 0 Target mode requested is not a disabled mode 1 Target mode requested is a disabled mode
S_NMA	**Non-existing Mode Access status** — This bit is set whenever the target mode requested is one of those non existing modes determined by ME_ME register. It is cleared by writing a '1' to this bit. 0 Target mode requested is an existing mode 1 Target mode requested is a non-existing mode
S_SEA	***SAFE* Event Active status** — This bit is set whenever the device is in ***SAFE*** mode, ***SAFE*** event bit is pending and a new mode requested other than ***RESET***/***SAFE*** modes. It is cleared by writing a '1' to this bit. 0 No new mode requested other than ***RESET***/***SAFE*** while ***SAFE*** event is pending 1 New mode requested other than ***RESET***/***SAFE*** while ***SAFE*** event is pending

33.3.2.7 Debug Mode Transition Status Register (ME_DMTS)

Figure 712. Debug Mode Transition Status Register (ME_DMTS)

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

	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	IRCOSC_SC	SCSRC_SC	SYSCLK_SW	reserved	FLASH_SC	CDP_PRPH_0_143	0	0			CDP_PRPH_64_95	CDP_PRPH_32_63	CDP_PRPH_0_31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

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

Table 556. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions

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

Table 556. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions (continued)

Field	Description
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
IRCOSC_SC	IRCOSC State Change during mode transition indicator — This bit is set when the 16 MHz internal RC oscillator is requested to change its power up/down state. It is cleared when the 16 MHz internal RC oscillator has completed its state change. 0 No state change is taking place 1A state change is taking place
SCSRC_SC	Secondary Clock Sources State Change during mode transition indicator — This bit is set when a secondary clock source is requested to change its power up/down state. It is cleared when all secondary system clock sources have completed their state changes. (A 'secondary clock source' is a clock source other than IRCOSC.) 0 No state change is taking place 1A state change is taking place
SYSCLK_SW	System Clock Switching pending status — 0 No system clock source switching is pending 1A system clock source switching is pending
FLASH_SC	FLASH State Change during mode transition indicator — This bit is set when the FLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change. 0 No state change is taking place 1A state change is taking place
CDP_PRPH_0_143	Clock Disable Process Pending status for Peripherals 0…143(1) — This bit is set when any peripheral has been requested to have its clock disabled. It is cleared when all the peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1Clock disabling is pending for at least one peripheral
CDP_PRPH_64_95	Clock Disable Process Pending status for Peripherals 64…95(1) — This bit is set when any peripheral appearing in **ME_PS2** has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1Clock disabling is pending for at least one peripheral

Table 556. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions (continued)

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

1. For cut2/3: Peripheral *n* corresponds to the **ME_PCTL*n*** register. Please see *Table 548* for the **ME_PCTL*n*** locations actually occupied, which in turn indicates which peripherals are reported in the **ME_DMTS** register.

33.3.2.8 *RESET* Mode Configuration Register (ME_RESET_MC)

Figure 713. RESET Mode Configuration Register (ME_RESET_MC)

Address 0xC3FD_C020 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	PDO	0	0	MVRON	reserved		FLAON	
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									PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***RESET*** mode. Please refer to *Table 557* for details.

33.3.2.9 *TEST* Mode Configuration Register (ME_TEST_MC)

Figure 714. 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	reserved		FLAON	
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	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***TEST*** mode. Please refer to *Table 557* for details.

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

33.3.2.10 *SAFE* Mode Configuration Register (ME_SAFE_MC)

Figure 715. 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	reserved		FLAON	
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									PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***SAFE*** mode. Please refer to *Table 557* for details.

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

33.3.2.11 *DRUN* Mode Configuration Register (ME_DRUN_MC)

Figure 716. 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	reserved		FLAON	
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	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***DRUN*** mode. Please refer to *Table 557* for details.

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

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

Figure 717. 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	reserved		FLAON	
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	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***RUN0…3*** modes. Please refer to *Table 557* for details.

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

33.3.2.13 *HALT0* Mode Configuration Register (ME_HALT0_MC)

Figure 718. HALT0 Mode Configuration Register (ME_HALT0_MC)

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

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***HALT0*** mode. Please refer to *Table 557* for details.

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

33.3.2.14 *STOP0* Mode Configuration Register (ME_STOP0_MC)

Figure 719. STOP0 Mode Configuration Register (ME_STOP0_MC)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	reserved		FLAON	
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	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***STOP0*** mode. Please refer to *Table 557* for details.

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

Table 557. Mode Configuration Registers (ME_<*mode*>_MC) Field Descriptions

Field	Description
PDO	**I/O output power-down control** — This bit controls the output power-down of I/Os. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled 1 In ***SAFE***/***TEST*** modes, outputs of pads are forced to high impedance state and pads power sequence driver is disabled. The inputs are level unchanged. In ***STOP0*** mode, only the pad power sequence driver is disabled, but the state of the output remains functional.
MVRON	**Main voltage regulator control** — This bit specifies whether main voltage regulator is switched off or not while entering this mode. 1 Main voltage regulator is switched on
FLAON	Flash **power-down control** — This bit specifies the operating mode of the code flash after entering this mode. 00 reserved 01 Flash is in power-down mode 10 Flash is in low-power mode 11 Flash is in normal mode
PLL1ON	**secondary FMPLL control** 0 secondary FMPLL is switched off 1 secondary FMPLL is switched on
PLL0ON	**system FMPLL control** 0 system FMPLL is switched off 1 system FMPLL is switched on
XOSCON	**4-40 MHz crystal oscillator control** 0 4-40 MHz crystal oscillator is switched off 1 4-40 MHz crystal oscillator is switched on
IRCOSCON	**16 MHz internal RC oscillator control** 0 16 MHz internal RC oscillator is switched off 1 16 MHz internal RC oscillator is switched on
SYSCLK	**System clock switch control** — These bits specify the system clock to be used by the system. 0000 16 MHz int. RC osc. 0001 reserved 0010 4–40 MHz crystal osc. 0011 reserved 0100 system 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 ***TEST*** mode, reserved in all other modes

33.3.2.15 Peripheral Status Register 0 (ME_PS0)

Figure 720. 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								S_FlexRay							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										S_DSPI2	S_DSPI1	S_DSPI0				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of the peripherals. Please refer to *Table 558* for details.

33.3.2.16 Peripheral Status Register 1 (ME_PS1)

Figure 721. 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		S_SWG				S_CRC									S_LIN_FLEX1	S_LIN_FLEX0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R						S_FlexPWM1	S_FlexPWM0	S_eTimer2	S_eTimer1	S_eTimer0			S_CTU		S_ADC1	S_ADC0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of the peripherals. Please refer to *Table 558* for details.

33.3.2.17 Peripheral Status Register 2 (ME_PS2)

Figure 722. 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				S_PIT												
W																
Reset	0	0	0	0	0	0	0	0	0	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of the peripherals. Please refer to *Table 558* for details.

Table 558. Peripheral Status Registers 0…4 (ME_PS0…4) Field Descriptions

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

33.3.2.18 Run Peripheral Configuration Registers (ME_RUN_PC0…7)

Figure 723. Run Peripheral Configuration Registers (ME_RUN_PC0…7)

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

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

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

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

Table 559. 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

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

Figure 724. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7)

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

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

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

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

Table 560. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7) Field Descriptions

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

33.3.2.20 Peripheral Control Registers (ME_PCTL0…143)

Figure 725. Peripheral Control Registers (ME_PCTL0…143)

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

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

These registers select the configurations during run and non-run modes for each peripheral.

Table 561. Peripheral Control Registers (ME_PCTL0…143) Field Descriptions

Field	Description
DBG_F	Peripheral control in debug mode — This bit controls the state of the peripheral in debug mode 0 Peripheral state depends on RUN_CFG/LP_CFG bits and the device mode 1 Peripheral is frozen if not already frozen in device modes. *Note: This feature is useful to freeze the peripheral state while entering debug. For example, this may be used to prevent a reference timer from running while making a debug accesses.*
LP_CFG	**Peripheral configuration select for non-run modes** — These bits associate a configuration as defined in the **ME_LP_PC0…7** registers to the peripheral. 000 Selects ME_LP_PC0 configuration 001 Selects ME_LP_PC1 configuration 010 Selects ME_LP_PC2 configuration 011 Selects ME_LP_PC3 configuration 100 Selects ME_LP_PC4 configuration 101 Selects ME_LP_PC5 configuration 110 Selects ME_LP_PC6 configuration 111 Selects ME_LP_PC7 configuration
RUN_CFG	**Peripheral configuration select for run modes** — These bits associate a configuration as defined in the **ME_RUN_PC0…7** registers to the peripheral. 000 Selects ME_RUN_PC0 configuration 001 Selects ME_RUN_PC1 configuration 010 Selects ME_RUN_PC2 configuration 011 Selects ME_RUN_PC3 configuration 100 Selects ME_RUN_PC4 configuration 101 Selects ME_RUN_PC5 configuration 110 Selects ME_RUN_PC6 configuration 111 Selects ME_RUN_PC7 configuration

*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.*

33.4 Functional description

33.4.1 Mode Transition Request

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

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

Once a valid mode transition request is detected, the target mode configuration information is loaded from the corresponding **ME_<*mode*>_MC** register. The mode transition request may require a number of cycles depending on the programmed configuration, and software should check the **S_CURRENT_MODE** bit field and the **S_MTRANS** bit of the global status register **ME_GS** to verify when the mode has been correctly entered and the transition process has completed. For a description of valid mode requests, please refer to *Section 33.4.5, "Mode Transition Interrupts"*.

Any modification of the mode configuration register of the currently selected mode will not be taken into account immediately but on the next request to enter this mode. This means that transition requests such as ***RUN0…3*** → ***RUN0…3***, ***DRUN*** → ***DRUN***, ***SAFE*** → ***SAFE***, and ***TEST*** → ***TEST*** are considered valid mode transition requests. As soon as the mode request is accepted as valid, the **S_MTRANS** bit is set till the status in the **ME_GS** register matches the configuration programmed in the respective **ME_<*mode*>_MC** register.

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

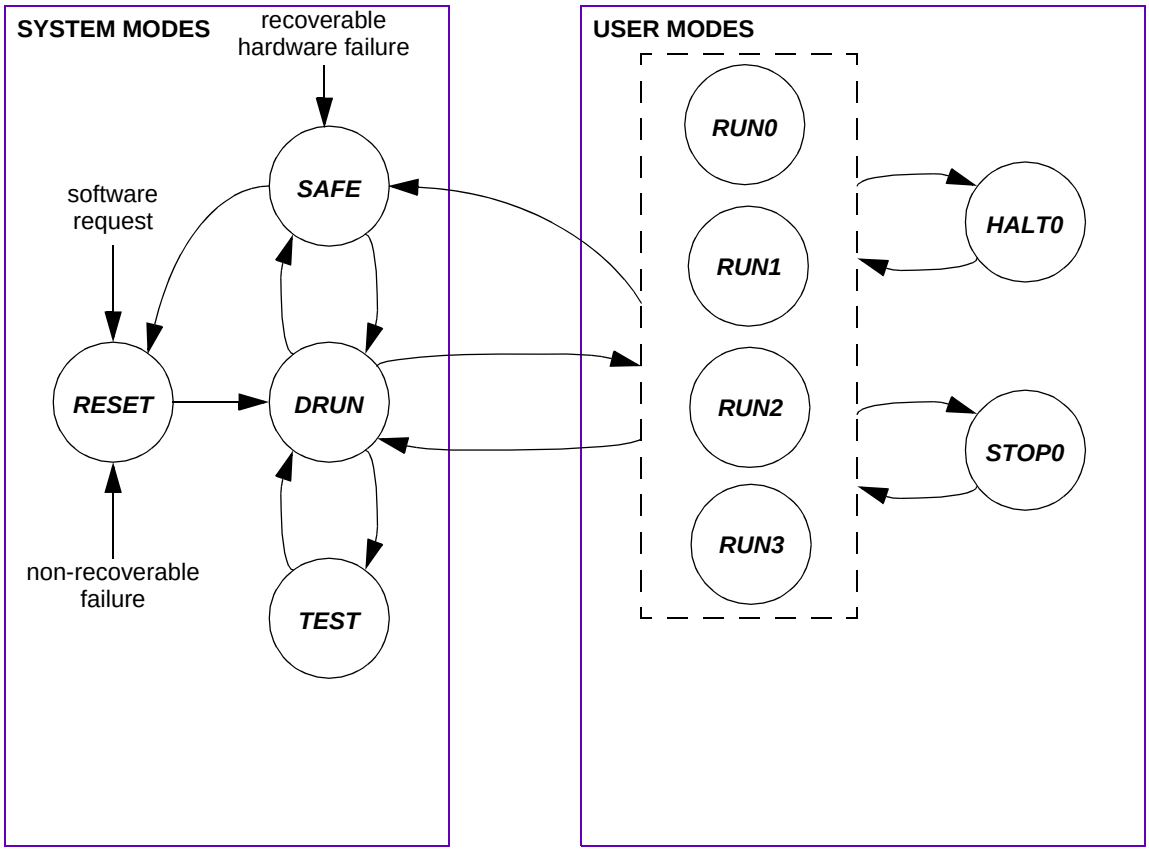


Figure 726. MC_ME Mode Diagram

33.4.2 Mode details

33.4.2.1 *RESET* mode

The device enters this mode on the following events:

- from ***SAFE***, ***DRUN***, ***RUN0…3***, or ***TEST*** mode when the **TARGET_MODE** bit field of the **ME_MCTL** register is written with:
  - Cut1: “0000”
  - cut2/3: 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 16 MHz int. RC osc. is selected as the system clock.

33.4.2.2 *DRUN* mode

The device enters this mode on the following events:

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

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 flash, all clock sources, and the system clock configuration can be controlled by software as required. After system reset, the software execution starts with the default configuration selecting the 16 MHz int. RC osc. as the system clock.

This mode is intended to be used by software

- to initialize all registers as per the system needs

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

33.4.2.3 *SAFE* mode

The device enters this mode on the following events:

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

*Note: For cut2/3: 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 16 MHz int. RC osc. is selected as the system clock.

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

*Note: If software requests to change to the **SAFE** mode and then requests to change back to the parent mode before the mode transition is completed, the device’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 device via the ***DRUN*** mode, or
  - completely reset the device via the ***RESET*** mode.

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

33.4.2.4 *TEST* Mode

The device enters this mode on the following events:

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

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

This mode is intended to be used by software

- to execute software test routines

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

33.4.2.5 *RUN0…3* Modes

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

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

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

These modes are intended to be used by software

- to execute application routines

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

33.4.2.6 *HALT0* Mode

The device enters this mode on the following events:

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

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

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

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

- the core clock frozen
- only a few peripherals running

and to be used by software

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

33.4.2.7 *STOP0* mode

The device enters this mode on the following events:

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

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

- the system FMPLL
- the secondary FMPLL

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

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

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

- the core clock frozen
- almost all peripherals stopped

and to be used by software

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

This mode can be used to stop all clock sources and thus preserve the device status. When exiting the ***STOP0*** mode, the 16 MHz internal RC oscillator clock is selected as the system clock until the target clock is available.

*Note: For cut2/3: It is good practice for software to ensure that the **S_MTRANS** bit in the **ME_GS** register has been cleared on **STOP0** mode exit to ensure that the previous **RUN0…3** mode configuratoin has been fully restored before executing critical code.*

33.4.3 Mode transition process

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

33.4.3.1 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. Refer to *Section 33.4.5, 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.

33.4.3.2 Target Mode Configuration Loading

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

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

Table 562. MC_ME Resource Control Overview

Resource	Mode						
	RESET	***TEST***	***SAFE***	***DRUN***	***RUN0…3***	***HALT0***	***STOP0***
IRCOSC	on	√ on	on	on	on	√ on	√ on
XOSC	off	√ off	off	√ off	√ off	√ off	√ off
PLL0	off	√ off	off	√ off	√ off	√ off	off
PLL1	off	√ off	off	√ off	√ off	√ off	√ off
FLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down
MVREG	on	on	on	on	on	√ on	√ on

Table 562. MC_ME Resource Control Overview (continued)

Resource	Mode						
	RESET	***TEST***	***SAFE***	***DRUN***	***RUN0…3***	***HALT0***	***STOP0***
PDO	off	√ off	√ on	off	off	off	√ off

33.4.3.3 Peripheral Clocks Disable

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

- the peripheral is configured to be disabled via the target mode, the peripheral configuration registers **ME_RUN_PC0…7** and **ME_LP_PC0…7**, and the peripheral control registers **ME_PCTL0…143**

Caution: For cut1: The MC_ME does not automatically request peripherals to enter their stop modes if the power domains in which they are residing are to be turned off due to a mode change. Therefore, it is software's responsibility to ensure that those peripherals that are to be powered down are configured in the MC_ME to be frozen.

Note: For cut2/3: 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 33.4.6, "Peripheral Clock Gating"* for more details.

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

33.4.3.4 Processor Low-Power Mode Entry

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

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

33.4.3.5 Processor and System Memory Clock Disable

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

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

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

33.4.3.6 Clock Sources Switch-On

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

- the 16 MHz internal RC oscillator
- the 4-40 MHz crystal oscillator
- the system FMPLL
- the secondary FMPLL

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.

33.4.3.7 Flash Module Switch-On

On completion of the step, if the flash needs to be switched to normal mode from its low-power or power-down mode based on the **FLAON** bit field of the **ME_<*current mode*>_MC** and **ME_<*target mode*>_MC** registers, the MC_ME requests the flash to exit from its low-power/power-down mode. When the flash is available for access, the **S_FLA** bit field of the **ME_GS** register is updated to “11” by hardware.

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

33.4.3.8 Pad Outputs-On

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

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

33.4.3.9 Peripheral Clocks Enable

Based on the current and target device modes, the peripheral configuration registers **ME_RUN_PC0…7**, **ME_LP_PC0…7**, and the peripheral control registers **ME_PCTL0…143**, the MC_ME enables the clocks for selected modules as required. This step is executed only after the process is completed.

33.4.3.10 Processor and Memory Clock Enable

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

33.4.3.11 Processor Low-Power Mode Exit

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

33.4.3.12 System clock switching

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

- The target clock configuration for the 16 MHz int. RC osc. takes effect only after the **S_IRCOSC** bit of the **ME_GS** register is set by hardware (i.e. the 16 MHz internal RC oscillator has stabilized).
- For cut2/3: The target clock configuration for the 4-40 MHz crystal osc. takes effect only after the **S_XOSC** bit of the **ME_GS** register is set by hardware (i.e., the 4-40 MHz crystal oscillator has stabilized).
- The target clock configuration for the system FMPLL takes effect only after the **S_PLL0** bit of the **ME_GS** register is set by hardware (i.e. the system 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***TEST*** mode.

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

System clock switching starts only after

- the *Clock Sources Switch-On* process has completed if the target system clock source is one of the following:
  - the 16 MHz internal RC oscillator
  - the system FMPLL
- the *Peripheral Clocks Disable* process has completed in order not to change the system clock frequency before peripherals close their internal activities

An overview of system clock source selection possibilities for each mode is shown in *Table 563*. A ‘√’ indicates that a given clock source is selectable for a given mode.

Table 563. MC_ME system clock selection overview

System Clock Source	Mode						
	RESET	***TEST***	***SAFE***	***DRUN***	***RUN0…3***	***HALT0***	***STOP0***
16 MHz int. RC osc.	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)
For cut2/3: 4-40 MHz crystal osc		√		√	√	√	√
system FMPLL		√		√	√	√	
system clock is disabled		√(1)					

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

33.4.3.13 Pad Switch-Off

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

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

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

33.4.3.14 Clock Sources (with no Dependencies) Switch-Off

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

- the system FMPLL
- the secondary FMPLL

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

33.4.3.15 Clock Sources (with Dependencies) Switch-Off

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

- the 16 MHz internal RC oscillator
- the 4-40 MHz crystal oscillator

This step is executed only after

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

33.4.3.16 Flash Switch-Off

Based on the **FLAON** bit field of the **ME_<*current mode*>_MC** and **ME_<*target mode*>_MC** registers, if the flash is to be put in its low-power or power-down mode, the MC_ME requests the flash to enter the corresponding power mode and waits for the flash to acknowledge. The exact power mode status of the flash is updated in the **S_FLA** bit field of the **ME_GS** register. This step is executed only when the *Processor and System Memory Clock Disable* process has completed.

33.4.3.17 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: For cut2/3: **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.

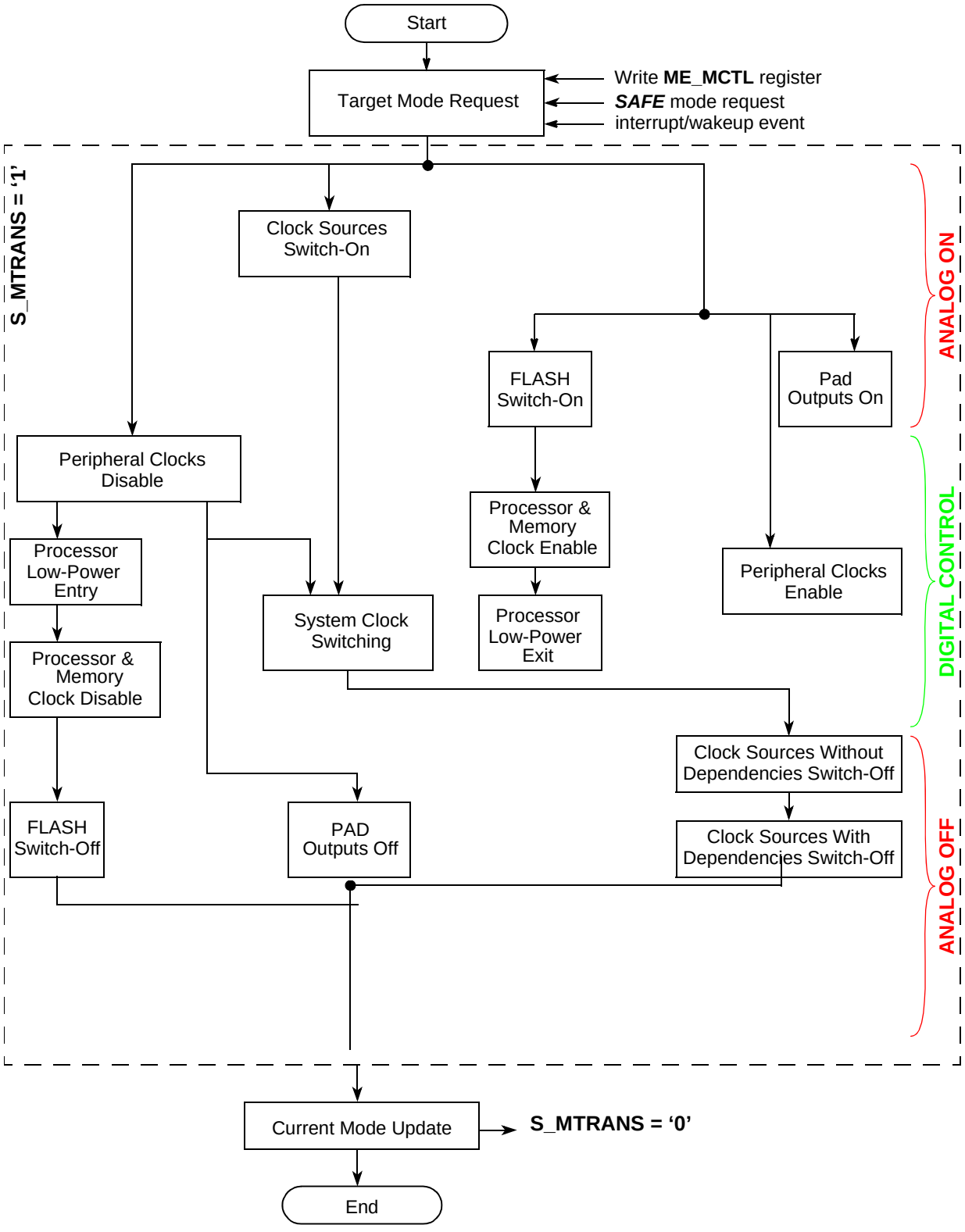


Figure 727. MC_ME Transition Diagram

33.4.4 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 16 MHz int. RC osc. is selected as the system clock, IRCOSC must be on.
- For cut1:
  - If **PLL0ON** is 1, **XOSCON** must also be 1.
  - If **PLL1ON** is 1, **XOSCON** must also be 1.
- For cut2/3:
  - If the 4-40 MHz crystal osc. clock is selected as the system clock, **XOSC** must be on.
  - If the system PLL clock is selected as the system clock, **PLL0** must be on.
  - If **PLL0** is on, **XOSC** must also be on.
  - If **PLL1** is on, **XOSC** must also be on.

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

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

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

33.4.5 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.

33.4.5.1 Invalid Mode Configuration Interrupt

Whenever a write operation is attempted to the **ME_<*mode*>_MC** registers violating the protection rules mentioned in the *Section 33.4.4, Protection of mode configuration registers*, the interrupt pending bit **I_ICONF** of the **ME_IS** register is set and an interrupt request is generated if the mask bit **M_ICONF** of **ME_IM** register is ‘1’.

For cut2/3: In addition, during a mode transition, if a clock source has been configured in the **ME_<*target mode*>_MC** register to be off and a peripheral requiring this clock source to be on has been enabled via the **ME_RUN_PC0…7/ME_LP_PC0…7** and **ME_PCTL*n*** registers, the interrupt pending bit **I_ICONF_CU** of the **ME_IS** register is set and an iterrupt request is generated if the mask bit **M_ICONF_CU** of the **ME_IM** register is ‘1’.

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

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

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

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

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

33.4.5.3 *SAFE* Mode Transition Interrupt

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

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

33.4.5.4 Mode Transition Complete interrupt

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

33.4.6 Peripheral Clock Gating

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

The run peripheral configuration registers **ME_RUN_PC0…7** are chosen only during the software running modes ***DRUN***, ***TEST***, ***SAFE***, and ***RUN0…3***. All configurations are programmable by software according to the needs of the application. Each configuration register contains a mode bit which determines whether or not a peripheral clock is to be gated. Run configuration selection for each peripheral is done by the **RUN_CFG** bit field of the **ME_PCTL0…143** registers.

The low-power peripheral configuration registers **ME_LP_PC0…7** are chosen only during the low-power modes ***HALT0*** and ***STOP0***. All configurations are programmable by software according to the needs of the application. Each configuration register contains a mode bit which determines whether or not a peripheral clock is to be gated. Low-power configuration selection for each peripheral is done by the **LP_CFG** bit field of the **ME_PCTL0…143** registers.

Any modifications to the **ME_RUN_PC0…7**, **ME_LP_PC0…7**, and **ME_PCTL0…143** registers do not affect the clock gating behavior until a new mode transition request is generated.

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

- The clock is gated if the **DBG_F** bit of the associated **ME_PCTL0…143** register is set. Otherwise, the peripheral clock gating status depends on the **RUN_CFG** and **LP_CFG** bits. Any further modifications of the **ME_RUN_PC0…7**, **ME_LP_PC0…7**, and **ME_PCTL0…143** registers during a debug session will take affect immediately without requiring any new mode request.

33.4.7 Application example

Figure 728 shows an example application flow for requesting a mode change and then waiting until the mode transition has completed.

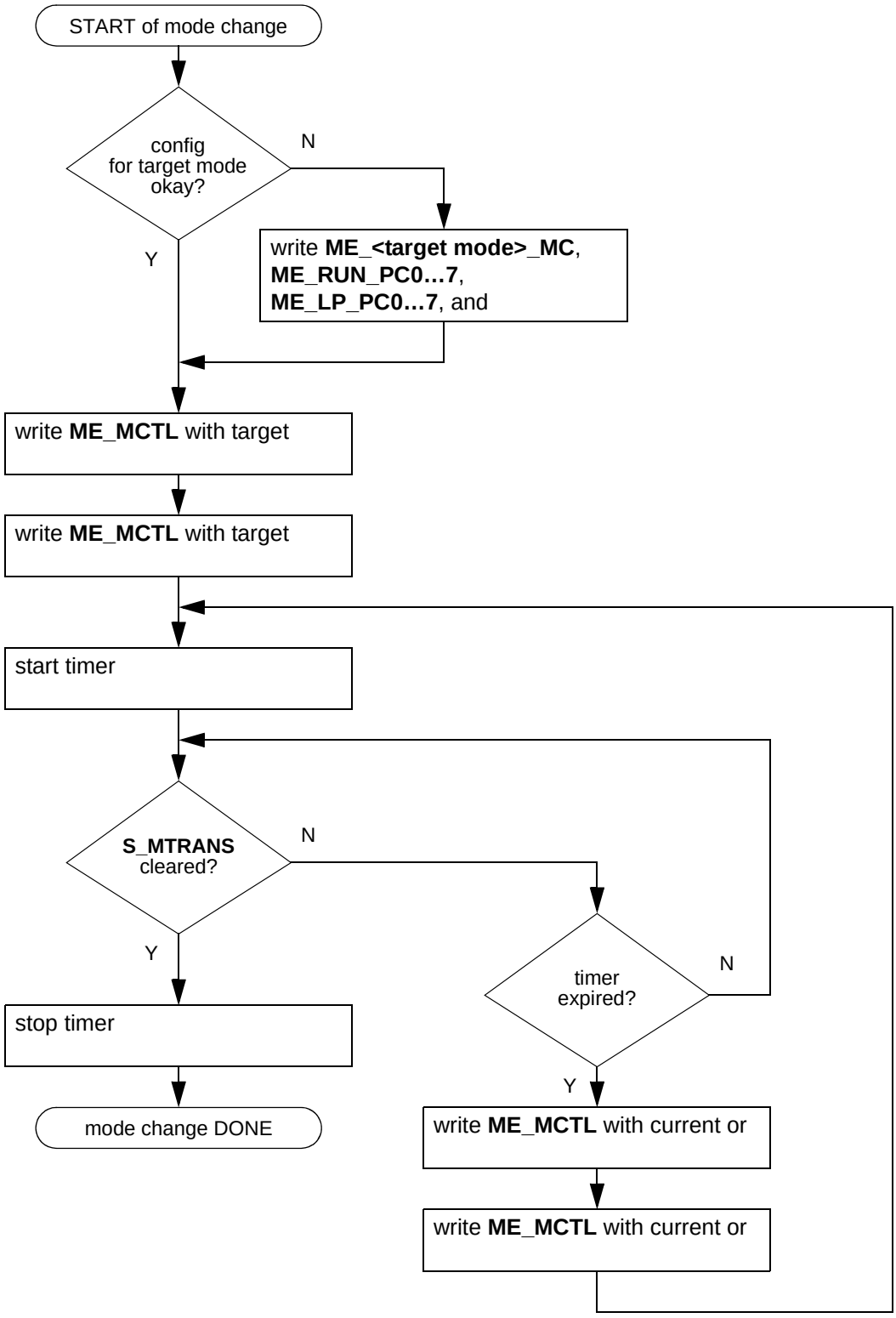


Figure 728. MC_ME Application Example Flow Diagram

34 Nexus Crossbar Slave Port Data Trace Module (NXSS) [cut2/3 only]

34.1 Introduction

The NXSS modules on cut2/3 of this device provide the data trace and watchpoint messaging features defined in the Class 3 IEEE-ISTO 5001-2003 standard.

The SPC56ELxx cut2/3 provides two NXSS modules:

- NXSS_0 can be programmed to trace data accesses to the System SRAM0.
- NXSS_1 can be programmed to trace data accesses to the System SRAM1.

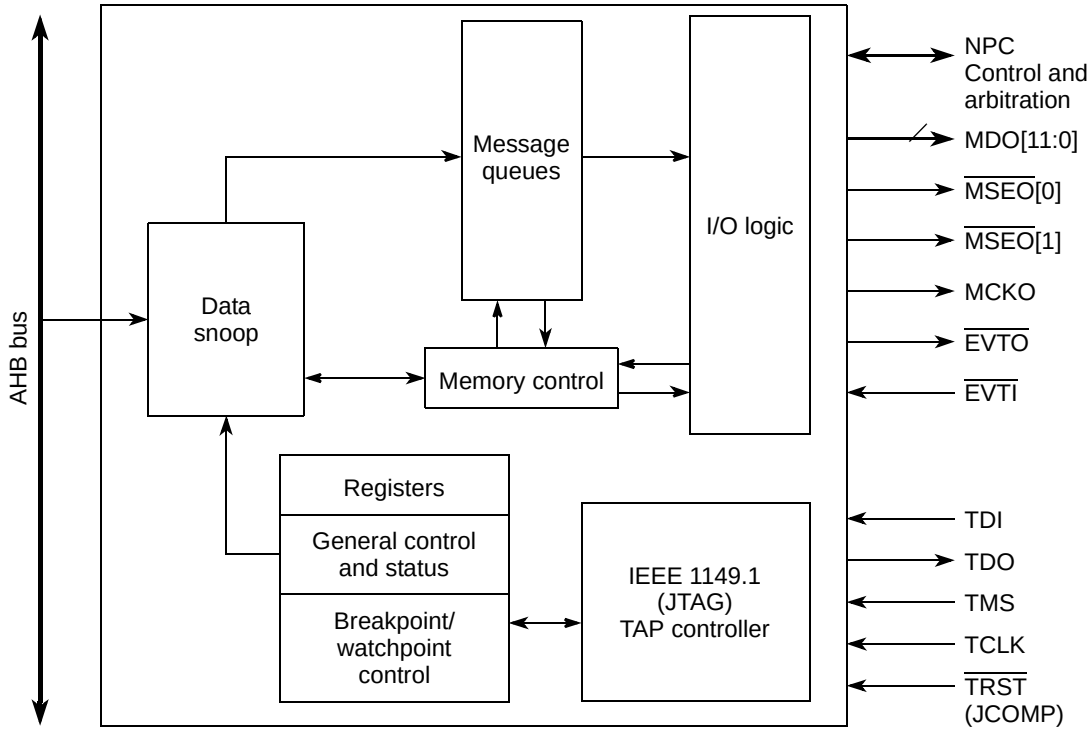
All output messages and register accesses are compliant with the protocol defined in the IEEE-ISTO 5001-2003 standard.

Note: The auxiliary port and its signals, such as MCKO, $\overline{\text{MSEO}}$[1:0], MDO[11:0] and others, are referenced. The device NPC module arbitrates the access of the single auxiliary port. The functions of the NXSS_0 and NXSS_1 modules are described without the interaction of the NPC, as if Nexus has a dedicated auxiliary port, to simplify the description.

34.2 Block diagram

Figure 729 shows a block diagram of the NXSS. In Nexus Full Port mode there are 12 MDOs and in Nexus Reduced Port mode there are 4 MDOs.

Figure 729. NXSS block diagram



34.3 Features

Features include the following:

- Data trace via data write messaging (DWM) and data read messaging (DRM). This provides the capability for the development tool to trace reads and/or writes to System SRAM0 and System SRAM1.
- Watchpoint messaging via the auxiliary pins
- Watchpoint trigger enable of data trace messaging (DTM)
- Registers for data trace, watchpoint generation, and watchpoint trigger
- All features controllable and configurable via the JTAG port

34.4 External signal description

34.4.1 Rules for output messages

The NXSS module observes the same rules for output messages as the NPC.

34.4.2 Auxiliary port arbitration

The NXSS_0 and NXSS_1 modules arbitrate for the shared Nexus port. This arbitration is handled by the NPC and is based on prioritized requests from the NXSS_0, NXSS_1, and the other Nexus clients sharing the port.

34.5 NXSS programmer model

This section describes the programmer model. Nexus registers are accessed using the JTAG port in compliance with IEEE 1149.1.

Table 564. Registers available in the NXSS programmer model

Register[1]	Nexus access opcode	Read address	Write address	Location
Development Control 1 (DC1_*n*)	0x2	0x04	0x05	*on page 1131*
Development Control 2 (DC2_*n*)	0x3	0x06	0x07	*on page 1131*
Watchpoint Trigger (WT_*n*)	0xB	0x16	0x17	*on page 1132*
Data Trace Control (DTC_*n*)	0xD	0x1A	0x1B	*on page 1133*
Data Trace Start Address 1 (DTSA1_*n*)	0xE	0x1C	0x1D	*on page 1134*
Data Trace Start Address 2 (DTSA2_*n*)	0xF	0x1E	0x1F	*on page 1134*
Data Trace End Address 1 (DTEA1_*n*)	0x12	0x24	0x25	*on page 1135*
Data Trace End Address 2 (DTEA2_*n*)	0x13	0x26	0x27	*on page 1135*
Breakpoint/Watchpoint Control Register 1 (BWC1_*n*)	0x16	0x2C	0x2D	*on page 1135*
Breakpoint/Watchpoint Control Register 2 (BWC2_*n*)	0x17	0x2E	0x2F	*on page 1136*
Breakpoint/Watchpoint Address Register 1 (BWA1_*n*)	0x1E	0x3C	0x3D	*on page 1137*
Breakpoint/Watchpoint Address Register 2 (BWA2_*n*)	0x1F	0x3E	0x3F	*on page 1137*

1. All registers have read/write access.

34.5.1 Development Control Registers (DC1 and DC2)

The Development Control Registers (DC1 and DC2) control the basic development features of the NXSS_0 and NXSS_1 modules.

Figure 730. Development Control Register 1 (DC1)

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OPC	0	0	EOC		0	0	WEN	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	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	EIC		TM		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 565. DC1 field descriptions

Field	Description
OPC[1]	Output port mode control. 0 Reduced port mode configuration 1 Full port mode configuration
EOC	$\overline{\text{EVTO}}$ control. 00 $\overline{\text{EVTO}}$ upon occurrence of watchpoint (internal or external) 01 $\overline{\text{EVTO}}$ upon entry into system-level debug mode 1X Reserved
WEN	Watchpoint trace enable. 0 Watchpoint messaging disabled 1 Watchpoint messaging enabled.
EIC	$\overline{\text{EVTI}}$ control. 00 $\overline{\text{EVTI}}$ for synchronization (Data Trace) 01 Reserved 10 $\overline{\text{EVTI}}$ disabled for this module 11 Reserved
TM	Trace mode. 000 No Trace 1XX Reserved X1X Data trace enabled XX1 Reserved

1. The output port mode control bit (OPC) is shown for clarity. This function is controlled globally by the NPC port control register (PCR).

Figure 731. Development Control Register 2 (DC2)

Access: R/W

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

Table 566. DC2 field description

Field	Description
EWC[1]	$\overline{\text{EVTO}}$ Watchpoint Configuration 00000000 = No watchpoints trigger $\overline{\text{EVTO}}$ 1XXXXXXX = Reserved X1XXXXXX = Reserved XX1XXXXX = Reserved XXX1XXXX = Reserved XXXX1XXX = Internal watchpoint #1 triggers $\overline{\text{EVTO}}$ XXXXX1XX = Internal watchpoint #2 triggers $\overline{\text{EVTO}}$ XXXXXX1X = Reserved XXXXXXX1 = Reserved

1. The EOC bits in DC1 must be programmed to trigger $\overline{\text{EVTO}}$ on watchpoint occurrence for the EWC bits to have any effect.

34.5.2 Watchpoint Trigger Register (WT)

The Watchpoint Trigger Register (WT) allows the watchpoints defined internally to the NXSS_0 and NXSS_1 modules to trigger actions. These watchpoints can control data trace enable and disable. The WT bits can be used to produce an address related window for triggering trace messages.

Figure 732. Watchpoint Trigger Register (WT)

Access: R/W

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

Table 567. WT field descriptions

Field	Description
DTS	Data trace start control. 000 Trigger disabled 001–100 Reserved 101 Use internal watchpoint #1 (BWA1 register) 110 Use internal watchpoint #2 (BWA2 register) 111 Reserved
DTE	Data trace end control 000 Trigger disabled 001–100 Reserved 101 Use internal watchpoint #1 (BWA1 register) 110 Use internal watchpoint #2 (BWA2 register) 111 Reserved

Note: The WT bits only enable data trace if the TM bits within the Development Control register (DC) have not already been set to enable data trace.

34.5.3 Data Trace Control Register (DTC)

The Data Trace Control Register (DTC) controls whether DTM Messages are restricted to reads, writes or both for a user programmable address range. There are two data trace channels controlled by the DTC for the NXSS_0 and NXSS_1 modules.

Figure 733. Data Trace Control Register (DTC)

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RWT1		RWT2		0	0	0	0	0	0	0	0	AMID				NSID				0	0	0	0	RC1	RC2	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

Table 568. DTC field descriptions

Bit	Description
RWT1	Read/write trace 1 00 No trace messages generated X1 Enable data read trace 1X Enable data write trace
RWT2	Read/write trace 2 00 No trace messages generated X1 Enable data read trace 1X Enable data write trace

Table 568. DTC field descriptions (continued)

Bit	Description
AMID	AHB Master ID Select This field selects which crossbar master will have its accesses to the SRAM traced. Only one master can have its accesses to the SRAM traced at a time. Lock-Step Mode Decoding 0000 Core_0 / Core_1 0001 Reserved 0010 eDMA_0 / eDMA_1 0011 FlexRay 0100–1111 Reserved Decoupled Parallel Mode Decoding 0000 Core_0 0001 Core_1 0010 eDMA_0 0011 FlexRay 0100 Reserved 0101 Reserved 0110 DMA 1 0111 Reserved 1000–1111 Reserved
NSID	Nexus Source ID This field defines the Nexus Source ID that will be used in the Nexus trace messages. This should be set by tools to: Lock-Step Mode Decoding 0000–1001 Reserved 1010 NXSS_0 / NXSS_1 1011–1111 Reserved Decoupled Parallel Mode Decoding 0000-1001 Reserved 1010 NXSS_0 1011 NXSS_1 1100–1111 Reserved
RC1	Range control 1 0 Condition trace on address within range (endpoints inclusive) 1 Condition trace on address outside of range (endpoints exclusive)
RC2	Range control 2 0 Condition trace on address within range (endpoints inclusive) 1 Condition trace on address outside of range (endpoints exclusive)

34.5.4 Data Trace Start Address Registers 1 and 2 (DTSA1 and DTSA2)

The Data Trace Start Address Registers 1 and 2 (DTSA1 and DTSA2) define the start addresses for each trace channel.

Figure 734. Data Trace Start Address Registers (DTSA1, DTSA2)

Access: R/W

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

34.5.5 Data Trace End Address Registers 1 and 2 (DTEA1 and DTEA2)

The Data Trace End Address Registers 1 and 2 (DTEA1 and DTEA2) define the end addresses for each trace channel.

Figure 735. Data Trace Start Address Registers (DTEA1, DTEA2)

Access: R/W

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

Table 569 illustrates the range that is selected for data trace for various cases of DTSA being less than, greater than, or equal to DTEA.

Table 569. 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	—	Invalid range, no trace

Note: DTSA must be less than or equal to DTEA to guarantee correct data write/read traces. When the range control bit is 0 (internal range), accesses to DTSA and DTEA addresses are traced. When the range control bit is 1 (external range), accesses to DTSA and DTEA are not traced.

34.5.6 Breakpoint / Watchpoint Control Register 1 (BWC1)

The Breakpoint / Watchpoint Control Register 1 (BWC1) controls attributes for generation of NXSS_0 and NXSS_1 watchpoint number 1.

Figure 736. Break / Watchpoint Control Register 1 (BWC1)

Access: R/W

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16	17 18 19	20 21 22 23	24 25 26 27	28 29 30 31
R	BWE1 BRW1			BWR1	BWT1				
W									
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0	0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Table 570. BWC1 field descriptions

Field	Description
BWE1	Breakpoint/watchpoint #1 enable 00 Internal Nexus watchpoint #1 disabled 01–10 Reserved 11 Internal Nexus watchpoint #1 enabled
BRW1	Breakpoint/watchpoint #1 read/write select 00 Watchpoint #1 hit on read accesses 01 Watchpoint #1 hit on write accesses 10 Watchpoint #1 on read or write accesses 11 Reserved
BWR1	Breakpoint/watchpoint #1 register compare 00 No register compare (same as BWC1[31:30] = 2'b00) 01 Reserved 10 Compare with BWA1 value 11 Reserved
BWT1	Breakpoint/watchpoint #1 type 0 Reserved 1 Watchpoint #1 on data accesses

34.5.7 Breakpoint / Watchpoint Control Register 2 (BWC2)

The Breakpoint / Watchpoint Control Register 2 (BWC2) controls attributes for generation of NXSS_0 and NXSS_1 watchpoint number 2.

Figure 737. Break / Watchpoint Control Register 2 (BWC2)

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BWE2		BRW2												BWR2		BWT2															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 571. BWC2 field descriptions

Field	Description
BWE2	Breakpoint/watchpoint #2 enable 00 Internal Nexus watchpoint #2 disabled 01–10 Reserved 11 Internal Nexus watchpoint #2 enabled
BRW2	Breakpoint/watchpoint #2 read/write select 00 Watchpoint #2 hit on read accesses 01 Watchpoint #2 hit on write accesses 10 Watchpoint #2 on read or write accesses 11 Reserved

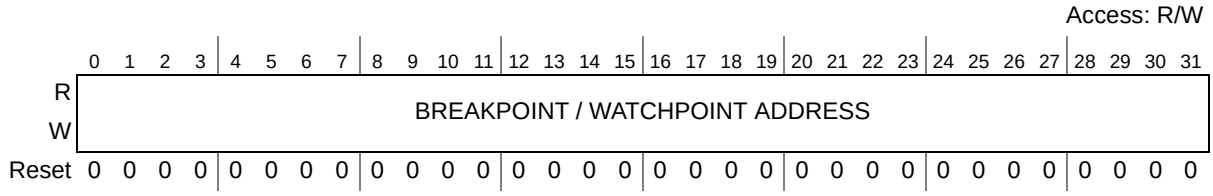
Table 571. BWC2 field descriptions (continued)

Field	Description
BWR2	Breakpoint/watchpoint #2 register compare 00 No register compare (same as BWC1[31:30] = 2'b00) 01 Reserved 10 Compare with BWA2 value 11 Reserved
BWT2	Breakpoint/watchpoint #2 Type 0 Reserved 1 Watchpoint #2 on data accesses

34.5.8 Breakpoint/Watchpoint Address Registers 1 and 2 (BWA1 and BWA2)

The breakpoint/watchpoint address registers are compared with bus addresses to generate internal watchpoints.

Figure 738. Breakpoint / Watchpoint Address Registers (BWA1, BWA2)



34.5.9 Unimplemented registers

Unimplemented registers are those with client select and index value combinations other than those listed in *Table 564*. For unimplemented registers, the NXSS_0 and NXSS_1 modules drives TDO to zero during the "SHIFT-DR" state. It also transmits an error message with the invalid access opcode encoding.

34.6 Functional description

34.6.1 TCODEs supported by NXSS_0 and NXSS_1

The NXSS_0 and NXSS_1 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 5001-2003 defines a set of public messages. The NXSS_0 and NXSS_1 modules support the public TCODEs as shown in *Table 572*.

Table 572. Public TCODEs supported

Message name	Packet size bits		Packet name	Packet type	Packet description
	Min.	Max.			
Data Trace - Date Write Message	6	6	TCODE	Fixed	TCODE number = 5
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to *Table 574*)
	1	32	U-ADDR	Variable	Unique portion of the data write value
	1	64	DATA	Variable	Data write value
Data Trace - Data Read Message	6	6	TCODE	Fixed	TCODE number = 6
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to *Table 574*)
	1	32	U-ADDR	Variable	Unique portion of the data read value
	1	64	DATA	Variable	Data read value
Error Message	6	6	TCODE	Fixed	TCODE number = 8
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	5	5	ECODE	Fixed	Error code (refer to *Table 573*)
Data Trace - Data Write Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 13 (0xD)
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to *Table 574*)
	1	32	F-ADDR	Variable	Full access address (leading zero (0) truncated)
	1	64	DATA	Variable	Data write value
Data Trace - Data Read Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 14 (0xE)
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to *Table 574*)
	1	32	F-ADDR	Variable	Full access address (leading zero (0) truncated)
	1	64	DATA	Variable	Data read valued
Watchpoint Message	6	6	TCODE	Fixed	TCODE number = 15 (0xF)
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	4	4	WPHIT	Fixed	Number indicating watchpoint sources

Table 573. Error code (ECODE) encoding (TCODE = 8)

Error code (ECODE)	Description
00000	Reserved
00001	Reserved
00010	Data Trace overrun

Table 573. Error code (ECODE) encoding (TCODE = 8)

Error code (ECODE)	Description
00011	Reserved
00100	Reserved
00101	Invalid access opcode (Nexus Register unimplemented)
00110	Watchpoint overrun
00111	Reserved
01000	Data Trace and Watchpoint overrun
01001–11111	Reserved

Table 574. Data Trace Size (DSZ) encodings (TCODE = 5, 6, 13, 14)

DTM size encoding	Transfer size
000	Byte
001	Halfword (two bytes)
010	Word (four bytes)
011	Doubleword (eight bytes)
100–111	Reserved

34.6.2 Data Trace

This section deals with the data trace mechanism supported by the NXSS_0 and NXSS_1 modules. Data trace is implemented via data write messaging (DWM) and data read messaging (DRM).

34.6.3 Data Trace Messaging (DTM)

NXSS_0 and NXSS_1 data trace messaging is accomplished by snooping the NXSS_0 and NXSS_1 data bus, and storing the information for qualifying accesses (based on enabled features and matching AHB Master ID and target addresses). The NXSS module traces all data access that meet the selected range and attributes.

34.6.4 DTM Message Formats

The NXSS module supports five types of DTM Messages — data write, data read, data write synchronization, data read synchronization, and error messages.

34.6.4.1 Data Write and Data Read Messages

The data write and data read messages contain the data write/read value and the address of the write/read access, relative to the previous data trace message. Data write message and data read message information is messaged out in the following format:

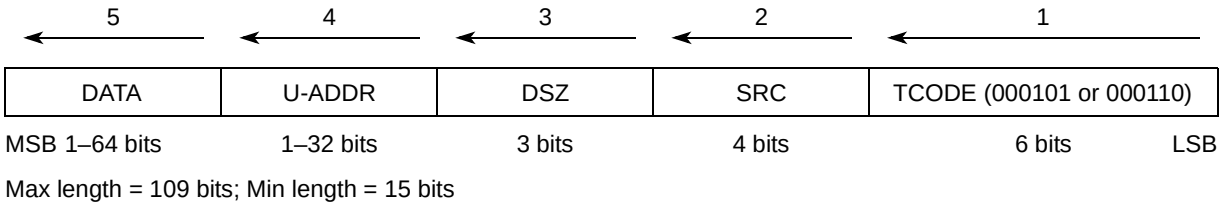


Figure 739. Data Write/Read Message Format

34.6.4.2 DTM Overflow Error Messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards incoming messages until it has completely emptied the queue. After it is emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied.

If only a data trace message attempts to enter the queue while it is being emptied, the error message incorporates the data trace only error encoding (00010). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

Error information is messaged out in the following format:

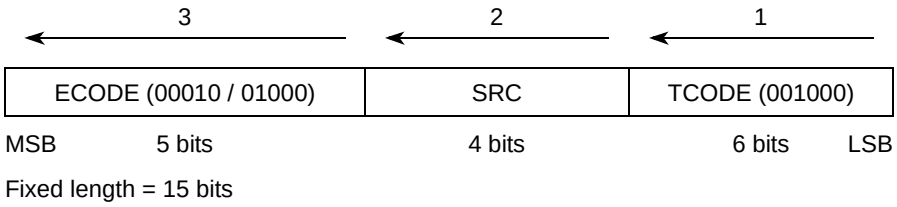


Figure 740. Error Message Format

34.6.4.3 Data Trace Synchronization Messages

A data trace write/read w/ sync. message is messaged via the auxiliary port (provided data trace is enabled) for the following conditions (refer to *Table 575*):

- Initial data trace message upon exit from system reset or whenever data trace is enabled is a synchronization message.
- Upon returning from debug mode, the first data trace message is a synchronization message.
- After occurrence of queue overrun (can be caused by any trace message), the first data trace message is a synchronization message.
- After the periodic data trace counter has expired indicating 255 *without-sync* data trace messages have occurred since the last *with-sync* message occurred.
- Upon assertion of the Event In ($\overline{\text{EVTI}}$) pin, the first data trace message is a synchronization message if the EIC bits of the DC register have enabled this feature.
- Upon data trace write/read after the previous DTM message was lost due to an attempted access to a secure memory location.
- Upon data trace write/read after the previous DTM message was lost due to a collision entering the FIFO between the DTM message and any of the following: error message, or watchpoint message.

Data trace synchronization messages provide the full address (without leading zeros) and ensure that development tools fully synchronize with data trace regularly. Synchronization messages provide a reference address for subsequent DTMs, in which only the unique portion of the data trace address is transmitted. The format for data trace write/read w/ sync. messages is as follows:

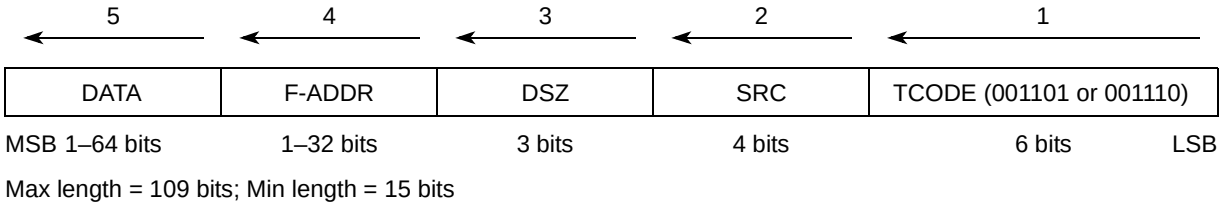


Figure 741. Data Write/Read w/ Sync Message Format

Exception conditions that result in data trace synchronization are summarized in *Table 575*.

Table 575. Data trace exception summary

Exception condition	Exception handling
System Reset Negation	At the negation of JTAG reset (JCOMP), queue pointers, counters, state machines, and registers within the NXSS_0 and NXSS_1 module are reset. If data trace is enabled, the first data trace message is a data write/read w/ sync. message.
Data Trace Enabled	The first data trace message (after data trace has been enabled) is a synchronization message.
Exit from Debug	Upon exit from debug mode the next data trace message is converted to a data write/read w/ sync. message.

Table 575. Data trace exception summary (continued)

Exception condition	Exception handling
Queue Overrun	An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. After it is emptied, an error message is queued. The error encoding indicates the types of messages that attempted to be queued while the FIFO was being emptied. The next DTM message in the queue is a data write/read w/ sync. message.
Periodic Data Trace Synchronization	A forced synchronization occurs periodically after 255 data trace messages have been queued. A data write/read w/ sync. message is queued. The periodic data trace message counter then resets.
Event In	If the Nexus module is enabled, an $\overline{\text{EVTI}}$ assertion initiates a data trace write/read w/ sync. message upon the next data write/read (if data trace is enabled and the EIC bits of the DC register have enabled this feature).
Attempted Access to Secure Memory	Any attempted read or write to secure memory locations temporarily disable data trace & cause the corresponding DTM to be lost. A subsequent read/write queues a data trace read/write with sync. message.
Collision Priority	All messages have the following priority: Error → WPM → DTM. A DTM message which attempts to enter the queue at the same time as an error message, or watchpoint message is lost. A subsequent read/write queues a data trace read/write with sync. message.

34.6.5 DTM operation

34.6.5.1 Enabling data trace messaging

Data trace messaging can be enabled in one of two ways.

- Setting the DC1[TM] field to enable data trace
- Using the WT[DTS] field to enable data trace on watchpoint hits

34.6.5.2 DTM queueing

NXSS_0 and NXSS_1 implement a programmable depth queue for queuing all messages. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

Note: If multiple trace messages must be queued at the same time, watchpoint messages have the highest priority (WPM → DTM).

34.6.5.3 Relative addressing

The relative address feature is compliant with IEEE-ISTO Nexus 5001-2003 and is designed to reduce the number of bits transmitted for addresses of data trace messages.

34.6.5.4 Data trace windowing

Data write/read messages are enabled via the RWT1(2) field in the data trace control register (DTC) for each DTM channel. Data trace windowing is achieved via the address range defined by the DTEA and DTSA registers and by the RC1(2) field in the DTC. All read/write accesses by the selected AHB Master ID (See DTC register for more details) that fall inside or outside these address ranges, as programmed, are candidates to be traced.

34.7 Watchpoint support

The NXSS_0 and NXSS_1 module provides watchpoint messaging via the auxiliary pins, as defined by IEEE-ISTO 5001-2003.

Watchpoint messages can be generated using the NXSS_0 and NXSS_1 defined internal watchpoints.

34.7.1 Watchpoint messaging

Enabling watchpoint messaging is accomplished by setting the watchpoint messaging enable bit, DC1[WEN]. Using the BWC1 and BWC2 registers, two independently controlled internal watchpoints can be initialized. When the selected AHB Master ID address matches on BWA1 or BWA2, a watchpoint message is transmitted.

The Nexus module provides watchpoint messaging using the TCODE. When either of the two possible watchpoint sources asserts, a message is sent to the queue to be messaged out. This message indicates the watchpoint number.

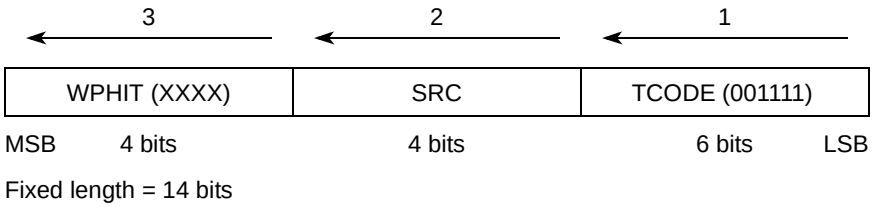


Figure 742. Watchpoint message format

Table 576. Watchpoint source description (WPHIT field descriptor)

Watchpoint Source (4 bits)	Watchpoint Description
XXX1	Reserved
XX1X	Reserved
X1XX	Internal Watchpoint #1 (BWA1 match)
1XXX	Internal Watchpoint #2 (BWA2 match)

34.7.2 Watchpoint error message

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. After it is emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied.

If only a watchpoint message attempts to enter the queue while it is being emptied, the error message incorporates the watchpoint only error encoding (00110). If a data trace message also attempts to enter the queue while it is being emptied, the error message incorporates error encoding (01000).

Error information is messaged out in the following format (refer to *Figure 743*).

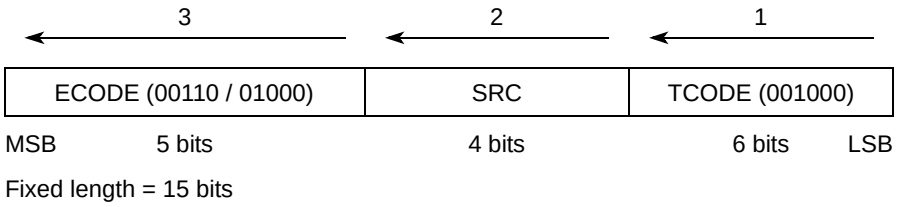


Figure 743. Error message format

35 Nexus Port Controller (NPC)

35.1 Information specific to this device

This section presents device-specific parameterization, customization, and feature availability information not specifically referenced in the remainder of this chapter.

35.1.1 Parameter values

The parameter values for this device are shown in *Table 577*.

Table 577. Device parameter values

Parameter	Value
Number of MDO pins available in reduced-port mode	4
Number of MDO pins available in full-port mode	12
Part identification number	0x2A2
Manufacturer identity code (MIC)	0x020
Design center code (DC)	0x2B

35.1.2 Unavailable features

MCKO clock gating is not supported on cut1 of this device. (It is supported on cut2/3.)

This device does not support an MCKO division factor of 3. Therefore, the setting PCR[MCKO_DIV] = 2 (see *Table 582*) is not available on this device.

Nexus DDR mode works only for MCKO division factors of 2, 4, and 8. Therefore, the setting PCR[MCKO_DIV] = 0 (see *Table 582*) will not support Nexus DDR mode on this device.

35.2 Introduction

Figure 744 is a block diagram of the Nexus Port Controller (NPC) block.

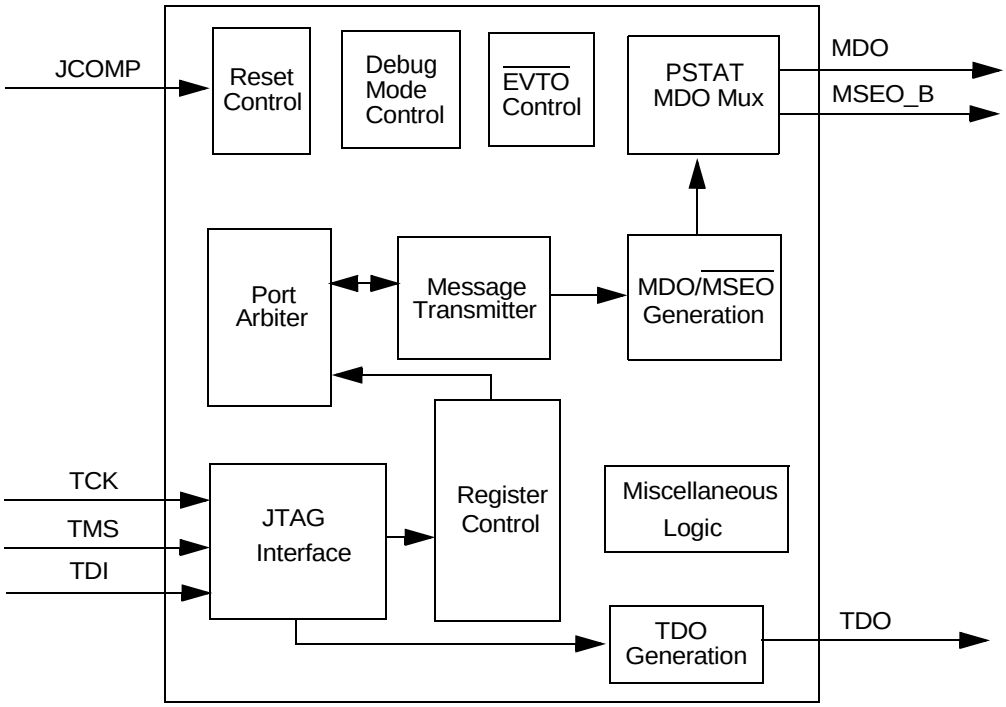


Figure 744. Nexus Port Controller Block Diagram

35.2.1 Overview

On a system-on-a-chip device, there are often multiple blocks that require development support. Each of these blocks implements a development interface based on the IEEE-ISTO 5001-2003 standard. The blocks share input and output ports that interface with the development tool. The NPC controls the usage of the input and output port in a manner that allows all the individual development interface blocks to share the port, and appear to the development tool to be a single block.

35.2.2 Features

The NPC block performs the following functions:

- Controls arbitration for ownership of the Nexus Auxiliary Output Port
- Nexus Device Identification Register and Messaging
- Generates MCKO enable and frequency division control signals
- Controls sharing of $\overline{\text{EVTO}}$
- Generates an MCKO clock gating control signal to enable gating of MCKO when the auxiliary output port is idle
- Control of the device-wide debug mode
- Generates asynchronous reset signal for Nexus blocks based on JCOMP input, censorship status, and power-on reset status
- System clock locked status indication via MDO[0] following power-on reset
- Provides Nexus support for censorship mode
- $\overline{\text{RDY}}$ pin support to increase the transfer rate of the IEEE 1149.1 port for large data read requests (supported on cut2/3 only)

35.2.3 Modes of operation

The NPC block uses the JCOMP input, the censorship status , and an internal power-on reset indication as its primary reset signals. Upon exit of reset, the mode of operation is determined by the Port Configuration Register (PCR) settings.

35.2.3.1 Reset

The NPC block is asynchronously placed in reset when either power-on reset is asserted, JCOMP is not set for Nexus access, the device enters censored mode , or the TAP controller state machine is in the Test-Logic-Reset state. Holding TMS high for 5 consecutive rising edges of TCK guarantees entry into the Test-Logic-Reset state regardless of the current TAP controller state. Following negation of power-on reset, the NPC remains in reset until the system clock achieves lock. The NPC is unaffected by other sources of reset. While in reset, the following actions occur:

- The TAP controller is forced into the Test-Logic-Reset state
- The auxiliary output port pins are negated
- The TDI, TMS, and TCK TAP inputs are ignored (when in power-on reset, censored mode or JCOMP not set for NPC operation only)
- Registers default back to their reset values

35.2.3.2 Disabled-Port Mode

In disabled-port mode, auxiliary output pin port enable signals are negated, thereby disabling message transmission. Any debug feature that generates messages can not be used. The primary features available are class 1 features and read/write access to the registers. Class 1 features include the ability to trigger a breakpoint event indication through $\overline{\text{EVTO}}$.

35.2.3.3 Full-Port Mode

Full-port mode (FPM) is entered by asserting the MCKO_EN and FPM bits in the PCR. All trace features are enabled or can be enabled by writing the configuration registers via the TAP. The number of MDO pins available is device-specific.

35.2.3.4 Reduced-Port Mode

Reduced-port mode (RPM) is entered by asserting the MCKO_EN bit and deasserting the FPM bit in the PCR. All trace features are enabled or can be enabled by writing the configuration registers via the TAP. The number of MDO pins available is device-specific.

35.2.3.5 Censored Mode

When the device is in censored mode, reading the contents of internal flash externally is not allowed. To prevent Nexus modules from violating censorship, the NPC is held in reset when in censored mode, asynchronously holding all other Nexus modules in reset as well. This prevents Nexus read/write to memory mapped resources and the transmission of Nexus trace messages.

35.2.3.6 Nexus Double Data Rate Mode

Nexus double data rate (DDR) mode is enabled by asserting the DDR_EN bit in the PCR. In double data rate mode, message data is updated as follows, effectively doubling message throughput:

- For cut1: On each edge (both rising and falling) of MCKO
- For cut2/3: Between the edges (both rising and falling) of MCKO

35.3 External signal description

35.3.1 Overview

The NPC pin interface provides for the transmission of messages from Nexus blocks to the external development tools and for access to Nexus client registers. The NPC pin definition is outlined in *Table 578*.

Table 578. NPC Signal Properties

Name	Port	Function	Reset State	Pull(1)
$\overline{\text{EVTO}}$	Auxiliary	Event Out pin	0b1	—
JCOMP	JTAG	JTAG Compliancy and TAP Sharing Control	—	Down
MDO	Auxiliary	Message Data Out pins	0(2)	—
MSEO	Auxiliary	Message Start/End Out pins	0b11	—
TCK(3)	JTAG	Test Clock Input	—	Up
TDI	JTAG	Test Data Input	—	Up
TDO	JTAG	Test Data Output	High Z(4)	—
TMS	JTAG	Test Mode Select Input	—	Up
$\overline{\text{RDY}}$	JTAG	Data ready for transfer to/from NRRs	—	—

1. The pull is not implemented in this block. Pullup/pulldown devices are implemented in the pads.

2. *Following a power-on reset, MDO[0] remains asserted until power-on reset is exited and the system clock achieves lock.*
3. *TCK frequency must be lower than system clock frequency during low power and normal operation modes for communication*
4. *TDO output buffer enable is negated when the NPC is not in the Shift-IR or Shift-DR states. A weak pull may be implemented on TDO at the SoC level.*

35.3.2 Detailed signal descriptions

This section describes each of the signals listed in *Table 578* in more detail. The JTAG test clock (TCK) input from the pin is not a direct input to the NPC. The NPC requires two separate input clocks for TCK clocked logic, one for posedge (rising edge TCK) logic and one for negedge (falling edge TCK) logic. Both clocks are derived from the pin TCK, and generated external to the NPC.

35.3.2.1 $\overline{\text{EVTO}}$ - Event Out

Event Out ($\overline{\text{EVTO}}$) is an output pin that is asserted upon breakpoint occurrence to provide breakpoint status indication. The $\overline{\text{EVTO}}$ output of the NPC is generated based on the values of the individual $\overline{\text{EVTO}}$ signals from all Nexus blocks that implement the signal.

35.3.2.2 JCOMP - JTAG Compliancy

The JCOMP signal provides the ability to share the TAP. The NPC TAP controller is enabled when JCOMP is set to the NPC enable encoding, otherwise the NPC TAP controller remains in reset.

35.3.2.3 MDO - Message Data Out

Message Data Out (MDO) are output pins used for uploading OTM, BTM, DTM, and other messages to the development tool. The development tool should sample MDO on the rising edge of MCKO. The width of the MDO bus used is determined by reset configuration.

35.3.2.4 MSEO_B - Message Start/End Out

Message Start/End Out ($\overline{\text{MSEO}}$) is an output pin that indicates when a message on the MDO pins has started, when a variable length packet has ended, or when the message has ended. The development tool should sample $\overline{\text{MSEO}}$ on the rising edge of MCKO.

35.3.2.5 TCK - Test Clock Input

Test Clock Input (TCK) pin is used to synchronize the test logic and control register access through the JTAG port.

The JTAG Clock (TCK) typically operates at a frequency well below the system clock frequency, as specified in the electrical data sheets. In some cases, however, such as low power mode (if the device supports low power modes), the system clock frequency may be lowered significantly from the normal operating range. If the system clock frequency is reduced below the frequency of TCK it will no longer be possible to communicate with the Nexus Port Controller Port Configuration Register (NPC_PCR). Therefore, if the tool needs to update the NPC_PCR Low Power Debug Enable (NPC[PCR[LP_DBG]) or Low Power Synchronization bits (NPC[PCR[LP1_SYN] and NPC[PCR[LP2_SYN]), the clock frequency must be lowered.

Ensure that the frequency of TCK does not exceed the system clock frequency during normal operation and during low power operation.

Note: TCK clock frequency needs to be smaller than SYSCLK/2 for correct operation of NPC/Nexus/NXSS subsystem

35.3.2.6 TDI - Test Data Input

Test Data Input (TDI) pin receives serial test instruction and data. TDI is sampled on the rising edge of TCK.

35.3.2.7 TDO - Nexus Test Data Output

Test Data Output (TDO) pin transmits serial output for instructions and data. TDO is three-stateable and is actively driven in the SHIFT-IR and SHIFT-DR controller states. TDO is updated on the falling edge of TCK and sampled by the development tool on the rising edge of TCK.

35.3.2.8 TMS - Test Mode Select

Test Mode Select Input (TMS) pin is used to sequence the IEEE 1149.1-2001 TAP controller state machine. TMS is sampled on the rising edge of TCK.

35.3.2.9 $\overline{\text{RDY}}$ — Data ready for transfer (on cut2/3 only)

The $\overline{\text{RDY}}$ pin, supported on cut2/3 only, exists to increase the transfer rate of the IEEE 1149.1 port. It is used to signal when data are ready to be transferred to and from NRRs. This may eliminate the need to poll NRRs for status information for synchronization purposes. This capability becomes especially important when performing read/write access transfers to different speed target memories.

The $\overline{\text{RDY}}$ pin asserts (asynchronously) a logic low whenever the read/write access transfer has completed without error and then deasserts when the IEEE 1149.1 state machine has reached the CAPTURE_DR state.

The $\overline{\text{RDY}}$ pin is controlled by the TEST_CTRL register in the JTAGC module (see *Section 30.3.1.4, TEST_CTRL register (cut2/3 only)*).

35.4 Register definition

This section provides a detailed description of the NPC registers accessible to the end user. Individual bit-level descriptions and reset states of the registers are included.

Table 579 shows the NPC registers by index values. The registers are not memory-mapped and can only be accessed via the TAP. The NPC block does not implement the client select control register because the value does not matter when accessing the registers. Note that the bypass and instruction registers have no index values. These registers are not accessed in the same manner as Nexus client registers. Refer to the individual register descriptions for more detail.

Table 579. NPC Registers

Index	Register
0	Device ID Register (DID)
127	Port Configuration Register (PCR)

35.4.1 Register descriptions

This section consists of NPC register descriptions.

35.4.1.1 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.

35.4.1.2 Instruction Register

The NPC block uses a 4-bit instruction register as shown in *Table 745*. The instruction register is accessed via the SELECT_IR_SCAN path of the tap controller state machine, and allows instructions to be loaded into the block to enable the NPC for register access (NEXUS_ENABLE) or select the bypass register as the shift path from TDI to TDO (BYPASS or unimplemented instructions).

Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and Test-Logic-Reset TAP controller states. Synchronous entry into the Test-Logic-Reset state results in synchronous loading of the BYPASS instruction. Asynchronous entry into the Test-Logic-Reset state results in asynchronous loading of the BYPASS instruction. During the Capture-IR TAP controller state, the instruction register is loaded with the value of the previously executed instruction, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

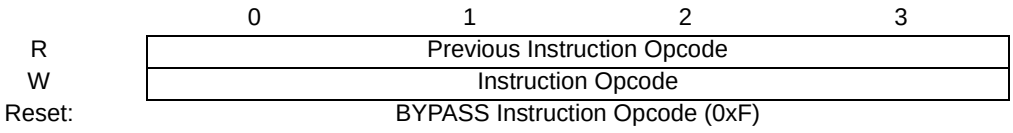


Figure 745. 4-Bit Instruction Register

35.4.1.3 Nexus Device ID Register (DID)

The device identification register, shown in *Figure 746*, allows the part revision number, design center, part identification number, and manufacturer identity code of the part to be determined through the auxiliary output port.

Register index: 0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	Part Revision Number				Design Center						Part Identification Number					
W																
RESET:	PRN				DC						PIN					

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Part Identification Number				Manufacturer Identity Code											1
W																
RESET:	PIN (cont'd.)				0	0	0	0	0	0	0	1	1	1	0	1

= Reserved

Figure 746. Nexus Device ID Register

Table 580. DID Field Descriptions

Bit	Name	Description
31:28	PRN	Part Revision Number These bits contain the revision number of the part
27:22	DC	Design Center These bits indicate the device design center
21:12	PIN	Part Identification Number These bits contain the part number of the device
11:1	MIC	Manufacturer Identity Code These bits contain the reduced Joint Electron Device Engineering Council (JEDEC) ID
0	Bit [0]	IDCODE Register ID Bit [0] identifies this register as the device identification register and not the bypass register

35.4.1.4 Port Configuration Register (PCR)

Register index: 127

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FPM	MCKO_GT	MCKO_EN	MCKO_DIV			EVT_EN	DDR_EN	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LP_DBG	0	0	0	0	0	LP2_SYN	LP1_SYN	0	0	0	0	0	0	0	PSTAT_EN
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Reserved

Figure 747. Port Configuration Register (PCR)

The PCR, shown in *Figure 747*, is used to select the NPC mode of operation, enable MCKO and select the MCKO frequency, and enable or disable MCKO gating. This register should be configured as soon as the NPC is enabled.

The PCR register may be rewritten by the debug tool subsequent to the enabling of the NPC for low power debug support. In this case, the debug tool may set and clear the LP_DBG and LPn_SYN bits, but must preserve the original state of the remaining bits in the register.

Note: The mode or clock division must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.

Table 581. 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. All MDO pins are used to transmit messages A subset of MDO pins are used to transmit messages
MCKO_GT	MCKO Clock Gating Control This bit is used to enable or disable MCKO clock gating. If clock gating is enabled, the MCKO clock is gated when the NPC is in enabled mode but not actively transmitting messages on the auxiliary output port. When clock gating is disabled, MCKO is allowed to run even if no auxiliary output port messages are being transmitted. MCKO gating is enabled MCKO gating is disabled
MCKO_EN	MCKO Enable This bit enables the MCKO clock to run. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. MCKO clock is enabled MCKO clock is driven to zero

Table 581. PCR field descriptions (continued)

Name	Description
MCKO_DIV	MCKO Division Factor The value of this signal determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. *Table 582* shows the meaning of MCKO_DIV Values. In this table, SYS_CLK represents the system clock frequency.
EVT_EN	$\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ Enable This bit enables the $\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ port functions. $\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ port enabled $\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ port disabled
DDR_EN	Double Data Rate Mode Enable This bit enables Nexus double data rate (DDR) mode. In DDR mode, message data is updated on both rising and falling edges of MCKO, effectively doubling message throughput. DDR mode enabled DDR mode disabled
LP_DBG_EN	Low Power Debug Enable This bit enables debug functionality on exit from low power modes on supported devices. Low power debug enabled Low power debug disabled
LPn_SYN	Low Power Mode n Synchronization These bits are used to synchronize the entry into low power modes between the device and debug tool. Supported devices set these bits before a pending entry into low power mode. After reading the bit as set, the debug tool then clears the bit to acknowledge to the device that it may enter the low power mode. Low power mode entry pending Low power mode entry acknowledged
PSTAT_EN	Processor Status Mode Enable(1) This bit enables processor status (PSTAT) mode. In PSTAT mode, all auxiliary output port MDO pins are used to transmit processor status information, and Nexus messaging is unavailable. PSTAT mode enabled PSTAT mode disabled

1. PSTAT Mode is intended for factory processor debug only. The PSTAT_EN bit should be written to disable PSTAT mode if Nexus messaging is desired. No Nexus messages are transmitted under any circumstances when PSTAT mode is enabled.

Table 582. MCKO_DIV Values

MCKO_DIV[2:0]	MCKO Frequency
0	SYS_CLK(1)
1	SYS_CLK/2
2	SYS_CLK/3
3	SYS_CLK/4
4	Reserved
5	Reserved

Table 582. MCKO_DIV Values

MCKO_DIV[2:0]	MCKO Frequency
6	Reserved
7	SYS_CLK/8

1. The SYS_CLK setting for MCKO frequency should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins.

35.5 Functional description

35.5.1 NPC reset configuration

The NPC is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the PCR is then required to enable the NPC and select the mode of operation. Asserting MCKO_EN places the NPC in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting or negating the FPM bit selects full-port or reduced-port mode, respectively.

Table 583 describes the NPC reset configuration options.

Table 583. NPC Reset Configuration Options

JCOMP Equal to npc_jcomp_plug?	MCKO_EN bit of the Port Configuration Register	FPM bit of the Port Configuration Register	Configuration
no	X	X	Reset
yes	0	X	Disabled
yes	1	1	Full-Port Mode
yes	1	0	Reduced-Port Mode

35.5.2 Auxiliary Output Port

The auxiliary output port is shared by each of the Nexus modules on the device. The NPC communicates with each of the Nexus modules and arbitrates for access to the port.

35.5.2.1 Output Message Protocol

The protocol for transmitting messages via the auxiliary port is accomplished with the MSEO functions. The MSEO pins are used to signal the end of variable-length packets and the end of messages. They are not required to indicate the end of fixed-length packets. MDO and MSEO are sampled on the rising edge of MCKO.

Figure 748 illustrates the state diagram for MSEO transfers. All transitions not included in the figure are reserved, and must not be used.

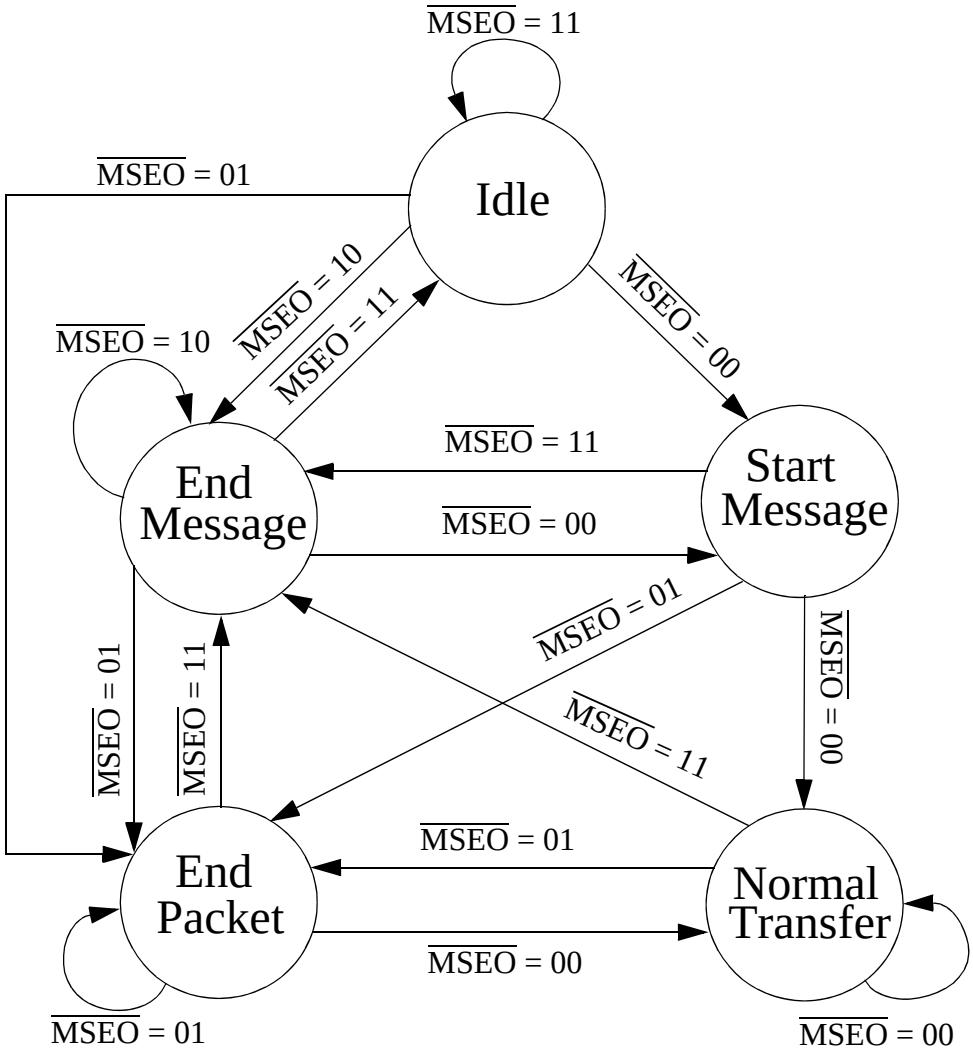


Figure 748. $\overline{\text{MSEO}}$ Transfers (for 2-bit $\overline{\text{MSEO}}$)

35.5.2.2 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 584 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 584. 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 749 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 749. Message Field Sizes

Message	TCODE	Field #1	Field #2	Field #3	Field #4	Field #5	Min. Size[1] (bits)	Max Size[2] (bits)
Device ID Message	1	Fixed = 32	NA	NA	NA	NA	38	38

NOTES:

1. Minimum information size. The actual number of bits transmitted depends on the number of MDO pins
2. Maximum information size. The actual number of bits transmitted depends on the number of MDO pins

The double edges in *Figure 749* 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.

35.5.2.3 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 750* shows the transmission sequence of a message that is made up of a TCODE followed by two fields.

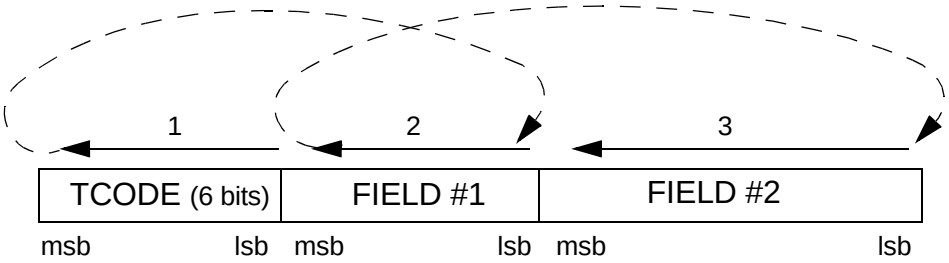


Figure 750. Transmission Sequence of Messages

35.5.3 IEEE 1149.1-2001 (JTAG) TAP

The NPC block uses the IEEE 1149.1-2001 TAP for accessing registers. Each of the individual Nexus blocks on the device implements a TAP controller for accessing its registers as well. TAP signals include TCK, TDI, TMS, and TDO. There may also be other blocks on the MCU that use the TAP and implement a TAP controller. The value of the JCOMP input controls ownership of the port between Nexus and non-Nexus blocks sharing the TAP.

Refer to the IEEE 1149.1-2001 specification for further detail on electrical and pin protocol compliance requirements.

The NPC implements a Nexus controller state machine that transitions based on the state of the IEEE 1149.1-2001 state machine shown in *Figure 752*. The Nexus controller state machine is defined by the IEEE-ISTO 5001-2003 standard. It is shown in *Figure 753*.

The instructions implemented by the NPC TAP controller are listed in *Table 585*. The value of the NEXUS-ENABLE instruction is 0b0000. Each unimplemented instruction acts like the BYPASS instruction. The size of the NPC instruction register is 4-bits.

Table 585. Implemented Instructions

Instruction Name	Private/ Public	Opcode	Description
NEXUS-ENABLE	public	0x0	Activate Nexus controller state machine to read and write NPC registers.
BYPASS	private	0xF	NPC BYPASS instruction. Also the value loaded into the NPC IR upon exit of reset.

Data is shifted between TDI and TDO starting with the least significant bit as illustrated in *Figure 751*. This applies for the instruction register and all Nexus tool-mapped registers.

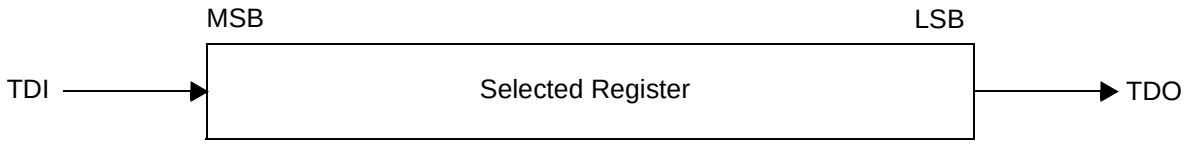
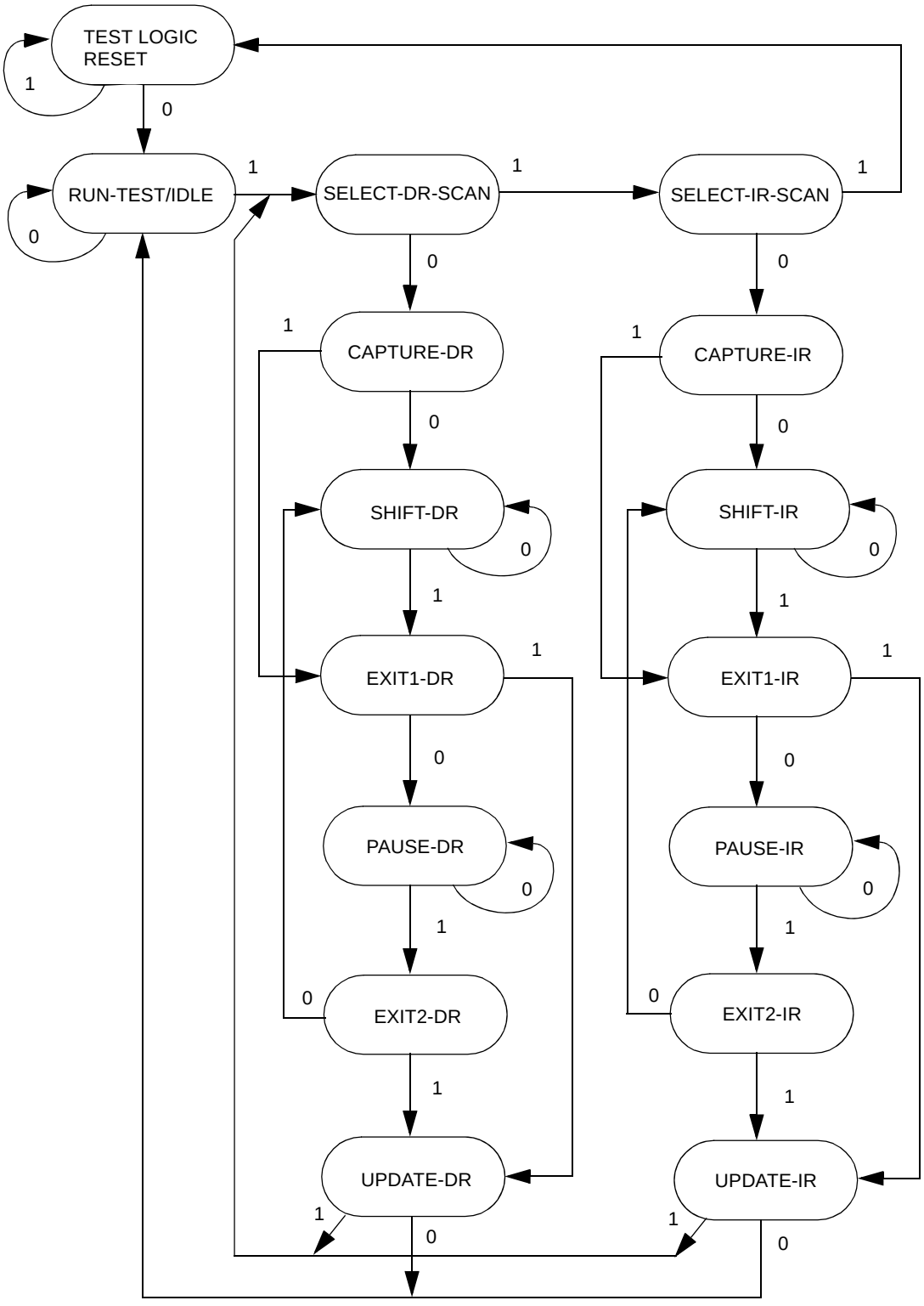


Figure 751. Shifting Data Into Register

35.5.3.1 Enabling the NPC TAP controller

Assertion of the power-on reset signal, entry into censored mode, or setting JCOMP to a value other than the NPC enable encoding resets the NPC TAP controller. When not in power-on reset or censored mode, the NPC TAP controller is enabled by driving JCOMP with the NPC enable value and exiting the Test-Logic-Reset state. Loading the NEXUS-ENABLE instruction then grants access to Nexus debug.

Figure 752. IEEE 1149.1-2001 TAP Controller State Machine



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.

35.5.3.2 Retrieving device IDCODE

The Nexus TAP controller does not implement the IDCODE instruction. However, the device identification message can be output by the NPC through the auxiliary output port or shifted out serially by accessing the Nexus Device ID register through the TAP. Transmission of the device identification message on the auxiliary output port MDO pins occurs immediately after a write to the PCR, if the NPC is enabled. Transmission of the device identification message serially via TDO is achieved by performing a read of the register contents as described in *Section 35.5.3.4, Selecting a Nexus Client Register*.

35.5.3.3 Loading NEXUS-ENABLE Instruction

Access to the NPC registers is enabled when the TAP controller instruction register is loaded with the NEXUS-ENABLE instruction. This instruction is shifted in via the SELECT-IR-SCAN path and loaded in the UPDATE-IR state. At this point, the Nexus controller state machine, shown in *Figure 753*, transitions to the REG_SELECT state. The Nexus controller has three states: idle, register select, and data access. *Table 586* illustrates the IEEE 1149.1 sequence to load the NEXUS-ENABLE instruction.

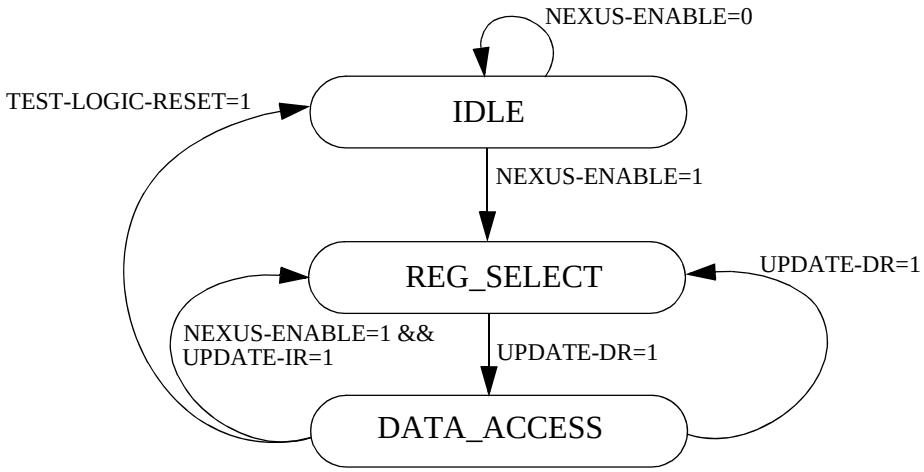


Figure 753. NEXUS Controller State Machine

Table 586. Loading NEXUS-ENABLE instruction

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
0	0	RUN-TEST/IDLE	IDLE	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	IDLE	Transitional state
2	1	SELECT-IR-SCAN	IDLE	Transitional state
3	0	CAPTURE-IR	IDLE	Internal shifter loaded with current instruction
4	0	SHIFT-IR	IDLE	TDO becomes active, and the IEEE 1149.1-2001 shifter is ready. Shift in all but the last bit of the NEXUS_ENABLE instruction.
3 TCKS				
12	1	EXIT1-IR	IDLE	Last bit of instruction shifted in

Table 586. Loading NEXUS-ENABLE instruction (continued)

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
13	1	UPDATE-IR	IDLE	NEXUS-ENABLE loaded into instruction register
14	0	RUN-TEST/IDLE	REG_SELECT	Ready to be read/write Nexus registers

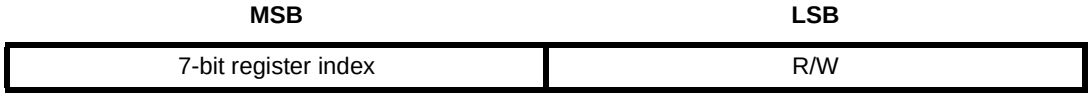
35.5.3.4 Selecting a Nexus Client Register

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 754*. The read/write control bit is set to 1 for writes and 0 for reads.

Figure 754. 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 587 illustrates a sequence which writes a 32-bit value to a register

Table 587. Write to a 32-Bit Nexus Client Register

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
0	0	RUN-TEST/IDLE	REG_SELECT	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	REG_SELECT	First pass through SELECT-DR-SCAN path
2	0	CAPTURE-DR	REG_SELECT	Internal shifter loaded with current value of controller command input.
3	0	SHIFT-DR	REG_SELECT	TDO becomes active, and write bit and 6 bits of register index shifted in.
7 TCKs				
12	1	EXIT1-DR	REG_SELECT	Last bit of register index shifted into TDI
13	1	UPDATE-DR	REG_SELECT	Controller decodes and selects register
14	1	SELECT-DR-SCAN	DATA_ACCESS	Second pass through SELECT-DR-SCAN path

Table 587. Write to a 32-Bit Nexus Client Register (continued)

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
15	0	CAPTURE-DR	DATA_ACCESS	Internal shifter loaded with current value of register
16	0	SHIFT-DR	DATA_ACCESS	TDO becomes active, and outputs current value of register while new value is shifted in through TDI
31 TCKs				
48	1	EXIT1-DR	DATA_ACCESS	Last bit of current value shifted out TDO. Last bit of new value shifted in TDI.
49	1	UPDATE-DR	DATA_ACCESS	Value written to register
50	0	RUN-TEST/IDLE	REG_SELECT	Controller returned to idle state. It could also return to SELECT-DR-SCAN to write another register.

35.5.4 Nexus JTAG Port Sharing

Each of the individual Nexus blocks on the device implements a TAP controller for accessing its registers. When Nexus has ownership of the TAP, only the block whose NEXUS-ENABLE instruction is loaded has control of the TAP. This allows the interface to all of these individual TAP controllers to appear to be a single port from outside the device. If no register is selected as the shift path for a Nexus block, that block acts like a single-bit shift register, or bypass register.

35.5.5 MCKO

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock and its frequency is determined by the value of the MCKO_DIV field in the PCR. Possible operating frequencies include system clock, one-half system clock, one-quarter system clock, and one-eighth system clock speed.

The NPC also generates an MCKO clock gating control output signal. This output can be used by the MCKO generation logic to gate the transmission of MCKO when the auxiliary port is enabled but not transmitting messages. The setting of the MCKO_GT bit inside the PCR determines whether or not MCKO gating control is active. The MCKO_GT bit resets to a logic 0. In this state gating of MCKO is disabled. To enable gating of MCKO, the MCKO_GT bit in the PCR is written to a logic 1.

35.5.6 $\overline{\text{EVTO}}$ Sharing

The NPC block controls sharing of the $\overline{\text{EVTO}}$ output between all Nexus clients that produce an $\overline{\text{EVTO}}$ signal. The NPC assumes incoming $\overline{\text{EVTO}}$ signals will be asserted for one system clock period. After receiving a single clock period of asserted $\overline{\text{EVTO}}$ from any Nexus client, the NPC latches the result, and drives $\overline{\text{EVTO}}$ for one MCKO period on the following clock. When there is no active MCKO, such as in disabled mode, the NPC drives $\overline{\text{EVTO}}$ for two system clock periods. $\overline{\text{EVTO}}$ sharing is active as long as the NPC is not in reset.

35.5.7 Nexus Reset Control

The JCOMP input that is used as the primary reset signal for the NPC is also used by the NPC to generate a single-bit reset signal for other Nexus blocks. If JCOMP is negated, an internal reset is asserted, indicating that all Nexus modules should be held in reset. Internal Nexus reset is also asserted when the device is in censored mode.

35.5.8 System Clock Locked Indication

Following a power-on reset, MDO[0] can be monitored to provide the lock status of the system clock. MDO[0] is driven to a logic 1 until the system clock achieves lock after exiting power-on reset. Once the system clock is locked, MDO[0] is negated and tools may begin Nexus configuration. Loss of lock conditions that occur subsequent to the exit of power-on reset and the initial lock of the system clock do not cause a Nexus reset, and therefore do not result in MDO[0] driven high.

35.6 Initialization/Application Information

35.6.1 Accessing NPC tool-mapped registers

To initialize the TAP for Nexus register accesses, the following sequence is required:

1. Enable the Nexus TAP controller
2. Load the TAP controller with the NEXUS-ENABLE instruction

To write control data to NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and set the write bit to select the register with a pass through the SELECT-DR-SCAN path in the TAP controller state machine.
2. Write the register value with a second pass through the SELECT-DR-SCAN path. Note that the prior value of this register is shifted out during the write.

To read status and control data from NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and clear the write bit to select register with a pass through SELECT-DR-SCAN path in the TAP controller state machine.
2. Read the register value with a second pass through the SELECT-DR-SCAN path. Data shifted in is ignored.

See the IEEE-ISTO 5001-2003 standard for more detail.

36 Oscillators

This chapter describes the following modules on this device:

- Internal RC oscillator (IRCOSC)
- External oscillator (XOSC)

36.1 IRCOSC 16 MHz internal RC oscillator

The internal RC oscillator has a nominal frequency of 16 MHz. The major features of the oscillator are:

- Glitch-free oscillation
- 6-bit trimming

The value of resistor is trimmed back using trim RCTRIM[5:0] of the RC_CTL register, so that the output clock frequency is never out of ±1% of 16 MHz over the process. After the oscillator is trimmed within this range using Trim Bits, the variation over the voltage and temperature is ±5%.

After power on reset, the trimming bits are provided by the flash options.

Table 588. RC Digital Interface Registers - base address 0xC3FE_0060

Register Name	Address Offset	Reset Value
RC_CTL	0x00	0x0000

Figure 755. RC Control Register (RC_CTL)

Offset 0x0

Access: User read
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	RCTRIM					
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	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

= Unimplemented or reserved

Table 589. RC_CTL field descriptions

Field	Description
RCTRIM	Main RC trimming bits

36.2 XOSC external oscillator

The external crystal oscillator (XOSC) works in the range of 4–40 MHz.

The XOSC digital interface contains control and status registers accessible to applications.

Main features are:

- Oscillator clock available interrupt
- Oscillator bypass mode

36.2.1 Functional description

The external crystal oscillator 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 external crystal oscillator can be controlled by the MC_ME. The OSCON bit of ME_xxx_MCR registers controls the powerdown of oscillator based on the current device mode while S_OSC of ME_GS register provides the oscillator clock available status.

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

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

Table 590 shows the truth table of different configurations of the oscillator.

Table 590. Truth table of the external crystal oscillator

OSCON	OSCBYP	XTAL	EXTAL	XOSC_CLK	XOSC MODE
0	0	No crystal, Hiz	No crystal, Hiz	0	Power down, 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

36.2.2 NVM interface

The NVM provides the XOSC with initial configuration information based on the flash memory option bit located at BIU4[1], the XOSC oscillation margin bit, as follows:

- If BIU4[1] = 1, the oscillation margin is higher, but the XOSC consumes more power.
- If BIU4[1] = 0, the oscillation margin is lower, and the XOSC consumes less power.

See the data sheet for the recommended value based on the oscillator frequency.

36.2.3 Register description

Figure 756. External Crystal Oscillator Control Register (OSC_CTL)

Offset 0x0

Access: User read
Supervisor read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M_OSC	0	0	0	0	0	0	0	I_OSC(2)	0						
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or reserved

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 591. OSC_CTL field descriptions

Field	Description
OSCBYP	External Crystal Oscillator bypass This bit specifies whether the oscillator should be bypassed or not. System reset is needed to reset this bit. 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. This counting period ensures that external oscillator clock signal is stable before it can be selected by the system. When oscillator counter reaches the value EOCV[7:0]*512, oscillator available interrupt request is generated. The reset value of this field depends on the device specification. The OSCCNT counter will be kept under reset if oscillator bypass mode is selected.
M_OSC	External Crystal oscillator clock interrupt mask 0: External crystal oscillator clock interrupt is masked. 1: External crystal oscillator clock interrupt is enabled.
I_OSC	External crystal oscillator clock interrupt This bit is set by hardware when OSCCNT counter reaches the count value EOCV[7:0]*512. It is cleared by software by writing ‘1’. 0: No oscillator clock interrupt occurred. 1: Oscillator clock interrupt pending.

37 Periodic Interrupt Timer (PIT)

37.1 Introduction

Figure 757 shows the PIT block diagram.

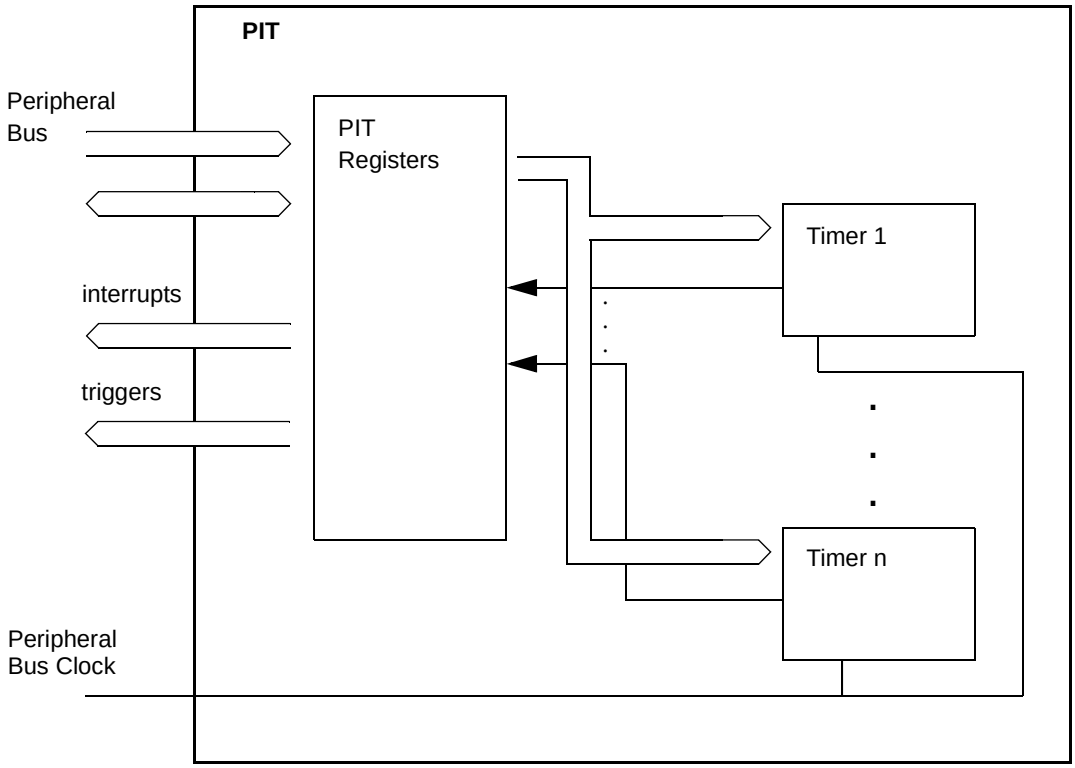


Figure 757. PIT block diagram

37.1.1 Overview

This specification describes the function of the Periodic Interrupt Timer block (PIT). The PIT is an array of timers that can be used to raise interrupts and trigger DMA channels.

Note: The PIT and the DMA need to run at the same synchronous frequency.
For the PIT frequency, it should be considered that the PIT runs with the peripheral set 0 clock, which also clocks DSPI's, FlexCAN's, and LinFlex.

37.1.2 Features

The main features of this block are:

- Timers can generate DMA trigger pulses
- Timers can generate interrupts
- All interrupts are maskable
- Independent timeout periods for each timer

37.2 Signal description

The PIT module has no external pins.

37.3 Memory map and register description

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

37.3.1 Memory map

Table 592 gives an overview on all PIT registers.

Table 592. PIT memory map

Address Offset	Use	Access
0x000	PIT Module Control Register	R/W
0x004 - 0x0FC	Reserved	R
0x100 - 0x10C	Timer Channel 0	(1)
0x110 - 0x11C	Timer Channel 1	(1)
0x120 - 0x12C	Timer Channel 2	(1)
0x130 - 0x13C	Timer Channel 3	(1)
0x140 - 0x1FC	Reserved	R

1. See *Table 593*

Table 593. Timer Channel n

Address Offset	Use	Access
channel + 0x00	Timer Load Value Register	R/W
channel + 0x04	Current Timer Value Register	R
channel + 0x08	Timer Control Register	R/W
channel + 0x0C	Timer Flag Register	R/W

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.

37.3.2 Register descriptions

This section describes in address order all the PIT registers and their individual bits.

37.3.2.1 PIT Module Control Register (PITMCR)

This register controls whether the timer clocks should be enabled and whether the timers should run in debug mode.

Figure 758. PIT Module Control Registers (PITMCR)

Offset 0x000 Access: Read/Write

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

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

Table 594. PITMCR Field Descriptions

Field	Description
MDIS	Module Disable. This is used to disable the module clock. This bit should be enabled before any other setup is done. 0 Clock for PIT Timers is enabled (default) 1 Clock for PIT Timers is disabled
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.

37.3.2.2 Timer Load Value Register (LDVAL)

These registers select the timeout period for the timer interrupts.

Figure 759. Timer Load Value Register (LDVAL)

Offset channel_base + 0x00 Access: Read/Write

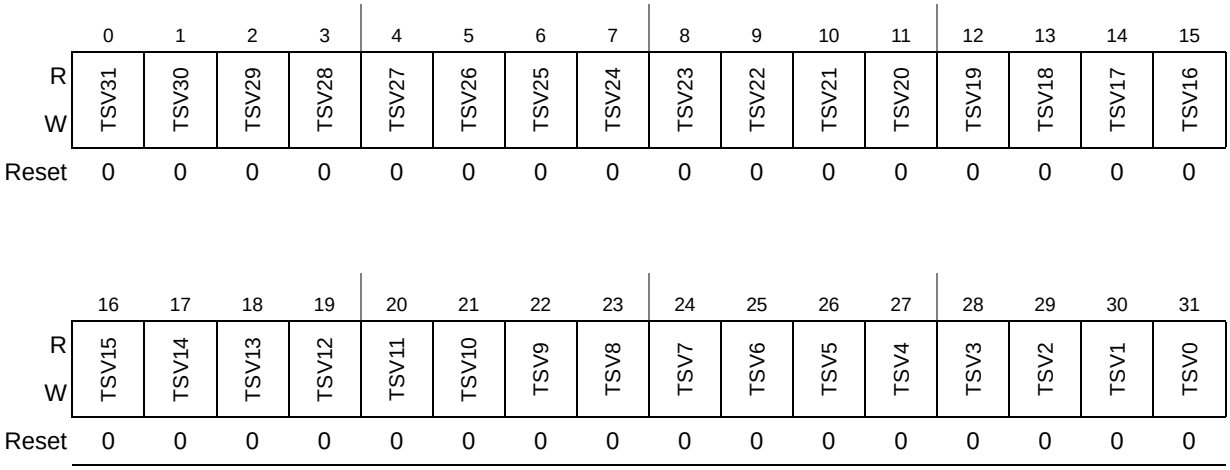


Table 595. LDVAL Field Descriptions

Field	Description
TSV*n*	Time Start Value Bits. These bits set the timer start value. The timer will count down until it reaches 0, then it will generate an interrupt and load this register value again. Writing a new value to this register will not restart the timer, instead the value will be loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see *Figure 764*).

37.3.2.3 Current Timer Value Register (CVAL)

These registers indicate the current timer position.

Figure 760. Current Timer Value Register (CVAL)

Offset channel_base + 0x04 Access: Read-only

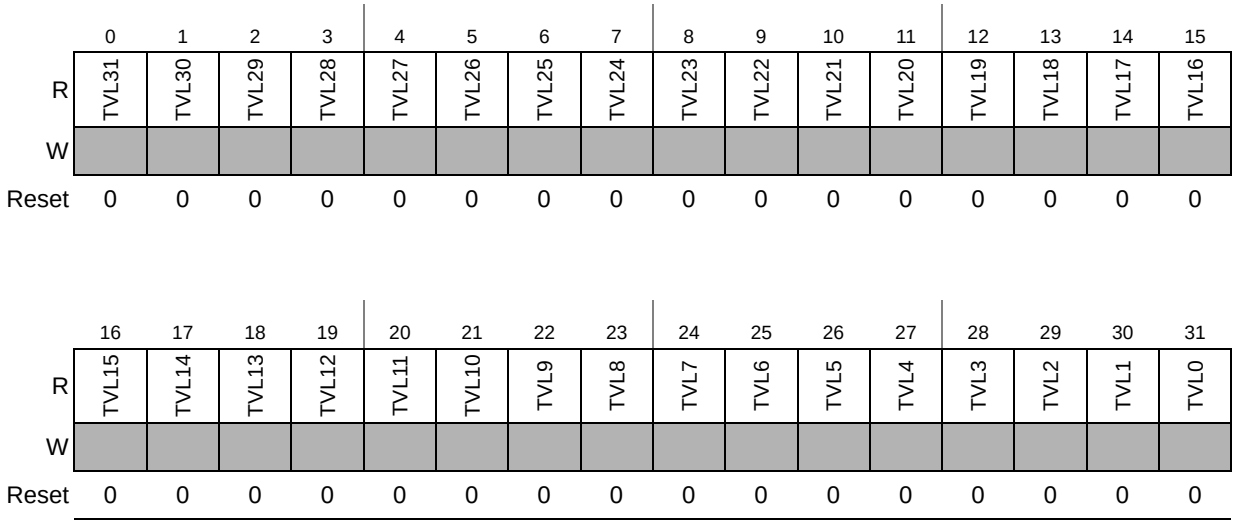


Table 596. CVAL Field Descriptions

Field	Description
TVL*n*	Current Timer Value. These bits represent the current timer value. Note that the timer uses a downcounter. NOTE: The timer values will be frozen in Debug mode if the FRZ bit is set in the PIT Module Control Register (see *Figure 758*).

37.3.2.4 Timer Control Register (TCTRL)

These register contain the control bits for each timer.

Figure 761. Timer Control Register (TCTRL)

Offset channel_base + 0x08 Access: Read/Write

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

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

Table 597. 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. 0Timer will be disabled 1Timer will be active

37.3.2.5 Timer Flag Register (TFLG)

These registers hold the PIT interrupt flags.

Figure 762. Timer Flag Register (TFLG)

Offset channel_base + 0x0C Access: Read/Write

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

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

Table 598. 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

37.4 Functional Description

37.4.1 General

This section gives detailed information on the internal operation of the module. Each timer can be used to generate trigger pulses as well as to generate interrupts, each interrupt will be available on a separate interrupt line.

37.4.1.1 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 763*).

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 764*).

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 765*).

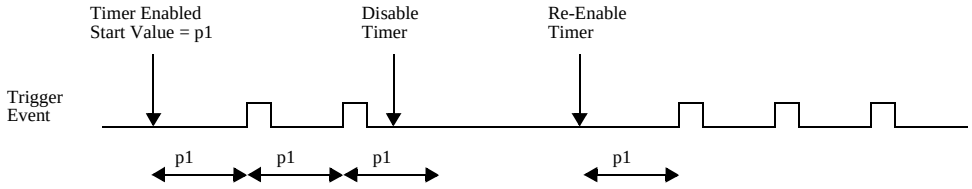


Figure 763. Stopping and Starting a Timer

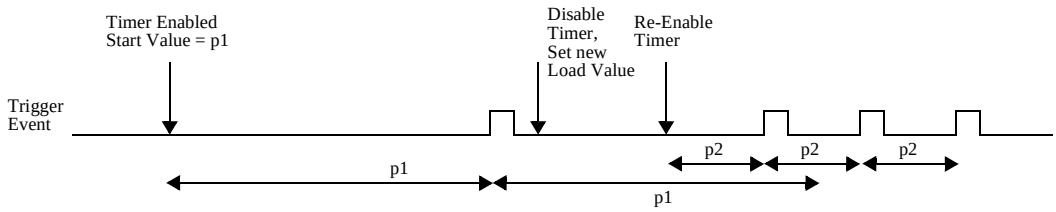


Figure 764. Modifying Running Timer Period

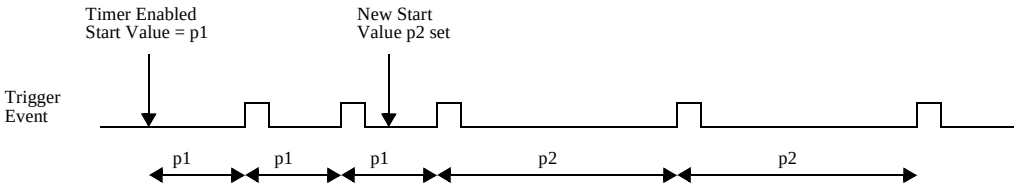


Figure 765. Dynamically Setting a New Load Value

37.4.1.2 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 (e.g. the timer values) and then continue the operation.

37.4.2 Interrupts

All of the timers support interrupt generation. Refer to the MCU specification 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.

37.5 Initialization and Application Information

37.5.1 Example Configuration

In the example configuration:

- the PIT clock has a frequency of 50 MHz
- timer 1 shall create an interrupt every 5.12 ms
- timer 3 shall create a trigger event every 30 ms

First the PIT module needs to be activated by writing a 0 to the MDIS bit in the PITMCR register.

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.

This means that LDVAL1 with 0003E7FF hex and LDVAL3 with 0016E35F hex.

The interrupt for Timer 1 is enabled by setting TIE in the TCTRL1 register. The timer is started by writing a 1 to bit TEN in the TCTRL1 register.

Timer 3 shall be used only for triggering. Therefore Timer 3 is started by writing a 1 to bit TEN in the TCTRL3 register, bit TIE stays at 0.

The following example code matches the described setup:

```
// turn on PIT
PIT_CTRL = 0x00;

// Timer 1
PIT_LDVAL1 = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_TCTRL1 = TIE; // enable Timer 1 interrupts
PIT_TCTRL1 |= TEN; // start timer 1

// Timer 3
PIT_LDVAL3 = 0x0016E35F; // setup timer 3for 1500000 cycles
PIT_TCTRL3 = TEN; // start timer 3
```

38 Peripheral Bridge (PBRIDGE)

38.1 Introduction

The peripheral bridge (PBRIDGE) is the interface between the system bus and on-chip peripherals. Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the slave bus interface.

The PBRIDGE will automatically manage peripheral access requests from software. In other words, you can access a peripheral using its memory-mapped interface without having to configure the PBRIDGE. However, the PBRIDGE also supports special peripheral access, such as memory protection, that is available for you to use if you need it.

This device includes two instantiations of the PBRIDGE. These are called PBRIDGE_0 and PBRIDGE_1.

38.2 Block interface

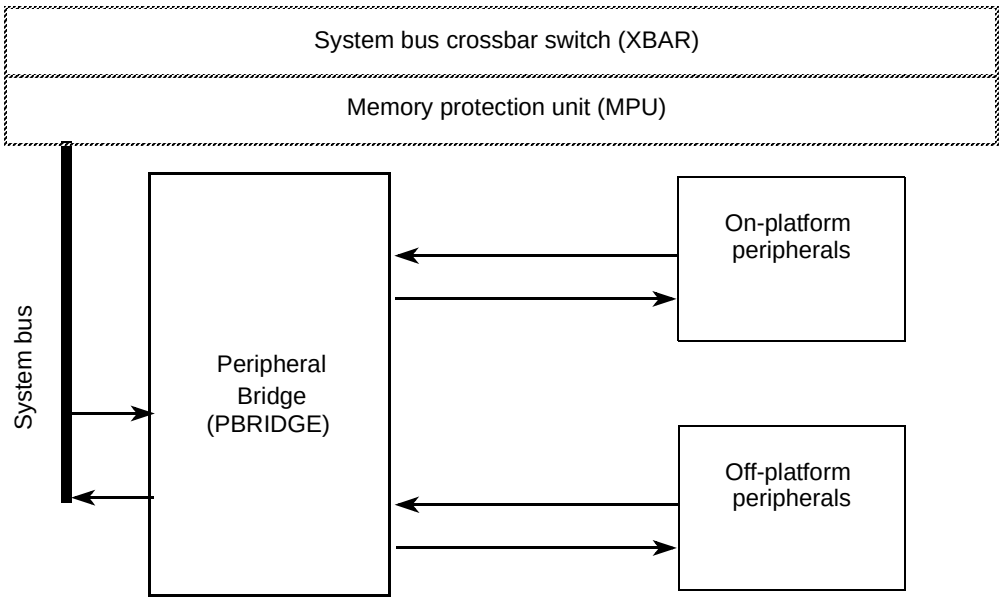


Figure 766. PBRIDGE interface

38.3 Features

The following list summarizes the key features of the PBRIDGE:

- Includes a 32-bit address bus and 64-bit data bus
- Supports 32-bit slave peripherals (byte, halfword, and word reads and writes are supported by the bridge)
- Provides configurable per-master access protections
- Provides configurable per-peripheral access protections for both on-platform and off-platform peripherals

38.4 Memory map and register description

38.4.1 Register access

All registers are 32-bit registers and can only be accessed in supervisor mode by trusted bus masters. Additionally, these registers must only be read from or written to by a 32-bit aligned access.

Two system clock cycles are required for read accesses and three system clock cycles are required for write accesses to the PBRIDGE registers.

38.4.2 Memory map

Each register in the PBRIDGE module has a size of 32 bits. The registers are listed in *Table 599* . The memory map organization is shown in *Table 600*. The organizational hierarchy is as follows:

- The module has multiple registers with the same register name (MPROT, PACR, OPACR), each at a different address offset.
- Each register has multiple similarly-named fields, each with a different number.
- Each field has subfields as defined elsewhere in this section.

Accesses to registers or register fields marked as reserved will return undefined data on reads, and will be ignored on writes.

Table 599. PBRIDGE registers

Offset from PBRIDGE_BASE PBRIDGE0_0 = 0xFFF0_0000 PBRIDGE_1 = 0xFFF0_4000	Register	Access (1)	Reset Value	Location
0x0000–0x0007 (2)	Master Protection Registers (MPROT)	R/W	—(3)	*on page 1180*
0x0008–0x001F	Reserved			
0x0020–0x003F (2)	Peripheral Access Control Registers (PACR)	R/W	—(3)	*on page 1181*
0x0040–0x006F (2)	Off-Platform Peripheral Access Control Registers (OPACR)	R/W	—(3)	*on page 1182*
0x0070–0x3FFF	Reserved			

1. In this column, R/W = Read/Write, R = Read-only, and W = Write-only.
2. This memory range contains reserved areas. See *Table 600*.

3. See the associated description for more information.

Table 600. PBRIDGE memory map

Address offset	Register name	Bit numbers							
		0–3	4–7	8–11	12–15	16–19	20–23	24–27	28–31
0x0000	MPROT	MPROT0	MPROT1	MPROT2	MPROT3	Reserved	Reserved	MPROT6	Reserved
0x0004 (cut1)		Reserved							
0x0004 (cut2/3)		MPROT8	MPROT9						
0x0020	PACR	PACR0	PACR1	Reserved		PACR4	Reserved		
0x0024		Reserved	PACR9	Reserved				PACR14	PACR15
0x0028		PACR16	PACR17	PACR18	Reserved				
0x002C		Reserved							
0x0040	OPACR	Reserved				OPACR4	OPACR5	OPACR6	Reserved
0x0044		Reserved							
0x0048		OPACR16	OPACR17	Reserved					OPACR23
0x004C		OPACR24	Reserved						OPACR31
0x0050		OPACR32	OPACR33	Reserved	OPACR35	Reserved		OPACR38	OPACR39
0x0054		OPACR40	OPACR41	OPACR42	Reserved				
0x0058		OPACR48	OPACR49	Reserved					
0x005C		Reserved		OPACR58	OPACR59	Reserved		OPACR62	Reserved
0x0060		Reserved		OPACR66	Reserved	OPACR68	OPACR69	Reserved	
0x0064		Reserved							
0x0068		Reserved						OPACR86	OPACR87
0x006C		OPACR88	OPACR89	OPACR90	Reserved	OPACR92	OPACR93	Reserved	

38.4.3 Register descriptions

38.4.3.1 Master Protection Registers (MPROT)

Each MPROT register contains one or more 4-bit fields, called MPROT*n*, as shown in *Table 600*. Each of these fields defines the access privilege level associated with bus master *n* in the platform. The registers provide one field per bus master. See *Section 15.1.4, Logical master IDs*, for a list of master numbers and names.

Each MPROT*n* field has the structure described in *Figure 767* and *Table 602*.

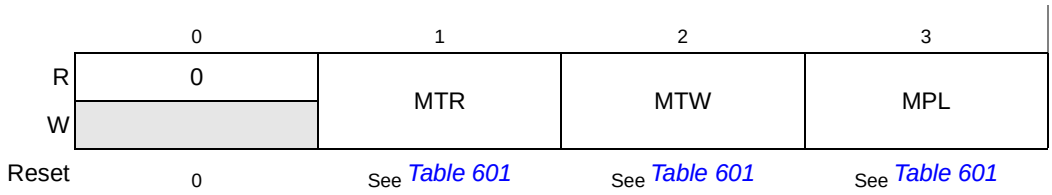


Figure 767. MPROT*n* field structure

Table 601. MPROT*n* field reset values

n	Mode	MTR	MTW	MPL
0	LSM and DPM	1	1	1
1	DPM	1	1	1
8	LSM and DPM	1	1	1
9 (on cut1)	LSM and DPM	1	1	1
9 (on cut2/3)	DPM (on cut2/3)	1	1	1
All others	LSM and DPM	0	0	0

Table 602. MPROT*n* field structure descriptions

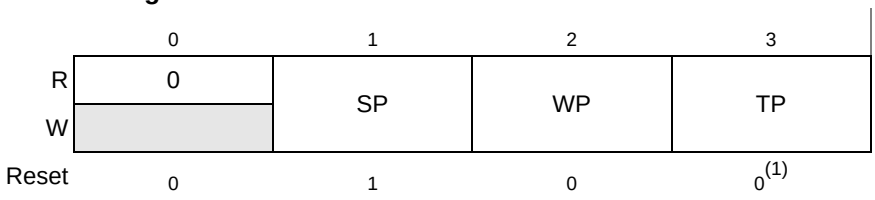
Subfield	Description
MTR	Master Trusted for Reads. This bit determines whether the master is trusted for read accesses. 0 This master is not trusted for read accesses. 1 This master is trusted for read accesses.
MTW	Master Trusted for Writes. This bit determines whether the master is trusted for write accesses. 0 This master is not trusted for write accesses. 1 This master is trusted for write accesses.
MPL	Master Privilege Level. This bit determines how the privilege level of the master is determined. 0 Accesses from this master are forced to user-mode. 1 Accesses from this master are not forced to user-mode.

38.4.3.2 Peripheral access control registers (PACR)

Each PACR register contains one or more 4-bit fields, called PACR*n*, as shown in *Table 600*. Each of these fields defines the access levels supported by the associated module. The lists of modules and their corresponding numbers are shown in *Table 604*.

Each PACR*n* field has the structure described in *Figure 768* and *Table 603*.

Figure 768. PACR*n* field structure



1. The reset state of PACR0[TP] is 1, not 0. PACR0[SP] and PACR0[TP] are hard wired to 1 and cannot be changed. Therefore, untrusted masters or user-mode accesses are always denied.

Table 603. PACR*n* field structure descriptions

Subfield	Description
SP	Supervisor Protect. This bit determines whether the peripheral requires supervisor privilege level for access. 0 This peripheral does not require supervisor privilege level for accesses. 1 This peripheral requires supervisor privilege level for accesses. The MPROTx[MPL] control bit for the master must be set. If not, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.
WP	Write Protect. This bit determines whether the peripheral allows write accesses. 0 This peripheral allows write accesses. 1 This peripheral is write protected. If a write access is attempted, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.
TP	Trusted Protect. This bit determines whether the peripheral allows accesses from an untrusted master. 0 Accesses from an untrusted master are allowed. 1 Accesses from an untrusted master are not allowed. If an access is attempted by an untrusted master, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.

Table 604. On-platform peripherals and PACR numbers

Peripheral	PACR number
PBRIDGE	0
XBAR	1
MPU	4
SEMA4	9
SWT	14
STM	15
ECSM	16
eDMA	17
INTC	18

38.4.3.3 Off-platform peripheral access control registers (OPACR)

The OPACR defines the access levels supported by the associated module. Each OPACR has a format identical to the PACR described in *Section 38.4.3.2, Peripheral access control registers (PACR).*

The lists of off-platform modules and their corresponding numbers are listed in *Table 605*. The OPACR number mirrors the PCTL column of *Table 3* in *Chapter 3: Memory Map*.

Table 605. Off-platform peripherals and OPACR numbers

Peripheral	OPACR number
DSPI 0	4
DSPI 1	5

Table 605. Off-platform peripherals and OPACR numbers (continued)

Peripheral	OPACR number
DSPI 2	6
FlexCAN 0	16
FlexCAN 1	17
DMA_MUX	23
FlexRay controller	24
BAM	31
ADC0	32
ADC1	33
CTU	35
eTimer 0	38
eTimer 1	39
eTimer 2	40
FlexPWM 0	41
FlexPWM 1	42
LINFlex 0	48
LINFlex 1	49
CRC	58
FCCU	59
SWG	62
FLASH0	66
SIUL	68
WKPU	69
SSCM	86
MC_ME	87
MC_CGM	88
MC_RGM	89
MC_PCU	90
PIT	92
STCU	93

38.5 Functional description

The PBRIDGE serves as an interface between a system bus and the peripheral (slave) bus. It functions as a protocol translator. Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the slave bus interface.

There is no need to configure the PBRIDGE registers unless the default master and/or peripheral access privileges need to be changed.

38.5.1 Access support

Aligned 64-bit word accesses, halfword accesses, and byte accesses are supported for the peripherals. Peripheral registers must not be misaligned, although no explicit checking is performed by the PBRIDGE.

Note: Data accesses that cross a 32-bit boundary are not supported.

38.5.1.1 Peripheral write buffering

Buffered writes are not supported by the PBRIDGE.

38.5.1.2 Read cycles

Two-clock read accesses are possible with the PBRIDGE when the requested access size is 32-bits or smaller, and is not misaligned across a 32-bit boundary.

38.5.1.3 Write cycles

Three clock write accesses are possible with the PBRIDGE when the requested access size is 32-bits or smaller. Misaligned writes that cross a 32-bit boundary are not supported.

38.5.2 General operation

Slave peripherals are modules that contain readable/writable control and status registers. The system bus master reads and writes these registers through the PBRIDGE. The PBRIDGE generates module enables, the module address, transfer attributes, byte enables, and write data as inputs to the slave peripherals. The PBRIDGE captures read data from the slave interface and drives it on the system bus.

The PBRIDGE occupies a 64 MB portion of the address space. The register maps of the slave peripherals are located on 16-KB boundaries. Each slave peripheral is allocated one 16-KB block of the memory map, and is activated by one of the module enables from the PBRIDGE.

The PBRIDGE is responsible for indicating to slave peripherals if an access is in supervisor or user mode. All eDMA and FlexRay transfers are done in supervisor mode.

39 Power Control Unit (MC_PCU)

39.1 Introduction

39.1.1 Overview

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

Figure 769 depicts the MC_PCU block diagram.

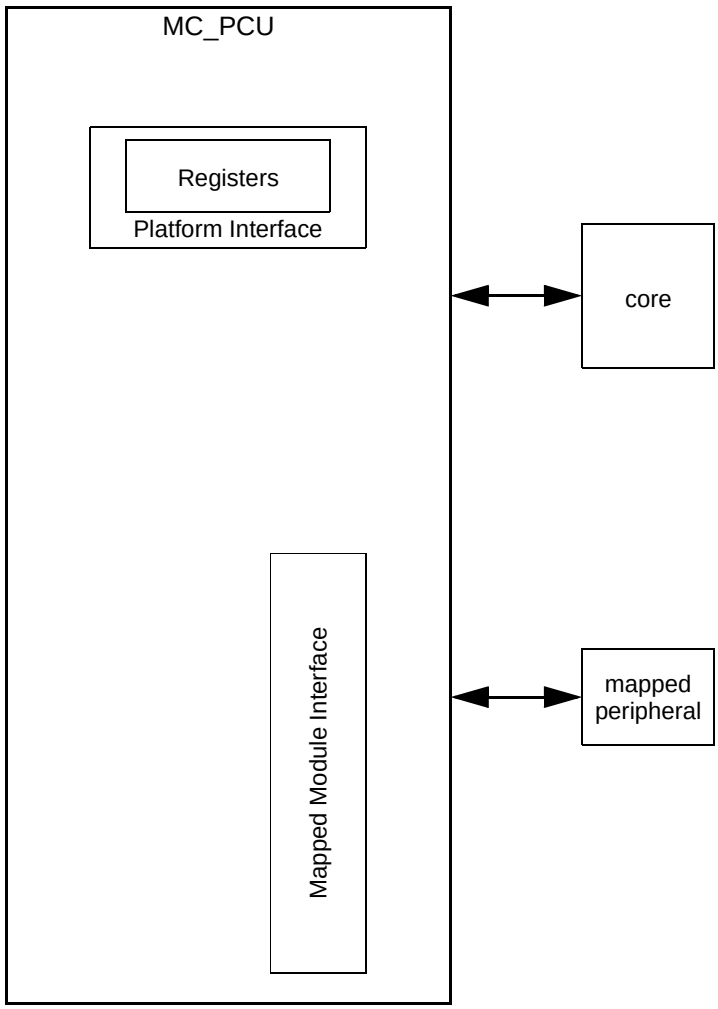


Figure 769. MC_PCU Block Diagram

39.1.2 Features

The MC_PCU includes the following features:

- maps the PMU registers to the MC_PCU address space

39.2 External Signal Description

The MC_PCU has no connections to any external pins.

39.3 Memory Map and Register Definition

39.3.1 Memory Map

Table 606. MC_PCU Register Description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_8040	PCU_PSTAT	Power Domain Status Register	word	read	read	read	*on page 1187*

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 607. MC_PCU Memory Map

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_8000 … 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																PD0
		W																
0xC3FE_8044 … 0xC3FE_807C	reserved																	
0xC3FE_8080 … 0xC3FE_80FC	PMU registers																	
0xC3FE_8100 … 0xC3FE_BFFC	reserved																	

39.3.2 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.

39.3.2.1 Power Domain Status Register (PCU_PSTAT)

Figure 770. 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																PD0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

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

Table 608. 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

40 Power Management Unit (PMU)

40.1 Overview

The PMU generates the 1.2 V core logic supply from a 3.3 V (nominal) input supply by means of a linear voltage regulator driving an external NPN bipolar transistor (emitter-follower configuration) or an internal PMOS FET.

The PMU always starts up using the internal ballast transistor. It then executes an automatic procedure that detects (during the system reset phase) whether an external ballast transistor is operational. If a functional external ballast transistor is detected, the system is supplied by the external transistor only. The information whether the internal or the external ballast transistor is used is available via a bit in one of the PMU registers.

The operating voltages are monitored by a set of on-chip supervisory circuits to ensure that this device works within the correct voltage range. These circuits are:

- Low-Voltage Inhibit (LVI), also known as Low-Voltage Detector (LVD)
- High-Voltage Inhibit (HVI), also known as High-Voltage Detector (HVD)
- Comparators

Main digital low- and high-voltage monitoring circuits are tested by integrated self test circuitry. The Voltage Regulator, IO and Flash dedicated Low Voltage monitoring circuitries are redundant in order to improve the safety coverage. The LVDs and the comparators provide their output signals to the Reset Generation Module (MC_RGM) and to the FCCU.

40.2 Block diagram

The PMU block diagram and its components are shown in *Figure 771* and *Table 609*, respectively.

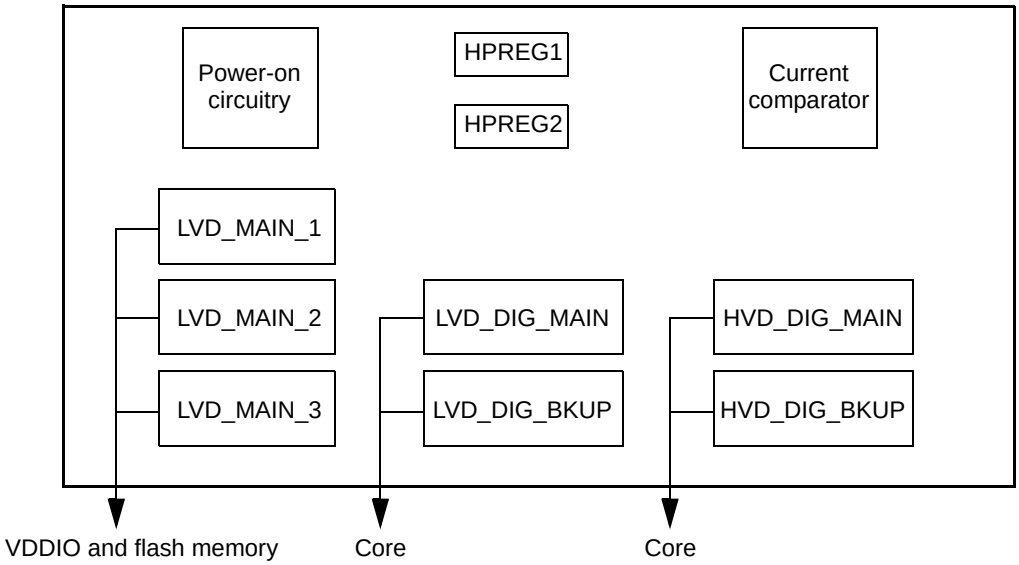


Figure 771. PMU block diagram

Table 609. PMU blocks

Abbreviation	Block name	Block function
HPREG1	High-power regulator 1	Provides internal ballast to support core current
HPREG2	High-power regulator 2	Provides external NPN to support core current
LVD_MAIN_1	Low-voltage detector 1	Monitors the 3.3 V supply of VDDIO
LVD_MAIN_2	Low-voltage detector 2	Monitors the 3.3 V supply of VDDREG
LVD_MAIN_3	Low-voltage detector 3	Monitors the 3.3 V supply of VDDFLASH
LVD_DIG_MAIN	Main digital low-voltage detector	Monitors the 1.2 V digital core supply (HPVDD)
LVD_DIG_BKUP	Backup digital low-voltage detector	Assists in the self-test of LVD_DIG_MAIN
HVD_DIG_MAIN	Main digital high-voltage detector	Monitors the 1.2 V digital core supply (HPVDD)
HVD_DIG_BKUP	Backup digital high-voltage detector	Assists in the self-test of LVD_DIG_MAIN

40.3 High-power regulators

The PMU includes two high-power regulators, HPREG1 and HPREG2.

HPREG1 uses internal ballast to support the core current, whereas HPREG2 is turned ON only when external NPN is present on board to supply core current. The device always powers up using HPREG1. If an external NPN is present, the device then makes a transition from HPREG1 to HPREG2. This transition is dynamic. Once HPREG2 is fully operational, controller part of HPREG1 is switched OFF. HPREG1 is used if no external ballast is used.

If an external NPN is connected to the device, HPREG2 takes control. The output voltage of HPREG2 is higher than HPREG1. This guarantees that internal Ballast (HPREG1) is turned OFF as soon as external NPN takes control. The base of the bipolar is driven by a control signal generated by the control part and available on a dedicated pin (BCTRL).

The supported bipolar transistors are the BCP68 (ON Semiconductor) and the BCX68 (Infineon).

Immediately after power-on reset, the PMU trims the output voltage using factory-specified parameters. The nominal target output after this trimming is 1.28 V.

The stabilization for HPREG is achieved using an external capacitance. The minimum required value is two 6 µF capacitors.

Note: The aforementioned minimum external capacitance value has to take into account the tolerance, temperature, voltage and aging variation. Be sure to limit the series inductance per pad to less than 13 nH.

40.4 High- and low-voltage detectors (HVD, LVD)

Two kind of LVDs are available:

- LVD_MAIN (3 blocks) for the 3.3V input supply with thresholds at 3.3 V level
- LVD_DIG for the 1.2 V output voltage

LVD_MAIN_1 senses the VDDIO supply and provides information to the system if VDDIO is not in the proper range. If the voltage is not in the proper range, the system responds with a reset.

LVD_MAIN_2 senses the VDDREG supply and provides information to the system if VDDREG is not in the proper range. If the voltage is not in the proper range, the system responds with a reset.

LVD_MAIN_3 senses the VDDFLASH supply and provides information to the system if VDDFLASH is not in the proper range. If the voltage is not in the proper range, the system responds with a reset.

The LVD_DIG_MAIN and LVD_DIG_BKUP blocks sense the HPREG output and provides information to the system if the output is not in the proper range. If the output is not in the proper range, the system responds with a reset.

The HVD_DIG_MAIN and HVD_DIG_BKUP blocks also sense the HPREG output and provides information to the system if the output is not in the proper range. If the output is not in the proper range, the system responds with a reset.

The POR is required to initialize the device during supply rise. The POR works only on the rising edge of main supply. To ensure its functioning during the following rising edge of the supply, it is reset by the output of either of MAIN_LVD_1 or MAIN_LVD_2 or MAIN_LVD_3. When main supply reaches below the lower voltage threshold of any of these LVDs, POR is reset and ready to work on next rising edge. When supply is below the POR threshold, POR output will be high and will follow the supply. When sufficient supply threshold is reached, POR will trigger and give ‘0’ level as output

40.4.1 Current comparator

This block mirrors a portion of the current flowing through the internal ballast. This current is compared with a reference current. If this ballast current is more than the reference current, the output of the block is logic “0” which indicates that there is no external NPN available on the board. In reverse case the output of the block is logic “1” which indicates the availability of the external NPN. Along with this circuit an option byte might also available to give the information about the availability of external NPN.

40.5 Power-up and initialization

After the device powers up, the PMU:

- Automatically detects whether an external ballast is present
- Applies the factory-specified trimming to the output
- Performs the built-in self-test (BIST)

The high-voltage detector is disabled while the PMU is initializating (detecting the external ballast and applying proper trimming), especially during the phase where the switchover occurs. Therefore, if during that time there is an overshoot of the 1.2 V supply, no reset is triggered.

40.6 Built In self-test (BIST)

The PMU provides (to the user) the software capability to check the run of the BIST procedure, generating non-critical faults (NCFs) or critical faults (CF) conditions for the FCCU module.

To provide the necessary redundancy, the logic AND of the two comparator outputs (from the main and back-up circuitries) is used as global system reset.

At each power-on, the self-test circuitry is able to detect a failure of one of the two LVDs and to provide an NCF to the FCCU.

The BIST flow is shown in *Figure 772*. The BIST execution is controlled by the PMUCTRL_CTRL[SILHT] field.

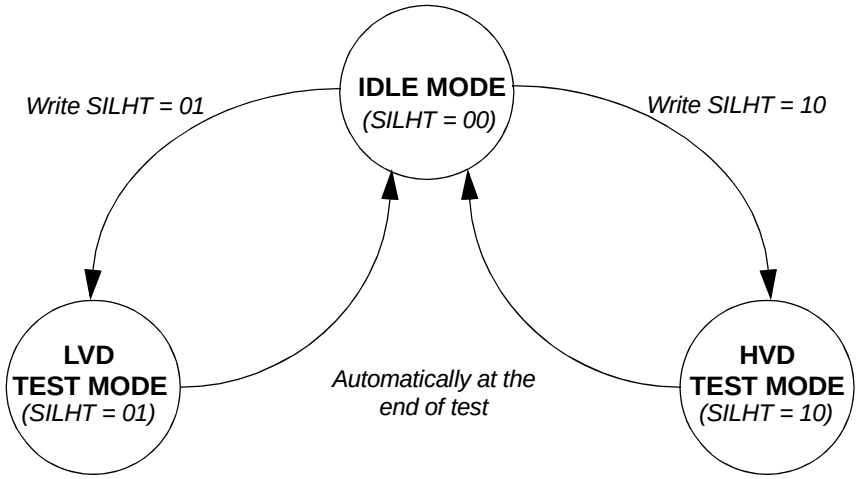


Figure 772. Built-In self test flow

Table 610 shows the critical and non-critical faults generated by the PMU during BIST mode.

Table 610. Fault assertion conditions

Fault number	Signal
NCF[13]	LVD BIST OK in test mode/ LVD NOK in user mode
NCF[14]	HVD BIST OK in test mode/ HVD NOK in user mode
NCF[15](1)	LVD VREG fault detected by self-checking
NCF[16](1)	LVD FLASH fault detected by self-checking
NCF[17](1)	LVD IO fault detected by self-checking
CF[21]	LVD/HVD BIST failure result in test mode

1. It can only be checked once after destructive reset. Once cleared they will be disabled permanently until the next destructive reset

All faults are monitored by the following fields in the PMUCTRL_FAULT register:

- LHCF
- LNCF
- HNCF

Before clearing the HVD and LVD critical fault by means of the FCCU, you must clear the following four pending fields in the PMUCTRL_IRQS register:

- MLVDP
- BLVDP
- MHVDP
- BHVDP

The critical fault number will be kept asserted as long as one of these fields is '1'.

When the PMU enters LVD or HVD test mode, it is not allowed to enter STOP mode. If the device enters stop mode after the pending status pin has been set it is not possible to reset it by SW until the CPU clock is

stopped.

The PMU can assert two faults on FLASH voltage monitor, two faults on IO voltage monitor and two faults on Regulator voltage monitor (corresponding to failure of either the main or backup LVDs). They are asserted (active low) during the power up phase, if a rising edge is not detected on the related LVD.

Each fault from the PMU sets:

- Its corresponding pending bit inside the PMUCTRL_IRQS register. If the associated interrupt in the PMUCTRL_IRQE register is enabled, the PMU will send an IRQ to the INTC.
- Its corresponding non critical fault

40.7 Memory map and register description

The PMU memory map is shown in *Table 611*.

The PMU registers are mapped into the MC_PCU address space (base address: 0xC3FE_8080).

Table 611. PMU memory map

Register Name	Address offset	Location
Reserved	0x00–0x3F	—
PMU status register (PMUCTRL_STATUS)	0x40	*on page 1193*
PMU control register (PMUCTRL_CTRL)	0x44	*on page 1194*
Reserved	0x48–0x6F	—
PMU mask fault register (PMUCTRL_MASKF)	0x70	*on page 1195*
PMU fault monitor register (PMUCTRL_FAULT)	0x74	*on page 1196*
PMU interrupt request status register (PMUCTRL_IRQS)	0x78	*on page 1197*
PMU interrupt request enable register (PMUCTRL_IRQE)	0x7C	*on page 1198*

40.7.1 PMUCTRL status register (PMUCTRL_STATUS)

Address: Base + 0x40

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

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

Figure 773. PMUCTRL status register (PMUCTRL_STATUS)

Table 612. PMUCTRL_STATUS field descriptions

Field	Description
ENPN	External NPN status flag 0 External NPN not detected 1 External NPN detected
CTB	Configuration Trace Bits. This field describes the PMU use case after initialization. 00 Reserved (not used on this device) 01 Internal ballast mode without NPN 10 External ballast mode detection with NPN 11 Reserved (not used on this device)

40.7.2 PMUCTRL control register (PMUCTRL_CTRL)

This register allows you to start or stop the LVD/HVD BIST procedure.

Address: Base + 0x44

	0	1	2	3	4	5	6	7	8	9	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	SILHT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 774. PMUCTRL control register (PMUCTRL_CTRL)

Table 613. PMUCTRL_CTRL field descriptions

Field	Description
SILHT	Start Idle or LVD or HVD BIST on main and backup lines 00 Idle mode 01 Start main and backup LVD BIST mode 10 Start main and backup HVD BIST mode 11 Reserved *Note: On the completion of the LVD/HVD BIST operation, the SILHT field clears itself (returns to idle mode).*

40.7.3 PMUCTRL mask fault register (PMUCTRL_MASKF)

This register allows you to mask the possible non-critical faults that are described in the PMUCTRL_FAULT register.

Address: Base + 0x70

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MF_BB				0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	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 775. PMUCTRL mask fault register (PMUCTRL_MASKF)

Table 614. PMUCTRL_MASKF field descriptions

Field	Description
MF_BB	Mask Fault Bypass Ballast. This field defines the mask for the nBYPASS_BALLAST_LV[3:0] bus. 0 MF_BB[n], means BYPASS_BALLAST_LV[n] = ‘0’; where n goes from 3 down to 0. 1 MF_BB[n], means BYPASS_BALLAST_LV[n] = NOT nBYPASS_BALLAST_LV[n]; where n goes from 3 down to 0.

40.7.4 PMUCTRL fault monitor register (PMUCTRL_FAULT)

Figure 776. PMUCTRL fault monitor register (PMUCTRL_FAULT)

Address: Base + 0x74

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BB_LV				0	0	0	0	0	0	0	0	0	FLNCF	IONCF	RENCF
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	LHCF	LNCF	HNCF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 615. PMUCTRL_FAULT field descriptions

Field	Description
BB_LV	Bypass Ballast Low Voltage. This field monitors the NOT of nBYPASS_BALLAST_LV[3:0] bus. 0 BB_LV[n] = NOT (nBYPASS_BALLAST_LV[n] = ‘1’); where n goes from 3 down to 0 1 BB_LV[n] = NOT (nBYPASS_BALLAST_LV[n] = ‘0’); where n goes from 3 down to 0
FLNCF	Flash memory voltage monitor non-critical fault 0 No flash memory voltage monitor fault is active 1 Flash memory voltage monitor fault is active
IONCF	IO voltage monitor non-critical fault 0 No IO voltage monitor fault is active 1 IO voltage monitor fault is active
RENCF	Regulator voltage monitor non-critical fault 0 No regulator voltage monitor fault is active 1 Regulator voltage monitor fault is active
LHCF	Low high voltage detector critical fault 0 No low high voltage fault is active 1 Low high voltage fault is active
LNCF	Low voltage detector non-critical fault 0 No low voltage fault is active 1 Low voltage fault is active
HNCF	High voltage detector non-critical fault 0 No high voltage fault is active 1 High voltage fault is active

40.7.5 PMUCTRL interrupt request status register (PMUCTRL_IRQS)

This register allows you to read and clear the various PMU interrupt status bits. To clear these bits, write a ‘1’ to them.

Address: Base + 0x78

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	MFVMP	BFVMP	MIVMP	BIVMP	MRVMP	BRVMP
W											w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	MLVDP	BLVDP	MHVDP	BHVDP
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 777. PMUCTRL interrupt request status register (PMUCTRL_IRQS)

Table 616. PMUCTRL_IRQS field descriptions

Field	Description
MFVMP	Main Flash Voltage Monitor interrupt Pending. 0 NO Main Flash Voltage Monitor interrupt fault error 1 Main Flash Voltage Monitor interrupt fault error
BFVMP	Backup Flash Voltage Monitor interrupt Pending. 0 NO Backup Flash Voltage Monitor interrupt fault error 1 Backup Flash Voltage Monitor interrupt fault error
MIVMP	Main Io Voltage Monitor interrupt Pending. 0 NO Main IO Voltage Monitor interrupt fault error 1 Main IO Voltage Monitor interrupt fault error
BIVMP	Backup IO Voltage Monitor interrupt Pending. 0 NO Back_up IO Voltage Monitor interrupt fault error 1 Back_up IO Voltage Monitor interrupt fault error
MRVMP	Main Regulator Voltage Monitor interrupt Pending. 0 NO Main Regulator Voltage Monitor interrupt fault error 1 Main Regulator Voltage Monitor interrupt fault error
BRVMP	Backup Regulator Voltage Monitor interrupt Enable. 0 NO Backup Regulator Voltage Monitor interrupt fault error 1 Backup Regulator Voltage Monitor interrupt fault error
MLVDP	Main Low Voltage Detector Error interrupt Pending. 0 NO Main Low Voltage Detector OUT interrupt error 1 Main Low Voltage Detector OUT interrupt error

Table 616. PMUCTRL_IRQS field descriptions (continued)

Field	Description
BLVDP	Backup Low Voltage Detector Error interrupt Pending. 0 NO Backup Low Voltage Detector OUT interrupt error 1 Backup Low Voltage Detector OUT interrupt error
MHVDP	Main High Voltage Detector Error interrupt Pending. 0 NO Main High Voltage Detector OUT interrupt error 1 Main High Voltage Detector OUT interrupt error
BHVDP	Backup High Voltage Detector Error interrupt Pending. 0 NO Back_up High Voltage Detector OUT interrupt error 1 Back_up Low Voltage Detector OUT interrupt error

40.7.6 PMUCTRL interrupt request enable register (PMUCTRL_IRQE)

This register allows you to enable or disable the various PMU interrupts.

Address: Base + 0x7C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	MFVME	BFVME	MIVME	BIVME	MRVME	BRVME
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	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	MLVDE	BLVDE	MHVDE	BHVDE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 778. PMUCTRL interrupt request enable register (PMUCTRL_IRQE)

Table 617. PMUCTRL_IRQE field descriptions

Field	Description
MFVME	Main Flash Voltage Monitor interrupt enable. 0 Main Flash Voltage Monitor interrupt fault error disabled 1 Main Flash Voltage Monitor interrupt fault error enabled
BFVME	Backup Flash Voltage Monitor interrupt enable. 0 Backup Flash Voltage Monitor interrupt fault error disabled 1 Backup Flash Voltage Monitor interrupt fault error enabled
MIVME	Main Io Voltage Monitor interrupt Enable. 0 Main IO Voltage Monitor interrupt fault error disabled 1 Main IO Voltage Monitor interrupt fault error enabled
BIVME	Backup IO Voltage Monitor interrupt Enable. 0 Backup IO Voltage Monitor interrupt fault error disabled 1 Backup IO Voltage Monitor interrupt fault error enabled

Table 617. PMUCTRL_IRQE field descriptions (continued)

Field	Description
MRVME	Main Regulator Voltage Monitor interrupt Enable. 0 Main Regulator Voltage Monitor interrupt fault error disabled 1 Main Regulator Voltage Monitor interrupt fault error enabled
BRVME	Backup Regulator Voltage Monitor interrupt Enable. 0 Back_up Regulator Voltage Monitor interrupt fault error disabled 1 Back_up Regulator Voltage Monitor interrupt fault error enabled
MLVDE	Main Low Voltage Detector Error interrupt enable. 0 Main Low Voltage Detector OUT interrupt error disabled 1 Main Low Voltage Detector OUT interrupt error enabled
BLVDE	Backup Low Voltage Detector Error interrupt enable. 0 Backup Low Voltage Detector OUT interrupt error disabled 1 Backup Low Voltage Detector OUT interrupt error enabled
MHVDE	Main High Voltage Detector Error interrupt enable. 0 Main High Voltage Detector OUT interrupt error disabled 1 Main High Voltage Detector OUT interrupt error enabled
BHVDE	Backup High Voltage Detector Error interrupt enable. 0 NO Backup High Voltage Detector OUT interrupt error disabled 1 Backup Low Voltage Detector OUT interrupt error enabled

41 Register Protection (REG_PROT)

41.1 Introduction

41.1.1 Overview

The REG_PROT module offers a mechanism to protect defined memory-mapped address locations in a module under protection from being written. The address locations that can be protected are module-specific.

The protection module is located between the module under protection and the PBRIDGE. (For an explanation of the REG_PROT and module base addresses, see *Section 41.3, Memory map and register definition*.) This is shown in *Figure 779*.

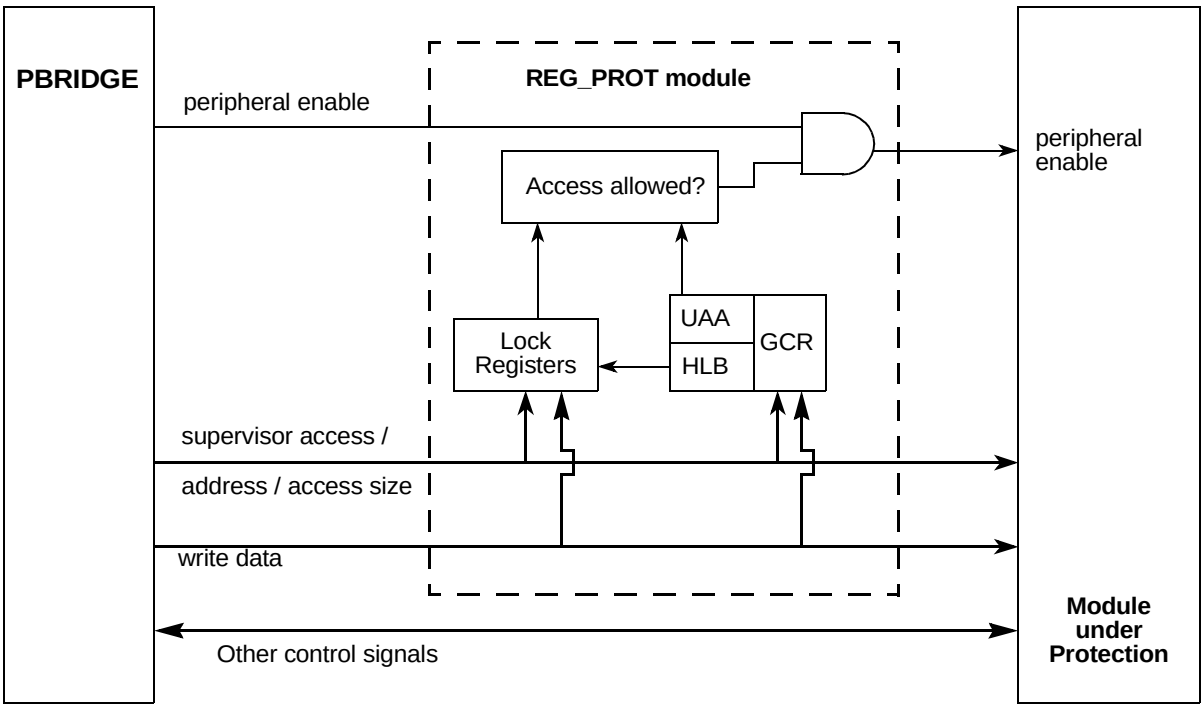


Figure 779. REG_PROT block diagram

41.1.2 Features

The REG_PROT includes these distinctive features:

- Restrict write accesses for the module under protection to supervisor mode only
- Lock registers for first 6 KB of memory-mapped address space
- Write to address mirror automatically sets corresponding lock bit
- Once configured lock bits can be protected from changes

41.1.3 Modes of operation

The REG_PROT module is operable when the module under protection is operable.

41.2 External signal description

There are no external signals.

41.3 Memory map and register definition

This section provides a detailed description of the memory map of a module using the REG_PROT. The original 16 KB module memory space is divided into 5 areas as shown in *Figure 780*.

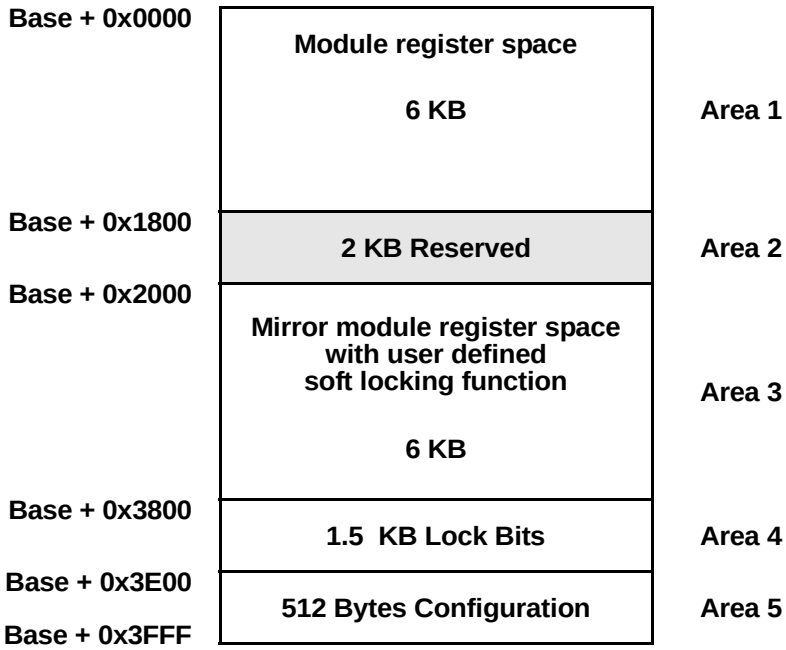


Figure 780. REG_PROT Memory Diagram

Area 1 is 6 KB and holds the normal functional module registers and is transparent for all read/write operations.

Area 2 is 2 KB starting at address 0x1800 is a reserved area, which shall not be accessed.

Area 3 is 6 KB, starting at address 0x2000 and is a mirror of area 1. A read/write access to these 0x2000+X addresses will read/write the register at address X. However, a write access to address 0x2000+X will additionally set the optional Soft Lock Bits for this address X in the same cycle as the register at address X is written. This provides an atomic write and lock operation. Not all registers in area 1 need to have protection defined by associated Soft Lock Bits. For unprotected registers at address Y, accesses to address 0x2000+Y will be

identical to accesses at address Y. Only for registers implemented in area 1 and defined as protectable Soft Lock Bits will be available in area 4.

Area 4 is 1.5 KB and holds the Soft Lock Bits, one bit per byte in area 1. The four Soft Lock Bits associated with one module register word are arranged at byte boundaries in the memory map. The Soft Lock Bit registers can be directly written using a bit mask.

Area 5 is 512 bytes and holds the configuration bits of the protection mode. There is one configuration hard lock bit per module that prevents all further modifications to the Soft Lock Bits and can only be cleared by a system reset once set. The other bits, if set, will allow user access to the protected module.

If any locked byte is accessed with a write transaction, a transfer error will be issued to the system and the write transaction will not be executed. This is true even if not all accessed bytes are locked.

Accessing unimplemented 32-bit registers in Areas 4 and 5 will result in a transfer error.

41.3.1 Memory map

Figure 618 gives an overview on the REG_PROT registers implemented.

Table 618. REG_PROT memory map

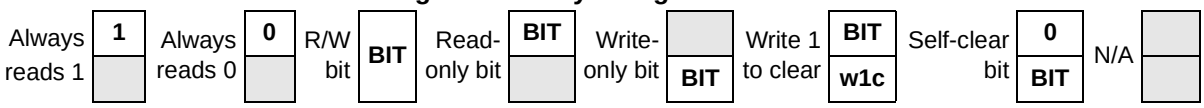
Address offset	Use	Location
0x0000	Module Register 0 (MR0)	*on page 1203*
0x0001	Module Register 1 (MR1)	*on page 1203*
0x0002	Module Register 2 (MR2)	*on page 1203*
0x0003–0x17FF	Module Register 3 (MR3) - Module Register 6143(MR6143)	*on page 1203*
0x1800–0x1FFF	Reserved	
0x2000	Module Register 0 (MR0) + Set Soft Lock Bit 0 (LMR0)	*on page 1203*
0x2001	Module Register 1 (MR1) + Set Soft Lock Bit 1 (LMR1)	*on page 1203*
0x2002–0x37FF	Module Register 2 (MR2) + Set Soft Lock Bit 2 (LMR2) - Module Register 6143 (MR6143) + Set Soft Lock Bit 6143 (LMR6143)	*on page 1203*
0x3800	Soft Lock Bit Register 0 (SLBR0): Soft Lock Bits 0-3	*on page 1203*
0x3801	Soft Lock Bit Register 1 (SLBR1): Soft Lock Bits 4-7	*on page 1203*
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 1203*
0x3E00–0x3FFB	Reserved	
0x3FFC	Global Configuration Register (GCR)	*on page 1205*

Note: Reserved registers in area #2 will be handled according to the protected IP (module under protection).

41.3.2 Register descriptions

This section describes in address order all the REG_PROT 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.

Figure 781. Key to register fields



41.3.2.1 Module registers (MR0-6143)

This is the lower 6 KB module memory space which holds all the functional registers of the module that is protected by the REG_PROT module.

41.3.2.2 Module register and set soft lock bit (LMR0-6143)

This is memory area #3 that provides mirrored access to the MR0-6143 registers with the side effect of setting Soft Lock Bits in case of a write access to a MR that is defined as protectable by the locking mechanism. Each MR is protectable by one associated bit in a SLBR*n*[SLB*m*], according to the mapping described in *Table 619*.

41.3.2.3 Soft lock bit register (SLBR0-1535)

These registers hold the soft lock bits for the protected registers in memory area #1.

Figure 782. Soft lock bit register (SLBR*n*)

Offset: 0x3800-0x3DFF

Access: Read always
Supervisor write

	0	1	2	3	4	5	6	7
R	0	0	0	0	SLB0	SLB1	SLB2	SLB3
W	WE0	WE1	WE2	WE3				
Reset	0	0	0	0	0	0	0	0

Table 619. SLBR*n* field descriptions

Field	Description
WE0 WE1 WE2 WE3	Write Enable Bits for Soft Lock Bits (SLB): WE0 enables writing to SLB0 WE1 enables writing to SLB1 WE2 enables writing to SLB2 WE3 enables writing to SLB3 1 Value is written to SLB 0 SLB is not modified
SLB0 SLB1 SLB2 SLB3	Soft Lock Bits for one MRn register: SLB0 can block accesses to MR{n *4 + 0} SLB1 can block accesses to MR{n *4 + 1} SLB2 can block accesses to MR{n *4 + 2} SLB3 can block accesses to MR{n *4 + 3} 1 Associated MRn byte is locked against write accesses 0 Associated MRn byte is unprotected and writeable

Table 620 gives some examples how SLBR*n*[SLB] and MR*n* go together.

Table 620. Soft lock bits vs. protected address

Soft lock bit	Protected address
SLBR0[SLB0]	MR0
SLBR0[SLB1]	MR1
SLBR0[SLB2]	MR2
SLBR0[SLB3]	MR3
SLBR1[SLB0]	MR4
SLBR1[SLB1]	MR5
SLBR1[SLB2]	MR6
SLBR1[SLB3]	MR7
SLBR2[SLB0]	MR8
...	...

41.3.2.4 Global configuration register (GCR)

This register is used to make global configurations related with the REG_PROT.

Figure 783. Global configuration register (GCR)

Offset: 0x3FFC

Access: Read Always
Supervisor 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	HLB	0	0	0	0	0	0	0	UAA	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

Table 621. 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 REG_PROT module.

41.4 Functional description

41.4.1 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 list of protected modules and registers is shown in *Section 41.6, SPC56ELxx registers under 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 only be read but not written in any mode (supervisor/normal). If an address is unprotected the corresponding SLBR*n*[SLB*m*] bit is always 0b0 no matter what software is writing to.

41.4.2 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.

41.4.2.1 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 784 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}].

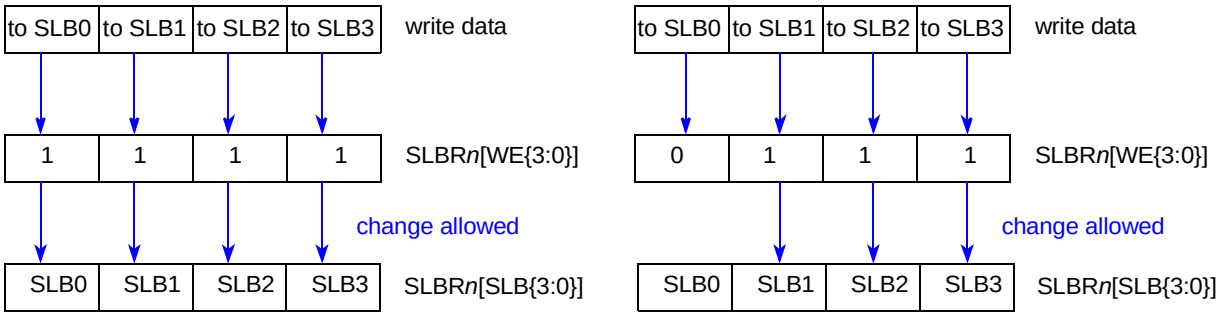


Figure 784. Change lock settings directly via area #4

Figure 784 showed four registers that can be protected with 8-bit protection. Registers with 16- and 32- bit protection are shown in *Figure 785* and *Figure 786*, respectively.

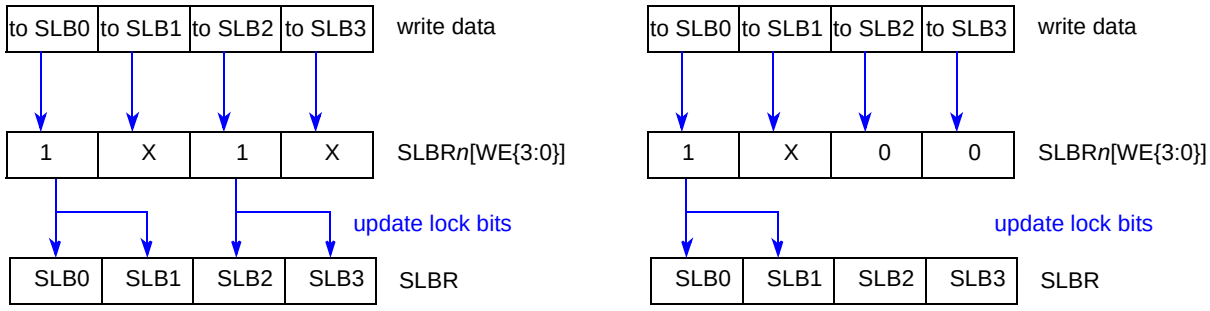


Figure 785. Change lock settings for 16-bit protected addresses

On the right side of *Figure 785* you can see that the data written to SLBR*n*[SLB{0}] is automatically written to SLBR*n*[SLB{1}] as well. This is done because the address reflected by SLBR*n*[SLB{0}] is has 16-bit protection. 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 785* 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.

Figure 786 shows a register with 32-bit protection. 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.

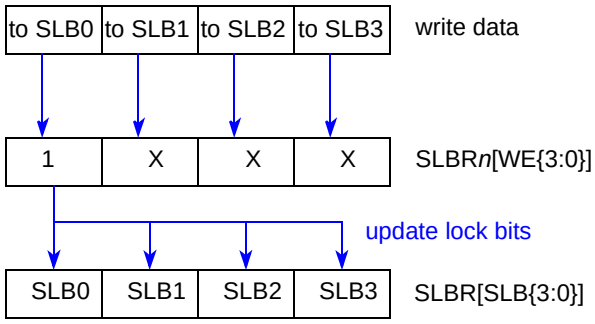


Figure 786. Change lock settings for 32-bit protected addresses

Figure 787 shows an example of mixed protection size configuration.

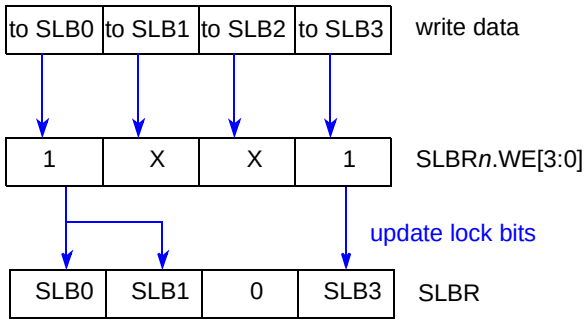


Figure 787. Change lock settings for mixed protection

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}].

41.4.2.2 Enable locking via mirror module space (area #3)

It is possible to enable locking for a register after writing to it. To do so, you must use the mirrored module address space. *Figure 788* shows one example.

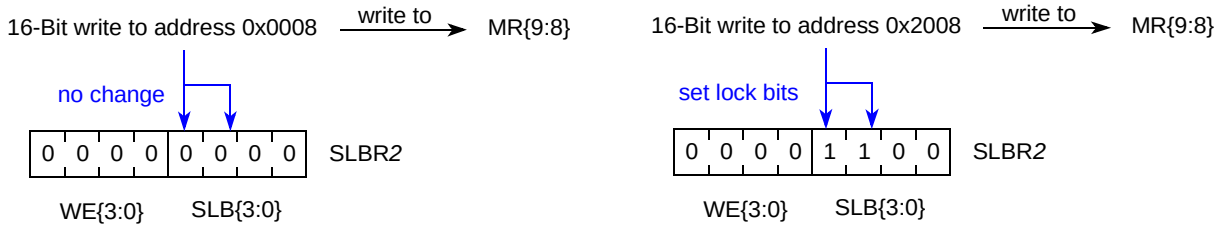


Figure 788. Enable locking via mirror module space (area #3)

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 785*).

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 785*).

Figure 789 shows an example where some addresses are protected and some are not:

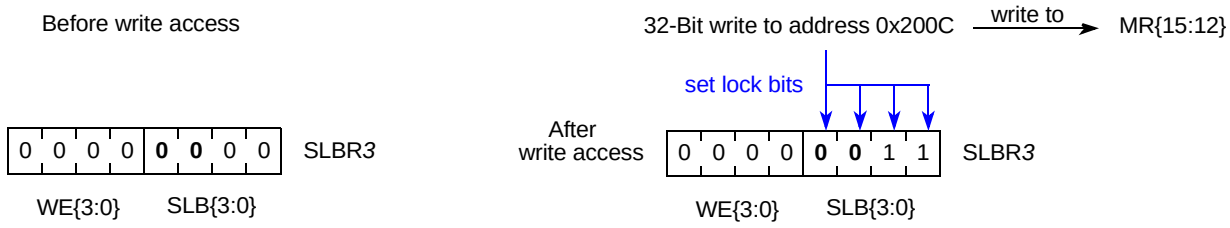


Figure 789. Enable locking for protected and unprotected addresses

In the example in *Figure 789* addresses 0x0C and 0x0D are unprotected. Therefore their corresponding lock bits SLBR3[SLB{1:0}] are always 0b0 (shown in bold). When doing a 32-bit write access to address 0x200C only lock bits SLBR3[SLB{3:2}] are set while bits SLBR3[SLB{1:0}] stay cleared.

Note: Lock bits can only be set via writes to the mirror module space. Reads from the mirror module space will not change the lock bits.

41.4.2.3 Write protection for locking bits

Changing the locking bits through any of the procedures mentioned in *Section 41.4.2.1, Change lock settings directly via area #4,* and *Section 41.4.2.2, 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 was a system reset.

41.4.3 Access errors

The protection module generates transfer errors under several circumstances. For the area definition refer to *Figure 780*

1. If accessing area #1 or area #3, the protection module will pass on any access error from the underlying protected module.
2. If user mode is not allowed, user writes to all areas will assert a transfer error and the writes will be blocked.
3. If accessing the reserved area #2, a transfer error will be asserted.
4. If accessing unimplemented 32-bit registers in area #4 and area #5 a transfer error will be asserted.
5. If writing to a register in area #1 and area #3 with Soft Lock Bit set for any of the affected bytes a transfer error is asserted and the write will be blocked. Also the complete write operation to non-protected bytes in this word is ignored.
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 Hard Lock Bit GCR.HLB is set

41.5 Initialization/application information

41.5.1 Reset

The reset state of each individual bit is shown within the register description section (See *Section 41.3.2, Register descriptions*). In summary, after reset, locking for all MR*n* registers is disabled. The registers can be accessed in supervisor mode only.

41.5.2 Writing C code using the register protection scheme

There is a set of macros provided as part of the device specific header file, which defines the memory map, the peripheral registers, and the fields of these registers. These macros are intended to make working with the register protection scheme easier for the developers of device driver software and/or other application code. This section describes these macros and how to use them.

A first macro is made available to perform a write to the mirrored module register space (Area 3). As described in this document, this results in concurrently setting the corresponding soft lock bit, while writing to the module register. There are three flavors of this macro, to account for the different size of the related module register `<thereg>` (the value of `<size>` is either 8, 16, or 32 bit):

```
WRITE_WITH_LOCK<size>(<thereg>, <newvalue>)
```

This macro writes the value `<newvalue>` into the register <thereg> assuming a register size of `<size>`. The parameter `<thereg>` must be the name of a register using the notation in the device specific header file.

A second set of macros is made available to work with the soft lock bits provided by the register protection scheme. The value of these bits associated with a particular module register `<thereg>` can be retrieved, set, and cleared. The macros provided for this purpose are:

```
GET_SOFTLOCK(<thereg>,<dest>)
SET_SOFTLOCK<size>(<thereg>)
CLR_SOFTLOCK<size>(<thereg>)
```

The macro GET_SOFTLOCK retrieves the value of the soft lock bit register associated with the given register `<thereg>` and stores it in the variable `<dest>`; which is always an eight bit

value. The other two macros (SET_SOFTLOCK, CLR_SOFTLOCK) set or clear the softlock bits associated with the given register <thereg>; assuming this register has a size of 8, 16, or 32 bit as indicated by the macro name. For all three macros, the parameter <thereg> must be the name of a register using the notation in the device specific header file.

Three more macros are made available to modify bits in the Global Configuration Register (GCR) of the protection gasket for the corresponding block. In contrast to the earlier ones, these macros receive the base address of the corresponding block (usually named <block_name>, which is set to the START Address as specified in the memory maps in *Table 3*, SPC56ELxx memory map, ordered by start address and *Table 4*, SPC56ELxx memory map, ordered by module name) as a parameter.

SET_HARDLOCK(base)

Sets the Hard Lock Bit HLB in the GCR register associated with the module identified by the given base address <base>.

USER_ACCESS_FORBIDDEN(base)

Clears the User Access Allowed Bit UAA in the GCR register associated with the module identified by the given base address <base>. When cleared, this bit denies any write access to the protected module in user mode and generates a transfer error in case of an attempt to write a protected register.

USER_ACCESS_ALLOWED(base)

Sets the User Access Allowed Bit UAA in the GCR register associated with the module identified by the given base address <base>. When set, this bit permits write accesses to the protected module in user mode when the corresponding register are not additionally protected by other means.

Finally, two more macros are provided to permit direct access to the registers in the register protection gasket using the same semantic as other register definitions in the device specific header file:

LOCK_SLB(thereg)

provides the content of the soft lock bit register associated with the register <thereg>; the parameter <thereg> must be the name of the corresponding module register using the notation in the device specific header file.

LOCK_GCR(base)

provides the content of the Global Configuration Register GCR for the block identified by its base address <base>; the parameter <base> must be the base address of the corresponding block.

41.6 SPC56ELxx registers under protection

Table 622. SPC56ELxx register protection

Module	Register	Size	Offset	Protect size
ADC_0	MCR	32	0x0	32-bit
ADC_0	IMR	32	0x20	32-bit
ADC_0	CIMR0	32	0x24	32-bit
ADC_0	WTIMR	32	0x34	32-bit
ADC_0	eDMAE	32	0x40	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
ADC_0	eDMAR0	32	0x44	32-bit
ADC_0	THRHLR0	32	0x60	32-bit
ADC_0	THRHLR1	32	0x64	32-bit
ADC_0	THRHLR2	32	0x68	32-bit
ADC_0	THRHLR3	32	0x6C	32-bit
ADC_0	PSCR	32	0x80	32-bit
ADC_0	PSR0	32	0x84	32-bit
ADC_0	CTR0	32	0x94	32-bit
ADC_0	NCMR0	32	0xA4	32-bit
ADC_0	JCMR0	32	0xB4	32-bit
ADC_0	PDEDR	32	0xC8	32-bit
ADC_0	THRHLR4	32	0x280	32-bit
ADC_0	THRHLR5	32	0x284	32-bit
ADC_0	THRHLR6	32	0x288	32-bit
ADC_0	THRHLR7	32	0x28C	32-bit
ADC_0	THRHLR8	32	0x290	32-bit
ADC_0	THRHLR9	32	0x294	32-bit
ADC_0	THRHLR10	32	0x298	32-bit
ADC_0	THRHLR11	32	0x29C	32-bit
ADC_0	THRHLR12	32	0x2A0	32-bit
ADC_0	THRHLR13	32	0x2A4	32-bit
ADC_0	THRHLR14	32	0x2A8	32-bit
ADC_0	THRHLR15	32	0x2AC	32-bit
ADC_0	CWSELR0	32	0x2B0	32-bit
ADC_0	CWSELR1	32	0x2B4	32-bit
ADC_0	CWENR0	32	0x2E0	32-bit
ADC_0	AWORR0	32	0x2F0	32-bit
ADC_0	STCR1	32	0x340	32-bit
ADC_0	STCR2	32	0x344	32-bit
ADC_0	STCR3	32	0x348	32-bit
ADC_0	STBRR	32	0x34C	32-bit
ADC_0	STAW0R	32	0x380	32-bit
ADC_0	STAW1AR	32	0x384	32-bit
ADC_0	STAW1BR	32	0x388	32-bit
ADC_0	STAW2R	32	0x38C	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
ADC_0	STAW3R	32	0x390	32-bit
ADC_0	STAW4R	32	0x394	32-bit
ADC_1	MCR	32	0x0	32-bit
ADC_1	IMR	32	0x20	32-bit
ADC_1	CIMR0	32	0x24	32-bit
ADC_1	WTIMR	32	0x34	32-bit
ADC_1	eDMAE	32	0x40	32-bit
ADC_1	eDMAR0	32	0x44	32-bit
ADC_1	THRHLR0	32	0x60	32-bit
ADC_1	THRHLR1	32	0x64	32-bit
ADC_1	THRHLR2	32	0x68	32-bit
ADC_1	THRHLR3	32	0x6C	32-bit
ADC_1	PSCR	32	0x80	32-bit
ADC_1	PSR0	32	0x84	32-bit
ADC_1	CTR0	32	0x94	32-bit
ADC_1	NCMR0	32	0xA4	32-bit
ADC_1	JCMR0	32	0xB4	32-bit
ADC_1	PDEDR	32	0xC8	32-bit
ADC_1	THRHLR4	32	0x280	32-bit
ADC_1	THRHLR5	32	0x284	32-bit
ADC_1	THRHLR6	32	0x288	32-bit
ADC_1	THRHLR7	32	0x28C	32-bit
ADC_1	THRHLR8	32	0x290	32-bit
ADC_1	THRHLR9	32	0x294	32-bit
ADC_1	THRHLR10	32	0x298	32-bit
ADC_1	THRHLR11	32	0x29C	32-bit
ADC_1	THRHLR12	32	0x2A0	32-bit
ADC_1	THRHLR13	32	0x2A4	32-bit
ADC_1	THRHLR14	32	0x2A8	32-bit
ADC_1	THRHLR15	32	0x2AC	32-bit
ADC_1	CWSELR0	32	0x2B0	32-bit
ADC_1	CWSELR1	32	0x2B4	32-bit
ADC_1	CWENR0	32	0x2E0	32-bit
ADC_1	AWORR0	32	0x2F0	32-bit
ADC_1	STCR1	32	0x340	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
ADC_1	STCR2	32	0x344	32-bit
ADC_1	STCR3	32	0x348	32-bit
ADC_1	STBRR	32	0x34C	32-bit
ADC_1	STAW0R	32	0x380	32-bit
ADC_1	STAW1AR	32	0x384	32-bit
ADC_1	STAW1BR	32	0x388	32-bit
ADC_1	STAW2R	32	0x38C	32-bit
ADC_1	STAW3R	32	0x390	32-bit
ADC_1	STAW4R	32	0x394	32-bit
CMU_0	CSR	32	0x0	32-bit
CMU_0	HFREFR_A	32	0x8	32-bit
CMU_0	LFREFR_A	32	0xC	32-bit
CMU_0	MDR	32	0x18	32-bit
CMU_1	CSR	32	0x0	32-bit
CMU_1	HFREFR_A	32	0x8	32-bit
CMU_1	LFREFR_A	32	0xC	32-bit
CMU_1	MDR	32	0x18	32-bit
CMU_2	CSR	32	0x0	32-bit
CMU_2	HFREFR_A	32	0x8	32-bit
CMU_2	LFREFR_A	32	0xC	32-bit
CMU_2	MDR	32	0x18	32-bit
CRC	CFG0	32	0x0	32-bit
CRC	CFG1	32	0x10	32-bit
CRC	CFG2	32	0x20	32-bit
CTU	TGSISR	32	0x0	32-bit
CTU	TGSCR	16	0x4	16-bit
CTU	T0CR	16	0x6	16-bit
CTU	T1CR	16	0x8	16-bit
CTU	T2CR	16	0x000A	16-bit
CTU	T3CR	16	0x000C	16-bit
CTU	T4CR	16	0x000E	16-bit
CTU	T5CR	16	0x10	16-bit
CTU	T6CR	16	0x12	16-bit
CTU	T7CR	16	0x14	16-bit
CTU	TGSCCR	16	0x16	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
CTU	TGSCRR	16	0x18	16-bit
CTU	CLCR1	32	0x001C	32-bit
CTU	CLCR2	32	0x20	32-bit
CTU	THCR1	32	0x24	32-bit
CTU	THCR2	32	0x28	32-bit
CTU	CLR1	16	0x002C	16-bit
CTU	CLR2	16	0x002E	16-bit
CTU	CLR3	16	0x30	16-bit
CTU	CLR4	16	0x32	16-bit
CTU	CLR5	16	0x34	16-bit
CTU	CLR6	16	0x36	16-bit
CTU	CLR7	16	0x38	16-bit
CTU	CLR8	16	0x003A	16-bit
CTU	CLR9	16	0x003C	16-bit
CTU	CLR10	16	0x003E	16-bit
CTU	CLR11	16	0x40	16-bit
CTU	CLR12	16	0x42	16-bit
CTU	CLR13	16	0x44	16-bit
CTU	CLR14	16	0x46	16-bit
CTU	CLR15	16	0x48	16-bit
CTU	CLR16	16	0x004A	16-bit
CTU	CLR17	16	0x004C	16-bit
CTU	CLR18	16	0x004E	16-bit
CTU	CLR19	16	0x50	16-bit
CTU	CLR20	16	0x52	16-bit
CTU	CLR21	16	0x54	16-bit
CTU	CLR22	16	0x56	16-bit
CTU	CLR23	16	0x58	16-bit
CTU	CLR24	16	0x005A	16-bit
CTU	CR	16	0x006C	16-bit
CTU	FCR	32	0x70	32-bit
CTU	TH1	32	0x74	32-bit
CTU	CTUIR	16	0x00C4	16-bit
CTU	COTR	16	0x00C6	16-bit
CTU	CTUCR	16	0x00C8	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
CTU	FILTER	16	0xCA	16-bit
CTU	EXPECTED_A	16	0xCC	16-bit
CTU	EXPECTED_B	16	0xCE	16-bit
CTU	CNT_RANGE	16	0xD0	16-bit
CodeFlash	MCR	32	0x0	32-bit
CodeFlash	LML	32	0x4	32-bit
CodeFlash	HBL	32	0x8	32-bit
CodeFlash	SLL	32	0xC	32-bit
CodeFlash	BIU0	32	0x1c	32-bit
CodeFlash	BIU1	32	0x20	32-bit
CodeFlash	BIU2	32	0x24	32-bit
CodeFlash	UT0	32	0x3C	32-bit
CodeFlash	UT1	32	0x40	32-bit
CodeFlash	UT2	32	0x44	32-bit
CodeFlash	UM0	32	0x48	32-bit
CodeFlash	UM1	32	0x4C	32-bit
CodeFlash	UM2	32	0x50	32-bit
CodeFlash	UM3	32	0x54	32-bit
CodeFlash	UM4	32	0x58	32-bit
CodeFlash	MCR	32	0x0	32-bit
CodeFlash	LML	32	0x4	32-bit
CodeFlash	HBL	32	0x8	32-bit
CodeFlash	SLL	32	0xC	32-bit
CodeFlash	BIU0	32	0x1c	32-bit
CodeFlash	BIU1	32	0x20	32-bit
CodeFlash	BIU2	32	0x24	32-bit
CodeFlash	UT0	32	0x3C	32-bit
CodeFlash	UT1	32	0x40	32-bit
CodeFlash	UT2	32	0x44	32-bit
CodeFlash	UM0	32	0x48	32-bit
CodeFlash	UM1	32	0x4C	32-bit
CodeFlash	UM2	32	0x50	32-bit
CodeFlash	UM3	32	0x54	32-bit
CodeFlash	UM4	32	0x58	32-bit
eDMA	CHCONFIG0	8	0x0	8-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
eDMA	CHCONFIG1	8	0x1	8-bit
eDMA	CHCONFIG2	8	0x2	8-bit
eDMA	CHCONFIG3	8	0x3	8-bit
eDMA	CHCONFIG4	8	0x4	8-bit
eDMA	CHCONFIG5	8	0x5	8-bit
eDMA	CHCONFIG6	8	0x6	8-bit
eDMA	CHCONFIG7	8	0x7	8-bit
eDMA	CHCONFIG8	8	0x8	8-bit
eDMA	CHCONFIG9	8	0x9	8-bit
eDMA	CHCONFIG10	8	0xA	8-bit
eDMA	CHCONFIG11	8	0xB	8-bit
eDMA	CHCONFIG12	8	0xC	8-bit
eDMA	CHCONFIG13	8	0xD	8-bit
eDMA	CHCONFIG14	8	0xE	8-bit
eDMA	CHCONFIG15	8	0xF	8-bit
DSPIA	MCR	32	0x0	32-bit
DSPIA	TCR	32	0x8	32-bit
DSPIA	CTAR0	32	0xC	32-bit
DSPIA	CTAR1	32	0x10	32-bit
DSPIA	CTAR2	32	0x14	32-bit
DSPIA	CTAR3	32	0x18	32-bit
DSPIA	RSER	32	0x30	32-bit
DSPIB	MCR	32	0x0	32-bit
DSPIB	TCR	32	0x8	32-bit
DSPIB	CTAR0	32	0xC	32-bit
DSPIB	CTAR1	32	0x10	32-bit
DSPIB	CTAR2	32	0x14	32-bit
DSPIB	CTAR3	32	0x18	32-bit
DSPIB	RSER	32	0x30	32-bit
DSPIC	MCR	32	0x0	32-bit
DSPIC	TCR	32	0x8	32-bit
DSPIC	CTAR0	32	0xC	32-bit
DSPIC	CTAR1	32	0x10	32-bit
DSPIC	CTAR2	32	0x14	32-bit
DSPIC	CTAR3	32	0x18	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
DSPIC	RSER	32	0x30	32-bit
eTIMER0	COMP10	16	0x0	16-bit
eTIMER0	COMP20	16	0x2	16-bit
eTIMER0	LOAD0	16	0x8	16-bit
eTIMER0	CTRL10	16	0xE	16-bit
eTIMER0	CTRL20	16	0x10	16-bit
eTIMER0	CTRL30	16	0x12	16-bit
eTIMER0	INTeDMA0	16	0x16	16-bit
eTIMER0	CMPLD10	16	0x18	16-bit
eTIMER0	CMPLD20	16	0x1A	16-bit
eTIMER0	CCCTRL0	16	0x1C	16-bit
eTIMER0	FILT0	16	0x1E	16-bit
eTIMER0	COMP11	16	0x20	16-bit
eTIMER0	COMP21	16	0x22	16-bit
eTIMER0	LOAD1	16	0x28	16-bit
eTIMER0	CTRL11	16	0x2E	16-bit
eTIMER0	CTRL21	16	0x30	16-bit
eTIMER0	CTRL31	16	0x32	16-bit
eTIMER0	INTeDMA1	16	0x36	16-bit
eTIMER0	CMPLD11	16	0x38	16-bit
eTIMER0	CMPLD21	16	0x3A	16-bit
eTIMER0	CCCTRL1	16	0x3C	16-bit
eTIMER0	FILT1	16	0x3E	16-bit
eTIMER0	COMP12	16	0x40	16-bit
eTIMER0	COMP22	16	0x42	16-bit
eTIMER0	LOAD2	16	0x48	16-bit
eTIMER0	CTRL12	16	0x4E	16-bit
eTIMER0	CTRL22	16	0x50	16-bit
eTIMER0	CTRL32	16	0x52	16-bit
eTIMER0	INTeDMA2	16	0x56	16-bit
eTIMER0	CMPLD12	16	0x58	16-bit
eTIMER0	CMPLD22	16	0x5A	16-bit
eTIMER0	CCCTRL2	16	0x5C	16-bit
eTIMER0	FILT2	16	0x5E	16-bit
eTIMER0	COMP13	16	0x60	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
eTIMER0	COMP23	16	0x62	16-bit
eTIMER0	LOAD3	16	0x68	16-bit
eTIMER0	CTRL13	16	0x6E	16-bit
eTIMER0	CTRL23	16	0x70	16-bit
eTIMER0	CTRL33	16	0x72	16-bit
eTIMER0	INTeDMA3	16	0x76	16-bit
eTIMER0	CMPLD13	16	0x78	16-bit
eTIMER0	CMPLD23	16	0x7A	16-bit
eTIMER0	CCCTRL3	16	0x7C	16-bit
eTIMER0	FILT3	16	0x7E	16-bit
eTIMER0	COMP14	16	0x80	16-bit
eTIMER0	COMP24	16	0x82	16-bit
eTIMER0	LOAD4	16	0x88	16-bit
eTIMER0	CTRL14	16	0x8E	16-bit
eTIMER0	CTRL24	16	0x90	16-bit
eTIMER0	CTRL34	16	0x92	16-bit
eTIMER0	INTeDMA4	16	0x96	16-bit
eTIMER0	CMPLD14	16	0x98	16-bit
eTIMER0	CMPLD24	16	0x9A	16-bit
eTIMER0	CCCTRL4	16	0x9C	16-bit
eTIMER0	FILT4	16	0x9E	16-bit
eTIMER0	COMP15	16	0xA0	16-bit
eTIMER0	COMP25	16	0xA2	16-bit
eTIMER0	LOAD5	16	0xA8	16-bit
eTIMER0	CTRL15	16	0xAE	16-bit
eTIMER0	CTRL25	16	0xB0	16-bit
eTIMER0	CTRL35	16	0xB2	16-bit
eTIMER0	INTeDMA5	16	0xB6	16-bit
eTIMER0	CMPLD15	16	0xB8	16-bit
eTIMER0	CMPLD25	16	0xBA	16-bit
eTIMER0	CCCTRL5	16	0xBC	16-bit
eTIMER0	FILT5	16	0xBE	16-bit
eTIMER0	WDTOL	16	0x100	16-bit
eTIMER0	WDTOH	16	0x102	16-bit
eTIMER0	ENBL	16	0x10C	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
eTIMER0	DREQ0	16	0x110	16-bit
eTIMER0	DREQ1	16	0x112	16-bit
eTIMER1	COMP10	16	0x0	16-bit
eTIMER1	COMP20	16	0x2	16-bit
eTIMER1	LOAD0	16	0x8	16-bit
eTIMER1	CTRL10	16	0xE	16-bit
eTIMER1	CTRL20	16	0x10	16-bit
eTIMER1	CTRL30	16	0x12	16-bit
eTIMER1	INTeDMA0	16	0x16	16-bit
eTIMER1	CMPLD10	16	0x18	16-bit
eTIMER1	CMPLD20	16	0x1A	16-bit
eTIMER1	CCCTRL0	16	0x1C	16-bit
eTIMER1	FILT0	16	0x1E	16-bit
eTIMER1	COMP11	16	0x20	16-bit
eTIMER1	COMP21	16	0x22	16-bit
eTIMER1	LOAD1	16	0x28	16-bit
eTIMER1	CTRL11	16	0x2E	16-bit
eTIMER1	CTRL21	16	0x30	16-bit
eTIMER1	CTRL31	16	0x32	16-bit
eTIMER1	INTeDMA1	16	0x36	16-bit
eTIMER1	CMPLD11	16	0x38	16-bit
eTIMER1	CMPLD21	16	0x3A	16-bit
eTIMER1	CCCTRL1	16	0x3C	16-bit
eTIMER1	FILT1	16	0x3E	16-bit
eTIMER1	COMP12	16	0x40	16-bit
eTIMER1	COMP22	16	0x42	16-bit
eTIMER1	LOAD2	16	0x48	16-bit
eTIMER1	CTRL12	16	0x4E	16-bit
eTIMER1	CTRL22	16	0x50	16-bit
eTIMER1	CTRL32	16	0x52	16-bit
eTIMER1	INTeDMA2	16	0x56	16-bit
eTIMER1	CMPLD12	16	0x58	16-bit
eTIMER1	CMPLD22	16	0x5A	16-bit
eTIMER1	CCCTRL2	16	0x5C	16-bit
eTIMER1	FILT2	16	0x5E	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
eTIMER1	COMP13	16	0x60	16-bit
eTIMER1	COMP23	16	0x62	16-bit
eTIMER1	LOAD3	16	0x68	16-bit
eTIMER1	CTRL13	16	0x6E	16-bit
eTIMER1	CTRL23	16	0x70	16-bit
eTIMER1	CTRL33	16	0x72	16-bit
eTIMER1	INTeDMA3	16	0x76	16-bit
eTIMER1	CMPLD13	16	0x78	16-bit
eTIMER1	CMPLD23	16	0x7A	16-bit
eTIMER1	CCCTRL3	16	0x7C	16-bit
eTIMER1	FILT3	16	0x7E	16-bit
eTIMER1	COMP14	16	0x80	16-bit
eTIMER1	COMP24	16	0x82	16-bit
eTIMER1	LOAD4	16	0x88	16-bit
eTIMER1	CTRL14	16	0x8E	16-bit
eTIMER1	CTRL24	16	0x90	16-bit
eTIMER1	CTRL34	16	0x92	16-bit
eTIMER1	INTeDMA4	16	0x96	16-bit
eTIMER1	CMPLD14	16	0x98	16-bit
eTIMER1	CMPLD24	16	0x9A	16-bit
eTIMER1	CCCTRL4	16	0x9C	16-bit
eTIMER1	FILT4	16	0x9E	16-bit
eTIMER1	COMP15	16	0xA0	16-bit
eTIMER1	COMP25	16	0xA2	16-bit
eTIMER1	LOAD5	16	0xA8	16-bit
eTIMER1	CTRL15	16	0xAE	16-bit
eTIMER1	CTRL25	16	0xB0	16-bit
eTIMER1	CTRL35	16	0xB2	16-bit
eTIMER1	INTeDMA5	16	0xB6	16-bit
eTIMER1	CMPLD15	16	0xB8	16-bit
eTIMER1	CMPLD25	16	0xBA	16-bit
eTIMER1	CCCTRL5	16	0xBC	16-bit
eTIMER1	FILT5	16	0xBE	16-bit
eTIMER1	ENBL	16	0x10C	16-bit
eTIMER1	DREQ0	16	0x110	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
eTIMER1	DREQ1	16	0x112	16-bit
eTIMER2	COMP10	16	0x0	16-bit
eTIMER2	COMP20	16	0x2	16-bit
eTIMER2	LOAD0	16	0x8	16-bit
eTIMER2	CTRL10	16	0xE	16-bit
eTIMER2	CTRL20	16	0x10	16-bit
eTIMER2	CTRL30	16	0x12	16-bit
eTIMER2	INTeDMA0	16	0x16	16-bit
eTIMER2	CMPLD10	16	0x18	16-bit
eTIMER2	CMPLD20	16	0x1A	16-bit
eTIMER2	CCCTRL0	16	0x1C	16-bit
eTIMER2	FILT0	16	0x1E	16-bit
eTIMER2	COMP11	16	0x20	16-bit
eTIMER2	COMP21	16	0x22	16-bit
eTIMER2	LOAD1	16	0x28	16-bit
eTIMER2	CTRL11	16	0x2E	16-bit
eTIMER2	CTRL21	16	0x30	16-bit
eTIMER2	CTRL31	16	0x32	16-bit
eTIMER2	INTeDMA1	16	0x36	16-bit
eTIMER2	CMPLD11	16	0x38	16-bit
eTIMER2	CMPLD21	16	0x3A	16-bit
eTIMER2	CCCTRL1	16	0x3C	16-bit
eTIMER2	FILT1	16	0x3E	16-bit
eTIMER2	COMP12	16	0x40	16-bit
eTIMER2	COMP22	16	0x42	16-bit
eTIMER2	LOAD2	16	0x48	16-bit
eTIMER2	CTRL12	16	0x4E	16-bit
eTIMER2	CTRL22	16	0x50	16-bit
eTIMER2	CTRL32	16	0x52	16-bit
eTIMER2	INTeDMA2	16	0x56	16-bit
eTIMER2	CMPLD12	16	0x58	16-bit
eTIMER2	CMPLD22	16	0x5A	16-bit
eTIMER2	CCCTRL2	16	0x5C	16-bit
eTIMER2	FILT2	16	0x5E	16-bit
eTIMER2	COMP13	16	0x60	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
eTIMER2	COMP23	16	0x62	16-bit
eTIMER2	LOAD3	16	0x68	16-bit
eTIMER2	CTRL13	16	0x6E	16-bit
eTIMER2	CTRL23	16	0x70	16-bit
eTIMER2	CTRL33	16	0x72	16-bit
eTIMER2	INTeDMA3	16	0x76	16-bit
eTIMER2	CMPLD13	16	0x78	16-bit
eTIMER2	CMPLD23	16	0x7A	16-bit
eTIMER2	CCCTRL3	16	0x7C	16-bit
eTIMER2	FILT3	16	0x7E	16-bit
eTIMER2	COMP14	16	0x80	16-bit
eTIMER2	COMP24	16	0x82	16-bit
eTIMER2	LOAD4	16	0x88	16-bit
eTIMER2	CTRL14	16	0x8E	16-bit
eTIMER2	CTRL24	16	0x90	16-bit
eTIMER2	CTRL34	16	0x92	16-bit
eTIMER2	INTeDMA4	16	0x96	16-bit
eTIMER2	CMPLD14	16	0x98	16-bit
eTIMER2	CMPLD24	16	0x9A	16-bit
eTIMER2	CCCTRL4	16	0x9C	16-bit
eTIMER2	FILT4	16	0x9E	16-bit
eTIMER2	COMP15	16	0xA0	16-bit
eTIMER2	COMP25	16	0xA2	16-bit
eTIMER2	LOAD5	16	0xA8	16-bit
eTIMER2	CTRL15	16	0xAE	16-bit
eTIMER2	CTRL25	16	0xB0	16-bit
eTIMER2	CTRL35	16	0xB2	16-bit
eTIMER2	INTeDMA5	16	0xB6	16-bit
eTIMER2	CMPLD15	16	0xB8	16-bit
eTIMER2	CMPLD25	16	0xBA	16-bit
eTIMER2	CCCTRL5	16	0xBC	16-bit
eTIMER2	FILT5	16	0xBE	16-bit
eTIMER2	ENBL	16	0x10C	16-bit
eTIMER2	DREQ0	16	0x110	16-bit
eTIMER2	DREQ1	16	0x112	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FCCU	CFG	32	0x8	32-bit
FCCU	CF_CFG0	32	0xC	32-bit
FCCU	NCF_CFG0	32	0x1C	32-bit
FCCU	CFS_CFG0	32	0x2C	32-bit
FCCU	CFS_CFG1	32	0x30	32-bit
FCCU	NCFS_CFG0	32	0x4C	32-bit
FCCU	NCFS_CFG1	32	0x50	32-bit
FCCU	NCFE0	32	0x94	32-bit
FCCU	NCF_TOE0	32	0xA4	32-bit
FCCU	NCF_TO	32	0xB4	32-bit
FCCU	CFG_TO	32	0xB8	32-bit
FCCU	CFF	32	0xD8	32-bit
FCCU	NCFF	32	0xDC	32-bit
FCCU	IRQ_EN	32	0xE4	32-bit
FlexCAN_A	MCR	32	0x0000	32-bit
FlexCAN_A	CTRL	32	0x0004	32-bit
FlexCAN_A	RXGMASK	32	0x0010	32-bit
FlexCAN_A	RX14MASK	32	0x0014	32-bit
FlexCAN_A	RX15MASK	32	0x0018	32-bit
FlexCAN_A	IMASK1	32	0x0028	32-bit
FlexCAN_A	MSG0_CS	32	0x80	32-bit
FlexCAN_A	MSG0_ID	32	0x84	32-bit
FlexCAN_A	MSG1_CS	32	0x90	32-bit
FlexCAN_A	MSG1_ID	32	0x94	32-bit
FlexCAN_A	MSG2_CS	32	0xA0	32-bit
FlexCAN_A	MSG2_ID	32	0xA4	32-bit
FlexCAN_A	MSG3_CS	32	0xB0	32-bit
FlexCAN_A	MSG3_ID	32	0xB4	32-bit
FlexCAN_A	MSG4_CS	32	0xC0	32-bit
FlexCAN_A	MSG4_ID	32	0xC4	32-bit
FlexCAN_A	MSG5_CS	32	0xD0	32-bit
FlexCAN_A	MSG5_ID	32	0xD4	32-bit
FlexCAN_A	MSG6_CS	32	0xE0	32-bit
FlexCAN_A	MSG6_ID	32	0xE4	32-bit
FlexCAN_A	MSG7_CS	32	0xF0	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexCAN_A	MSG7_ID	32	0xF4	32-bit
FlexCAN_A	MSG8_CS	32	0x100	32-bit
FlexCAN_A	MSG8_ID	32	0x104	32-bit
FlexCAN_A	MSG9_CS	32	0x110	32-bit
FlexCAN_A	MSG9_ID	32	0x114	32-bit
FlexCAN_A	MSG10_CS	32	0x120	32-bit
FlexCAN_A	MSG10_ID	32	0x124	32-bit
FlexCAN_A	MSG11_CS	32	0x130	32-bit
FlexCAN_A	MSG11_ID	32	0x134	32-bit
FlexCAN_A	MSG12_CS	32	0x140	32-bit
FlexCAN_A	MSG12_ID	32	0x144	32-bit
FlexCAN_A	MSG13_CS	32	0x150	32-bit
FlexCAN_A	MSG13_ID	32	0x154	32-bit
FlexCAN_A	MSG14_CS	32	0x160	32-bit
FlexCAN_A	MSG14_ID	32	0x164	32-bit
FlexCAN_A	MSG15_CS	32	0x170	32-bit
FlexCAN_A	MSG15_ID	32	0x174	32-bit
FlexCAN_A	MSG16_CS	32	0x180	32-bit
FlexCAN_A	MSG16_ID	32	0x184	32-bit
FlexCAN_A	MSG17_CS	32	0x190	32-bit
FlexCAN_A	MSG17_ID	32	0x194	32-bit
FlexCAN_A	MSG18_CS	32	0x1A0	32-bit
FlexCAN_A	MSG18_ID	32	0x1A4	32-bit
FlexCAN_A	MSG19_CS	32	0x1B0	32-bit
FlexCAN_A	MSG19_ID	32	0x1B4	32-bit
FlexCAN_A	MSG20_CS	32	0x1C0	32-bit
FlexCAN_A	MSG20_ID	32	0x1C4	32-bit
FlexCAN_A	MSG21_CS	32	0x1D0	32-bit
FlexCAN_A	MSG21_ID	32	0x1D4	32-bit
FlexCAN_A	MSG22_CS	32	0x1E0	32-bit
FlexCAN_A	MSG22_ID	32	0x1E4	32-bit
FlexCAN_A	MSG23_CS	32	0x1F0	32-bit
FlexCAN_A	MSG23_ID	32	0x1F4	32-bit
FlexCAN_A	MSG24_CS	32	0x200	32-bit
FlexCAN_A	MSG24_ID	32	0x204	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexCAN_A	MSG25_CS	32	0x210	32-bit
FlexCAN_A	MSG25_ID	32	0x214	32-bit
FlexCAN_A	MSG26_CS	32	0x220	32-bit
FlexCAN_A	MSG26_ID	32	0x224	32-bit
FlexCAN_A	MSG27_CS	32	0x230	32-bit
FlexCAN_A	MSG27_ID	32	0x234	32-bit
FlexCAN_A	MSG28_CS	32	0x240	32-bit
FlexCAN_A	MSG28_ID	32	0x244	32-bit
FlexCAN_A	MSG29_CS	32	0x250	32-bit
FlexCAN_A	MSG29_ID	32	0x254	32-bit
FlexCAN_A	MSG30_CS	32	0x260	32-bit
FlexCAN_A	MSG30_ID	32	0x264	32-bit
FlexCAN_A	MSG31_CS	32	0x270	32-bit
FlexCAN_A	MSG31_ID	32	0x274	32-bit
FlexCAN_A	RXIMR0	32	0x0880	32-bit
FlexCAN_A	RXIMR1	32	0x0884	32-bit
FlexCAN_A	RXIMR2	32	0x0888	32-bit
FlexCAN_A	RXIMR3	32	0x088C	32-bit
FlexCAN_A	RXIMR4	32	0x0890	32-bit
FlexCAN_A	RXIMR5	32	0x0894	32-bit
FlexCAN_A	RXIMR6	32	0x0898	32-bit
FlexCAN_A	RXIMR7	32	0x089C	32-bit
FlexCAN_A	RXIMR8	32	0x08A0	32-bit
FlexCAN_A	RXIMR9	32	0x08A4	32-bit
FlexCAN_A	RXIMR10	32	0x08A8	32-bit
FlexCAN_A	RXIMR11	32	0x08AC	32-bit
FlexCAN_A	RXIMR12	32	0x08B0	32-bit
FlexCAN_A	RXIMR13	32	0x08B4	32-bit
FlexCAN_A	RXIMR14	32	0x08B8	32-bit
FlexCAN_A	RXIMR15	32	0x08BC	32-bit
FlexCAN_A	RXIMR16	32	0x08C0	32-bit
FlexCAN_A	RXIMR17	32	0x08C4	32-bit
FlexCAN_A	RXIMR18	32	0x08C8	32-bit
FlexCAN_A	RXIMR19	32	0x08CC	32-bit
FlexCAN_A	RXIMR20	32	0x08D0	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexCAN_A	RXIMR21	32	0x08D4	32-bit
FlexCAN_A	RXIMR22	32	0x08D8	32-bit
FlexCAN_A	RXIMR23	32	0x08DC	32-bit
FlexCAN_A	RXIMR24	32	0x08E0	32-bit
FlexCAN_A	RXIMR25	32	0x08E4	32-bit
FlexCAN_A	RXIMR26	32	0x08E8	32-bit
FlexCAN_A	RXIMR27	32	0x08EC	32-bit
FlexCAN_A	RXIMR28	32	0x08F0	32-bit
FlexCAN_A	RXIMR29	32	0x08F4	32-bit
FlexCAN_A	RXIMR30	32	0x08F8	32-bit
FlexCAN_A	RXIMR31	32	0x08FC	32-bit
FlexCAN_B	MCR	32	0x0000	32-bit
FlexCAN_B	CTRL	32	0x0004	32-bit
FlexCAN_B	RXGMASK	32	0x0010	32-bit
FlexCAN_B	RX14MASK	32	0x0014	32-bit
FlexCAN_B	RX15MASK	32	0x0018	32-bit
FlexCAN_B	IMASK1	32	0x0028	32-bit
FlexCAN_B	MSG0_CS	32	0x80	32-bit
FlexCAN_B	MSG0_ID	32	0x84	32-bit
FlexCAN_B	MSG1_CS	32	0x90	32-bit
FlexCAN_B	MSG1_ID	32	0x94	32-bit
FlexCAN_B	MSG2_CS	32	0xA0	32-bit
FlexCAN_B	MSG2_ID	32	0xA4	32-bit
FlexCAN_B	MSG3_CS	32	0xB0	32-bit
FlexCAN_B	MSG3_ID	32	0xB4	32-bit
FlexCAN_B	MSG4_CS	32	0xC0	32-bit
FlexCAN_B	MSG4_ID	32	0xC4	32-bit
FlexCAN_B	MSG5_CS	32	0xD0	32-bit
FlexCAN_B	MSG5_ID	32	0xD4	32-bit
FlexCAN_B	MSG6_CS	32	0xE0	32-bit
FlexCAN_B	MSG6_ID	32	0xE4	32-bit
FlexCAN_B	MSG7_CS	32	0xF0	32-bit
FlexCAN_B	MSG7_ID	32	0xF4	32-bit
FlexCAN_B	MSG8_CS	32	0x100	32-bit
FlexCAN_B	MSG8_ID	32	0x104	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexCAN_B	MSG9_CS	32	0x110	32-bit
FlexCAN_B	MSG9_ID	32	0x114	32-bit
FlexCAN_B	MSG10_CS	32	0x120	32-bit
FlexCAN_B	MSG10_ID	32	0x124	32-bit
FlexCAN_B	MSG11_CS	32	0x130	32-bit
FlexCAN_B	MSG11_ID	32	0x134	32-bit
FlexCAN_B	MSG12_CS	32	0x140	32-bit
FlexCAN_B	MSG12_ID	32	0x144	32-bit
FlexCAN_B	MSG13_CS	32	0x150	32-bit
FlexCAN_B	MSG13_ID	32	0x154	32-bit
FlexCAN_B	MSG14_CS	32	0x160	32-bit
FlexCAN_B	MSG14_ID	32	0x164	32-bit
FlexCAN_B	MSG15_CS	32	0x170	32-bit
FlexCAN_B	MSG15_ID	32	0x174	32-bit
FlexCAN_B	MSG16_CS	32	0x180	32-bit
FlexCAN_B	MSG16_ID	32	0x184	32-bit
FlexCAN_B	MSG17_CS	32	0x190	32-bit
FlexCAN_B	MSG17_ID	32	0x194	32-bit
FlexCAN_B	MSG18_CS	32	0x1A0	32-bit
FlexCAN_B	MSG18_ID	32	0x1A4	32-bit
FlexCAN_B	MSG19_CS	32	0x1B0	32-bit
FlexCAN_B	MSG19_ID	32	0x1B4	32-bit
FlexCAN_B	MSG20_CS	32	0x1C0	32-bit
FlexCAN_B	MSG20_ID	32	0x1C4	32-bit
FlexCAN_B	MSG21_CS	32	0x1D0	32-bit
FlexCAN_B	MSG21_ID	32	0x1D4	32-bit
FlexCAN_B	MSG22_CS	32	0x1E0	32-bit
FlexCAN_B	MSG22_ID	32	0x1E4	32-bit
FlexCAN_B	MSG23_CS	32	0x1F0	32-bit
FlexCAN_B	MSG23_ID	32	0x1F4	32-bit
FlexCAN_B	MSG24_CS	32	0x200	32-bit
FlexCAN_B	MSG24_ID	32	0x204	32-bit
FlexCAN_B	MSG25_CS	32	0x210	32-bit
FlexCAN_B	MSG25_ID	32	0x214	32-bit
FlexCAN_B	MSG26_CS	32	0x220	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexCAN_B	MSG26_ID	32	0x224	32-bit
FlexCAN_B	MSG27_CS	32	0x230	32-bit
FlexCAN_B	MSG27_ID	32	0x234	32-bit
FlexCAN_B	MSG28_CS	32	0x240	32-bit
FlexCAN_B	MSG28_ID	32	0x244	32-bit
FlexCAN_B	MSG29_CS	32	0x250	32-bit
FlexCAN_B	MSG29_ID	32	0x254	32-bit
FlexCAN_B	MSG30_CS	32	0x260	32-bit
FlexCAN_B	MSG30_ID	32	0x264	32-bit
FlexCAN_B	MSG31_CS	32	0x270	32-bit
FlexCAN_B	MSG31_ID	32	0x274	32-bit
FlexCAN_B	RXIMR0	32	0x0880	32-bit
FlexCAN_B	RXIMR1	32	0x0884	32-bit
FlexCAN_B	RXIMR2	32	0x0888	32-bit
FlexCAN_B	RXIMR3	32	0x088C	32-bit
FlexCAN_B	RXIMR4	32	0x0890	32-bit
FlexCAN_B	RXIMR5	32	0x0894	32-bit
FlexCAN_B	RXIMR6	32	0x0898	32-bit
FlexCAN_B	RXIMR7	32	0x089C	32-bit
FlexCAN_B	RXIMR8	32	0x08A0	32-bit
FlexCAN_B	RXIMR9	32	0x08A4	32-bit
FlexCAN_B	RXIMR10	32	0x08A8	32-bit
FlexCAN_B	RXIMR11	32	0x08AC	32-bit
FlexCAN_B	RXIMR12	32	0x08B0	32-bit
FlexCAN_B	RXIMR13	32	0x08B4	32-bit
FlexCAN_B	RXIMR14	32	0x08B8	32-bit
FlexCAN_B	RXIMR15	32	0x08BC	32-bit
FlexCAN_B	RXIMR16	32	0x08C0	32-bit
FlexCAN_B	RXIMR17	32	0x08C4	32-bit
FlexCAN_B	RXIMR18	32	0x08C8	32-bit
FlexCAN_B	RXIMR19	32	0x08CC	32-bit
FlexCAN_B	RXIMR20	32	0x08D0	32-bit
FlexCAN_B	RXIMR21	32	0x08D4	32-bit
FlexCAN_B	RXIMR22	32	0x08D8	32-bit
FlexCAN_B	RXIMR23	32	0x08DC	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexCAN_B	RXIMR24	32	0x08E0	32-bit
FlexCAN_B	RXIMR25	32	0x08E4	32-bit
FlexCAN_B	RXIMR26	32	0x08E8	32-bit
FlexCAN_B	RXIMR27	32	0x08EC	32-bit
FlexCAN_B	RXIMR28	32	0x08F0	32-bit
FlexCAN_B	RXIMR29	32	0x08F4	32-bit
FlexCAN_B	RXIMR30	32	0x08F8	32-bit
FlexCAN_B	RXIMR31	32	0x08FC	32-bit
FlexPWM0	INIT0	16	0x0002	16-bit
FlexPWM0	CTRL20	16	0x0004	16-bit
FlexPWM0	CTRL10	16	0x0006	16-bit
FlexPWM0	VAL_00	16	0x0008	16-bit
FlexPWM0	VAL_10	16	0x000A	16-bit
FlexPWM0	VAL_20	16	0x000C	16-bit
FlexPWM0	VAL_30	16	0x000E	16-bit
FlexPWM0	VAL_40	16	0x0010	16-bit
FlexPWM0	VAL_50	16	0x0012	16-bit
FlexPWM0	OCTRL0	16	0x0018	16-bit
FlexPWM0	INTEN0	16	0x001C	16-bit
FlexPWM0	eDMAEN0	16	0x001E	16-bit
FlexPWM0	TCTRL0	16	0x0020	16-bit
FlexPWM0	DISMAP0	16	0x0022	16-bit
FlexPWM0	DTCNT00	16	0x0024	16-bit
FlexPWM0	DTCNT10	16	0x0026	16-bit
FlexPWM0	CAPTCTRLX0	16	0x0030	16-bit
FlexPWM0	CAPTCMPX0	16	0x0032	16-bit
FlexPWM0	INIT1	16	0x0052	16-bit
FlexPWM0	CTRL21	16	0x0054	16-bit
FlexPWM0	CTRL11	16	0x0056	16-bit
FlexPWM0	VAL_01	16	0x0058	16-bit
FlexPWM0	VAL_11	16	0x005A	16-bit
FlexPWM0	VAL_21	16	0x005C	16-bit
FlexPWM0	VAL_31	16	0x005E	16-bit
FlexPWM0	VAL_41	16	0x0060	16-bit
FlexPWM0	VAL_51	16	0x0062	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexPWM0	OCTRL1	16	0x0068	16-bit
FlexPWM0	INTEN1	16	0x006C	16-bit
FlexPWM0	eDMAEN1	16	0x006E	16-bit
FlexPWM0	TCTRL1	16	0x0070	16-bit
FlexPWM0	DISMAP1	16	0x0072	16-bit
FlexPWM0	DTCNT01	16	0x0074	16-bit
FlexPWM0	DTCNT11	16	0x0076	16-bit
FlexPWM0	CAPTCTRLX1	16	0x0080	16-bit
FlexPWM0	CAPTCMPX1	16	0x0082	16-bit
FlexPWM0	INIT2	16	0x00A2	16-bit
FlexPWM0	CTRL22	16	0x00A4	16-bit
FlexPWM0	CTRL12	16	0x00A6	16-bit
FlexPWM0	VAL_02	16	0x00A8	16-bit
FlexPWM0	VAL_12	16	0x00AA	16-bit
FlexPWM0	VAL_22	16	0x00AC	16-bit
FlexPWM0	VAL_32	16	0x00AE	16-bit
FlexPWM0	VAL_42	16	0x00B0	16-bit
FlexPWM0	VAL_52	16	0x00B2	16-bit
FlexPWM0	OCTRL2	16	0x00B8	16-bit
FlexPWM0	INTEN2	16	0x00BC	16-bit
FlexPWM0	eDMAEN2	16	0x00BE	16-bit
FlexPWM0	TCTRL2	16	0x00C0	16-bit
FlexPWM0	DISMAP2	16	0x00C2	16-bit
FlexPWM0	DTCNT02	16	0x00C4	16-bit
FlexPWM0	DTCNT12	16	0x00C6	16-bit
FlexPWM0	CAPTCTRLX2	16	0x00D0	16-bit
FlexPWM0	CAPTCMPX2	16	0x00D2	16-bit
FlexPWM0	INIT3	16	0x00F2	16-bit
FlexPWM0	CTRL23	16	0x00F4	16-bit
FlexPWM0	CTRL13	16	0x00F6	16-bit
FlexPWM0	VAL_03	16	0x00F8	16-bit
FlexPWM0	VAL_13	16	0x00FA	16-bit
FlexPWM0	VAL_23	16	0x00FC	16-bit
FlexPWM0	VAL_33	16	0x00FE	16-bit
FlexPWM0	VAL_43	16	0x0100	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexPWM0	VAL_53	16	0x0102	16-bit
FlexPWM0	OCTRL3	16	0x108	16-bit
FlexPWM0	INTEN3	16	0x010C	16-bit
FlexPWM0	eDMAEN3	16	0x010E	16-bit
FlexPWM0	TCTRL3	16	0x0110	16-bit
FlexPWM0	DISMAP3	16	0x0112	16-bit
FlexPWM0	DTCNT03	16	0x0114	16-bit
FlexPWM0	DTCNT13	16	0x0116	16-bit
FlexPWM0	CAPTCTRLX3	16	0x0120	16-bit
FlexPWM0	CAPTCMPX3	16	0x0122	16-bit
FlexPWM0	OUTEN	16	0x0140	16-bit
FlexPWM0	MASK	16	0x0142	16-bit
FlexPWM0	SWCOUT	16	0x0144	16-bit
FlexPWM0	DTSRCSEL	16	0x0146	16-bit
FlexPWM0	MCTRL	16	0x0148	16-bit
FlexPWM0	FCTRL	16	0x014C	16-bit
FlexPWM1	INIT0	16	0x0002	16-bit
FlexPWM1	CTRL20	16	0x0004	16-bit
FlexPWM1	CTRL10	16	0x0006	16-bit
FlexPWM1	VAL_00	16	0x0008	16-bit
FlexPWM1	VAL_10	16	0x000A	16-bit
FlexPWM1	VAL_20	16	0x000C	16-bit
FlexPWM1	VAL_30	16	0x000E	16-bit
FlexPWM1	VAL_40	16	0x0010	16-bit
FlexPWM1	VAL_50	16	0x0012	16-bit
FlexPWM1	OCTRL0	16	0x0018	16-bit
FlexPWM1	INTEN0	16	0x001C	16-bit
FlexPWM1	eDMAEN0	16	0x001E	16-bit
FlexPWM1	TCTRL0	16	0x0020	16-bit
FlexPWM1	DISMAP0	16	0x0022	16-bit
FlexPWM1	DTCNT00	16	0x0024	16-bit
FlexPWM1	DTCNT10	16	0x0026	16-bit
FlexPWM1	CAPTCTRLX0	16	0x0030	16-bit
FlexPWM1	CAPTCMPX0	16	0x0032	16-bit
FlexPWM1	INIT1	16	0x0052	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexPWM1	CTRL21	16	0x0054	16-bit
FlexPWM1	CTRL11	16	0x0056	16-bit
FlexPWM1	VAL_01	16	0x0058	16-bit
FlexPWM1	VAL_11	16	0x005A	16-bit
FlexPWM1	VAL_21	16	0x005C	16-bit
FlexPWM1	VAL_31	16	0x005E	16-bit
FlexPWM1	VAL_41	16	0x0060	16-bit
FlexPWM1	VAL_51	16	0x0062	16-bit
FlexPWM1	OCTRL1	16	0x0068	16-bit
FlexPWM1	INTEN1	16	0x006C	16-bit
FlexPWM1	eDMAEN1	16	0x006E	16-bit
FlexPWM1	TCTRL1	16	0x0070	16-bit
FlexPWM1	DISMAP1	16	0x0072	16-bit
FlexPWM1	DTCNT01	16	0x0074	16-bit
FlexPWM1	DTCNT11	16	0x0076	16-bit
FlexPWM1	CAPTCTRLX1	16	0x0080	16-bit
FlexPWM1	CAPTCMPX1	16	0x0082	16-bit
FlexPWM1	INIT2	16	0x00A2	16-bit
FlexPWM1	CTRL22	16	0x00A4	16-bit
FlexPWM1	CTRL12	16	0x00A6	16-bit
FlexPWM1	VAL_02	16	0x00A8	16-bit
FlexPWM1	VAL_12	16	0x00AA	16-bit
FlexPWM1	VAL_22	16	0x00AC	16-bit
FlexPWM1	VAL_32	16	0x00AE	16-bit
FlexPWM1	VAL_42	16	0x00B0	16-bit
FlexPWM1	VAL_52	16	0x00B2	16-bit
FlexPWM1	OCTRL2	16	0x00B8	16-bit
FlexPWM1	INTEN2	16	0x00BC	16-bit
FlexPWM1	eDMAEN2	16	0x00BE	16-bit
FlexPWM1	TCTRL2	16	0x00C0	16-bit
FlexPWM1	DISMAP2	16	0x00C2	16-bit
FlexPWM1	DTCNT02	16	0x00C4	16-bit
FlexPWM1	DTCNT12	16	0x00C6	16-bit
FlexPWM1	CAPTCTRLX2	16	0x00D0	16-bit
FlexPWM1	CAPTCMPX2	16	0x00D2	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexPWM1	INIT3	16	0x00F2	16-bit
FlexPWM1	CTRL23	16	0x00F4	16-bit
FlexPWM1	CTRL13	16	0x00F6	16-bit
FlexPWM1	VAL_03	16	0x00F8	16-bit
FlexPWM1	VAL_13	16	0x00FA	16-bit
FlexPWM1	VAL_23	16	0x00FC	16-bit
FlexPWM1	VAL_33	16	0x00FE	16-bit
FlexPWM1	VAL_43	16	0x0100	16-bit
FlexPWM1	VAL_53	16	0x0102	16-bit
FlexPWM1	OCTRL3	16	0x108	16-bit
FlexPWM1	INTEN3	16	0x010C	16-bit
FlexPWM1	eDMAEN3	16	0x010E	16-bit
FlexPWM1	TCTRL3	16	0x0110	16-bit
FlexPWM1	DISMAP3	16	0x0112	16-bit
FlexPWM1	DTCNT03	16	0x0114	16-bit
FlexPWM1	DTCNT13	16	0x0116	16-bit
FlexPWM1	CAPTCTRLX3	16	0x0120	16-bit
FlexPWM1	CAPTCMPX3	16	0x0122	16-bit
FlexPWM1	OUTEN	16	0x0140	16-bit
FlexPWM1	MASK	16	0x0142	16-bit
FlexPWM1	SWCOUT	16	0x0144	16-bit
FlexPWM1	DTSRCSEL	16	0x0146	16-bit
FlexPWM1	MCTRL	16	0x0148	16-bit
FlexPWM1	FCTRL	16	0x014C	16-bit
FlexRay	MCR	16	0x2	16-bit
FlexRay	SYMBADHR	16	0x4	16-bit
FlexRay	SYMBADLR	16	0x6	16-bit
FlexRay	STBSCR	16	0x8	16-bit
FlexRay	MBDSR	16	0xC	16-bit
FlexRay	MBSSUTR	16	0xE	16-bit
FlexRay	POCR	16	0x14	16-bit
FlexRay	GIFER	16	0x16	16-bit
FlexRay	PIER0	16	0x1C	16-bit
FlexRay	PIER1	16	0x1E	16-bit
FlexRay	SYMATOR	16	0x3E	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexRay	SFTOR	16	0x42	16-bit
FlexRay	SFTCCSR	16	0x44	16-bit
FlexRay	SFIDRFR	16	0x46	16-bit
FlexRay	SFIDAFVR	16	0x48	16-bit
FlexRay	SFIDAFMR	16	0x4A	16-bit
FlexRay	NMVLR	16	0x58	16-bit
FlexRay	TICCR	16	0x5A	16-bit
FlexRay	TI1CYSR	16	0x5C	16-bit
FlexRay	TI1MTOR	16	0x5E	16-bit
FlexRay	TI2CR0	16	0x60	16-bit
FlexRay	TI2CR1	16	0x62	16-bit
FlexRay	MTSACFR	16	0x80	16-bit
FlexRay	MTSBCFR	16	0x82	16-bit
FlexRay	RFRFCTR	16	0x9A	16-bit
FlexRay	PCR0	16	0xA0	16-bit
FlexRay	PCR1	16	0xA2	16-bit
FlexRay	PCR2	16	0xA4	16-bit
FlexRay	PCR3	16	0xA6	16-bit
FlexRay	PCR4	16	0xA8	16-bit
FlexRay	PCR5	16	0xAA	16-bit
FlexRay	PCR6	16	0xAC	16-bit
FlexRay	PCR7	16	0xAE	16-bit
FlexRay	PCR8	16	0xB0	16-bit
FlexRay	PCR9	16	0xB2	16-bit
FlexRay	PCR10	16	0xB4	16-bit
FlexRay	PCR11	16	0xB6	16-bit
FlexRay	PCR12	16	0xB8	16-bit
FlexRay	PCR13	16	0xBA	16-bit
FlexRay	PCR14	16	0xBC	16-bit
FlexRay	PCR15	16	0xBE	16-bit
FlexRay	PCR16	16	0xC0	16-bit
FlexRay	PCR17	16	0xC2	16-bit
FlexRay	PCR18	16	0xC4	16-bit
FlexRay	PCR19	16	0xC6	16-bit
FlexRay	PCR20	16	0xC8	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexRay	PCR21	16	0xCA	16-bit
FlexRay	PCR22	16	0xCC	16-bit
FlexRay	PCR23	16	0xCE	16-bit
FlexRay	PCR24	16	0xD0	16-bit
FlexRay	PCR25	16	0xD2	16-bit
FlexRay	PCR26	16	0xD4	16-bit
FlexRay	PCR27	16	0xD6	16-bit
FlexRay	PCR28	16	0xD8	16-bit
FlexRay	PCR29	16	0xDA	16-bit
FlexRay	PCR30	16	0xDC	16-bit
FlexRay	MBCCFR0	16	0x102	16-bit
FlexRay	MBFIDR0	16	0x104	16-bit
FlexRay	MBCCFR1	16	0x10A	16-bit
FlexRay	MBFIDR1	16	0x10C	16-bit
FlexRay	MBCCFR2	16	0x112	16-bit
FlexRay	MBFIDR2	16	0x114	16-bit
FlexRay	MBCCFR3	16	0x11A	16-bit
FlexRay	MBFIDR3	16	0x11C	16-bit
FlexRay	MBCCFR4	16	0x122	16-bit
FlexRay	MBFIDR4	16	0x124	16-bit
FlexRay	MBCCFR5	16	0x12A	16-bit
FlexRay	MBFIDR5	16	0x12C	16-bit
FlexRay	MBCCFR6	16	0x132	16-bit
FlexRay	MBFIDR6	16	0x134	16-bit
FlexRay	MBCCFR7	16	0x13A	16-bit
FlexRay	MBFIDR7	16	0x13C	16-bit
FlexRay	MBCCFR8	16	0x142	16-bit
FlexRay	MBFIDR8	16	0x144	16-bit
FlexRay	MBCCFR9	16	0x14A	16-bit
FlexRay	MBFIDR9	16	0x14C	16-bit
FlexRay	MBCCFR10	16	0x152	16-bit
FlexRay	MBFIDR10	16	0x154	16-bit
FlexRay	MBCCFR11	16	0x15A	16-bit
FlexRay	MBFIDR11	16	0x15C	16-bit
FlexRay	MBCCFR12	16	0x162	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexRay	MBFIDR12	16	0x164	16-bit
FlexRay	MBCCFR13	16	0x16A	16-bit
FlexRay	MBFIDR13	16	0x16C	16-bit
FlexRay	MBCCFR14	16	0x172	16-bit
FlexRay	MBFIDR14	16	0x174	16-bit
FlexRay	MBCCFR15	16	0x17A	16-bit
FlexRay	MBFIDR15	16	0x17C	16-bit
FlexRay	MBCCFR16	16	0x182	16-bit
FlexRay	MBFIDR16	16	0x184	16-bit
FlexRay	MBCCFR17	16	0x18A	16-bit
FlexRay	MBFIDR17	16	0x18C	16-bit
FlexRay	MBCCFR18	16	0x192	16-bit
FlexRay	MBFIDR18	16	0x194	16-bit
FlexRay	MBCCFR19	16	0x19A	16-bit
FlexRay	MBFIDR19	16	0x19C	16-bit
FlexRay	MBCCFR20	16	0x1A2	16-bit
FlexRay	MBFIDR20	16	0x1A4	16-bit
FlexRay	MBCCFR21	16	0x1AA	16-bit
FlexRay	MBFIDR21	16	0x1AC	16-bit
FlexRay	MBCCFR22	16	0x1B2	16-bit
FlexRay	MBFIDR22	16	0x1B4	16-bit
FlexRay	MBCCFR23	16	0x1BA	16-bit
FlexRay	MBFIDR23	16	0x1BC	16-bit
FlexRay	MBCCFR24	16	0x1C2	16-bit
FlexRay	MBFIDR24	16	0x1C4	16-bit
FlexRay	MBCCFR25	16	0x1CA	16-bit
FlexRay	MBFIDR25	16	0x1CC	16-bit
FlexRay	MBCCFR26	16	0x1D2	16-bit
FlexRay	MBFIDR26	16	0x1D4	16-bit
FlexRay	MBCCFR27	16	0x1DA	16-bit
FlexRay	MBFIDR27	16	0x1DC	16-bit
FlexRay	MBCCFR28	16	0x1E2	16-bit
FlexRay	MBFIDR28	16	0x1E4	16-bit
FlexRay	MBCCFR29	16	0x1EA	16-bit
FlexRay	MBFIDR29	16	0x1EC	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexRay	MBCCFR30	16	0x1F2	16-bit
FlexRay	MBFIDR30	16	0x1F4	16-bit
FlexRay	MBCCFR31	16	0x1FA	16-bit
FlexRay	MBFIDR31	16	0x1FC	16-bit
FlexRay	MBCCFR32	16	0x202	16-bit
FlexRay	MBFIDR32	16	0x204	16-bit
FlexRay	MBCCFR33	16	0x20A	16-bit
FlexRay	MBFIDR33	16	0x20C	16-bit
FlexRay	MBCCFR34	16	0x212	16-bit
FlexRay	MBFIDR34	16	0x214	16-bit
FlexRay	MBCCFR35	16	0x21A	16-bit
FlexRay	MBFIDR35	16	0x21C	16-bit
FlexRay	MBCCFR36	16	0x222	16-bit
FlexRay	MBFIDR36	16	0x224	16-bit
FlexRay	MBCCFR37	16	0x22A	16-bit
FlexRay	MBFIDR37	16	0x22C	16-bit
FlexRay	MBCCFR38	16	0x232	16-bit
FlexRay	MBFIDR38	16	0x234	16-bit
FlexRay	MBCCFR39	16	0x23A	16-bit
FlexRay	MBFIDR39	16	0x23C	16-bit
FlexRay	MBCCFR40	16	0x242	16-bit
FlexRay	MBFIDR40	16	0x244	16-bit
FlexRay	MBCCFR41	16	0x24A	16-bit
FlexRay	MBFIDR41	16	0x24C	16-bit
FlexRay	MBCCFR42	16	0x252	16-bit
FlexRay	MBFIDR42	16	0x254	16-bit
FlexRay	MBCCFR43	16	0x25A	16-bit
FlexRay	MBFIDR43	16	0x25C	16-bit
FlexRay	MBCCFR44	16	0x262	16-bit
FlexRay	MBFIDR44	16	0x264	16-bit
FlexRay	MBCCFR45	16	0x26A	16-bit
FlexRay	MBFIDR45	16	0x26C	16-bit
FlexRay	MBCCFR46	16	0x272	16-bit
FlexRay	MBFIDR46	16	0x274	16-bit
FlexRay	MBCCFR47	16	0x27A	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FlexRay	MBFIDR47	16	0x27C	16-bit
FlexRay	MBCCFR48	16	0x282	16-bit
FlexRay	MBFIDR48	16	0x284	16-bit
FlexRay	MBCCFR49	16	0x28A	16-bit
FlexRay	MBFIDR49	16	0x28C	16-bit
FlexRay	MBCCFR50	16	0x292	16-bit
FlexRay	MBFIDR50	16	0x294	16-bit
FlexRay	MBCCFR51	16	0x29A	16-bit
FlexRay	MBFIDR51	16	0x29C	16-bit
FlexRay	MBCCFR52	16	0x2A2	16-bit
FlexRay	MBFIDR52	16	0x2A4	16-bit
FlexRay	MBCCFR53	16	0x2AA	16-bit
FlexRay	MBFIDR53	16	0x2AC	16-bit
FlexRay	MBCCFR54	16	0x2B2	16-bit
FlexRay	MBFIDR54	16	0x2B4	16-bit
FlexRay	MBCCFR55	16	0x2BA	16-bit
FlexRay	MBFIDR55	16	0x2BC	16-bit
FlexRay	MBCCFR56	16	0x2C2	16-bit
FlexRay	MBFIDR56	16	0x2C4	16-bit
FlexRay	MBCCFR57	16	0x2CA	16-bit
FlexRay	MBFIDR57	16	0x2CC	16-bit
FlexRay	MBCCFR58	16	0x2D2	16-bit
FlexRay	MBFIDR58	16	0x2D4	16-bit
FlexRay	MBCCFR59	16	0x2DA	16-bit
FlexRay	MBFIDR59	16	0x2DC	16-bit
FlexRay	MBCCFR60	16	0x2E2	16-bit
FlexRay	MBFIDR60	16	0x2E4	16-bit
FlexRay	MBCCFR61	16	0x2EA	16-bit
FlexRay	MBFIDR61	16	0x2EC	16-bit
FlexRay	MBCCFR62	16	0x2F2	16-bit
FlexRay	MBFIDR62	16	0x2F4	16-bit
FlexRay	MBCCFR63	16	0x2FA	16-bit
FlexRay	MBFIDR63	16	0x2FC	16-bit
FMPLL0	CR	32	0x0	32-bit
FMPLL0	MR	32	0x4	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
FMPLL1	CR	32	0x0	32-bit
FMPLL1	MR	32	0x4	32-bit
LINFlex_0	LINCR1	32	0x0	32-bit
LINFlex_0	LINIER	32	0x4	32-bit
LINFlex_0	UARTCR	32	0x10	32-bit
LINFlex_0	LINTCSR	32	0x18	32-bit
LINFlex_0	LINOCR	32	0x1c	32-bit
LINFlex_0	LINTOCR	32	0x20	32-bit
LINFlex_0	LINFBRR	32	0x24	32-bit
LINFlex_0	LINIBRR	32	0x28	32-bit
LINFlex_0	LINCR2	32	0x30	32-bit
LINFlex_0	BIDR	32	0x34	32-bit
LINFlex_0	IFER	32	0x40	32-bit
LINFlex_0	IFMR	32	0x48	32-bit
LINFlex_0	IFCR20	32	0x4C	32-bit
LINFlex_0	IFCR21	32	0x50	32-bit
LINFlex_0	IFCR22	32	0x54	32-bit
LINFlex_0	IFCR23	32	0x58	32-bit
LINFlex_0	IFCR24	32	0x5C	32-bit
LINFlex_0	IFCR25	32	0x60	32-bit
LINFlex_0	IFCR26	32	0x64	32-bit
LINFlex_0	IFCR27	32	0x68	32-bit
LINFlex_0	IFCR28	32	0x6C	32-bit
LINFlex_0	IFCR29	32	0x70	32-bit
LINFlex_0	IFCR210	32	0x74	32-bit
LINFlex_0	IFCR211	32	0x78	32-bit
LINFlex_0	IFCR212	32	0x7C	32-bit
LINFlex_0	IFCR213	32	0x80	32-bit
LINFlex_0	IFCR214	32	0x84	32-bit
LINFlex_0	IFCR215	32	0x88	32-bit
LINFlex_0	GCR	32	0x8c	32-bit
LINFlex_0	UARTPTO	32	0x90	32-bit
LINFlex_0	eDMATXE	32	0x98	32-bit
LINFlex_0	eDMARXE	32	0x9c	32-bit
LINFlex_1	LINCR1	32	0x0	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
LINFlex_1	LINIER	32	0x4	32-bit
LINFlex_1	UARTCR	32	0x10	32-bit
LINFlex_1	LINTCSR	32	0x18	32-bit
LINFlex_1	LINOCR	32	0x1c	32-bit
LINFlex_1	LINTOCR	32	0x20	32-bit
LINFlex_1	LINFBRR	32	0x24	32-bit
LINFlex_1	LINIBRR	32	0x28	32-bit
LINFlex_1	LINCR2	32	0x30	32-bit
LINFlex_1	BIDR	32	0x34	32-bit
LINFlex_1	IFER	32	0x40	32-bit
LINFlex_1	IFMR	32	0x48	32-bit
LINFlex_1	IFCR20	32	0x4C	32-bit
LINFlex_1	IFCR21	32	0x50	32-bit
LINFlex_1	IFCR22	32	0x54	32-bit
LINFlex_1	IFCR23	32	0x58	32-bit
LINFlex_1	IFCR24	32	0x5C	32-bit
LINFlex_1	IFCR25	32	0x60	32-bit
LINFlex_1	IFCR26	32	0x64	32-bit
LINFlex_1	IFCR27	32	0x68	32-bit
LINFlex_1	IFCR28	32	0x6C	32-bit
LINFlex_1	IFCR29	32	0x70	32-bit
LINFlex_1	IFCR210	32	0x74	32-bit
LINFlex_1	IFCR211	32	0x78	32-bit
LINFlex_1	IFCR212	32	0x7C	32-bit
LINFlex_1	IFCR213	32	0x80	32-bit
LINFlex_1	IFCR214	32	0x84	32-bit
LINFlex_1	IFCR215	32	0x88	32-bit
LINFlex_1	GCR	32	0x8c	32-bit
LINFlex_1	UARTPTO	32	0x90	32-bit
LINFlex_1	eDMATXE	32	0x98	32-bit
LINFlex_1	eDMARXE	32	0x9c	32-bit
MC_CGM (cut1)	CGM_OC_EN	32	0x0	32-bit
MC_CGM (cut1)	CGM_OCDS_SC	32	0x4	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
MC_CGM (cut1)	CGM_SC_DC0	8	0xC	8-bit
MC_CGM (cut1)	CGM_AC0_SC	32	0x10	32-bit
MC_CGM (cut1)	CGM_AC0_DC0	8	0x14	8-bit
MC_CGM (cut1)	CGM_AC0_DC1	8	0x15	8-bit
MC_CGM (cut1)	CGM_AC1_SC	32	0x18	32-bit
MC_CGM (cut1)	CGM_AC1_DC0	8	0x1C	8-bit
MC_CGM (cut1)	CGM_AC2_SC	32	0x20	32-bit
MC_CGM (cut1)	CGM_AC2_DC0	8	0x24	8-bit
MC_CGM (cut1)	CGM_AC3_SC	32	0x28	32-bit
MC_CGM (cut1)	CGM_AC4_SC	32	0x30	32-bit
MC_CGM (cut2/3)	CGM_OC_EN	32	0x370	32-bit
MC_CGM (cut2/3)	CGM_OCDS_SC	32	0x374	32-bit
MC_CGM (cut2/3)	CGM_SC_DC0	8	0x37C	8-bit
MC_CGM (cut2/3)	CGM_AC0_SC	32	0x380	32-bit
MC_CGM (cut2/3)	CGM_AC0_DC0	8	0x384	8-bit
MC_CGM (cut2/3)	CGM_AC0_DC1	8	0x385	8-bit
MC_CGM (cut2/3)	CGM_AC1_SC	32	0x388	32-bit
MC_CGM (cut2/3)	CGM_AC1_DC0	8	0x38C	8-bit
MC_CGM (cut2/3)	CGM_AC2_SC	32	0x390	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
MC_CGM (cut2/3)	CGM_AC2_DC0	8	0x394	8-bit
MC_CGM (cut2/3)	CGM_AC3_SC	32	0x398	32-bit
MC_CGM (cut2/3)	CGM_AC4_SC	32	0x3A0	32-bit
MC_ME	MC_ME	32	0x8	32-bit
MC_ME	ME_IM	32	0x10	32-bit
MC_ME	ME_TEST_MC	32	0x024	32-bit
MC_ME	ME_SAFE_MC	32	0x028	32-bit
MC_ME	ME_DRUN_MC	32	0x02C	32-bit
MC_ME	ME_RUN0_MC	32	0x030	32-bit
MC_ME	ME_RUN1_MC	32	0x034	32-bit
MC_ME	ME_RUN2_MC	32	0x038	32-bit
MC_ME	ME_RUN3_MC	32	0x03C	32-bit
MC_ME	ME_HALT0_MC	32	0x040	32-bit
MC_ME	ME_STOP0_MC	32	0x048	32-bit
MC_ME	ME_RUN_PC0	32	0x080	32-bit
MC_ME	ME_RUN_PC1	32	0x084	32-bit
MC_ME	ME_RUN_PC2	32	0x088	32-bit
MC_ME	ME_RUN_PC3	32	0x08C	32-bit
MC_ME	ME_RUN_PC4	32	0x090	32-bit
MC_ME	ME_RUN_PC5	32	0x094	32-bit
MC_ME	ME_RUN_PC6	32	0x098	32-bit
MC_ME	ME_RUN_PC7	32	0x09C	32-bit
MC_ME	ME_LP_PC0	32	0x0A0	32-bit
MC_ME	ME_LP_PC1	32	0x0A4	32-bit
MC_ME	ME_LP_PC2	32	0x0A8	32-bit
MC_ME	ME_LP_PC3	32	0x0AC	32-bit
MC_ME	ME_LP_PC4	32	0x0B0	32-bit
MC_ME	ME_LP_PC5	32	0x0B4	32-bit
MC_ME	ME_LP_PC6	32	0x0B8	32-bit
MC_ME	ME_LP_PC7	32	0x0BC	32-bit
MC_ME	ME_PCTL4	8	0xC4	8-bit
MC_ME	ME_PCTL5	8	0xC5	8-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
MC_ME	ME_PCTL6	8	0xC6	8-bit
MC_ME	ME_PCTL16	8	0xD0	8-bit
MC_ME	ME_PCTL17	8	0xD1	8-bit
MC_ME	ME_PCTL24	8	0xD8	8-bit
MC_ME	ME_PCTL32	8	0xE0	8-bit
MC_ME	ME_PCTL33	8	0xE1	8-bit
MC_ME	ME_PCTL35	8	0xE3	8-bit
MC_ME	ME_PCTL38	8	0xE6	8-bit
MC_ME	ME_PCTL39	8	0xE7	8-bit
MC_ME	ME_PCTL40	8	0xE8	8-bit
MC_ME	ME_PCTL41	8	0xE9	8-bit
MC_ME	ME_PCTL42	8	0xEA	8-bit
MC_ME	ME_PCTL48	8	0xF0	8-bit
MC_ME	ME_PCTL49	8	0xF1	8-bit
MC_ME	ME_PCTL58	8	0xFA	8-bit
MC_ME	ME_PCTL62	8	0xFE	8-bit
MC_ME	ME_PCTL92	8	0x11C	8-bit
MC_RGM	RGM_FEAR	16	0x10	16-bit
MC_RGM	RGM_FESS	16	0x18	16-bit
MC_RGM	RGM_FBRE	16	0x1C	16-bit
PIT	PITMCR	32	0x0	32-bit
PIT	LDVAL0	32	0x100	32-bit
PIT	TCTRL0	32	0x108	32-bit
PIT	LDVAL1	32	0x110	32-bit
PIT	TCTRL1	32	0x118	32-bit
PIT	LDVAL2	32	0x120	32-bit
PIT	TCTRL2	32	0x128	32-bit
PIT	LDVAL3	32	0x130	32-bit
PIT	TCTRL3	32	0x138	32-bit
PMU	CTRL	32	0x44	32-bit
PMU	IRQE	32	0x7C	32-bit
	CTL	32	0x0	32-bit
SIUL	IRER	32	0x0018	32-bit
SIUL	IREER	32	0x0028	32-bit
SIUL	IFEER	32	0x002C	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
SIUL	IFER	32	0x0030	32-bit
SIUL	PCR0	16	0x0040	16-bit
SIUL	PCR1	16	0x0042	16-bit
SIUL	PCR2	16	0x0044	16-bit
SIUL	PCR3	16	0x0046	16-bit
SIUL	PCR4	16	0x0048	16-bit
SIUL	PCR5	16	0x004A	16-bit
SIUL	PCR6	16	0x004C	16-bit
SIUL	PCR7	16	0x004E	16-bit
SIUL	PCR8	16	0x50	16-bit
SIUL	PCR9	16	0x52	16-bit
SIUL	PCR10	16	0x54	16-bit
SIUL	PCR11	16	0x56	16-bit
SIUL	PCR12	16	0x58	16-bit
SIUL	PCR13	16	0x005A	16-bit
SIUL	PCR14	16	0x005C	16-bit
SIUL	PCR15	16	0x005E	16-bit
SIUL	PCR16	16	0x60	16-bit
SIUL	PCR17	16	0x62	16-bit
SIUL	PCR18	16	0x64	16-bit
SIUL	PCR19	16	0x66	16-bit
SIUL	PCR20	16	0x68	16-bit
SIUL	PCR21	16	0x006A	16-bit
SIUL	PCR22	16	0x006C	16-bit
SIUL	PCR23	16	0x006E	16-bit
SIUL	PCR24	16	0x70	16-bit
SIUL	PCR25	16	0x72	16-bit
SIUL	PCR26	16	0x74	16-bit
SIUL	PCR27	16	0x76	16-bit
SIUL	PCR28	16	0x78	16-bit
SIUL	PCR29	16	0x007A	16-bit
SIUL	PCR30	16	0x007C	16-bit
SIUL	PCR31	16	0x007E	16-bit
SIUL	PCR32	16	0x80	16-bit
SIUL	PCR33	16	0x82	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
SIUL	PCR34	16	0x84	16-bit
SIUL	PCR36	16	0x88	16-bit
SIUL	PCR37	16	0x008A	16-bit
SIUL	PCR38	16	0x008C	16-bit
SIUL	PCR39	16	0x008E	16-bit
SIUL	PCR42	16	0x94	16-bit
SIUL	PCR43	16	0x96	16-bit
SIUL	PCR44	16	0x98	16-bit
SIUL	PCR45	16	0x009A	16-bit
SIUL	PCR46	16	0x009C	16-bit
SIUL	PCR47	16	0x009E	16-bit
SIUL	PCR48	16	0x00A0	16-bit
SIUL	PCR49	16	0x00A2	16-bit
SIUL	PCR50	16	0x00A4	16-bit
SIUL	PCR51	16	0x00A6	16-bit
SIUL	PCR52	16	0x00A8	16-bit
SIUL	PCR53	16	0x00AA	16-bit
SIUL	PCR54	16	0x00AC	16-bit
SIUL	PCR55	16	0x00AE	16-bit
SIUL	PCR56	16	0x00B0	16-bit
SIUL	PCR57	16	0x00B2	16-bit
SIUL	PCR58	16	0x00B4	16-bit
SIUL	PCR59	16	0x00B6	16-bit
SIUL	PCR60	16	0x00B8	16-bit
SIUL	PCR62	16	0x00BC	16-bit
SIUL	PCR64	16	0x00C0	16-bit
SIUL	PCR66	16	0x00C4	16-bit
SIUL	PCR68	16	0x00C8	16-bit
SIUL	PCR69	16	0x00CA	16-bit
SIUL	PCR70	16	0x00CC	16-bit
SIUL	PCR71	16	0x00CE	16-bit
SIUL	PCR73	16	0x00D2	16-bit
SIUL	PCR74	16	0x00D4	16-bit
SIUL	PCR75	16	0x00D6	16-bit
SIUL	PCR76	16	0x00D8	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
SIUL	PCR77	16	0x00DA	16-bit
SIUL	PCR78	16	0x00DC	16-bit
SIUL	PCR79	16	0x00DE	16-bit
SIUL	PCR80	16	0x00E0	16-bit
SIUL	PCR83	16	0x00E6	16-bit
SIUL	PCR84	16	0x00E8	16-bit
SIUL	PCR85	16	0x00EA	16-bit
SIUL	PCR86	16	0x00EC	16-bit
SIUL	PCR87	16	0x00EE	16-bit
SIUL	PCR88	16	0x00F0	16-bit
SIUL	PCR89	16	0x00F2	16-bit
SIUL	PCR90	16	0x00F4	16-bit
SIUL	PCR91	16	0x00F6	16-bit
SIUL	PCR92	16	0x00F8	16-bit
SIUL	PCR93	16	0x00FA	16-bit
SIUL	PCR94	16	0x00FC	16-bit
SIUL	PCR95	16	0x00FE	16-bit
SIUL	PCR98	16	0x104	16-bit
SIUL	PCR99	16	0x106	16-bit
SIUL	PCR100	16	0x108	16-bit
SIUL	PCR101	16	0x010A	16-bit
SIUL	PCR102	16	0x010C	16-bit
SIUL	PCR103	16	0x010E	16-bit
SIUL	PCR104	16	0x110	16-bit
SIUL	PCR105	16	0x112	16-bit
SIUL	PCR106	16	0x114	16-bit
SIUL	PCR107	16	0x116	16-bit
SIUL	PCR108	16	0x118	16-bit
SIUL	PCR109	16	0x11A	16-bit
SIUL	PCR110	16	0x11C	16-bit
SIUL	PCR111	16	0x11E	16-bit
SIUL	PCR112	16	0x120	16-bit
SIUL	PCR113	16	0x122	16-bit
SIUL	PCR114	16	0x124	16-bit
SIUL	PCR115	16	0x126	16-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
SIUL	PCR116	16	0x128	16-bit
SIUL	PCR117	16	0x12A	16-bit
SIUL	PCR118	16	0x12C	16-bit
SIUL	PCR119	16	0x12E	16-bit
SIUL	PCR120	16	0x130	16-bit
SIUL	PCR121	16	0x132	16-bit
SIUL	PCR122	16	0x134	16-bit
SIUL	PCR123	16	0x136	16-bit
SIUL	PCR124	16	0x138	16-bit
SIUL	PCR125	16	0x13A	16-bit
SIUL	PCR126	16	0x13C	16-bit
SIUL	PCR127	16	0x13E	16-bit
SIUL	PCR128	16	0x140	16-bit
SIUL	PCR129	16	0x142	16-bit
SIUL	PCR130	16	0x144	16-bit
SIUL	PCR131	16	0x146	16-bit
SIUL	PCR132 (cut2/3 only)	16	0x148	16-bit
SIUL	PSMI0	8	0x0500	8-bit
SIUL	PSMI1	8	0x0501	8-bit
SIUL	PSMI2	8	0x0502	8-bit
SIUL	PSMI3	8	0x0503	8-bit
SIUL	PSMI7	8	0x0507	8-bit
SIUL	PSMI8	8	0x0508	8-bit
SIUL	PSMI9	8	0x0509	8-bit
SIUL	PSMI10	8	0x050A	8-bit
SIUL	PSMI11	8	0x050B	8-bit
SIUL	PSMI12	8	0x050C	8-bit
SIUL	PSMI13	8	0x050D	8-bit
SIUL	PSMI14	8	0x050E	8-bit
SIUL	PSMI15	8	0x050F	8-bit
SIUL	PSMI16	8	0x0510	8-bit
SIUL	PSMI17	8	0x0511	8-bit
SIUL	PSMI18	8	0x0512	8-bit
SIUL	PSMI19	8	0x0513	8-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
SIUL	PSMI20	8	0x0514	8-bit
SIUL	PSMI21	8	0x0515	8-bit
SIUL	PSMI22	8	0x0516	8-bit
SIUL	PSMI23	8	0x0517	8-bit
SIUL	PSMI24	8	0x0518	8-bit
SIUL	PSMI25	8	0x0519	8-bit
SIUL	PSMI26	8	0x051A	8-bit
SIUL	PSMI27	8	0x051B	8-bit
SIUL	PSMI28	8	0x051C	8-bit
SIUL	PSMI29	8	0x051D	8-bit
SIUL	PSMI30	8	0x051E	8-bit
SIUL	PSMI31	8	0x051F	8-bit
SIUL	PSMI32	8	0x0520	8-bit
SIUL	PSMI33	8	0x0521	8-bit
SIUL	PSMI34	8	0x0522	8-bit
SIUL	PSMI35	8	0x0523	8-bit
SIUL	PSMI36	8	0x0524	8-bit
SIUL	PSMI37	8	0x0525	8-bit
SIUL	PSMI38	8	0x0526	8-bit
SIUL	PSMI39	8	0x0527	8-bit
SIUL	PSMI40	8	0x0528	8-bit
SIUL	PSMI41	8	0x0529	8-bit
SIUL	PSMI42	8	0x052A	8-bit
SIUL	MPGPDO0	32	0x0C80	32-bit
SIUL	MPGPDO1	32	0x0C84	32-bit
SIUL	MPGPDO2	32	0x0C88	32-bit
SIUL	MPGPDO3	32	0x0C8C	32-bit
SIUL	MPGPDO4	32	0x0C90	32-bit
SIUL	MPGPDO5	32	0x0C94	32-bit
SIUL	MPGPDO6	32	0x0C98	32-bit
SIUL	MPGPDO7	32	0x0C9C	32-bit
SIUL	MPGPDO8	32	0x0CA0	32-bit
SIUL	IFMC0	32	0x1000	32-bit
SIUL	IFMC1	32	0x1004	32-bit
SIUL	IFMC2	32	0x1008	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
SIUL	IFMC3	32	0x100C	32-bit
SIUL	IFMC4	32	0x1010	32-bit
SIUL	IFMC5	32	0x1014	32-bit
SIUL	IFMC6	32	0x1018	32-bit
SIUL	IFMC7	32	0x101C	32-bit
SIUL	IFMC8	32	0x1020	32-bit
SIUL	IFMC9	32	0x1024	32-bit
SIUL	IFMC10	32	0x1028	32-bit
SIUL	IFMC11	32	0x102C	32-bit
SIUL	IFMC12	32	0x1030	32-bit
SIUL	IFMC13	32	0x1034	32-bit
SIUL	IFMC14	32	0x1038	32-bit
SIUL	IFMC15	32	0x103C	32-bit
SIUL	IFMC16	32	0x1040	32-bit
SIUL	IFMC17	32	0x1044	32-bit
SIUL	IFMC18	32	0x1048	32-bit
SIUL	IFMC19	32	0x104C	32-bit
SIUL	IFMC20	32	0x1050	32-bit
SIUL	IFMC21	32	0x1054	32-bit
SIUL	IFMC22	32	0x1058	32-bit
SIUL	IFMC23	32	0x105C	32-bit
SIUL	IFMC24	32	0x1060	32-bit
SIUL	IFMC25	32	0x1064	32-bit
SIUL	IFMC26	32	0x1068	32-bit
SIUL	IFMC27	32	0x106C	32-bit
SIUL	IFMC28	32	0x1070	32-bit
SIUL	IFMC29	32	0x1074	32-bit
SIUL	IFMC30	32	0x1078	32-bit
SIUL	IFMC31	32	0x107C	32-bit
SIUL	IFCPR	32	0x1080	32-bit
SSCM	ERROR	16	0x6	16-bit
SSCM	DEBUGPORT	16	0x8	16-bit
SSCM	SCTR	32	0x24	32-bit
SWG	CTRL	32	0x0	32-bit

Table 622. SPC56ELxx register protection (continued)

Module	Register	Size	Offset	Protect size
WKPU	NCR	32	0x8	32-bit
XOSC	CTL	32	0x0	32-bit

42 Reset Generation Module (MC_RGM)

42.1 Introduction

42.1.1 Overview

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

Figure 790 depicts the MC_RGM block diagram.

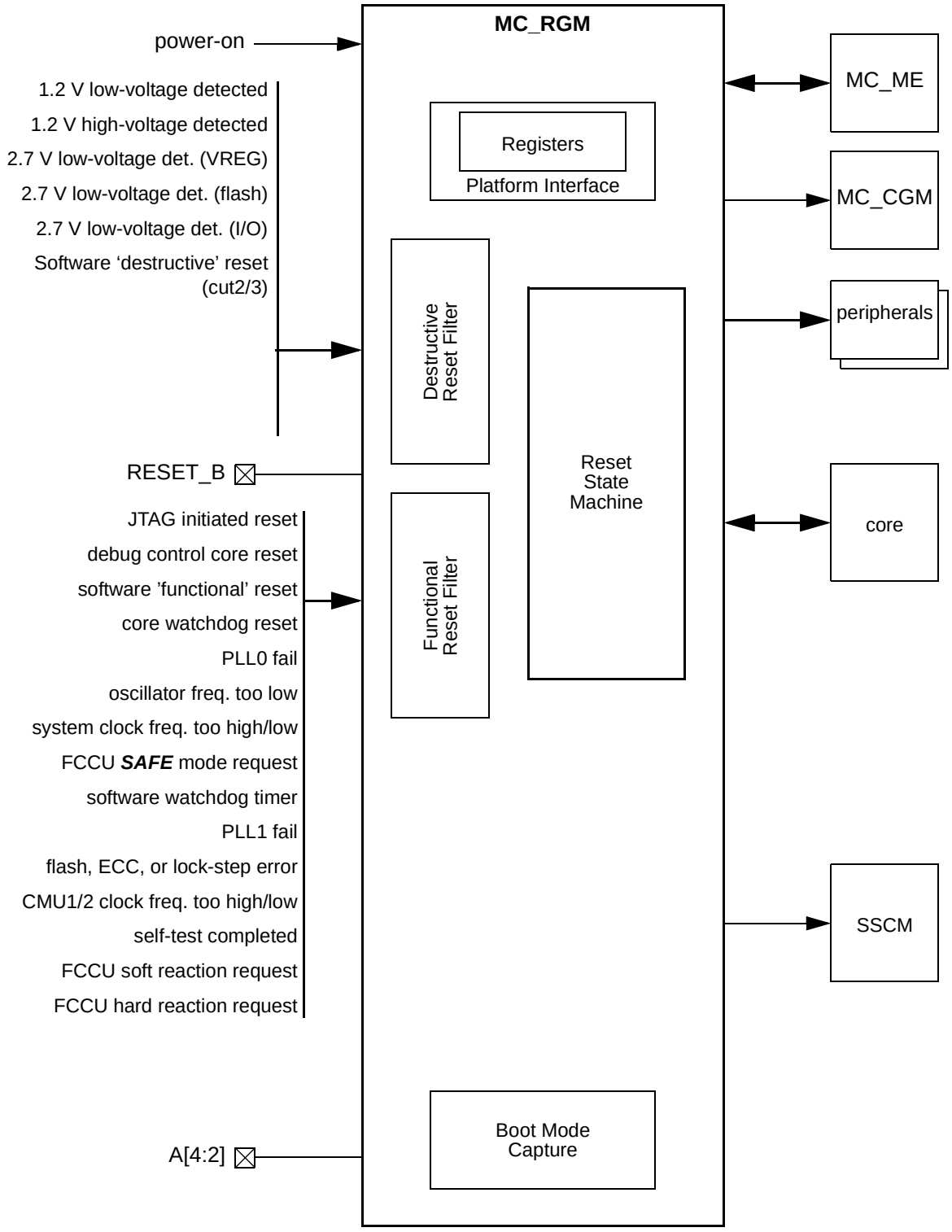


Figure 790. MC_RGM block diagram

42.1.2 Features

The MC_RGM contains the functionality for the following features:

- ‘destructive’ resets management
- ‘functional’ resets management
- signalling of reset events after each reset sequence (reset status flags)
- conversion of reset events to ***SAFE*** mode or interrupt request events
- short reset sequence configuration
- bidirectional reset behavior configuration
- boot mode capture on RESET_B deassertion

42.1.3 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 device starting from ***PHASE0***. This resets the full device ensuring a safe start-up state for both digital and analog modules. ‘Destructive’ resets are
  - power-on reset
  - 1.2 V low-voltage detected
  - 1.2 V high-voltage detected
  - 2.7 V low-voltage det. (VREG)
  - 2.7 V low-voltage det. (flash)
  - 2.7 V low-voltage det. (I/O)
  - cut2/3: 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 device starting from ***PHASE1***. In this case, most digital modules are reset normally, while analog modules or specific digital modules’ (e.g. debug modules, flash modules) state is preserved. ‘Functional’ resets are
  - external reset
  - JTAG initiated reset
  - debug control core reset
  - software ’functional’ reset
  - core watchdog reset
  - PLL0 fail
  - oscillator freq. too low
  - system clock freq. too high/low
  - FCCU ***SAFE*** mode request
  - software watchdog timer
  - PLL1 fail
  - flash, ECC, or lock-step error
  - CMU1/2 clock freq. too high/low
  - self-test completed

- FCCU soft reaction request
- FCCU hard reaction request

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

Alternatively, it is possible for software to configure some reset source events to be converted from a reset to either a ***SAFE*** mode request issued to the MC_ME or to an interrupt issued to the core (see *Section 42.3.1.3, Functional Event Reset Disable Register (RGM_FERD)* and *Section 42.3.1.5, Functional Event Alternate Request Register (RGM_FEAR)* for 'functional' resets).

42.2 External signal description

The MC_RGM interfaces to the bidirectional reset pin RESET_B and the boot mode pins PA[4:2].

42.3 Memory map and register definition

Table 623. MC_RGM register summary

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 1257*
0xC3FE_4002	RGM_DES	Destructive Event Status	half-word	read	read/write(1)	read/write1	*on page 1259*
0xC3FE_4004	RGM_FERD	Functional Event Reset Disable	half-word	read	read/write(2)	read/write(2)	*on page 1261*
0xC3FE_4006	RGM_DERD	Destructive Event Reset Disable	half-word	read	read	read	*on page 1263*
0xC3FE_4010	RGM_FEAR	Functional Event Alternate Request	half-word	read	read/write	read/write	*on page 1264*
0xC3FE_4018	RGM_FESS	Functional Event Short Sequence	half-word	read	read/write	read/write	*on page 1265*
0xC3FE_401C	RGM_FBRE	Functional Bidirectional Reset Enable	half-word	read	read/write	read/write	*on page 1267*

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 624. MC_RGM memory map

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_4000	RGM_FES /	R	F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT (cut1) F_SOFT_FUNC (cut2/3)	F_CORE	F_JTAG
		W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
0xC3FE_4002	RGM_DES (cut1)	R	F_POR	0	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	0	0	F_HVD12	F_LVD12
		W	w1c									w1c	w1c	w1c			w1c	w1c
	RGM_DES (cut2/3)	R	F_POR	F_SOFT_DEST	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	0	0	F_HVD12	F_LVD12
		W	w1c	w1c								w1c	w1c	w1c			w1c	w1c
0xC3FE_4004	RGM_FERD	R	D_EXR	D_FCCU_HARD	D_FCCU_SOFT	D_ST_DONE	D_CMU12_FHL	D_FL_ECC_RCC	D_PLL1	D_SWT	D_FCCU_SAFE	D_CMU0_FHL	D_CMU0_OLR	D_PLL0	D_CWD	D_SOFT (cut1) D_SOFT_FUNC (cut2/3)	D_CORE	D_JTAG
		W																
0xC3FE_4006	RGM_DERD (cut1)	R	0	0	0	0	0	0	0	0	0	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	0	0	D_HVD12	D_LVD12
		W																
	RGM_DERD (cut2/3)	R	D_SOFT_DEST	0	0	0	0	0	0	0	0	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	0	0	D_HVD12	D_LVD12
		W																

Table 624. MC_RGM memory map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_4008 … 0xC3FE_400C	reserved																	
0xC3FE_4010	RGM_FEAR	R	0	0	0	0	AR_CMU12_FHL	0	AR_PLL1	0	AR_FCCU_SAFE	AR_CMU0_FHL	AR_CMU0_OLR	AR_PLL0	AR_CWD	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_4014	reserved																	
0xC3FE_4018	RGM_FESS	R	SS_EXR	SS_FCCU_HARD	SS_FCCU_SOFT	SS_ST_DONE	SS_CMU12_FHL	SS_FL_ECC_RCC	SS_PLL1	SS_SWT	0	SS_CMU0_FHL	SS_CMU0_OLR	SS_PLL0	SS_CWD	SS_SOFT (cut1) SS_SOFT_FUNC (cut2/3)	SS_CORE	SS_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_401C	RGM_FBRE	R	BE_EXR	BE_FCCU_HARD	BE_FCCU_SOFT	BE_ST_DONE	BE_CMU12_FHL	BE_FL_ECC_RCC	BE_PLL1	BE_SWT	0	BE_CMU0_FHL	BE_CMU0_OLR	BE_PLL0	BE_CWD	BE_SOFT (cut1) BE_SOFT_FUNC (cut2/3)	BE_CORE	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																	

42.3.1 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_4004 (for cut1) or 0xC3FE_4003 (for cut2/3).

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

42.3.1.1 Functional Event Status Register (RGM_FES)

Figure 791. 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_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT (cut1) F_SOFT_FUNC (cut2/3)	F_CORE	F_JTAG
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

* This register is reset if and only if one of thefollowing occurs:

– Power up

– 1.2 V low-voltage detection (i.e. when the core voltage drops below the point at which the flip-flops can reliably retain their value)

This register contains the status of the last asserted functional reset sources. Register bits are cleared as follows:

- For cut1: On write ‘1’
- For cut2/3: On write ‘1’ if the triggering event has already been cleared at the source

*Note: For cut2/3: 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 625. 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_FCCU_HARD	**Flag for FCCU hard reaction request** 0 No FCCU hard reaction request event has occurred since either the last clear or the last destructive reset assertion 1 A FCCU hard reaction request event has occurred
F_FCCU_SOFT	**Flag for FCCU soft reaction request** 0 No FCCU soft reaction request event has occurred since either the last clear or the last destructive reset assertion 1 A FCCU soft reaction request event has occurred

Table 625. Functional Event Status Register (RGM_FES) Field Descriptions (continued)

Field	Description
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_CMU12_FHL	**Flag for CMU1/2 clock freq. too high/low** 0 No CMU1/2 clock freq. too high/low event has occurred since either the last clear or the last destructive reset assertion 1 A CMU1/2 clock freq. too high/low event has occurred
F_FL_ECC_RCC	**Flag for flash, ECC, or lock-step error** 0 No flash, ECC, or lock-step error event has occurred since either the last clear or the last destructive reset assertion 1 A flash, ECC, or lock-step error event has occurred
F_PLL1	**Flag for PLL1 fail** 0 No PLL1 fail event has occurred since either the last clear or the last destructive reset assertion 1 A PLL1 fail 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 destructive reset assertion 1 A software watchdog timer event has occurred
F_FCCU_SAFE	**Flag for FCCU *SAFE* mode request** 0 No FCCU ***SAFE*** mode request event has occurred since either the last clear or the last destructive reset assertion 1 A FCCU ***SAFE*** mode request event has occurred
F_CMU0_FHL	**Flag for system clock freq. too high/low** 0 No system clock freq. too high/low event has occurred since either the last clear or the last destructive reset assertion 1 A system clock freq. too high/low event has occurred
F_CMU0_OLR	**Flag for oscillator freq. too low** 0 No oscillator freq. too low event has occurred since either the last clear or the last destructive reset assertion 1 A oscillator freq. too low event has occurred
F_PLL0	**Flag for PLL0 fail** 0 No PLL0 fail event has occurred since either the last clear or the last destructive reset assertion 1 A PLL0 fail event has occurred
F_CWD	**Flag for core watchdog reset** 0 No core watchdog reset event has occurred since either the last clear or the last destructive reset assertion 1 A core watchdog reset event has occurred
F_SOFT (cut1) F_SOFT_FUNC (cut2/3)	**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

Table 625. Functional Event Status Register (RGM_FES) Field Descriptions (continued)

Field	Description
F_CORE	**Flag for debug control core reset** 0 No debug control core reset event has occurred since either the last clear or the last destructive reset assertion 1 A debug control core reset event has occurred; this event can only be asserted when the RST field of the DBCR0 register is set by an external debugger. Refer to chapter “Debug Support” 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

42.3.1.2 Destructive Event Status Register (RGM_DES)

Figure 792. Destructive Event Status Register (RGM_DES) for cut1

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_POR	0	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	0	0	F_HVD12	F_LVD12
W	w1c									w1c	w1c	w1c			w1c	w1c
Reset*	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

* This register is reset if and only if one of thefollowing occurs:

– Power up

– 1.2 V low-voltage detection (i.e. when the core voltage drops below the point at which the flip-flops can reliably retain their value)

Figure 793. Destructive Event Status Register (RGM_DES) for cut2/3

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	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	0	0	F_HVD12	F_LVD12
W	w1c	w1c								w1c	w1c	w1c			w1c	w1c
Reset*	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

* This register is reset if and only if one of thefollowing occurs:

– Power up

– 1.2 V low-voltage detection (i.e. when the core voltage drops below the point at which the flip-flops can reliably retain their value)

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

Table 626. 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 (cut2/3)	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_IO	**Flag for 2.7 V low-voltage det. (I/O)** 0 No 2.7 V low-voltage det. (I/O) event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7 V low-voltage det. (I/O) event has occurred
F_LVD27_FLASH	**Flag for 2.7 V low-voltage det. (flash)** 0 No 2.7 V low-voltage det. (flash) event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7 V low-voltage det. (flash) event has occurred
F_LVD27_VREG	**Flag for 2.7 V low-voltage det. (VREG)** 0 No 2.7 V low-voltage det. (VREG) event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7 V low-voltage det. (VREG) event has occurred

Table 626. Destructive Event Status Register (RGM_DES) Field Descriptions (continued)

Field	Description
F_HVD12	**Flag for 1.2 V high-voltage detected** 0 No 1.2 V high-voltage detected event has occurred since either the last clear or the last power-on reset assertion 1 A 1.2 V high-voltage detected event has occurred
F_LVD12	**Flag for 1.2 V low-voltage detected** 0 No 1.2 V low-voltage detected event has occurred since either the last clear or the last power-on reset assertion 1 A 1.2 V low-voltage detected 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.*

42.3.1.3 Functional Event Reset Disable Register (RGM_FERD)

Figure 794. 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_FCCU_HARD	D_FCCU_SOFT	D_ST_DONE	D_CMU12_FHL	D_FL_ECC_RCC	D_PLL1	D_SWT	D_FCCU_SAFE	D_CMU0_FHL	D_CMU0_OLR	D_PLL0	D_CWD	D_SOFT (cut1) D_SOFT_FUNC (cut2/3)	D_CORE	D_JTAG
W																
Reset*	0(1)	0(1)	0(1)	0(1)	0	0(1)	0	0(1)	1(1)	0	0	0	0	0(1)	0(1)	0(1)

* This register is reset if and only if one of thefollowing occurs:

– Power up

– 1.2 V low-voltage detection (i.e. when the core voltage drops below the point at which the flip-flops can reliably retain their value)

1. This is a read-only field so the Reset value cannot be modified.

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 42.3.1.5, Functional Event Alternate Request Register (RGM_FEAR)*). It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode. Each byte can be written only once after power-on reset. Some fields are read-only, with a POR value of 1 or 0, and the corresponding behavior cannot be changed.

Table 627. 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_FCCU_HARD	**Disable FCCU hard reaction request** 0 A FCCU hard reaction request event triggers a reset sequence
D_FCCU_SOFT	**Disable FCCU soft reaction request** 0 A FCCU soft reaction request event triggers a reset sequence
D_ST_DONE	**Disable self-test completed** 0 A self-test completed event triggers a reset sequence
D_CMU12_FHL	**Disable CMU1/2 clock freq. too high/low** 0 A CMU1/2 clock freq. too high/low event triggers a reset sequence 1 A CMU1/2 clock freq. too high/low event generates either a ***SAFE*** mode or an interrupt request depending on the value of **RGM_FEAR.AR_CMU12_FHL**
D_FL_ECC_RCC	**Disable flash, ECC, or lock-step error** 0 A flash, ECC, or lock-step error event triggers a reset sequence
D_PLL1	**Disable PLL1 fail** 0 A PLL1 fail event triggers a reset sequence 1 A PLL1 fail event generates either a ***SAFE*** mode or an interrupt request depending on the value of **RGM_FEAR.AR_PLL1**
D_SWT	**Disable software watchdog timer** 0 A software watchdog timer event triggers a reset sequence
D_FCCU_SAFE	**Disable FCCU *SAFE*** mode request 1 A FCCU ***SAFE* mode request** event generates a ***SAFE*** mode request
D_CMU0_FHL	**Disable system clock freq. too high/low** 0 A system clock freq. too high/low event triggers a reset sequence 1 A system clock freq. too high/low event generates either a ***SAFE*** mode or an interrupt request depending on the value of **RGM_FEAR.AR_CMU0_FHL**
D_CMU0_OLR	**Disable oscillator freq. too low** 0 A oscillator freq. too low event triggers a reset sequence 1 A oscillator freq. too low event generates either a ***SAFE*** mode or an interrupt request depending on the value of **RGM_FEAR.AR_CMU0_OLR**
D_PLL0	**Disable PLL0 fail** 0 A PLL0 fail event triggers a reset sequence 1 A PLL0 fail event generates either a ***SAFE*** mode or an interrupt request depending on the value of **RGM_FEAR.AR_PLL0**
D_CWD	**Disable core watchdog reset** 0 A core watchdog reset event triggers a reset sequence 1 A core watchdog reset event generates either a ***SAFE*** mode or an interrupt request depending on the value of **RGM_FEAR.AR_CWD**
D_SOFT (cut1) D_SOFT_FUNC (cut2/3)	**Disable software 'functional' reset** 0 A software 'functional' reset event triggers a reset sequence

Table 627. Functional Event Reset Disable Register (RGM_FERD) Field Descriptions (continued)

Field	Description
D_CORE	**Disable debug control core reset** 0 A debug control core reset event triggers a reset sequence
D_JTAG	**Disable JTAG initiated reset** 0 A JTAG initiated reset event triggers a reset sequence

42.3.1.4 Destructive Event Reset Disable Register (RGM_DERD)

Figure 795. Destructive Event Reset Disable Register (RGM_DERD) for cut1

Address 0xC3FE_4006 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	0	0	D_HVD12	D_LVD12
W																
Reset*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

* This register is reset if and only if one of thefollowing occurs:

– Power up

– 1.2 V low-voltage detection (i.e. when the core voltage drops below the point at which the flip-flops can reliably retain their value)

Figure 796. Destructive Event Reset Disable Register (RGM_DERD) for cut2/3

Address 0xC3FE_4006 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	D_SOFT_DEST	0	0	0	0	0	0	0	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	0	0	D_HVD12	D_LVD12
W																
Reset*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

* This register is reset if and only if one of thefollowing occurs:

– Power up

– 1.2 V low-voltage detection (i.e. when the core voltage drops below the point at which the flip-flops can reliably retain their value)

This register provides dedicated bits to disable particular destructive reset sources. It can be accessed in read-only in supervisor mode, test mode, and user mode.

Table 628. Destructive Event Reset Disable Register (RGM_DERD) Field Descriptions

Field	Description
D_SOFT_DEST (cut2/3)	Disable software 'destructive' reset 0 A software 'destructive' reset event triggers a reset sequence
D_LVD27_IO	**Disable 2.7 V low-voltage det. (I/O)** 0 A 2.7 V low-voltage det. (I/O) event triggers a reset sequence
D_LVD27_FLASH	**Disable 2.7 V low-voltage det. (flash)** 0 A 2.7 V low-voltage det. (flash) event triggers a reset sequence
D_LVD27_VREG	**Disable 2.7 V low-voltage det. (VREG)** 0 A 2.7 V low-voltage det. (VREG) event triggers a reset sequence
D_HVD12	**Disable 1.2 V high-voltage detected** 0 A 1.2 V high-voltage detected event triggers a reset sequence
D_LVD12	**Disable 1.2 V low-voltage detected** 0 A 1.2 V low-voltage detected event triggers a reset sequence

42.3.1.5 Functional Event Alternate Request Register (RGM_FEAR)

Figure 797. Functional Event Alternate Request Register (RGM_FEAR)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	AR_CMU12_FHL	0	AR_PLL1	0	AR_FCCU_SAFE	AR_CMU0_FHL	AR_CMU0_OLR	AR_PLL0	AR_CWD	0	0	0
W																
Reset*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

* This register is reset if and only if one of thefollowing occurs:

– Power up

– 1.2 V low-voltage detection (i.e. when the core voltage drops below the point at which the flip-flops can reliably retain their value)

This register defines an alternate request to be generated when a reset on a functional event has been disabled. The alternate request can be either a ***SAFE*** mode request to MC_ME or an interrupt request to the system. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode.

Table 629. Functional Event Alternate Request Register (RGM_FEAR) Field Descriptions

Field	Description
AR_CMU12_FHL	**Alternate Request for CMU1/2 clock freq. too high/low** 0 Generate a ***SAFE*** mode request on a CMU1/2 clock freq. too high/low event if the reset is disabled 1Generate an interrupt request on a CMU1/2 clock freq. too high/low event if the reset is disabled
AR_PLL1	**Alternate Request for PLL1 fail** 0 Generate a ***SAFE*** mode request on a PLL1 fail event if the reset is disabled 1 Generate an interrupt request on a PLL1 fail event if the reset is disabled
AR_FCCU_SAFE	**Alternate Request for FCCU *SAFE* mode request** 0 Generate a ***SAFE*** mode request on a FCCU ***SAFE* mode request** event if the reset is disabled
AR_CMU0_FHL	**Alternate Request for system clock freq. too high/low** 0 Generate a ***SAFE*** mode request on a system clock freq. too high/low event if the reset is disabled 1 Generate an interrupt request on a system clock freq. too high/low event if the reset is disabled
AR_CMU0_OLR	**Alternate Request for oscillator freq. too low** 0 Generate a ***SAFE*** mode request on a oscillator freq. too low event if the reset is disabled 1 Generate an interrupt request on a oscillator freq. too low event if the reset is disabled
AR_PLL0	**Alternate Request for PLL0 fail** 0 Generate a ***SAFE*** mode request on a PLL0 fail event if the reset is disabled 1 Generate an interrupt request on a PLL0 fail event if the reset is disabled
AR_CWD	**Alternate Request for core watchdog reset** 0 Generate a ***SAFE*** mode request on a core watchdog reset event if the reset is disabled 1 Generate an interrupt request on a core watchdog reset event if the reset is disabled

42.3.1.6 Functional Event Short Sequence Register (RGM_FESS)

Figure 798. 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_FCCU_HARD	SS_FCCU_SOFT	SS_ST_DONE	SS_CMU12_FHL	SS_FL_ECC_RCC	SS_PLL1	SS_SWT	0	SS_CMU0_FHL	SS_CMU0_OLR	SS_PLL0	SS_CWD	SS_SOFT	SS_CORE	SS_JTAG
W																
Reset*	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

* This register is reset if and only if one of the following occurs:

– Power up

– 1.2 V low-voltage detection (i.e. when the core voltage drops below the point at which the flip-flops can reliably retain their value)

This register defines which reset sequence will be done when a functional reset sequence is triggered. The functional reset sequence can either start from ***PHASE1*** or from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***.

Note: This could be useful for fast reset sequence, for example to skip flash reset.

It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Table 630. Functional Event Short Sequence Register (RGM_FESS) Field Descriptions

Field	Description
SS_EXR	**Short Sequence for External Reset** 0 The reset sequence triggered by an **external** reset event will start from ***PHASE1*** 1 The reset sequence triggered by an **external** reset event will start from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***
SS_FCCU_HARD	**Short Sequence for FCCU hard reaction request** 0The reset sequence triggered by a FCCU hard reaction request event will start from ***PHASE1***
SS_FCCU_SOFT	**Short Sequence for FCCU soft reaction request** 1 The reset sequence triggered by a FCCU soft reaction request event will start from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***
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_CMU12_FHL	**Short Sequence for CMU1/2 clock freq. too high/low** 0 The reset sequence triggered by a CMU1/2 clock freq. too high/low event will start from ***PHASE1*** 1 The reset sequence triggered by a CMU1/2 clock freq. too high/low event will start from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***
SS_FL_ECC_RCC	**Short Sequence for flash, ECC, or lock-step error** 0 The reset sequence triggered by a flash, ECC, or lock-step error event will start from ***PHASE1***
SS_PLL1	**Short Sequence for PLL1 fail** 0 The reset sequence triggered by a PLL1 fail event will start from ***PHASE1*** 1 The reset sequence triggered by a PLL1 fail event will start from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***
SS_SWT	**Short Sequence for software watchdog timer** 0 The reset sequence triggered by a software watchdog timer event will start from ***PHASE1***
SS_CMU0_FHL	**Short Sequence for system clock freq. too high/low** 0 The reset sequence triggered by a system clock freq. too high/low event will start from ***PHASE1*** 1 The reset sequence triggered by a system clock freq. too high/low event will start from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***
SS_CMU0_OLR	**Short Sequence for oscillator freq. too low** 0 The reset sequence triggered by a oscillator freq. too low event will start from ***PHASE1*** 1 The reset sequence triggered by a oscillator freq. too low event will start from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***

Table 630. Functional Event Short Sequence Register (RGM_FESS) Field Descriptions

Field	Description
SS_PLL0	**Short Sequence for PLL0 fail** 0 The reset sequence triggered by a PLL0 fail event will start from ***PHASE1*** 1 The reset sequence triggered by a PLL0 fail event will start from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***
SS_CWD	**Short Sequence for core watchdog reset** 0 The reset sequence triggered by a core watchdog reset event will start from ***PHASE1*** 1 The reset sequence triggered by a core watchdog reset event will start from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***
SS_SOFT	**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***
SS_CORE	**Short Sequence for debug control core reset** 0 The reset sequence triggered by a debug control core reset event will start from ***PHASE1*** 1 The reset sequence triggered by a debug control core reset event will start from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***
SS_JTAG	**Short Sequence for JTAG initiated reset** 0 The reset sequence triggered by a JTAG initiated reset event will start from ***PHASE1*** 1 The reset sequence triggered by a JTAG initiated reset event will start from ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***

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

42.3.1.7 Functional Bidirectional Reset Enable Register (RGM_FBRE)

Figure 799. Functional Bidirectional Reset Enable Register (RGM_FBRE)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BE_EXR	BE_FCCU_HARD	BE_FCCU_SOFT	BE_ST_DONE	BE_CMU12_FHL	BE_FL_ECC_RCC	BE_PLL1	BE_SWT	0	BE_CMU0_FHL	BE_CMU0_OLR	BE_PLL0	BE_CWD	BE_SOFT	BE_CORE	BE_JTAG
W																
Reset*	0	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1

* This register is reset if and only if one of thefollowing occurs:

– Power up

– 1.2 V low-voltage detection (i.e. when the core voltage drops below the point at which the flip-flops can reliably retain their value)

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

Table 631. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions

Field	Description
BE_EXR	**Bidirectional Reset Enable for External Reset** 0RESET_B is asserted on an **external** reset event if the reset is enabled 1RESET_B is not asserted on an **external** reset event
BE_FCCU_HARD	**Bidirectional Reset Enable for FCCU hard reaction request** 0RESET_B is asserted on aFCCU hard reaction request event if the reset is enabled 1RESET_B is not asserted on a FCCU hard reaction request event
BE_FCCU_SOFT	**Bidirectional Reset Enable for FCCU soft reaction request** 0RESET_B is asserted on a FCCU soft reaction request event if the reset is enabled 1RESET_B is not asserted on a FCCU soft reaction request event
BE_ST_DONE	**Bidirectional Reset Enable for self-test completed** 0RESET_B is asserted on a self-test completed event if the reset is enabled 1RESET_B is not asserted on a self-test completed event
BE_CMU12_FHL	**Bidirectional Reset Enable for CMU1/2 clock freq. too high/low** 0RESET_B is asserted on a CMU1/2 clock freq. too high/low event if the reset is enabled 1RESET_B is not asserted on a CMU1/2 clock freq. too high/low event
BE_FL_ECC_RCC	**Bidirectional Reset Enable for flash, ECC, or lock-step error** 0RESET_B is asserted on a flash, ECC, or lock-step error event if the reset is enabled 1RESET_B is not asserted on a flash, ECC, or lock-step error event
BE_PLL1	**Bidirectional Reset Enable for PLL1 fail** 0RESET_B is asserted on a PLL1 fail event if the reset is enabled 1RESET_B is not asserted on a PLL1 fail event
BE_SWT	**Bidirectional Reset Enable for software watchdog timer** 0RESET_B is asserted on a software watchdog timer event if the reset is enabled 1RESET_B is not asserted on a software watchdog timer event
BE_CMU0_FHL	**Bidirectional Reset Enable for system clock freq. too high/low** 0RESET_B is asserted on a system clock freq. too high/low event if the reset is enabled 1RESET_B is not asserted on a system clock freq. too high/low event
BE_CMU0_OLR	**Bidirectional Reset Enable for oscillator freq. too low** 0RESET_B is asserted on a oscillator freq. too low event if the reset is enabled 1RESET_B is not asserted on a oscillator freq. too low event
BE_PLL0	**Bidirectional Reset Enable for PLL0 fail** 0RESET_B is asserted on a PLL0 fail event if the reset is enabled 1RESET_B is not asserted on a PLL0 fail event
BE_CWD	**Bidirectional Reset Enable for core watchdog reset** 0RESET_B is asserted on a core watchdog reset event if the reset is enabled 1RESET_B is not asserted on a core watchdog reset event
BE_SOFT	**Bidirectional Reset Enable for software 'functional' reset** 0RESET_B is asserted on a software 'functional' reset event if the reset is enabled 1RESET_B is not asserted on a software 'functional' reset event

Table 631. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions

Field	Description
BE_CORE	**Bidirectional Reset Enable for debug control core reset** 0RESET_B is asserted on a debug control core reset event if the reset is enabled 1RESET_B is not asserted on a debug control core reset event
BE_JTAG	**Bidirectional Reset Enable for JTAG initiated reset** 0RESET_B is asserted on a JTAG initiated reset event if the reset is enabled 1RESET_B is not asserted on a JTAG initiated reset event

42.4 Functional description

42.4.1 Reset State Machine

The main role of MC_RGM is the generation of the reset sequence which ensures that the correct parts of the device are reset based on the reset source event. This is summarized in *Table 632*.

Table 632. MC_RGM Reset Implications

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

1. 'external reset assertion' means that the RESET_B 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 800*.

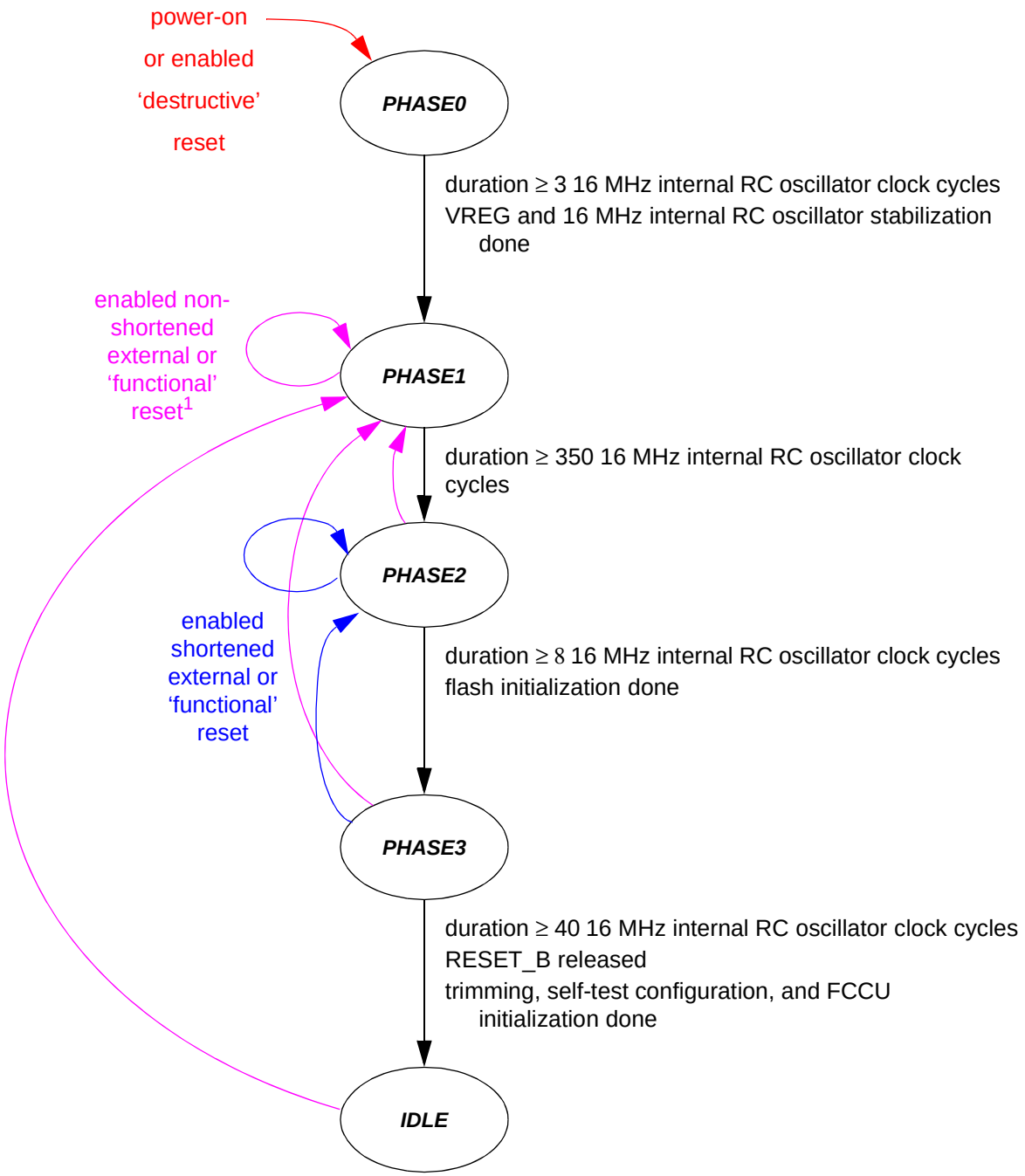


Figure 800. MC_RGM State Machine

The System Status and Configuration Module (SSCM) reads only the reset vector from the flash during reset PHASE3, if PHASE3 was entered from PHASE2.

Therefore, short external and short 'functional' resets will not trigger the SSCM to read the reset vector.

42.4.1.1 *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
  - VREG and 16 MHz internal RC oscillator stabilization
- a minimum of 3 16 MHz internal RC oscillator clock cycles have elapsed since power-up completion and the last enabled ‘destructive’ reset event

42.4.1.2 *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 350 16 MHz internal RC oscillator clock cycles have elapsed since the last enabled external or non-shortened ‘functional’ reset event

42.4.1.3 *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
  - flash initialization
- a minimum of 8 16 MHz internal RC oscillator clock cycles have elapsed since entering ***PHASE2***

42.4.1.4 *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
  - trimming, self-test configuration, and FCCU initialization
- a minimum of 40 16 MHz internal RC oscillator clock cycles have elapsed since the last enabled, shortened ‘functional’ reset event

42.4.1.5 *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 system to the platform and waits for new reset events that can trigger a reset sequence.

The actual MBIST and LBIST execution is performed during this phase. (See *Chapter 43: Self-Test Control Unit (STCU),* and *Chapter 49: System Status and Configuration Module (SSCM).*)

42.4.2 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 device’s low-voltage detector threshold ensures that, when 1.2 V low-voltage detected is enabled, the supply is sufficient to have the destructive event correctly propagated through the digital logic. Therefore, if a given ‘destructive’ reset is enabled, the MC_RGM ensures that the associated reset event will be correctly triggered to the full system. However, if the given ‘destructive’ reset is disabled and the voltage goes below the digital functional threshold, functionality can no longer be ensured, and the reset may or may not be asserted.

An enabled destructive reset will trigger a reset sequence starting from the beginning of ***PHASE0***.

42.4.3 External Reset

The MC_RGM manages the external reset coming from RESET_B. The detection of a falling edge on RESET_B 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***.

42.4.4 Functional resets

A ‘functional’ reset indicates that an event has occurred after which it can be guaranteed that critical register and memory content is still intact.

The status flag associated with a given 'functional' reset event (**RGM_FES.F_*<functional reset>*** bit) is set when the 'functional' reset is asserted and the power-on reset is not asserted. It is possible for multiple status bits to be set simultaneously, and it is software's responsibility to determine which reset source is the most critical for the application.

The 'functional' reset can be optionally disabled by software writing bit **RGM_FERD.D_*<functional reset>***.

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

An enabled functional reset will normally trigger a reset sequence starting from the beginning of ***PHASE1***. Nevertheless, the **RGM_FESS** register enables the further configuring of the reset sequence triggered by a functional reset. When **RGM_FESS.SS_*<functional reset>*** is set, the associated 'functional' reset will trigger a reset sequence starting directly from the beginning of ***PHASE3***, skipping ***PHASE1*** and ***PHASE2***. This can be useful especially in case a functional reset should not reset the flash module.

See *Chapter 33: Mode Entry Module (MC_ME)*, for details on the ***DRUN*** mode.

42.4.5 Alternate event generation

The MC_RGM provides alternative events to be generated on reset source assertion. When a reset source is asserted, the MC_RGM normally enters the reset sequence. Alternatively, it is possible for some reset source events to be converted from a reset to either a ***SAFE*** mode request issued to the MC_ME or to an interrupt request issued to the core.

Alternate event selection for a given reset source is made via the **RGM_FERD** and **RGM_FEAR** registers as shown in *Table 633*.

Table 633. MC_RGM Alternate Event Selection

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

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

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

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

42.4.6 Application Information

It is recommended that, whenever the chip comes out of reset, software checks the reset status via the RGM_DES and RGM_FES registers in order to determine the cause of the just ended reset sequence. Once the cause has been determined, software should immediately clear these registers to ensure that the old information does not make the cause of any later occurring reset unclear.

Following is a number of common reset causes and the corresponding reset status register contents. The power-on and 'destructive' reset cases are summarized in *Table 634*, and the external and 'functional' reset cases are summarized in *Table 635*. Please note that many combinations of flags can conceivably occur and that only some representative cases are shown here.

Table 634. Power-On and 'Destructive' Reset Exit Conditions Summary

RGM_DES																Description	Details
F_POR	F_SOFT_DEST	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	0	0	F_HVD12	F_LVD12		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Monotonic Power-Up	*on page 1276*
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Non-Monotonic Power-Up (dip on 1.2V supply)	*on page 1277*
																Post-Power-Up 1.2V LVD	*on page 1280*
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Non-Monotonic Power-Up (overshoot on 1.2V supply)	*on page 1278*
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	Non-Monotonic Power-Up (dip on 2.7V supply)	*on page 1279*
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0		
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Post-Power-Up 1.2V HVD	*on page 1281*
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	Post-Power-Up 2.7V LVD	*on page 1282*
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0		
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0		
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Software Triggered 'Destructive' Reset	*on page 1283*

Table 635. External and 'Functional' Reset Exit Conditions Summary

RGM_FES																Description	Details
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	External Reset	*on page 1284*
1	0	0	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	1	0	0	Software Triggered 'Destructive' Reset	*on page 1285*
0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0		
0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0		
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	Core Watchdog Timeout	*on page 1286*
0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0		
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0		
0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0		
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	PLL0 Fail	*on page 1287*
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0		
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0		
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	Oscillator Frequency Too Low	*on page 1289*
0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0		
0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0		
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	System Clock Frequency Too High/Low	*on page 1290*
0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0		
0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0		
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	Software Watchdog Timeout	*on page 1291*
0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0		
0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0		
0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0		
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0		
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	PLL1 Fail	*on page 1292*
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0		
0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0		
0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	Flash, ECC, or Lock-Step Error	*on page 1293*
0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0		
0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0		
0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0		

Table 635. External and 'Functional' Reset Exit Conditions Summary (continued)

RGM_FES																Description	Details
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG		
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Motor Control or FlexRay PE Clock Frequency Too High/Low	*on page 1295*
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0		
0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0		
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0		
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	Self-Test Completed	*on page 1296*
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0		

42.4.6.1 Power-On and 'Destructive' Resets

On chip power up, depending on the Power-up monotonicity, various RGM_DES flags may be set in addition to the RGM_DES[F_POR] flag.

In post-power-up 'destructive' reset cases, the RGM_DES[F_POR] flag is not set with the notable exception of the 1.2V LVD case. Therefore, since the flags set in the non-monotonic power-up with a dip on the 1.2V supply and a post-power-up 1.2V LVD are indistinguishable, software must handle both cases as a power-on reset.

Note that if self testing is enabled as per the shadow flash content (see the STCU chapter for details on how to disable the self test), in addition to the RGM_DES flag(s), the RGM_FES[F_ST_DONE] flag is set.

42.4.6.1.1 Monotonic Power-Up

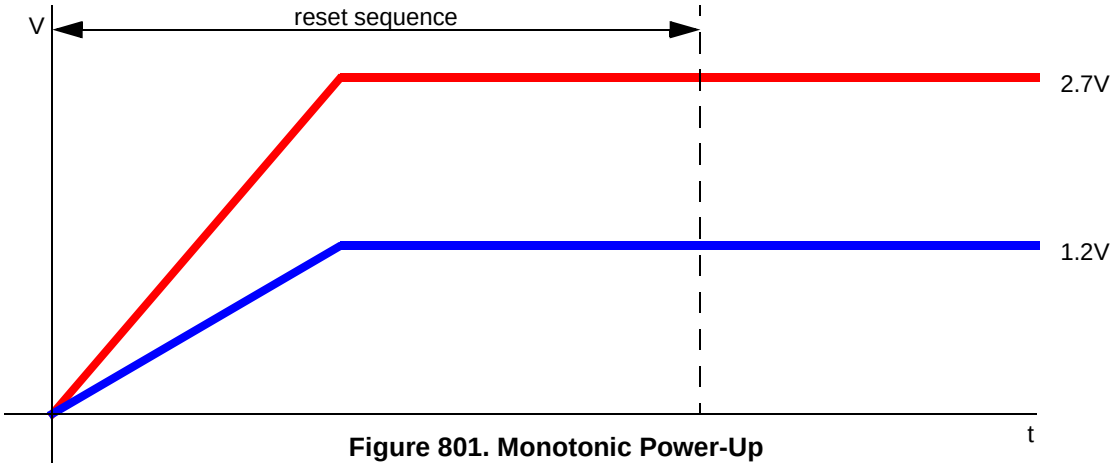


Figure 801. Monotonic Power-Up

Table 636. Reset Status After Monotonic Power-Up (without self test)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 637. Reset Status After Monotonic Power-Up (with self test)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

42.4.6.1.2 Non-Monotonic Power-Up (dip on 1.2V supply)

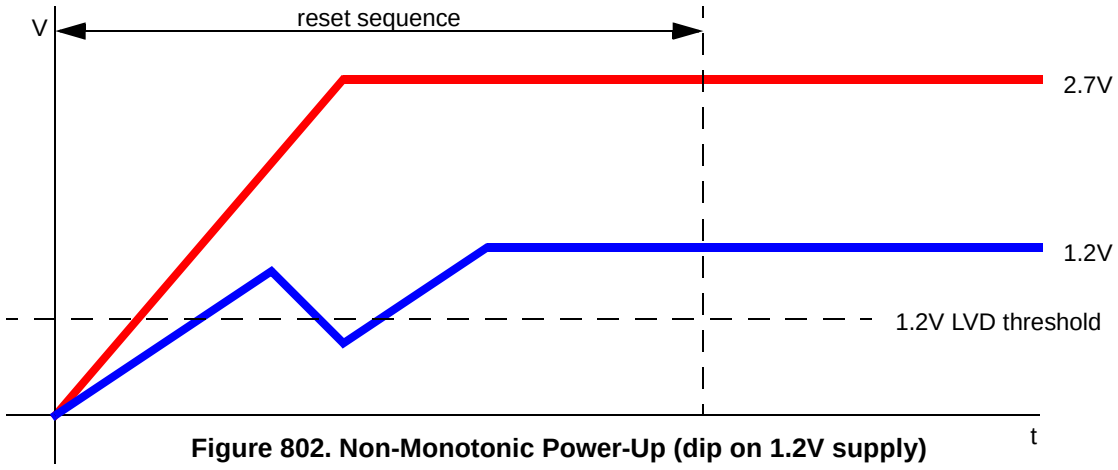


Figure 802. Non-Monotonic Power-Up (dip on 1.2V supply)

Table 638. Reset Status After Non-Monotonic Power-Up (dip on 1.2V supply, without self test)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 639. Reset Status After Non-Monotonic Power-Up (dip on 1.2V supply, with self test)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Caution: The **RGM_DES[F_POR]** bit is also set in post-power-up 1.2V LVD case, so it is not possible for software to differentiate between a post-power-up 1.2V LVD and a non-monotonic power-up with a dip on the 1.2V supply (see *Section 42.4.6.1.5, Post-Power-Up 1.2V LVD* for comparison).

42.4.6.1.3 Non-Monotonic Power-Up (overshoot on 1.2V supply)

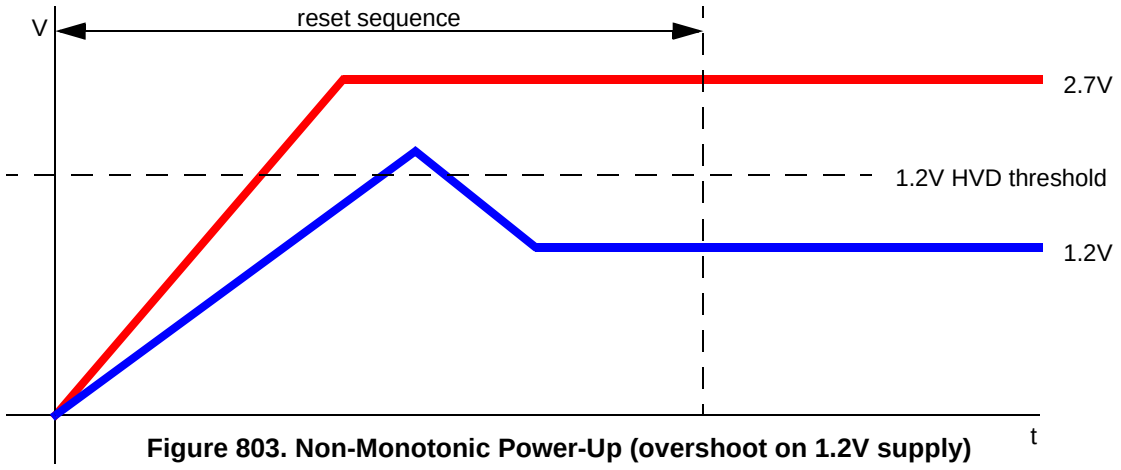


Figure 803. Non-Monotonic Power-Up (overshoot on 1.2V supply)

Table 640. Reset Status After Non-Monotonic Power-Up (overshoot on 1.2V supply, without self test)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Table 641. Reset Status After Non-Monotonic Power-Up (overshoot on 1.2V supply, with self test)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

42.4.6.1.4 Non-Monotonic Power-Up (dip on 2.7V supply)

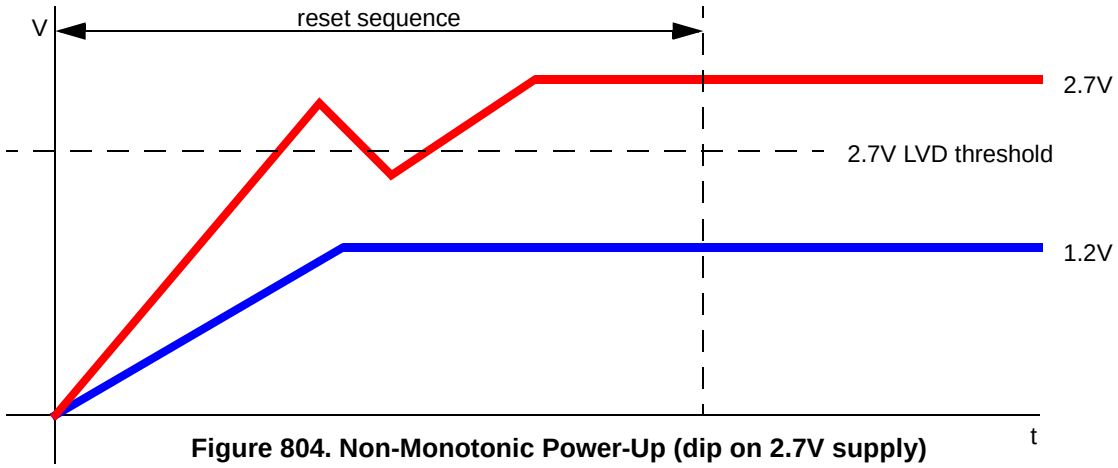


Figure 804. Non-Monotonic Power-Up (dip on 2.7V supply)

Table 642. Reset Status After Non-Monotonic Power-Up (dip on 2.7V supply, without self test)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	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	0	0	0	0

Table 643. Reset Status After Non-Monotonic Power-Up (dip on 2.7V supply, with self test)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Similarly, if the low-voltage condition occurred on the flash or I/O power rails, the RGM_DES[F_LVD27_FLASH] or RGM_DES[F_LVD27_IO] bit would be set, respectively.

42.4.6.1.5 Post-Power-Up 1.2V LVD

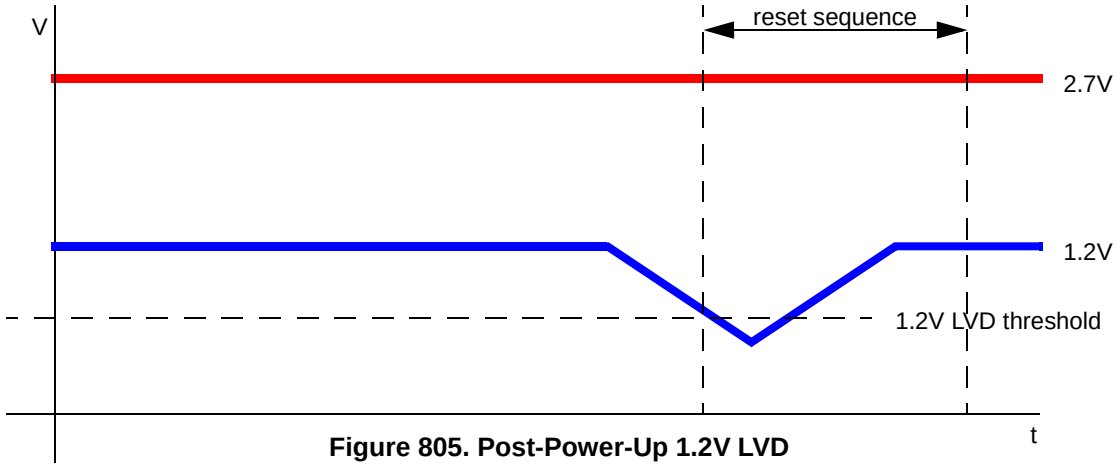


Figure 805. Post-Power-Up 1.2V LVD

Table 644. Post-Power-Up 1.2V LVD (without self test)

RGM_FES																RGM_DES														
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 645. Post-Power-Up 1.2V LVD (with self test)

RGM_FES																RGM_DES														
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Caution: The RGM_DES[F_POR] bit is set in this case, so it is not possible for software to differentiate between a post-power-up 1.2V LVD and a non-monotonic power-up with a dip on the 1.2V supply (see *Section 42.4.6.1.2, Non-Monotonic Power-Up (dip on 1.2V supply)* for comparison).

42.4.6.1.6 Post-Power-Up 1.2V HVD

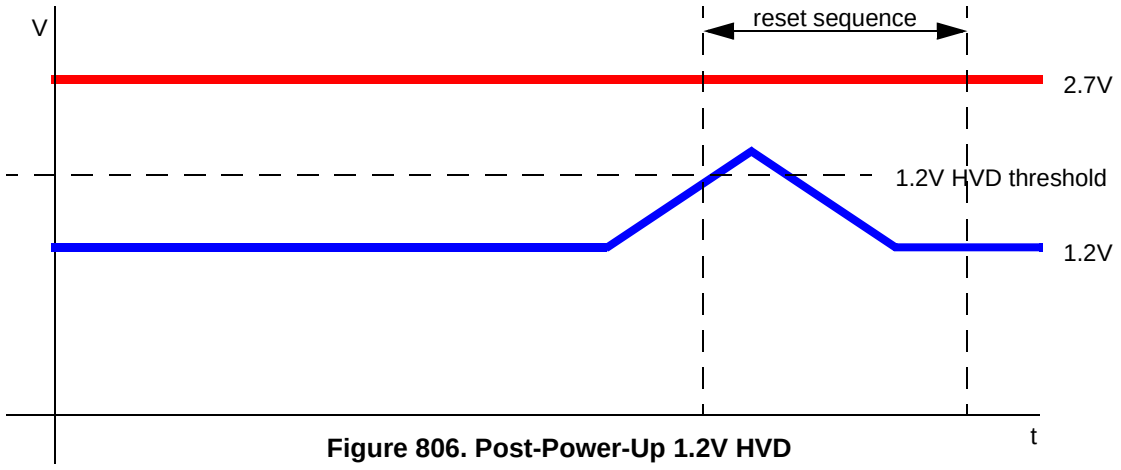


Figure 806. Post-Power-Up 1.2V HVD

Table 646. Post-Power-Up 1.2V HVD (without self test)

RGM_FES																RGM_DES														
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	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	0

Table 647. Post-Power-Up 1.2V HVD (with self test)

RGM_FES																RGM_DES														
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	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	1	0

42.4.6.1.7 Post-Power-Up 2.7V LVD

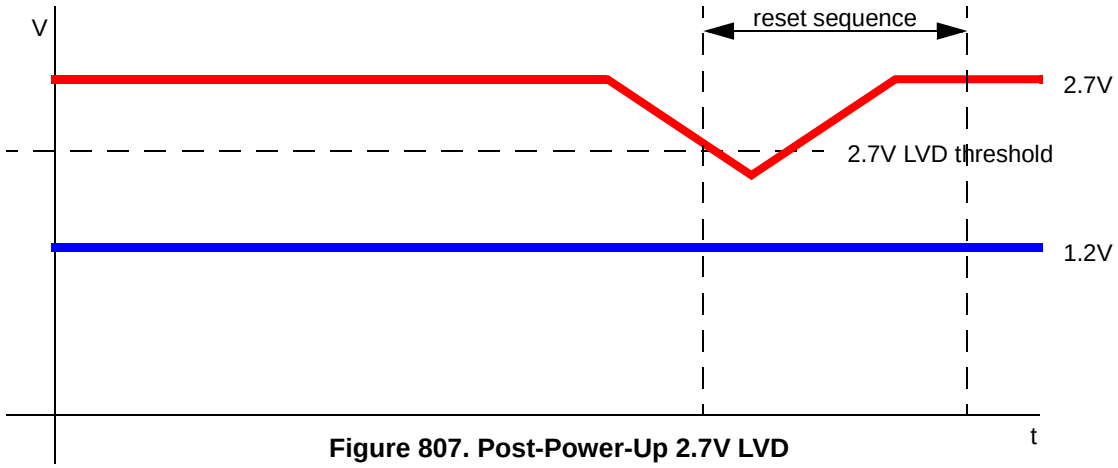


Figure 807. Post-Power-Up 2.7V LVD

Table 648. Post-Power-Up 2.7V LVD (without self test)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	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	0	0	0	0

Table 649. Post-Power-Up 2.7V LVD (with self test)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Similarly, if the low-voltage condition occurred on the flash or I/O power rails, the RGM_DES[F_LVD27_FLASH] or RGM_DES[F_LVD27_IO] bit would be set, respectively.

42.4.6.1.8 Software Triggered ‘Destructive’ Reset

This reset is triggered by software requesting a mode change to the ***RESET*** mode with the ME_MCTL[TARGET_MODE] = “1111” (see the MC_ME chapter for details on how to make a mode change request). Software may choose to trigger such a reset when, for example, software determines that a fault has occurred that cannot be cleared via a ‘functional’ reset and/or a self test needs to be executed.

Table 650. Software Triggered ‘Destructive’ Reset (without self test)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 651. Software Triggered 'Destructive' Reset (with self test)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

42.4.6.2 External Reset

Depending on whether the external reset is configured as long or short as per the RGM_FESS[SS_EXR] bit value ('0' = long, and '1' = short) and whether self testing is enabled as per the shadow flash content (see the STCU chapter for details on how to disable the self test), a detected falling edge on the RESET_B pin will result in the following reset status flag settings.

Table 652. External Reset (configured as short)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
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

Table 653. External Reset (configured as long and without self test)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
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

Table 654. External Reset (configured as long and with self test)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
1	0	0	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

If the off-chip assertion of the RESET_B pin is used to control the exit of reset ***PHASE3*** (For example, on power-up to define the actual reset exit time or to allow the debugger to connect to the device and set up break points before reset exit), the RGM_FES[F_EXR] bit is not set.

42.4.6.3 'Functional' Resets

Most 'functional' reset events are also mapped to FCCU critical faults (CFs) or non-critical faults (NCFs). Since many of these can be configured for various reactions in the MC_RGM and also in the FCCU, it is recommended to configure both modules to result in the same reaction. This is to avoid confusion as to the actual resulting reaction should the one or other module have a fault that prevents its configured reaction from occurring. If both modules behave properly, the more severe reaction of the two will determine the actual resulting reaction.

42.4.6.3.1 Software Triggered 'Functional' Reset

This reset is triggered by software requesting a mode change to the ***RESET*** mode with the **ME_MCTL[TARGET_MODE]** = "0000" (see the MC_ME chapter for details on how to make a mode change request). Software may choose to trigger such a reset when, for example, software determines that a fault has occurred that can be cleared without a 'destructive' reset and doesn't require a self test to be executed.

The software triggered 'functional' reset can be configured as a short or long 'functional' reset. In addition, it is mapped to NCF[21] in the FCCU (see the FCCU chapter for details on how to configure NCFs).

Table 655. Software Triggered 'Functional' Reset (configured as short or long with no reset reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 656. Software Triggered 'Functional' Reset (configured as short or long with soft reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 657. Software Triggered 'Functional' Reset (configured as short or long with hard reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

42.4.6.3.2 Core Watchdog Timeout

The core watchdog timeout event can be configured as a short or long 'functional' reset, a ***SAFE*** mode request, and as an interrupt request. In addition, the Core_0 watchdog timeout is mapped to NCF[0] and the Core_1 watchdog timeout is mapped to NCF[1] in the FCCU (see the FCCU chapter for details on how to configure NCFs).

In principle, NCF[0] and NCF[1] could be configured for different FCCU reactions. However, this is not a typical use case.

Since the RGM_FES[F_CWD] flag is set for both cores, software should check the FCCU_NCFS0[NCFS0] and FCCU_NCFS0[NCFS1] bits to determine which core's watchdog timed out.

Table 658. Core Watchdog Timeout (configured as short or long with no reset reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 659. Core Watchdog Timeout (configured as short, long, *SAFE* mode, or IRQ with soft reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 660. Core Watchdog Timeout (configured as short, long, *SAFE* mode, or IRQ with hard reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 661. Core Watchdog Timeout (configured as short, long, *SAFE* mode, or IRQ with both soft and hard reactions from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

If the core watchdog timeout event is configured as a ***SAFE*** mode request or an interrupt request, and NCF[0] and NCF[1] are both configured for no reset reaction in the FCCU, then no reset will occur when the even t occurs.

If the core watchdog timeout event is configured as a ***SAFE*** mode request and NCF[0] or NCF[1] is configured for a soft or hard reaction in the FCCU, then the chip will enter ***SAFE*** mode immediately after reset exit.

42.4.6.3.3 PLL0 Fail

The PLL0 fail event can be configured as a short or long ‘functional’ reset, a ***SAFE*** mode request, and as an interrupt request. In addition, it is mapped to NCF[2] in the FCCU (see the FCCU chapter for details on how to configure NCFs).

Table 662. PLL0 Fail (configured as short or long with no reset reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 663. PLL0 Fail (configured as short, long, *SAFE* mode, or IRQ with soft reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 664. PLL0 Fail (configured as short, long, *SAFE* mode, or IRQ with hard reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

If the PLL0 fail event is configured as a SAFE mode request or an interrupt request, and NCF[2] is configured for no reset reaction in the FCCU, then no reset will occur when the even t occurs.

Caution: Configuring the PLL0 fail event as a **SAFE** mode request or an interrupt request and NCF[2] for no reset reaction in the FCCU is strongly discouraged if PLL0 is selected as the source for the system clock (ME_<mode>_MC[SYSCLK] = “0100”), since the core may not behave predictably if the system clock frequency is too high, and software may not be able to service the **SAFE** mode or interrupt request correctly.

If the PLL0 fail event is configured as a ***SAFE*** mode request and NCF[2] is configured for a soft or hard reaction in the FCCU, then the chip will enter ***SAFE*** mode immediately after reset exit.

42.4.6.3.4 Oscillator Frequency Too Low

The oscillator frequency too low event can be configured as a short or long 'functional' reset, a ***SAFE*** mode request, and as an interrupt request. In addition, it is mapped to NCF[4] in the FCCU (see the FCCU chapter for details on how to configure NCFs).

Table 665. Oscillator Frequency Too Low (configured as short or long with no reset reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 666. Oscillator Frequency Too Low (configured as short, long, *SAFE* mode, or IRQ with soft reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 667. Oscillator Frequency Too Low (configured as short, long, *SAFE* mode, or IRQ with hard reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

If the oscillator frequency too low event is configured as a ***SAFE*** mode request or an interrupt request, and NCF[4] is configured for no reset reaction in the FCCU, then no reset will occur when the even t occurs.

If the oscillator frequency too low event is configured as a ***SAFE*** mode request and NCF[4] is configured for a soft or hard reaction in the FCCU, then the chip will enter ***SAFE*** mode immediately after reset exit.

42.4.6.3.5 System Clock Frequency Too High/Low

The system clock frequency too high/low event can be configured as a short or long 'functional' reset, a ***SAFE*** mode request, and as an interrupt request. In addition, it is mapped to NCF[5] in the FCCU (see the FCCU chapter for details on how to configure NCFs).

Table 668. System Clock Frequency Too High/Low (configured as short or long with no reset reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 669. System Clock Frequency Too High/Low (configured as short, long, *SAFE* mode, or IRQ with soft reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 670. System Clock Frequency Too High/Low (configured as short, long, *SAFE* mode, or IRQ with hard reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

If the system clock frequency too high/low event is configured as a ***SAFE*** mode request or an interrupt request, and NCF[5] is configured for no reset reaction in the FCCU, then no reset will occur when the even t occurs.

Caution: Configuring the system clock frequency too high/low event as a **SAFE** mode request or an interrupt request and NCF[5] for no reset reaction in the FCCU is strongly discouraged,

since the core may not behave predictably if the system clock frequency is too high, and software may not be able to service the **SAFE** mode or interrupt request correctly.

If the system clock frequency too high/low event is configured as a ***SAFE*** mode request and NCF[5] is configured for a soft or hard reaction in the FCCU, then the chip will enter ***SAFE*** mode immediately after reset exit.

42.4.6.3.6 Software Watchdog Timeout

The software watchdog timeout event always generates a long 'functional' reset. In addition, the SWT_0 timeout is mapped to CF[14] and the SWT_1 timeout is mapped to CF[15] in the FCCU (see the FCCU chapter for details on how to configure CFs).

In principle, CF[14] and CF[15] could be configured for different FCCU reactions. However, the only realistic use case is to configure them both for a hard reaction.

Since the RGM_FES[F_SWT] flag is set for both SWTs, software should check the FCCU_CFS0[CFS14] and FCCU_CFS0[CFS15] bits to determine which core's watchdog timed out.

Sine the software watchdog timer is running during reset ***PHASE2*** (in order to return to the beginning of ***PHASE1*** in case the flash does not complete its initialization sequence within a maximum amount of time), the flag settings shown below can also occur in combination with RGM_DES flags as well as with other RGM_FES flags for external and 'functional' resets configured as long. This would be an indication to software that environmental conditions are causing the chip to reset the flash multiple times before being able to get the flash to successfully complete its initialization sequence and become operational.

Table 671. Software Watchdog Timeout (with no reset reaction from FCCU)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	0	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

Table 672. Software Watchdog Timeout (with soft reaction from FCCU)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	1	0	0	0	0	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

Table 673. Software Watchdog Timeout (with hard reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	0	0	0	0	0	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

Table 674. Software Watchdog Timeout (with both soft and hard reactions from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	1	0	0	0	0	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

Table 675. Software Watchdog Timeout (during flash initialization on power-up with self test)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

If a software watchdog timeout event occurs, then the chip will always enter ***SAFE*** mode immediately after reset exit.

42.4.6.3.7 PLL1 Fail

The PLL1 fail event can be configured as a short or long ‘functional’ reset, a ***SAFE*** mode request, and as an interrupt request. In addition, it is mapped to NCF[3] in the FCCU (see the FCCU chapter for details on how to configure NCFs).

Table 676. PLL1 Fail (configured as short or long with no reset reaction from FCCU)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	0	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

Table 677. PLL1 Fail (configured as short, long, *SAFE* mode, or IRQ with soft reaction from FCCU)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	1	0	0	0	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

Table 678. PLL1Fail (configured as short, long, *SAFE* mode, or IRQ with hard reaction from FCCU)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	0	0	0	0	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

If the PLL1 fail event is configured as a ***SAFE*** mode request or an interrupt request, and NCF[2] is configured for no reset reaction in the FCCU, then no reset will occur when the even t occurs.

If the PLL1 fail event is configured as a ***SAFE*** mode request and NCF[3] is configured for a soft or hard reaction in the FCCU, then the chip will enter ***SAFE*** mode immediately after reset exit.

42.4.6.3.8 Flash, ECC, or Lock-Step Error

The flash, ECC, or lock-step error event always generates a long ‘functional’ reset. In addition, the flash and SRAM ECC errors are mapped to CF[16] and CF[17] and the RCCUs are mapped to CF[0] through CF[10] in the FCCU (see the FCCU chapter for details on how to configure CFs).

In principle, the various CFs could be configured for different FCCU reactions. However, the only realistic use case is to configure them all for a hard reaction.

Since the RGM_FES[F_FL_ECC_RCC] flag is set for all of the above mentioned events, software should check the related FCCU_CFS0 register bits to determine which event actually caused the resulting reset.

Table 679. Flash, ECC, or Lock-Step Error (with no reset reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 680. Flash, ECC, or Lock-Step Error (with soft reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 681. Flash, ECC, or Lock-Step Error (with hard reaction from FCCU)

RGM_FES																RGM_DES															
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 682. Flash, ECC, or Lock-Step Error (with both soft and hard reactions from FCCU)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

If a flash, ECC, or lock-step error event occurs, then the chip will always enter ***SAFE*** mode immediately after reset exit.

42.4.6.3.9 Motor Control or FlexRay PE Clock Frequency Too High/Low

The motor control or FlexRay PE clock frequency too high/low event can be configured as a short or long 'functional' reset, a ***SAFE*** mode request, and as an interrupt request. In addition, the motor control clock monitor is mapped to NCF[6] and the FlexRay PE clock monitor is mapped to NCF[7] in the FCCU (see the FCCU chapter for details on how to configure NCFs).

In principle, NCF[6] and NCF[7] could be configured for different FCCU reactions. However, this is not a typical use case.

Since the RGM_FES[F_CMU12_FHL] flag is set for both clock monitors, software should check the FCCU_NCFS0[NCFS6] and FCCU_NCFS0[NCFS7] bits to determine which clock's frequency is out of range.

Table 683. Motor Control or FlexRay PE Clock Frequency Too High/Low (configured as short or long with no reset reaction from FCCU)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	0	0	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

Table 684. Motor Control or FlexRay PE Clock Frequency Too High/Low (configured as short, long, *SAFE* mode, or IRQ with soft reaction from FCCU)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	0	1	0	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

Table 685. Motor Control or FlexRay PE Clock Frequency Too High/Low (configured as short, long, *SAFE* mode, or IRQ with hard reaction from FCCU)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	0	0	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

Table 686. Motor Control or FlexRay PE Clock Frequency Too High/Low (configured as short, long, *SAFE* mode, or IRQ with both soft and hard reactions from FCCU)

RGM_FES																RGM_DES													
F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG	F_POR	F_SOFT_DEST								F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG			F_HVD12	F_LVD12
0	1	1	0	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

If the motor control or FlexRay PE clock frequency too high/low event is configured as a ***SAFE*** mode request or an interrupt request, and NCF[6] and NCF[7] are both configured for no reset reaction in the FCCU, then no reset will occur when the even t occurs.

If the motor control or FlexRay PE clock frequency too high/low event is configured as a ***SAFE*** mode request and NCF[6] or NCF[7] is configured for a soft or hard reaction in the FCCU, then the chip will enter ***SAFE*** mode immediately after reset exit.

42.4.6.3.10 Self-Test Completed

Self test execution is triggered only on power-on, ‘destructive’, and long external resets, and then only if the self test is enabled (see the STCU chapter for details on how to disable the self test). Therefore, if the **RGM_FES[F_ST_DONE]** flag is set, one of the **RGM_DES** flags

or the **RGM_FES[F_EXR]** flag must be set. See *Section 42.4.6.1, Power-On and 'Destructive' Resets* and *Section 42.4.6.2, External Reset* for example flag combinations.

42.4.7 Boot mode capturing

The MC_RGM samples PA[4:2] whenever RESET_B is asserted until five IRCOSC 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_B has been deasserted for subsequent boots after reset sequences during which RESET_B 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_B *is asserted.*

Note: RESET_B *can be low as a consequence of the internal reset generation. This will force re-sampling of the boot mode pins. (See Table 632 for details.)*

43 Self-Test Control Unit (STCU)

43.1 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 (Apps SW). The goal of the self test is to detect physical defects in the digital logic and embedded memories with enough coverage to meet the required Safety Integrity Level (SIL) of the system.

The SSCM interface is able to write the configuration parameters once after one of the trigger events has been detected by the System. The IPS interface is mainly able to read the STCU registers after the self-test sequence is over.

On cut1, the user-visible purpose of the STCU is to display the results of the self test.

43.1.1 Acronyms, abbreviations, and terms

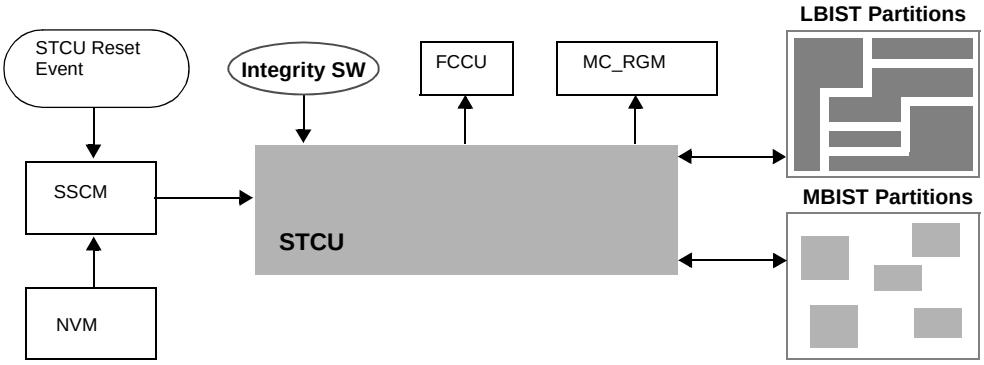
Table 687 contains acronyms, abbreviations, and terms used in this document.

Table 687. Acronyms and abbreviated terms

Term	Meaning
Apps SW	User applications software
BIST	Built-In Self Test (general term that includes both LBIST and MBIST)
CF	Critical Faults
CPU	Central Processing Unit
FCCU	Fault Collection Control Unit
HW	Hardware in general
IPS	Integrated Peripheral System Bus Interface
LBIST	Logic Built-In Self Test
MBIST	Memory Built-In Self Test
MC_RGM	Reset generation module
MISR	Multiple Input Shift Register (for LBIST result signatures)
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

43.1.2 Safety Integrity Subsystem

Figure 808 shows the STCU as a component of the Safety Integrity Subsystem on the device.



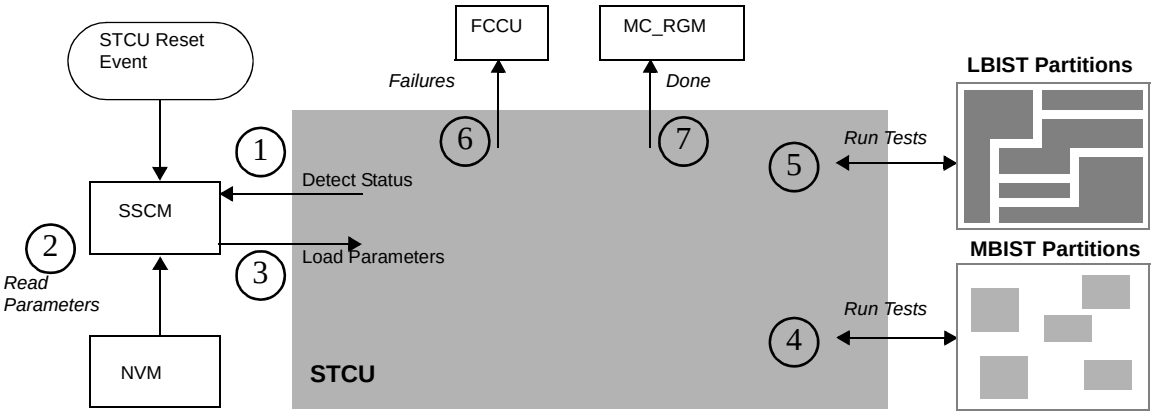
Component(1)	Description
FCCU	The Fault Collection Control Unit collects errors and controls the safety state of the device.
Integrity SW	The Safety Integrity Software checks the STCU status before passing control over to Apps SW.
LBIST Partitions	The set of individual logic block partitions included in the self test
MBIST Partitions	The set of individual embedded memory blocks included in the self test
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: – Destructive Reset, BIST enabled (for the definition of Destructive Reset, please see *Section 42.4.2, Destructive Resets*) – External Reset Long, BIST enabled

1. Components are the hardware and software that make up a subsystem. Events that affect subsystem behavior are also included.

Figure 808. The STCU within the Safety Integrity Subsystem

When an STCU Reset Event occurs, the device goes through a two-phase boot sequence:

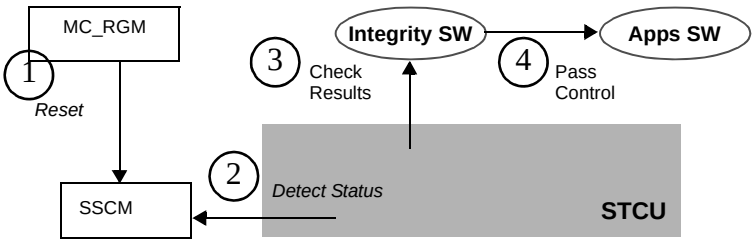
1. Self-test phase: See *Figure 809*.
2. Functional-reset phase: See *Figure 810*.



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.
4. The STCU manages the MBISTs and updates its internal status.
5. The STCU manages the LBISTs and updates its internal status.
6. If faults are detected, the STCU reports the test failures to the FCCU.
7. The STCU signals the MC_RGM that the tests are complete, and the boot sequence proceeds to the next phase (see *Figure 810*).

Note: All steps described in the box above are executed only after (not before) the MC_RGM is in IDLE phase.

Figure 809. Boot sequence phase 1: Self test



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. See *Section 43.1.3, Integrity SW operations*."
4. If the Integrity software check passes, the device can be considered as OK and control can be passed to the Apps software (applications).

Figure 810. Boot sequence phase 2: Functional reset

The functional reset shown in step 1 of *Figure 810* does not reset the MBIST or LBIST results. Therefore, after MC_RGM passes through PHASE1, PHASE2, PHASE3, and IDLE, the STCU is still aware of the MBIST and LBIST status and does not generate a new functional reset. It thus allows the SSCM to continue.

43.1.3 Integrity SW operations

The Integrity SW should perform operations based on the STCU status conditions after the self test. Even if no errors are reported, the Integrity SW should confirm that all MBIST and LBIST finished successfully and no further error is flagged. This software confirmation prevents a fault within the STCU itself from incorrectly indicating that the self test passed.

43.1.3.1 Reported errors

In the case of reported errors, the Integrity SW should:

- Read the STCU_LBS flag register to determine which LBISTs failed.
- Read the STCU_LBE flag register to determine which LBISTs did not finish.
- Read the STCU_MBSL and STCU_MBSH flag registers to determine which MBISTs failed.
- Read the STCU_MBEL and STCU_MBEH flag registers to determine which MBISTs did not finish.
- Read the STCU_ERR register to check whether there has been an internal STCU failure.

43.1.3.2 No reported errors

In the case of no reported errors, the Integrity SW should confirm the following:

- The signature registers of each of the LBIST results match their corresponding expected values:
  For each LBIST:
  - Read the STCU_LBMISREL/H and STCU_LBIST_NMISRRL/H registers to check the coherency with the STCU_LBS and STCU_LBE bits.
  - Read the registers described in *Section 43.1.3.1, Reported errors,"* and verify that their values are as expected.

43.2 STCU main features

The STCU features include the following:

- Performs a one-time self test after an STCU reset event
- Provides register interfaces for both software and hardware:
  - Hardware: SSCM write-one-time register interface
  - Software: IPS register interface
- Manages LBISTs
- Manages 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 LBIST and MBIST operations
  - Flags for STCU internal errors
- Software can confirm the integrity of the LBIST status information by directly comparing the expected and actual results the LBIST operations.

43.3 Block diagram and components

Figure 811 shows a block diagram of the STCU.

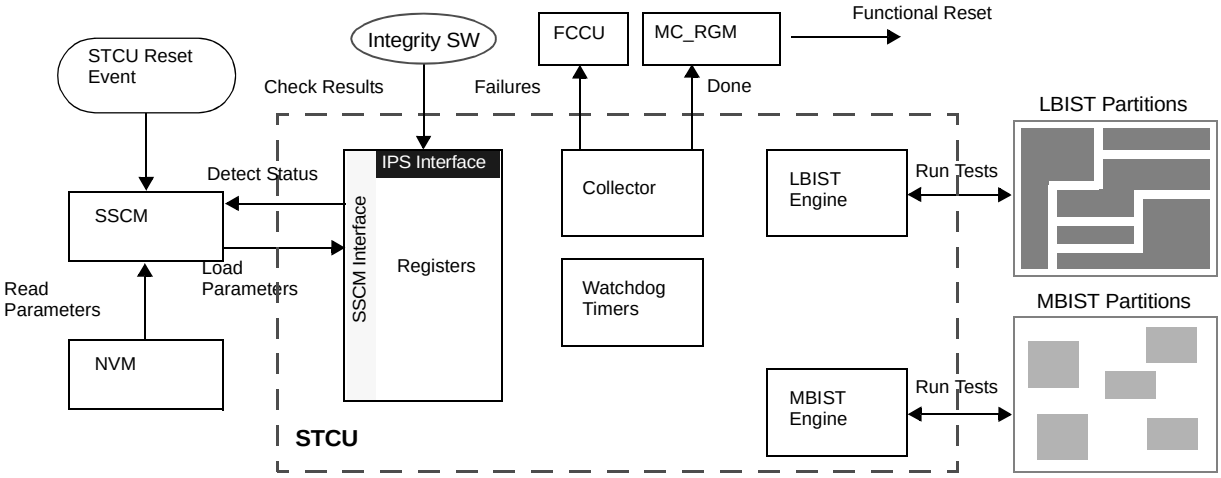


Figure 811. STCU block diagram

The main components of the STCU are:

- ***Registers***: Hold the self-test parameters and status flags: scheduling activity, LBIST setup, Critical/Non-Critical/Stay-in-Reset fault mapping, MISR expected values. See *Section 43.4, 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.
- ***Collector***: Collects and updates the status and error conditions related to the L/MBIST execution and STCU internal operation. The Collector sends the BIST error information to the FCCU and signals the MC_RGM to begin a functional reset.
- ***Watchdog Timers:*** Provide a double functionality to:
  - Protect against Dead-Lock or runaway condition during the self test
  - Verify that each L/MBIST has been completed in its assigned L/MBIST time slot
- ***LBIST Engine***: Manages the testing of logic blocks on the device.
- ***MBIST Engine***: Manages the testing of embedded memory blocks.

43.4 Memory map and register definition

All registers shown in this section are defined as visible by the IPS interface.

The STCU contains registers for:

- BIST status reporting (finished, successfull)
- STCU status reporting (internal errors)
- Bypassing selftest mode
- Key registers used to clear error flags

43.4.1 Memory map

The STCU memory map is listed in *Table 688*.

For LBIST partitioning, see *Section 43.5, LBIST partitioning."*

Table 688. STCU register map

Offset from STCU_BASE (0xC3FF_4000)(1)	Register Name	Access(2)	Location
0x0000–0x0004	Reserved		
0x0008 (cut1)	Reserved		
0x0008 (cut2/3)	STCU SK Code Register (STCU_SKC)	W	*on page 1304*
0x000C (cut1)	STCU Configuration Register (STCU_CFG)	R	*on page 1305*
0x000C (cut2/3)	Reserved		
0x0010–0x0018	Reserved		
0x001C	STCU Error Register (STCU_ERR)	R [cut1] R/W [cut2/3]	*on page 1306*
0x0020 (cut1)	Reserved		
0x0020 (cut2/3)	STCU Error Key Register (STCU_ERRK)	W	*on page 1307*
0x0024	STCU LBIST Status Register (STCU_LBS)	R	*on page 1308*
0x0028	STCU LBIST End Flag Register (STCU_LBE)	R	*on page 1309*
0x002C–0x0038	Reserved		
0x003C	STCU MBIST Status Low Register (STCU_MBSL)	R	*on page 1309*
0x0040	STCU MBIST Status High Register (STCU_MBSH)	R	*on page 1310*
0x0044	STCU MBIST End Flag Low Register (STCU_MBEL)	R	*on page 1311*
0x0048	STCU MBIST End Flag High Register (STCU_MBEH)cut2/3	R	*on page 1311*
0x004C–0x0084	Reserved		
0x0088 (cut1)	Reserved		

Table 688. STCU register map (continued)

Offset from STCU_BASE (0xC3FF_4000)[1]	Register Name	Access[2]	Location
0x0088 + (*n* × 0x20) (cut2/3)	STCU LBIST MISR Expected Low Register (STCU_LB_MISREL)	R	*on page 1312*
0x008C (cut1)	Reserved		
0x008C + (*n* × 0x20) (cut2/3)	STCU LBIST MISR Expected High Register (STCU_LB_MISREH)	R	*on page 1313*
0x0090 + (*n* × 0x20)	STCU LBIST MISR Read Low Register (STCU_LB_MISRRL)	R	*on page 1313*
0x0094 + (*n* × 0x20)	STCU LBIST MISR Read High Register (STCU_LB_MISRRH)	R	*on page 1314*
0x0098–0x7FFF	Reserved		

1. *n* is a variable representing the repeated register blocks of the multiple LBISTs and MBISTs: *n* ranges from 0 to 2.
2. In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

43.4.2 Register conventions

The following bus operations (contiguous byte enables) are supported:

- Word (32 bits) data read operations
- Word (32 bits) data write operation for write-enabled registers

The registers of the STCU are accessible in each access mode: user or supervisor. Reading and writing to reserved areas results in unexpected behavior.

43.4.3 Detailed register descriptions

43.4.3.1 STCU SK Code Register (STCU_SKC) [cut2/3 only]

The STCU_SKC register implements the security key code mechanism needed to access in write mode to the other STCU registers. In order to unlock the STCU access after:

- The Power-On, Destructive or External Reset
- The completation of the STCU run

the SW (IPs bus) or the SSCM interfaces have to apply the following sequence:

- write the key1 into the STCU_SKC register
- write the key2 into the STCU_SKC register

After the Self-Test sequence has been completed or the Bypass feature has been enabled (setting the bit BYP into the STCU_CFG), the SSCM interface is no longer 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, in order 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 0x00000000 is always returned in case of a read operation.

Figure 812. STCU SK Code Register (STCU_SKC) [cut2/3 only]

Address: Base + 0x0008 (cut2/3 only) Access: User write-only

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

Table 689. STCU_SKC field descriptions

Field	Description
SKC	SKC: *STCU security key code* = 0xABFC1893: Key1 to unlock the write access the STCU (when not protected) = 0x319A6C2F: Key2 to unlock the write access the STCU (when not protected)

43.4.3.2 STCU Configuration Register (STCU_CFG) [cut1 only]

The STCU_CFG register includes the global configuration of the STCU.

Figure 813. STCU Configuration Register (STCU_CFG) [cut1 only]

Address: Base + 0x000C [cut1 only] Access: User read-only

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

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

Table 690. STCU_CFG field descriptions

Field	Description
BYP	*Bypass mode* 0: STCU is active. 1: STCU is in bypass mode. The self test is bypassed.

43.4.3.3 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.

The STCU_ERR fields can be cleared by software depending on the applied unlocked sequence:

- Write the keys into the STCU_ERRK
- Set/clear the STCU_ERR register

where

- Key1 allows to clear the fields that have a value of 1 by writing a 0 to those fields
- Key2 allows to set the fields that have a value of 0 by writing a 1 to those fields

In case of invalid access (wrong or missing key), a transfer error is asserted (it depends on the selected bus interface) and the writing operation on STCU_ERR register is ignored.

Figure 814. STCU Error Register (STCU_ERR) for cut1

Address: Base + 0x001C Access: User read-only

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	CFSF	NCFSF	SIRSF	0	0	0	0	WDTO	0	ENGE	INVP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 815. STCU Error Register (STCU_ERR) for cut2/3

Address: Base + 0x001C Access: User read-only

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

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

Table 691. STCU_ERR field descriptions

Field	Description
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.
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 (cut1)	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.
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 LBIST or MBIST pointer is out of range – LBIST is selected when MBIST is concurrently running or vice versa – Error in the LBIST/MBIST linking (execution generates an infinite loop)

43.4.3.4 STCU Error Key Register (STCU_ERRK) [cut2/3 only]

The STCU_ERRK register, available on cut2/3 only, implements the security key code to access to the STCU_ERR register. In order to write the STCU_ERR register, software must:

- Write the keyx into the STCU_ERRK
- Set/clear the STCU_ERR register

wher:e

- 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 interface) 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 watchdog 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 a read operation.

Figure 816. STCU Error Key Register (STCU_ERRK) [cut2/3 only]

Address: Base + 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[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	ERR_SK[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 692. STCU_ERRK field descriptions

Field	Description
ERR_SK	STCU_ERRK security key 0xF175_9034: Key1 to reset the STCU_ERR bits at 1 0x9531_B0C6: Key2 to set the STCU_ERR bits at 0

43.4.3.5 STCU LBIST Status Register (STCU_LBS)

The STCU_LBS register includes the results corresponding to the execution of each LBIST. The STCU_LBS register is automatically set following the completion of the LBIST run.

Figure 817. STCU LBIST Status Register (STCU_LBS)

Address: Base + 0x0024 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	LBS2	LBS1	LBS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 693. STCU_LBS field descriptions

Field	Description
LBSx	*LBIST status* 0: Failed LBIST execution 1: No fault detected during the BIST execution

Note: A BIST counts as successfull when no fault was detected during execution (see LBS) and its execution is finished (see LBE).

Note: The status of LBIST4 is reflected in (mapped onto) LBS2.

43.4.3.6 STCU LBIST End Flag Register (STCU_LBE)

The STCU_LBE register includes the End Flag related to the execution of each LBIST. The STCU_LBE register is automatically updated following the completion of the LBIST run.

Figure 818. STCU LBIST End Flag Register (STCU_LBE)

Address: Base + 0x0028 Access: User read-only

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	LBE2	LBE1	LBE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 694. STCU_LBE field descriptions

Field	Description
LBEx	*LBIST End status* 0: LBIST execution is not finished. 1: LBIST execution is finished.

Note: A BIST counts as successfull when no fault was detected during execution (see LBS) and its execution is finished (see LBE).

Note: The status of LBIST4 is reflected in (mapped onto) LBE2.

43.4.3.7 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.

Figure 819. STCU MBIST Status Low Register (STCU_MBSL)

Address: Base + 0x003C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MBS 31	MBS 30	MBS 29	MBS 28	MBS 27	MBS 26	MBS 25	MBS 24	MBS 23	MBS 22	MBS 21	MBS 20	MBS 19	MBS 18	MBS 17	MBS 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	MBS 15	MBS 14	MBS 13	MBS 12	MBS 11	MBS 10	MBS 9	MBS 8	MBS 7	MBS 6	MBS 5	MBS 4	MBS 3	MBS 2	MBS 1	MBS 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 695. STCU_MBSL field descriptions

Field	Description
Bits 0:31	MBSx: *MBIST status* 0: Failed MBIST execution 1: No fault detected during the BIST execution

Note: A BIST counts as successfull when no fault was detected during execution (see MBSL, MBSH) and its execution is finished (see MBEL, MBEH).

43.4.3.8 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.

Figure 820. STCU MBIST Status High Register (STCU_MBSH)

Address: Base + 0x0040 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	MBS 34	MBS 33	MBS 32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 696. STCU_MBSH field descriptions

Field	Description
MBSx	*MBIST status* 0: Failed MBIST execution 1: No fault detected during the BIST execution

Note: A BIST counts as successfull when no fault was detected during execution (see MBSL, MBSH) and its execution is finished (see MBEL, MBEH).

43.4.3.9 STCU MBIST End Flag Low Register (STCU_MBEL)

The STCU_MBEL register includes the End Flag related to the execution of each MBIST. The STCU_MBEL register is automatically updated following the completion of the MBIST run.

Figure 821. STCU MBIST End Flag Low Register (STCU_MBEL)

Address: Base + 0x0044 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MBE 31	MBE 30	MBE 29	MBE 28	MBE 27	MBE 26	MBE 25	MBE 24	MBE 23	MBE 22	MBE 21	MBE 20	MBE 19	MBE 18	MBE 17	MBE 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	MBE 15	MBE 14	MBE 13	MBE 12	MBE 11	MBE 10	MBE 9	MBE 8	MBE 7	MBE 6	MBE 5	MBE 4	MBE 3	MBE 2	MBE 1	MBE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 697. STCU_MBEL field descriptions

Field	Description
MBEx	: *MBIST End status* 0: MBIST execution is not finished. 1: MBIST execution is finished.

Note: A BIST counts as successfull when no fault was detected during execution (see MBSL, MBSH) and its execution is finished (see MBEL, MBEH).

43.4.3.10 STCU MBIST End Flag High Register (STCU_MBEH)

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.

Figure 822. STCU MBIST End Flag High Register (STCU_MBEH)

Address: Base + 0x0048 Access: User read-only

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	MBE 34	MBE 33	MBE 32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 698. STCU_MBEH field descriptions

Field	Description
Bits 0:28	Reserved These are reserved bits. These bits are always read as ‘0’ and must always be written with ‘0’.
Bits 29:31	MBEx: *MBIST End status* 0: MBIST execution is not finished. 1: MBIST execution is finished.

Note: A BIST counts as successfull when no fault was detected during execution (see MBSL, MBSH) and its execution is finished (see MBEL, MBEH).

43.4.3.11 STCU LBIST MISR Expected Low Register (STCU_LB_MISREL) [cut2/3 only]

The STCU_LB_MISREL register defines the LSB part of the Expected MISR of each LBIST controller.

Figure 823. STCU LBIST MISR Expected Low Register (STCU_LB_MISREL) [cut2/3 only]

Address: Base + 0x0088 + (*n* × 0x20)[1] [cut2/3 only] Access: User read

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MISREL[15:0]															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1. The *n* variable represents the repeated register blocks of the multiple LBISTs: *n* ranges from 0 up to 2.

Table 699. STCU_LB_MISREL field descriptions

Field	Description
MISREL	MISR Expected low part MISREL defines the low part of the Expected MISR.

43.4.3.12 STCU LBIST MISR Expected High Register (STCU_LB_MISREH) [cut2/3 only]

The STCU_LB_MISREH register defines the MSB part of the Expected MISR of each LBIST controller.

Figure 824. STCU LBIST MISR Expected High Register (STCU_LB_MISREH) [cut2/3 only]

Address: Base + 0x008C + (*n* × 0x20)(1) [cut2/3 only] Access: User read

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MISREH[15:0]															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1. The *n* variable represents the repeated register blocks of the multiple LBISTs: *n* ranges from 0 up to 2.

Table 700. STCU_LB_MISREH field descriptions

Field	Description
MISREH	MISR Expected high part MISREH defines the high part of the Expected MISR.

43.4.3.13 STCU LBIST MISR Read Low Register (STCU_LB_MISRRL)

The STCU_LB_MISRRL registers report the LSB part of the MISR obtained at the end of each LBIST.

Figure 825. STCU LBIST MISR Read Low Register (STCU_LB_MISRRL)

Address: Base + 0x0090 + (*n* × 0x20)(1) Access: User read/write

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

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

1. The *n* variable represents the repeated register blocks of the multiple LBISTs: *n* ranges from 0 up to 2.

Table 701. STCU_LB_MISRRL field descriptions

Field	Description
Bits 0:31	MISRRL: *MISR Read low part* Contains the low word of the MISR obtained at the end of the LBIST

43.4.3.14 STCU LBIST MISR Read High Register (STCU_LB_MISRRH)

The STCU_LB_MISRRH registers report the MSB part of the MISR obtained at the end of each LBIST.

Figure 826. STCU LBIST MISR Read High Register (STCU_LB_MISRRH)

Address: Base + 0x0094 + (*n* × 0x20) (1) Access: User read-only

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

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

1. The *n* variable represents the repeated register blocks of the multiple LBISTs: *n* ranges from 0 up to 2.

Table 702. STCU_LB_MISRRH field descriptions

Field	Description
Bits 0:31	MISRRH: *MISR Read high part* Contains the high word of the MISR obtained at the end of the LBIST

43.5 LBIST partitioning

The LBIST partitioning scheme for cut1 and cut2/3 of this device is shown in *Table 703* and *Table 704*, respectively.

Table 703. LBIST partitioning for cut1

LBIST Partition	Module
0	Core_0
	AXBS_0
	MPU_0
	PBRIDGE_0
	PBRIDGE_0 Off-Platform Read Mux
	PBRIDGE_0 On-Platform Read Mux
	PFLASHC_0
	SRAMC_0
	DMA_0
	DMACHMUX_0
	STM_0
	SWT_0
	INTC_0
	ECSM_0
	SEMA4_0
	Cache Parity Propagation_0
	TSENS_0
	SRAMC_0 Read Mux

Table 703. LBIST partitioning for cut1 (continued)

LBIST Partition	Module
1	Core_1
	XBAR_1
	MPU_1
	PBRIDGE_1
	PBRIDGE_1 Off-Platform Read Mux
	PBRIDGE_1 On-Platform Read Mux
	PFLASHC_1
	SRAMC_1
	eDMA_1
	DMACHMUX_1
	STM_1
	SWT_1
	INTC_1
	ECSM_1
	SEMA4_1
	Cache Parity Propagation_1
	TSENS_1
	SRAMC_1 Read Mux
2	FlexPWM_0
	eTimer_0
	ADC_0
	CTU
3	FlexPWM_1
	eTimer_1
	ADC_1
4	SSCM

Table 704. LBIST partitioning for cut2/3

LBIST Partition	Module
0	Core_0
	XBAR_0
	MPU_0
	PBRIDGE_0
	PFLASHC_0
	SRAMC_0
	DMA_0
	DMACHMUX_0
	STM_0
	SWT_0
	CMU_0
	INTC_0
	ECSM_0
	SEMA4_0
	TSENS_0
	FlexPWM_0
	eTimer_0
	ADC_0
	CTU

Table 704. LBIST partitioning for cut2/3 (continued)

LBIST Partition	Module
1	Core_1
	AXBS_1
	MPU_1
	PBRIDGE_1
	PFLASHC_1
	SRAMC_1
	DMA_1
	DMACHMUX_1
	STM_1
	SWT_1
	INTC_1
	ECSM_1
	SEMA4_1
	TSENS_1
	FlexPWM_1
	eTimer_1
	ADC_1

Table 704. LBIST partitioning for cut2/3 (continued)

LBIST Partition	Module
4	RCCU5–0
	WKPU
	LINFlexD_0
	LINFlexD_1
	eTIMER_2
	DSPI_0
	DSPI_1
	DSPI_2
	FLEXRAY
	PIT
	FlexCAN_0
	FlexCAN_1
	CRC
	CMU_1
	CMU_2
	FMPLL_0
	FMPLL_1
	SIUL
	SWG
	NPC
	MC_ME
	MC_PCU
	FLASH
	FCCU
	IOMUX

43.6 MBIST partitioning

Table 705. MBIST partitioning

MBIST	Type	Module
0	SRAM	System RAM (128 KB plus ECC)
1	SRAM	System RAM (128 KB plus ECC)
2	SRAM	System RAM (128 KB plus ECC)
3	SRAM	System RAM (128 KB plus ECC)
4	SRAM	System RAM (128 KB plus ECC)

Table 705. MBIST partitioning (continued)

MBIST	Type	Module
5	SRAM	System RAM (128 KB plus ECC)
6	SRAM	System RAM (128 KB plus ECC)
7	SRAM	System RAM (128 KB plus ECC)
8	SRAM	Platform I-Cache, SoR 0
9	SRAM	Platform I-Cache, SoR 0
10	SRAM	Platform I-Cache, SoR 0
11	SRAM	Platform I-Cache, SoR 0
12	SRAM	Platform Cache Tag, SoR 0
13	SRAM	Platform Cache Tag, SoR 0
14	SRAM	Platform Cache Tag, SoR 0
15	SRAM	Platform Cache Tag, SoR 0
16	SRAM	DMA memory, SoR 0
17	SRAM	Platform I-Cache, SoR 1
18	SRAM	Platform I-Cache, SoR 1
19	SRAM	Platform I-Cache, SoR 1
20	SRAM	Platform I-Cache, SoR 1
21	SRAM	Platform Cache Tag, SoR 1
22	SRAM	Platform Cache Tag, SoR 1
23	SRAM	Platform Cache Tag, SoR 1
24	SRAM	Platform Cache Tag, SoR 1
25	SRAM	DMA memory, SoR 1
26	SRAM	FlexCAN RX Buffer, rxim_ram
27	SRAM	FlexCAN RX Buffer, rxim_ram
28	SRAM	FlexCAN TX Buffer, mb_ram
29	SRAM	FlexCAN TX Buffer, mb_ram
30	SRAM	FlexRay LUT
31	SRAM	FlexRay LUT
32	SRAM	FlexRay Data Table
33	ROM	FlexRay ROM
34	ROM	BAM ROM

43.7 Self-test bypass and MBIST-only mode

The default STCU configuration as it is located in the test flash memory leads to a full self-test after an STCU reset event (running all MBISTs and LBISTs).

Besides this, two configurations are available:

- Bypass self-test mode (disabling the whole self-test)
- MBIST-only mode (disabling all LBISTs)

To configure those cases, a specific hexadecimal code needs to be added to the shadow flash memory. The required hexadecimal codes for the optional selftest modes (Bypass, MBIST-only) are listed below:

SHADOWFLASH HEXCODE

- TO CONFIG SELFTEST_DISABLED (SELFTEST BYPASSED) MODE

Note: This code only works if there is a syntactic-valid stcu configuration in the testflash already.

Starting at first address in the shadowflash program:

```
05AA55AF_00000000_FFFFFFFF_FFFFFFFF
00000000_00080000_FFFFFFFF_FFFFFFFF
00000900_0008000C_FFFFFFFF_FFFFFFFF
FFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFF
```

- TO CONFIG MBIST_ONLY MODE

Note: This code only works if there is a syntactic-valid stcu configuration in the testflash already.

Starting at first address in the shadowflash program:

```
05AA55AF_00000000_FFFFFFFF_FFFFFFFF
7F140000_00080388_FFFFFFFF_FFFFFFFF
10000800_0008000C_FFFFFFFF_FFFFFFFF
913756AF_0008002C_FFFFFFFF_FFFFFFFF
0000FFFF_00080024_FFFFFFFF_FFFFFFFF
0000FFFF_00080028_FFFFFFFF_FFFFFFFF
00000002_00080010_FFFFFFFF_FFFFFFFF
FFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFF
```

44 Semaphore Unit (SEMA4)

44.1 Introduction

SPC56ELxx contains two SEMA4 units.

In a dual-processor chip, semaphores are used to let each processor know who has control of common memory. Before a core can update or read memory coherently, it has to check the semaphore to see if the other core is not already updating the memory. If the semaphore is clear, it can write common memory, but if it is set, it has to wait for the other core to finish and clear the semaphore.

The semaphore unit (SEMA4) provides the hardware support needed in multi-core systems for implementing semaphores and provide a simple mechanism to achieve lock/unlock operations via a single write access. This approach eliminates architecture-specific implementations like atomic (indivisible) read-modify-write instructions or reservation mechanisms. The result is an architecture-neutral solution that provides hardware-enforced gates as well as other useful system functions related to the gating mechanisms.

On SPC56ELxx, the SEMA4 unit is intended for use when using the device in dual processor mode. When using the device in Lock Step mode, the SEMA4 unit is disabled and its interrupt sources deasserted.

In this chapter, the two instantiations of the e200z4d core on SPC56ELxx are referred to as e200z4d_0 and e200z4d_1. (The e200z4d_0 instantiation runs from reset in dual processor mode.)

44.1.1 Block diagram

Figure 827 is a simplified block diagram of the SEMA4 unit that illustrates the functionality and interdependence of major blocks. In the diagram, the register blocks named gate0, gate1, ..., gate 15 include the finite state machines implementing the semaphore gates plus the interrupt notification logic.

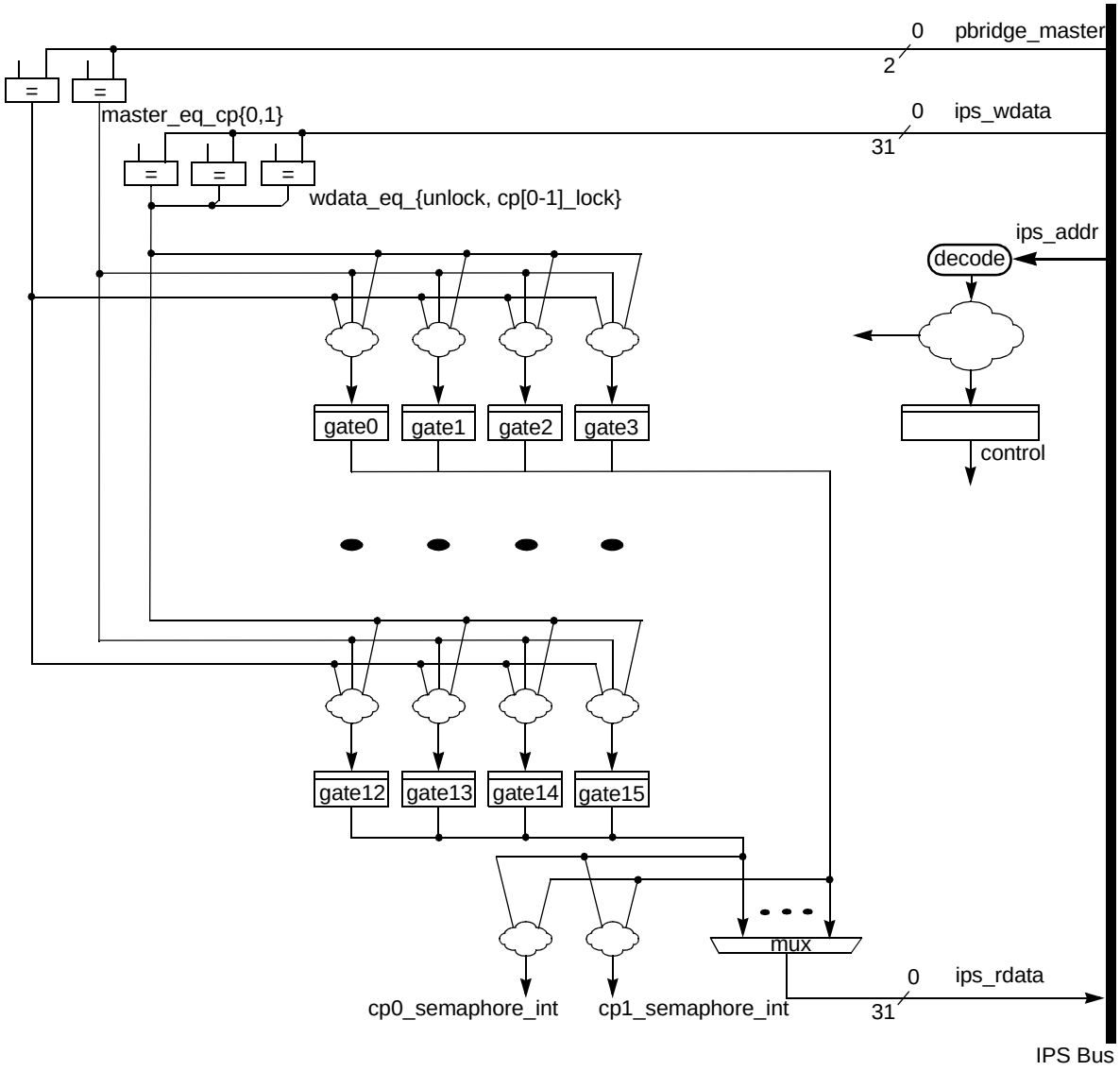


Figure 827. SEMA4 block diagram

44.1.2 Features

The SEMA4 unit implements hardware-enforced semaphores as a peripheral device and has these major features:

- Support for 16 hardware-enforced gates in a dual-processor configuration
- Each hardware gate appears as a three-state, 2-bit state machine, with all 16 gates mapped as an array of bytes
  - Three-state implementation
  - if gate = 0b00, then state = unlocked
  - if gate = 0b01, then state = locked by e200z4d_0 (master ID = 0)
  - if gate = 0b10, then state = locked by e200z4d_1 (master ID = 1)
  - Uses the bus master ID number as a reference attribute plus the specified data patterns to validate all write operations
  - After it is locked, the gate must be unlocked by a write of zeroes from the locking processor
- Optionally enabled interrupt notification after a failed lock write provides a mechanism to indicate the gate is unlocked
- Secure reset mechanisms are supported to clear the contents of individual semaphore gates or notification logic, and clear_all capability

Note: Semaphore gates that are locked when entering sleep mode are cleared by the internal reset generated when exiting sleep mode.

44.1.3 Modes of operation

The SEMA4 unit does not support any special modes of operation.

44.2 Signal description

The SEMA4 unit does not include any external signals.

44.3 Memory map and register description

The SEMA4 programming model map is shown in *Table 706*. The address of each register is given as an offset to the SEMA4 base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

Table 706. SEMA4 memory map

Address offset	Register	Location
0x0000	SEMA4_Gate00—Semaphores gate 0	*on page 1325*
0x0001	SEMA4_Gate01—Semaphores gate 1	*on page 1325*
0x0002	SEMA4_Gate02—Semaphores gate 2	*on page 1325*
0x0003	SEMA4_Gate03—Semaphores gate 3	*on page 1325*
0x0004	SEMA4_Gate04—Semaphores gate 4	*on page 1325*
0x0005	SEMA4_Gate05—Semaphores gate 5	*on page 1325*

Table 706. SEMA4 memory map (continued)

Address offset	Register	Location
0x0006	SEMA4_Gate06—Semaphores gate 6	*on page 1325*
0x0007	SEMA4_Gate07—Semaphores gate 7	*on page 1325*
0x0008	SEMA4_Gate08—Semaphores gate 8	*on page 1325*
0x0009	SEMA4_Gate09—Semaphores gate 9	*on page 1325*
0x000A	SEMA4_Gate10—Semaphores gate 10	*on page 1325*
0x000B	SEMA4_Gate11—Semaphores gate 11	*on page 1325*
0x000C	SEMA4_Gate12—Semaphores gate 12	*on page 1325*
0x000D	SEMA4_Gate13—Semaphores gate 13	*on page 1325*
0x000E	SEMA4_Gate14—Semaphores gate 14	*on page 1325*
0x000F	SEMA4_Gate15—Semaphores gate 15	*on page 1325*
0x0010–0x003F	Reserved	
00x040	SEMA4_CP0INE—Semaphores CP0 IRQ notification enable	*on page 1326*
0x0042–0x0047	Reserved	
0x0048	SEMA4_CP1INE—Semaphores CP1 IRQ notification enable	*on page 1326*
0x004A–0x07F	Reserved	
0x0080	SEMA4_CP0NTF—Semaphores CP0 IRQ notification	*on page 1327*
0x008 2–00x087	Reserved	
0x0088	SEMA4_CP1NTF—Semaphores CP1 IRQ notification	*on page 1326*
0x008A–0x00FF	Reserved	
0x0100	SEMA4_RSTGT—Semaphores reset gate	*on page 1327*
0x0102	Reserved	
0x0104	SEMA4_RSTNTF—Semaphores reset IRQ notification	*on page 1329*
0x0106–0x3FFF	Reserved	

44.3.1 Semaphores gate *n* register (SEMA4_GATE*n*)

Each semaphore gate is implemented in a 2-bit finite state machine, right-justified in a byte data structure. The hardware uses the bus master number in conjunction with the data patterns to validate all attempted write operations. Only processor bus masters can modify the gate registers. After it is locked, a gate must be opened (unlocked) by the locking processor core.

Multiple gate values can be read in a single access, but only a single gate at a time can be updated via a write operation. 16- and 32-bit writes to multiple gates are allowed, but the write data operand must update the state of a single gate only. A byte write data value of 0x03 is defined as no operation and does not affect the state of the corresponding gate register. Attempts to write multiple gates in a single-aligned access with a size larger than an 8-bit (byte) reference generate an error termination and do not allow any gate state changes.

Offset: *n* (*n* = 0x0, 0x1,0x2,..., 0xF) Access: User read/write

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

Figure 828. SEMA4 gate *n* register (SEMA4_GATE*n*)

Table 707. SEMA4_GATE*n* field descriptions

Field	Description
GTFSM	Gate Finite State Machine. The hardware gate is maintained in a three-state implementation, defined as: 00 The gate is unlocked (free). 01 The gate has been locked by processor 0. 10 The gate has been locked by processor 1. 11 This state encoding is never used and therefore reserved. Attempted writes of 0x03 are treated as no operation and do not affect the gate state machine. *Note: The state of the gate reflects the last processor that locked it, which can be useful during system debug.*

44.3.2 Semaphores processor *n* IRQ notification enable (SEMA4_CP{0,1}INE)

The application of a hardware semaphore module provides an opportunity for implementation of helpful system-level features. An example is an optional mechanism to generate a processor interrupt after a failed lock attempt. Traditional software gate functions execute a spin-wait loop in an effort to obtain and lock the referenced gate. With this module, the processor that fails in the lock attempt could continue with other tasks and allow a properly-enabled notification interrupt to return its execution to the original lock function.

The optional notification interrupt function consists of two registers for each processor: an interrupt notification enable register (SEMA4_CP*n*INE) and the interrupt request register (SEMA4_CP*n*NTF). To support implementations with more than 16 gates, these registers can be referenced with aligned 16- or 32-bit accesses. For the SEMA4_CP*n*INE registers, unimplemented bits read as zeroes and writes are ignored.

Offset : 0x0040 (SEMA4_CP0INE)
0x0048 (SEMA4_CP1INE) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	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 829. Semaphores processor *n* IRQ notification enable (SEMA4_CP{0,1}INE)

Table 708. 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.

44.3.3 Semaphores processor *n* IRQ notification (SEMA4_CP{0,1}NTF)

The notification interrupt is generated via a unique finite state machine, one per hardware gate. This machine operates in the following manner:

- When an attempted lock fails, the FSM enters a first state where it waits until the gate is unlocked.
- After it is unlocked, the FSM enters a second state where it generates an interrupt request to the failed lock processor.
- When the failed lock processor succeeds in locking the gate, the IRQ is automatically negated and the FSM returns to the idle state. However, if the other processor locks the gate again, the FSM returns to the first state, negates the interrupt request, and waits for the gate to be unlocked again.

The notification interrupt request is implemented in a 3-bit, five-state machine, where two specific states are encoded and program-visible as SEMA4_CP0NTF[GN*n*] and SEMA4_CP1NTF[GN*n*].

Offset: 0x0080 (SEMA4_CP0NTF)
0x0088 (SEMA4_CP1NTF)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	GN0	GN1	GN2	GN3	GN4	GN5	GN6	GN7	GN8	GN9	GN10	GN11	GN12	GN13	GN14	GN15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 830. Semaphores processor *n* IRQ notification (SEMA4_CP{0,1}NTF)

Table 709. 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.

44.3.4 Semaphores (secure) reset gate *n* (SEMA4_RSTGT)

Although the intent of the hardware gate implementation specifies a protocol where the locking processor must unlock the gate, it is recognized that system operation may require a reset function to re-initialize the state of any gate(s) without requiring a system-level reset.

To support this special gate reset requirement, the SEMA4 unit implements a secure reset mechanism which allows a hardware gate (or all the gates) to be initialized by following a

specific dual-write access pattern. Using a technique similar to that required for the servicing of a software watchdog timer, the secure gate reset requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the specified gate(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTGT memory location. The most significant byte (SEMA4_RSTGT[RSTGDP]) must be 0xE2; the least significant byte is a “don’t care” for this reference.
2. The same processor then performs a second 16-bit write to the SEMA4_RSTGT location. For this write, the upper byte (SEMA4_RSTGT[RSTGDP]) is the logical complement of the first data pattern (0x1D) and the lower byte (SEMA4_RSTGT[RSTGTN]) specifies the gate(s) to be reset. This gate field can specify a single gate be cleared or that all gates are cleared.
3. Reads of the SEMA4_RSTGT location return information on the 2-bit state machine (SEMA4_RSTGT[RSTGSM]) which implements this function, the bus master performing the reset (SEMA4_RSTGT[RSTGMS]) and the gate number(s) last cleared (SEMA4_RSTGT[RSTGTN]). Reads of the SEMA4_RSTGT register do not affect the secure reset finite state machine in any manner.

Offset: 0x0100 (SEMA4_RSTGT) Access: User read/write

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

Figure 831. Semaphores (secure) reset gate *n* (SEMA4_RSTGT)

Table 710. 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_0</td><td>0</td></tr><tr><td>e200z4d_1</td><td>1</td></tr><tr><td>eDMA</td><td>2</td></tr><tr><td>FlexRay</td><td>3</td></tr><tr><td>—</td><td>4</td></tr><tr><td>—</td><td>5</td></tr><tr><td>—</td><td>6</td></tr><tr><td>—</td><td>7</td></tr></table></td></tr>
<tr><td>RSTGTN</td><td>Reset Gate Number. This 8-bit field specifies the specific hardware gate to be reset. This field is updated by the second write.
If RSTGTN < 64, then reset the single gate defined by RSTGTN, else reset all the gates. The corresponding secure IRQ notification state machine(s) are also reset.</td></tr>
<tr><td>RSTGDP</td><td>Reset Gate Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the gate reset mechanism. For the first write, RSTGDP = 0xe2 while the second write requires RSTGDP = 0x1d.</td></tr>
</table>

44.3.5 Semaphores (secure) Reset IRQ notification (SEMA4_RSTNTF)

As with the case of the secure reset function and the hardware gates, it is recognized that system operation may require a reset function to re-initialize the state of the IRQ notification logic without requiring a system-level reset.

To support this special notification reset requirement, the SEMA4 unit implements a secure reset mechanism which allows an IRQ notification (or all the notifications) to be initialized by following a specific dual-write access pattern. When successful, the specified IRQ notification state machine(s) are reset. Using a technique similar to that required for the servicing of a software watchdog timer, the secure reset mechanism requires two

consecutive writes with predefined data patterns from the same processor to force the clearing of the IRQ notification(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTNTF memory location. The most significant byte (SEMA4_RSTNTF[RSTNDP]) must be 0x47; the least significant byte is a “don’t care” for this reference.
2. The same processor performs a second 16-bit write to the SEMA4_RSTNTF location. For this write, the upper byte (SEMA4_RSTNTF[RSTNDP]) is the logical complement of the first data pattern (0xb8) and the lower byte (SEMA4_RSTNTF[RSTNTN]) specifies the notification(s) to be reset. This field can specify a single notification be cleared or that all notifications are cleared.
3. Reads of the SEMA4_RSTNTF location return information on the 2-bit state machine (SEMA4_RSTNTF[RSTNSM]) that implements this function, the bus master performing the reset (SEMA4_RSTNTF[RSTNMS]) and the notification number(s) last cleared (SEMA4_RSTNTF[RSTNTN]). Reads of the SEMA4_RSTNTF register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0104 Access: User read/write

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

Figure 832. Semaphores (secure) Reset IRQ notification (SEMA4_RSTNTF)

Table 711. SEMA4_RSTNTF field descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>RSTNSM</td><td>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.</td></tr>
<tr><td>RSTNMS</td><td>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.
<table><tr><th>Master</th><th>Master ID</th></tr><tr><td>e200z4d_0</td><td>0</td></tr><tr><td>e200z4d_1</td><td>1</td></tr><tr><td>eDMA</td><td>2</td></tr><tr><td>FlexRay</td><td>3</td></tr><tr><td>—</td><td>4</td></tr><tr><td>—</td><td>5</td></tr><tr><td>—</td><td>6</td></tr><tr><td>—</td><td>7</td></tr></table></td></tr>
<tr><td>RSTNTN</td><td>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.</td></tr>
<tr><td>RSTNDP</td><td>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.</td></tr>
</table>

44.4 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.

44.4.1 Semaphore usage

Example 1: Inter-processor communication done with software interrupts and semaphores...

- The e200z4d_1 uses software interrupts to tell the e200z4d_0 that new data is available, or the e200z4d_0 does the same to tell the e200z4d_1 that there is new data available for transmission.
- Because only eight software interrupts are available, the user may need RAM locations or general-purpose registers in the SIU to refine the meaning of the software interrupt.
- Messages are passed between cores in a defined section of system RAM.
- Before a core updates a message, it must check the associated semaphore to see if the other core is in the process of updating the same message. If the RAM not being updated, then the semaphore must first be locked, then the message can be updated. A software interrupt can be sent to the other core and the semaphore can be unlocked. If the RAM is being updated, the CPU must wait for the other core to unlock the semaphore before proceeding with update.
- Using the same memory location for bidirectional communication might be difficult, so two one-way message areas might work better.
  - For example, if both cores want to update the same location, then the following sequence may occur.

    1. The e200z4d_1 locks the semaphore, updates the memory, unlocks the semaphore, and generates a software interrupt to the e200z4d_0.

    2. Before the e200z4d_0 takes the software interrupt request, it finds the semaphore to be unlocked, so it writes new data to the memory.

    3. The e200z4d_0 software interrupt ISR reads the data sent to the e200z4d_1, not the data sent from the e200z4d_1, and performs an incorrect operation.
  - Semaphores do not prevent this situation from occurring.

Example 2: Coherent read done with semaphores...

- The e200z4d_0 wants to coherently read a section of shared memory.
- The e200z4d_0 should check that the semaphore for the shared memory is not currently set.
- The e200z4d_0 should set the semaphore for the shared memory to prevent the e200z4d_1 from updating the shared memory.
- The e200z4d_0 reads the required data, then unlock the semaphore.

44.5 Initialization information

The reset state of the SEMA4 unit allows it to begin operation without the need for any further initialization. All the internal state machines are cleared by any reset event, allowing the unit to immediately begin operation.

44.6 Application information

In an operational multi-core system, most interactions involving the SEMA4 unit involves reads and writes to the SEMA4_GATE*n* registers for implementation of the hardware-

enforced software gate functions. Typical code segments for gate functions perform the following operations:

- To lock (close) a gate
  - The processor performs a byte write of logical_processor_number + 1 to gate[i]
  - The processor reads back gate[i] and checks for a value of logical_processor_number + 1

    If the compare indicates the expected value

    then the gate is locked; proceed with the protected code segment

    else

    lock operation failed;

    repeat process beginning with byte write to gate[i] in spin-wait loop, or

    proceed with another execution path and wait for failed lock interrupt notification

A simple C-language example of a gatelock function is shown in *Example 9*. This function follows the Hennessy/Patterson example.

Example 9 Sample Gatelock Function

```
#define UNLOCK      0
#define CP0_LOCK    1
#define CP1_LOCK    2

void gateLock (n)
int   n;    /* gate number to lock */
{
  int   i;
  int   current_value;
  int   locked_value;

  i = processor_number();  /* obtain logical CPU number */

  if (i == 0)
    locked_value = CP0_LOCK;
  else
    locked_value = CP1_LOCK;

  /* read the current value of the gate and wait until the state == UNLOCK
*/
  do {
    current_value = gate[n];
  } while (current_value != UNLOCK);

  /* the current value of the gate == UNLOCK. attempt to lock the gate for
this
     processor. spin-wait in this loop until gate ownership is obtained */
  do {
    gate[n] = locked_value;   /* write gate with processor_number + 1 */
```

```
    current_value = gate[n];  /* read gate to verify ownership was obtained
*/
  } while (current_value != locked_value);
}
```

- To unlock (open) a gate
  - After completing the protected code segment, the locking processor performs a byte write of zeroes to gate[i], unlocking (opening) the gate

In this example, a reference to `processor_number()` is used to retrieve this hardware configuration value. Typically, the logical processor numbers are defined by a hardwired input vector to the individual cores. For PowerPC cores, there is a processor ID register (PIR) which is SPR 286 and contains this value. A single instruction can be used to move the contents of the PIR into a general-purpose register: mfspr rx,286 where rx is the destination GPRn. Other architectures may support a specific instruction to move the contents of the logical processor number into a general-purpose register, e.g., **`rdcpn rx`** for a read CPU number instruction.

If the optional failed lock IRQ notification mechanisms are used, then accesses to the related registers (SEMA4_CP*n*INE, SEMA4_ CP*n*NTF) are required. There is no required negation of the failed lock write notification interrupt as the request is automatically negated by the SEMA4 unit once the gate has been successfully locked by the failing processor.

Finally, in the event a system state requires a software-controlled reset of a gate or IRQ notification register(s), accesses to the secure reset control registers (SEMA4_RSTGT, SEMA4_RSTNTF) are required. For these situations, it is recommended that the appropriate IRQ notification enable(s) (SEMA4_CP*n*INE) bits be disabled before initiating the secure reset 2-write sequence to avoid any race conditions involving spurious notification interrupt requests.

44.7 DMA requests

There are no DMA requests associated with the SEMA4 unit.

45 Sine Wave Generator (SWG)

45.1 Introduction

The Sine Wave Generator (SWG) generates a high-quality sinusoidal voltage signal. It can be programmed with the desired oscillation frequency and amplitude voltage. A wide frequency range (1–50 kHz in 16 Hz steps) is easily programmable through a simple register interface. The linearity/noise performances are carefully optimized through digital processing.

Figure 833 shows a block diagram of the SWG.

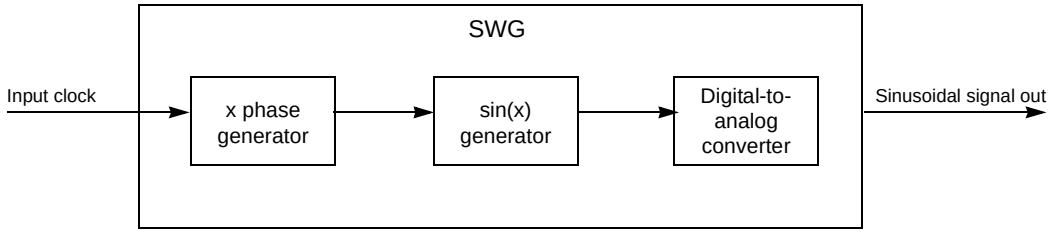


Figure 833. SWG block diagram

45.2 Features

- Input clock frequency range: 12–20 MHz
- Output sinusoidal signal:
  - Frequency range: 1–50 kHz
  - Peak-to-peak amplitude: 0.426–2.063 V

45.3 Memory map and register description

The memory map of the SWG is shown in *Table 712*. The address of each register is given as an offset to the SWG base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

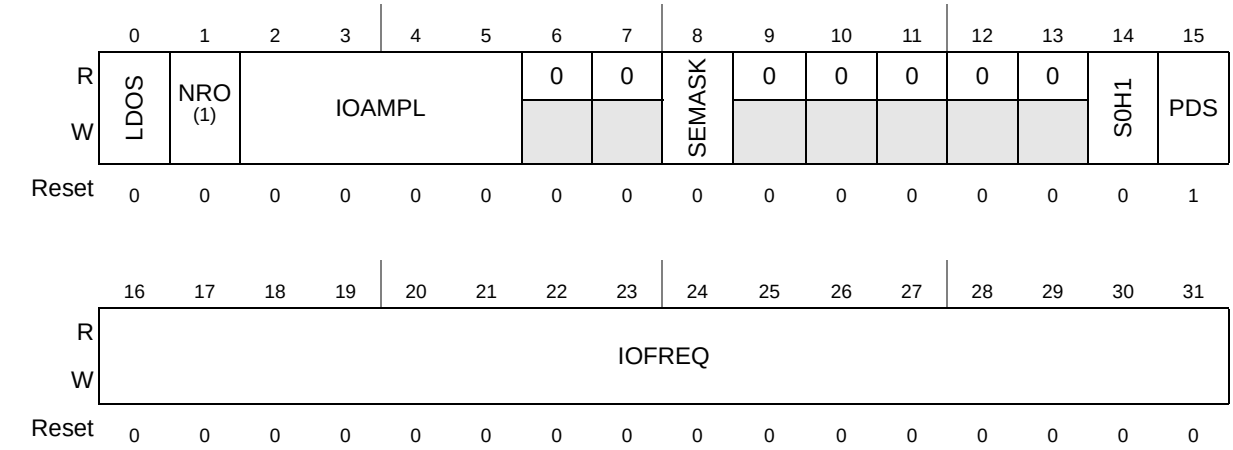
Table 712. SWG memory map

Address offset	Register	Location
0x0	SWG control register (SWG_CTRL)	*on page 1336*
0x4	SWG status register (SWG_STAT)	*on page 1338*

45.3.1 SWG control register (SWG_CTRL)

This register controls the operation of the SWG.

Address: Base + 0x0



1. For cut1, this field must be set to 1 by software before using the SWG. This field is reserved for cut2/3.

Figure 834. SWG control register (SWG_CTRL)

Table 713. SWG_CTRL field descriptions

Field	Description
LDOS	Load sine wave frequency 0 Wait for I/O sine wave frequency 1 Load I/O sine wave frequency *Note: The I/O frequency is loaded by waiting until LDOS = 0 and then setting LDOS = 1.*
NRO	Noise reduction enable 0 Noise reduction feature is enabled (signal-to-noise ratio improved) 1 Noise reduction feature is disabled (signal-to-noise ratio not improved)
IOAMPL	Output sine wave amplitude (see *Table 714*)
SEMASK	SWG error interrupt mask 0 Mask the SWG error interrupt source 1 Enable the SWG error interrupt source
S0H1	Stop mode behavior 0 In STOP mode, the SWG enters Power Down mode only if PDS = 1. 1 In STOP mode, the SWG always enters Power Down mode.
PDS	Enter/exit Power Down mode 0 Force the SWG to exit Power Down mode 1 Force the SWG to enter Power Down mode
IOFREQ	Output sine wave frequency (see *Section 45.4.2, Output sine wave frequency*)

Table 714. Sine wave amplitude (approximate) as a function of SWG_CTRL[IOAMPL]

SWG_CTRL[IOAMPL]	Sine wave amplitude (V)	Amplitude step (V)
0b0000	0.423	—
0b0001	0.487	0.064
0b0010	0.550	0.063

Table 714. Sine wave amplitude (approximate) as a function of SWG_CTRL[IOAMPL]

SWG_CTRL[IOAMPL]	Sine wave amplitude (V)	Amplitude step (V)
0b0011	0.613	0.063
0b0100	0.675	0.062
0b0101	0.737	0.062
0b0110	0.797	0.060
0b0111	0.857	0.060
0b1000	0.987	0.130
0b1001	1.136	0.149
0b1010	1.284	0.148
0b1011	1.430	0.146
0b1100	1.575	0.145
0b1101	1.719	0.144
0b1110	1.861	0.142
0b1111	2.00	0.139

45.3.2 SWG status register (SWG_STAT)

This register indicates whether an error interrupt is pending and allows you to force this interrupt.

Address: Base + 0x4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	SERR	0	0	0	FERR	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	0	0	0	0	0	0	0	0	0	0	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 835. SWG status register (SWG_STAT)

Table 715. SWG_STAT field descriptions

Field	Description
SERR	Error interrupt status bit 0 No error interrupt pending 1 Error interrupt pending *Note: You can clear this bit by writing a '1' to it.*
FERR	Force error interrupt 0 An error interrupt will not be forced 1 An error interrupt will be forced (SERR will be set two clock cycles after FERR is set)

45.4 Functional description

45.4.1 SWG operation after a power-on reset

After a power-on reset, the SWG produces a DC value (no sinusoidal fluctuations).

45.4.2 Output sine wave frequency

The output sine wave frequency *f* is controlled by the SWG_CTRL[IOFREQ] field according to *Equation 53*:

Equation 53 $$f = \left(\frac{\text{InputFrequency}}{1048576}\right) \times \text{IOFREQ}$$

where both frequencies are in Hz and IOFREQ is in decimal format. IOFREQ must be chosen to ensure that *f* remains between 1 and 50 kHz.

45.4.3 Output sine wave amplitude

The amplitude of the output sine wave is controlled by the SWG_CTRL[IOAMPL] field as described in *Table 714*.

45.5 Initialization / application information

45.5.1 Changing the output frequency

To change the output frequency:

1. Determine the required value of SWG_CTRL[IOFREQ] using *Equation 53*.
2. Ensure that SWG_CTRL[LDOS] = 0.
3. Write the new value of the SWG_CTRL[IOFREQ] field.
4. Set SWG_CTRL[LDOS].

The SWG will begin producing the new sine wave frequency two input clock cycles after this procedure is completed.

45.5.2 Preserving the SWG_CTRL data

After you make a change to the SWG_CTRL register, the SWG expects that you will preserve this data for at least 50 clock cycles. Failure to do so may cause unexpected output from the SWG.

46 Software Watchdog Timer (SWT)

46.1 Introduction

46.1.1 Overview

The Software Watchdog Timer (SWT) is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. When enabled, the SWT requires periodic execution of a watchdog servicing 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.

46.1.2 Features

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- Fixed selection of IRCOSC clock for timer operation
- 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

On this device, the SWT is always driven by the IRCOSC.

46.1.3 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.

46.2 External Signal Description

The SWT module does not have any external interface signals.

46.3 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.

46.3.1 Memory Map

The SWT memory map is shown in *Table 716*. The reset values of SWT_CR, SWT_TO and SWT_WN are device specific. These values are determined by SWT inputs.

Table 716. SWT Memory Map

Address Offset	Register Name	Register Description	Size (bits)	Access
0x0000	SWT_CR	SWT Control Register	32	R/W
0x0004	SWT_IR	SWT Interrupt Register	32	R/W
0x0008	SWT_TO	SWT Time-out Register	32	R/W
0x000C	SWT_WN	SWT Window Register	32	R/W
0x0010	SWT_SR	SWT Service Register	32	R/W
0x0014	SWT_CO	SWT Counter Output Register	32	R
0x0018	SWT_SK	SWT Service Key Register	32	R/W
0x001C-0x3FFF	-	Reserved	-	-

46.3.2 Register Descriptions

The following sections detail the individual registers within the SWT programming model.

46.3.2.1 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.

Figure 836. SWT Control Register (SWT_CR)

Offset 0x0000 Access: Read/Write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	KEY	RIA	WND	ITR	HLK	SLK	1	STP	FRZ	WEN
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	0

Table 717. SWT_CR Field Descriptions

Field	Description
MAPn	Master Access Protection for Master n. The platform bus master assignments are device specific. 0 = Access for the master is not enabled 1 = Access for the master is enabled *Note: The Master n refers to the **Logical Master ID**, for details please refer, Table 125, Logical master IDs.* *Master can have Logical Master ID higher than 0x7 and SWT only looks at the 3 LSB of this ID.*
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

Field	Description
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

46.3.2.2 SWT Interrupt Register (SWT_IR)

The SWT_IR contains the time-out interrupt flag.

Figure 837. SWT Interrupt Register (SWT_IR)

Offset 0x0004 Access: Read/Write

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

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

Table 718. 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.

46.3.2.3 SWT Time-Out Register (SWT_TO)

The SWT Time-Out (SWT_TO) register contains the 32-bit time-out period. The reset value for this register is device specific. This register is read only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

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 W	WTO																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0	0	0	0

Figure 838. SWT Time-Out Register (SWT_TO)

Table 719. 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.

46.3.2.4 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 W	WST																															
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 839. SWT Window Register (SWT_WN)

Table 720. 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.

46.3.2.5 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

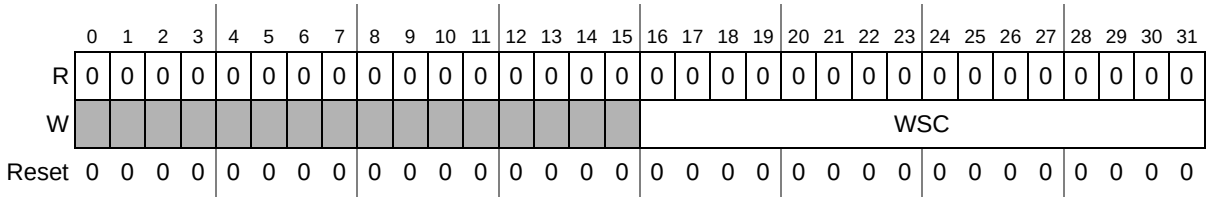


Figure 840. SWT Service Register (SWT_SR)

Table 721. 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 (46.4) 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.

46.3.2.6 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.

Caution: Do not access the SWT_CO register in Lock Step Mode (LSM).

Offset 0x014

Access: Read Only

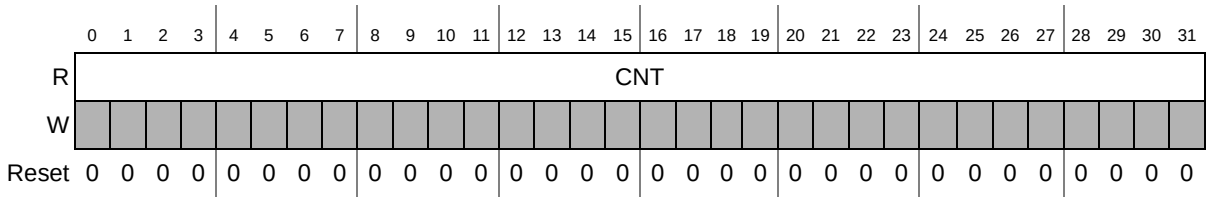


Figure 841. SWT Counter Output Register (SWT_CO)

Table 722. 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.

46.3.2.7 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 842. SWT Service Key Register (SWT_SK)

Table 723. SWT_SK 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}.

46.4 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 down counter is always driven by the IRCOSC clock . 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 843*. 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 843. 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.

47 Static RAM (SRAM)

47.1 Introduction

The SRAM provides the following features:

- 128 KB of general-purpose static RAM
- SRAM can be read/written from any bus master
- Byte, halfword, word and doubleword addressable
- Single-bit correction and double-bit error detection
- Ability to optimize number of SRAM wait-states to system clock frequency

47.2 SRAM operating mode

The SRAM has only one operating mode.There is no stand-by mode available.

Table 724. SRAM operating modes

Mode	Configuration
Normal (functional)	Allows reads and writes of SRAM.

47.3 Registers

The internal SRAM has no registers. Registers for the SRAM ECC are located in the ECSM.

47.4 SRAM ECC mechanism

The SRAM ECC detects the following conditions and produces the following results:

- Detects and corrects all 1-bit errors
- Detects and flags all 2-bit errors as non-correctable errors
- Detects 39-bit reads (32-bit data bus plus the 7-bit ECC) that return all zeros or all ones, asserts an error indicator on the bus cycle, and sets the error flag

SRAM does not detect all errors greater than 2 bits.

Internal SRAM write operations are performed on the following byte boundaries:

- 1 byte (0:7 bits)
- 2 bytes (0:15 bits)
- 4 bytes or 1 word (0:31 bits)

If the entire 32 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 32-bit data bus. The 8-bit ECC is appended to the data segment and written to SRAM.

If the write operation is less than the entire 32-bit data width (1-, or 2-byte segment), the following occurs:

1. The ECC mechanism checks the entire 32-bit data bus for errors, detecting and either correcting or flagging errors.
2. The write data bytes (1-, or 2-byte segment) are merged with the corrected 32 bits on the data bus.
3. The ECC is then calculated on the resulting 32 bits formed in the previous step.
4. The 7-bit ECC result is appended to the 32 bits from the data bus, and the 39-bit value is then written to SRAM.

47.4.1 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. *Table 725* 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 operationLists the type of SRAM operation executing currently

Previous operationLists the valid types of SRAM operations that can precede the current SRAM operation (valid operation during the preceding clock)

Wait statesLists the number of wait states (bus clocks) the operation requires which depends on the combination of the current and previous operation

Table 725. Number of Wait States Required for SRAM Operations

	Current Operation	Previous Operation	Number of Wait States Required
Read Operation	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 Operation	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	

47.4.2 Reset effects on SRAM accesses on cut1

On cut1, asynchronous reset will possibly corrupt RAM if it asserts during a read or write operation to SRAM. The completion of that access depends on the cycle at which the reset occurs. 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 RAM corruption does not happen.

Instead synchronous reset (SW reset) should be used in controlled function (without RAM accesses) in case initialization procedure is needed without RAM initialization.

47.5 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 read/write (R/W) operation. Because the ECC bits can contain random data after the device is powered on, the SRAM must be initialized by executing 32-bit write operations prior any read accesses. This is also true for implicit read accesses caused by any write accesses of less than 32-bit as discussed in "ECC Mechanism".

47.6 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. Refer to *Section 47.4, SRAM ECC mechanism*.

Note: You must initialize SRAM, even if the application does not use ECC reporting.

Note: SPC56ELxx defaults to one SRAM wait state. Low frequencies do not require a wait state. To set the number of wait-states, see Section 22.4.2.7, Miscellaneous User-Defined Control Register (MUDCR).

48 System Integration Unit Lite (SIUL)

48.1 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.

48.2 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 844* is a block diagram of the SIUL and its interfaces to other system components.

The module provides dedicated general-purpose pads that can be configured as either inputs or outputs. When configured as an output, you can write to an internal register to control the state driven on the associated output pad. When configured as an input, you can detect the state of the associated pad by reading the value from an internal register. When configured as an input and output, the pad value can be read back, which can be used a method of checking if the written value appeared on the pad.

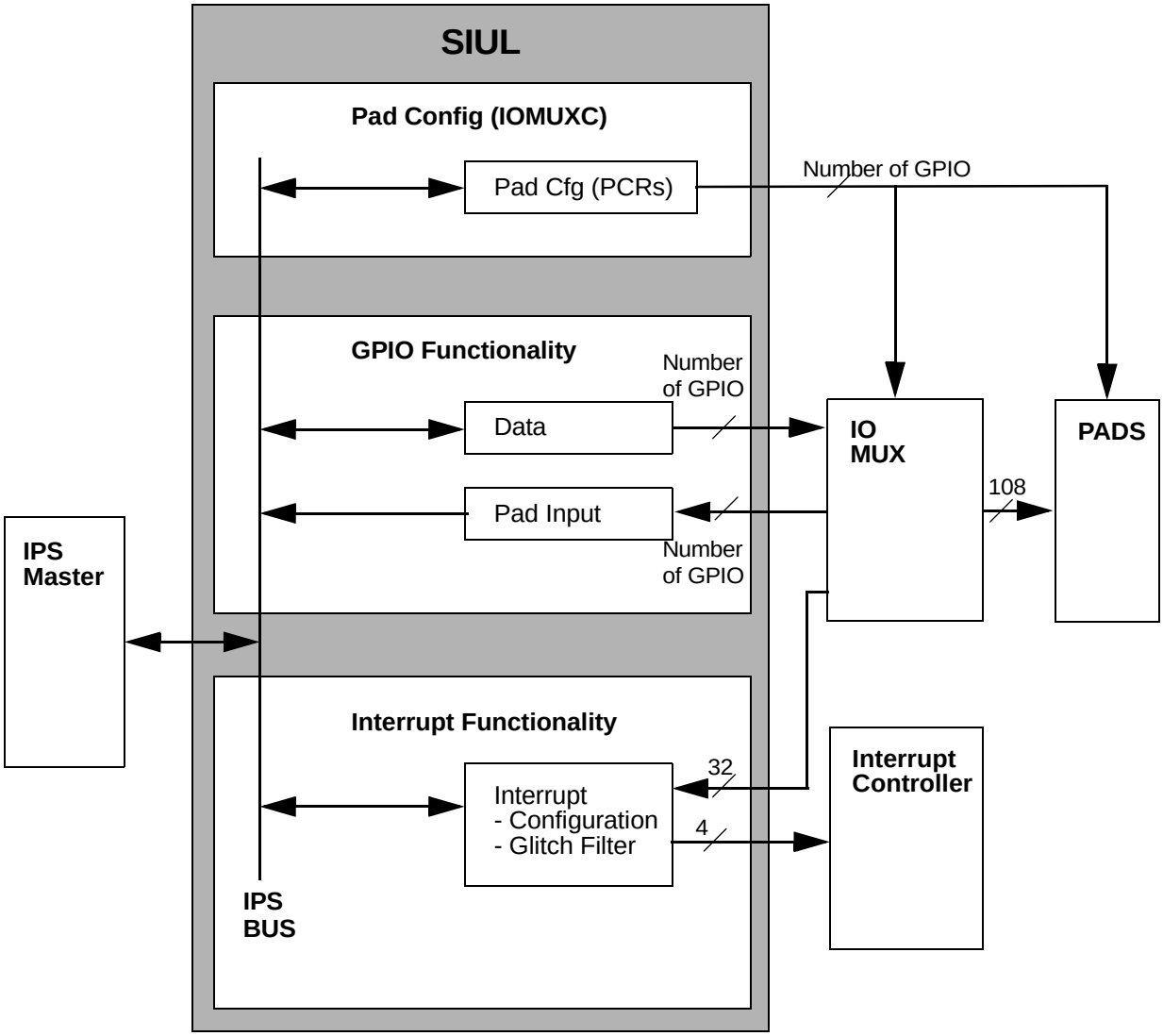


Figure 844. SIUL block diagram

48.3 Features

The System Integration Unit Lite supports these distinctive features:

- GPIO
  - 121 pins which are user-configurable inputs and/or outputs
  - 22 pins have user-configurable General Purpose Input (GPI) functionality
  - 99 pins have user-configurable General Purpose Input/Ouput (GPIO) functionality
  - Dedicated input and output registers for each GPIO pin
- External interrupts
  - 4 system interrupt vectors for 32 interrupt sources
  - 32 programmable digital glitch filters
  - Independent interrupt mask
  - Edge detection
- System configuration
  - Pad configuration control

48.3.1 Register protection

The individual registers of System Integration Unit Lite are protected from accidental writes, see *Chapter 41: Register Protection (REG_PROT)*.

48.4 External signal description

The pad configuration allows flexible, centralized control of the pin electrical characteristics of the MCU with the GPIO control providing centralized general purpose I/O for an MCU that multiplexes GPIO with other signals at the I/O pads. These other signals, or alternate functions, will normally be the peripherals functions. The internal multiplexing allows user selection of the input to chip-level signal multiplexors. Each GPIO port communicates via 16 I/O channels. In order to use the pad as a GPIO, the corresponding Pad Configuration Registers (PCR) for all pads used in the port must be configured as GPIO rather than as the alternate pad function.

Table 726 lists the external pins used by the SIUL.

Table 726. SIUL signal properties

Name	I/O Type	Function
System Configuration		
GPIO	I/O	General-Purpose I/O
External Interrupt		
EIRQ[0:8, 10:18, 22, 30:31, 35:38, 77:81, 96:97]	Input	External Interrupt Request Input

48.4.1 Detailed signal descriptions

48.4.1.1 General-purpose I/O pins

The GPIO pins provide general-purpose input and output function. The GPIO pins are generally multiplexed with other I/O pin functions. Each GPIO input and output is separately controlled by an input (GPDI*n_n*) or output (GPDO*n_n*) register. See *Section 48.5.2.10, GPIO Pad Data Output Registers (GPDO)* and *Section 48.5.2.11, GPIO Pad Data Input Registers (GPDI)*.

48.4.1.2 External interrupt request input pins (EIRQ[0:31])

The EIRQ[0:31] are connected to the SIU inputs. Rising or falling edge events are enabled by setting the corresponding bits in the SIU_IREER or the SIU_IFEER register. See *Section 48.5.2.5, Interrupt Rising-Edge Event Enable Register (IREER)* and *Section 48.5.2.6, Interrupt Falling-Edge Event Enable Register (IFEER)*.

48.5 Memory map and register description

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

48.5.1 SIUL memory map

Table 727 gives an overview on the SIUL registers implemented.

Table 727. SIUL memory map

Address offset	Register name	Description	Location
0x0004	MIDR1	MCU ID Register #1	*on page 1357*
0x0008	MIDR2	MCU ID Register #2	*on page 1358*
(0x000C–0x0013)	—	Reserved	
0x0014	ISR	Interrupt Status Flag Register	*on page 1359*
0x0018	IRER	Interrupt Request Enable Register	*on page 1359*
(0x001C–0x0027)	—	Reserved	
0x0028	IREER	Interrupt Rising Edge Event Enable	*on page 1360*
0x002C	IFEER	Interrupt Falling-Edge Event Enable	*on page 1360*
0x0030	IFER	IFER Interrupt Filter Enable Register	*on page 1361*
(0x0034–0x003F)	—	Reserved	
0x0040–0x0148	PCR0–PCR*n*	Pad Configuration Registers	*on page 1361*
(0x0150–0x04FF)	—	Reserved	
0x0500–0x0528	PSMI0_3–PSMI40_43	Pad Selection for Multiplexed Inputs	*on page 1364*
(0x052A–0x05FF)	—	Reserved	
0x0600–0x0668	GPDO(1)	GPIO Pad Data Output Register	*on page 1365*
(0x066C–0x07FF)	—	Reserved	

Table 727. SIUL memory map (continued)

Address offset	Register name	Description	Location
0x0800–0x0868	GPDI[2]	GPIO Pad Data Input Register	*on page 1366*
(0x086C–0x0BFF)	—	Reserved	
0x0C00–0x0C0C	PGPDO0–PGPDO3	Parallel GPIO Pad Data Out Register	*on page 1367*
(0x0C10–0x0C3F)	—	Reserved	
0x0C40–0x0C4C	PGPDI0–PGPDI3	Parallel GPIO Pad Data In Register	*on page 1368*
(0x0C50–0x0C7F)	—	Reserved	
0x0C80–0x0C98	MPGPDO0–MPGPDO6	Masked Parallel GPIO Pad Data Out Register	*on page 1369*
(0x0C9C–0x0FFF)	—	Reserved	
0x1000–0x107C	IFMC0–IFMC31	Interrupt Filter Maximum Counter Register	*on page 1370*
0x1080	IFCP	Interrupt Filter Clock Prescaler Register	*on page 1371*
(0x1084–0x3FFF)	—	Reserved	

1. Check the pin-muxing table (*Section 4.4, Pin muxing*) if GPO functionality is available.
2. Check the pin-muxing table (*Section 4.4, Pin muxing*) if GPI functionality is available.

Note: A transfer error will be issued when trying to access completely reserved register space.

48.5.2 Register description

This section describes in address order all the SIUL registers. Each description includes a standard register diagram. Details of register bit and field function follow the register diagrams, in bit order. The numbering convention of register is MSB=0; however the numbering of internal field is LSB=0, for example, PARTNUM[5] = MIDR1[10].

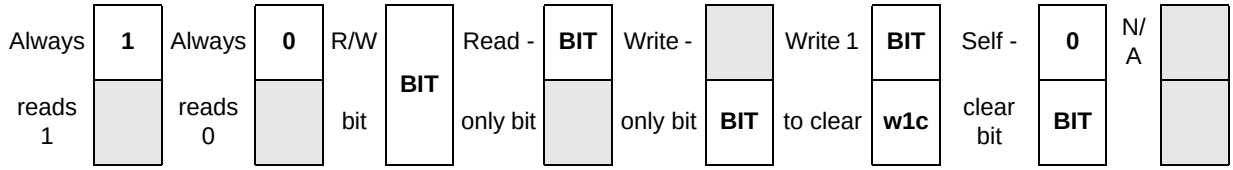


Figure 845. Key to Register Fields

48.5.2.1 MCU ID Register 1 (MIDR1)

This register contains information that identifies the device.

Figure 846. MCU ID Register 1 (MIDR1)

Address: Base + 0x0004 Access: Read only

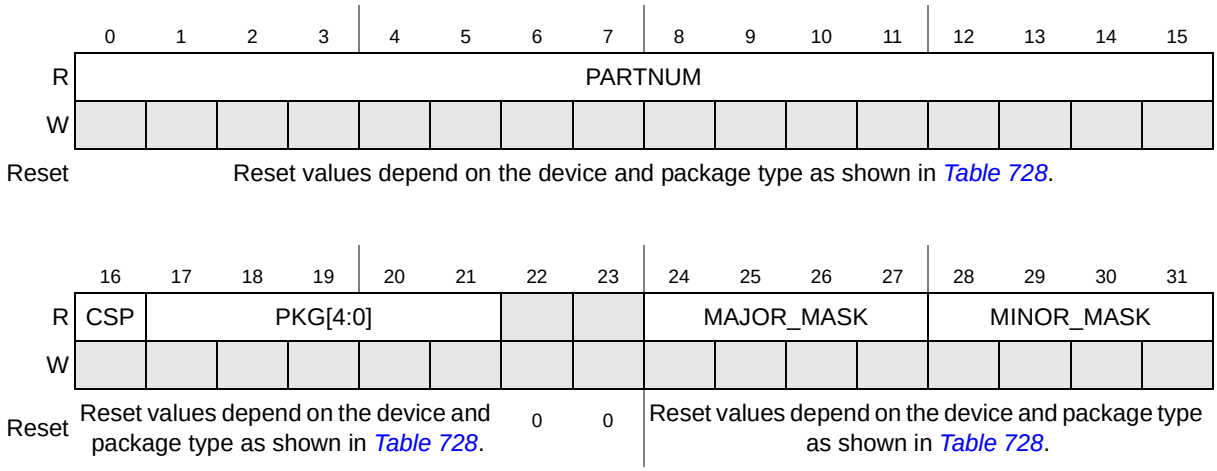


Table 728. MIDR1 Field Descriptions

Field	Description
PARTNUM	MCU Part Number Read-only, device part number of the MCU. 0101_0110_0100_0011: Device with 1 MB flash memory For the full part number this field needs to be combined with MIDR2.PARTNUM[23:16]
CSP	Always reads back 0
PKG	Package Settings Can by read by software to determine the package type that is used for the particular device: 0b01001: 100-pin QFP 0b01101: 144-pin QFP 0b01000: 257-pin BGA
MAJOR_MASK	Major Mask Revision cut1: 0x0 cut2: 0x1 cut3: 0x1 cut3.1: 0x1 cut3.2: 0x1
MINOR_MASK	Minor Mask Revision cut1: 0x0 cut2: 0x0 cut3: 0x1 cut3.1: 0x2 cut3.2: 0x3

48.5.2.2 MCU ID Register 2 (MIDR2)

Figure 847. MCU ID Register 2 (MIDR2)

Address: Base + 0x0008 Access: Read only

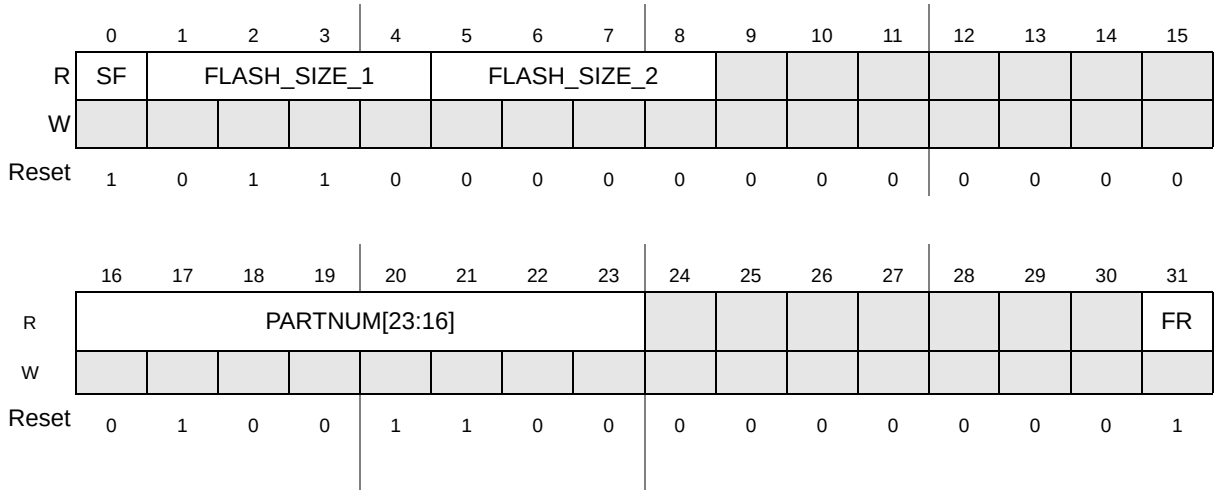


Table 729. MIDR2 field descriptions

Field	Description
SF	Manufacturer 1: ST
FLASH_SIZE_1	Coarse granularity for flash memory size Combine with FLASH_SIZE_2 to calculate the actual memory size. 0b0110: 1 MB
FLASH_SIZE_2	Fine granularity for Flash memory size Combine with FLASH_SIZE_1 to calculate the actual memory size. 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 0x4C: L (SPC56ELxx) family
FR	FlexRay present 1: FlexRay is present 0: FlexRay is not present

48.5.2.3 Interrupt Status Flag Register (ISR)

This register holds the interrupt flags.

Figure 848. Interrupt Status Flag Register (ISR)

Address: Base + 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	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 730. 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

48.5.2.4 Interrupt Request Enable Register (IRER)

This register is used to enable the interrupt messaging to the interrupt controller.

Figure 849. Interrupt Request Enable Register (IRER)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EIRE[31:0]																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 731. IRER Field Descriptions

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

48.5.2.5 Interrupt Rising-Edge Event Enable Register (IREER)

This register enables rising-edge triggered events on the corresponding interrupt pads.

Figure 850. Interrupt Rising-Edge Event Enable Register (IREER)

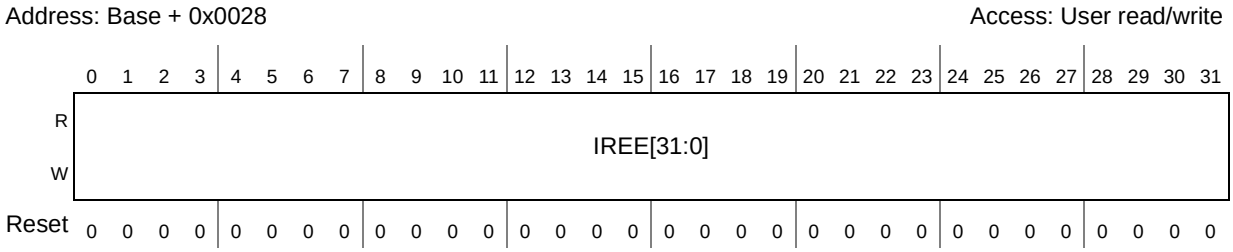


Table 732. IREER Field Descriptions

Field	Description
IREE[x]	Enable rising-edge events to cause the EIF[x] bit to be set. 1: Rising-edge event is enabled 0: Rising-edge event is disabled

48.5.2.6 Interrupt Falling-Edge Event Enable Register (IFEER)

This register enables falling-edge triggered events on the corresponding interrupt pads.

Figure 851. Interrupt Falling-Edge Event Enable Register (IFEER)

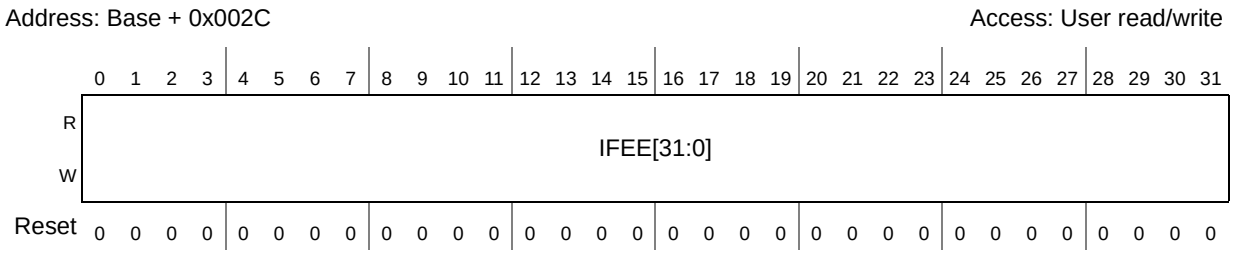


Table 733. IFEER Field Descriptions

Field	Description
IFEE[x]	Enable falling-edge events to cause the EIF[x] bit to be set. 1: Falling-edge event is enabled 0: Falling-edge event is disabled

Note: If both the IREE and IFEE bits are cleared for the same interrupt source, the interrupt status flag for the corresponding external interrupt will never be set.

48.5.2.7 Interrupt Filter Enable Register (IFER)

This register is used to enable or disable a digital filter counter on the interrupt pads to filter out glitches on the inputs.

Figure 852. Interrupt Filter Enable Register (IFER)

Address: Base + 0x0030 Access: User read/write

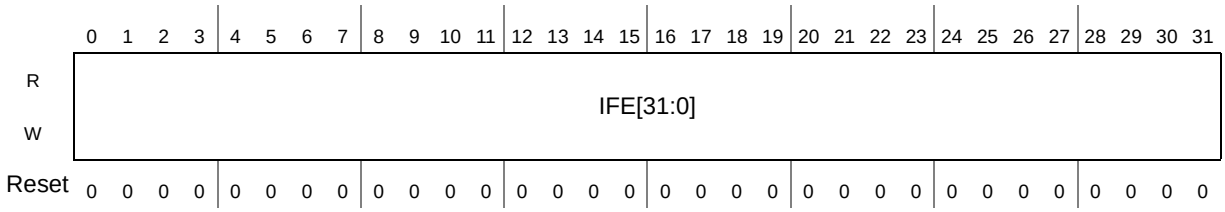


Table 734. IFER Field Descriptions

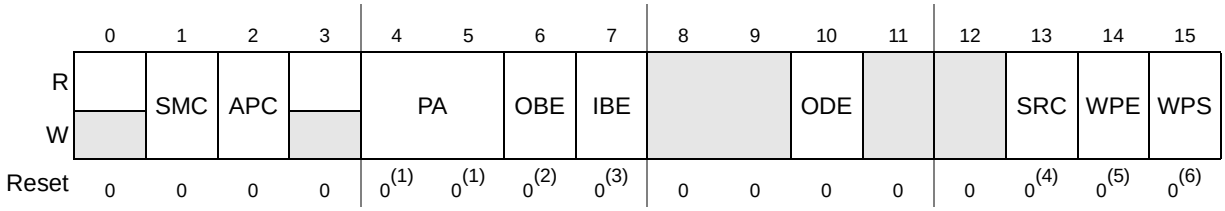
Field	Description
IFE[x]	Enable digital glitch filter on the interrupt pad input. 1: Filter is enabled 0: Filter is disabled

48.5.2.8 Pad Configuration Registers (PCR0–PCR132)

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. See *Table 12* for the mapping of PCR to pads.

Figure 853. Pad Configuration Registers (PCR0)

Address: Base + 0x0040 (PCR0)(122 registers)
Base + 0x0042 (PCR1)
...
Base + 0x0148 (PCR132)
Access: User read/write



1. The reset value of the PA- bits of the Pad Configuration Registers PCR20 is '01' in distinction from the remaining PCR registers
2. The reset value of the OBE- bit of the Pad Configuration Registers PCR20 is '1' in distinction from the remaining PCR registers
3. The reset value of the IBE- bit of the Pad Configuration Registers PCR2, 3, 4, 21 is '1' in distinction from the remaining PCR registers
4. The reset value of the SRC- bit of the Pad Configuration Registers PCR20 is '1' in distinction from the remaining PCR registers
5. The reset value of the WPE- bit of the Pad Configuration Registers PCR2, 3, 4, 21 is '1' in distinction from the

remaining PCR registers

6. The reset value of the WPS- bit of the Pad Configuration Registers PCR21 is '1' in distinction from the remaining PCR registers

Note: 16- and 32-bit accesses are supported.

Table 735. PCR0 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 SoC. 1: In SoC SAFE mode, the output buffer remains functional. 0: In SoC SAFE mode, the output buffer of the pad is disabled.
APC	Analog Pad Control This bit enables the usage of the pad as analog input. 1: Analog input path switch can be enabled by the ADC. 0: Analog input path from the pad is gated and can not be used.
PA[1:0]	Pad Output Assignment This field is used to select the function that is allowed to drive the output of a multiplexed pad. The PA field size can vary from zero to two bits, depending on the number of output functions associated with this pad. 00: Alternative Mode 0: GPIO. 01: Alternative Mode 1 10: Alternative Mode 2 11: Alternative Mode 3 *Note: Number of bit depending of the number of actual alternate function. Please refer to datasheet*
OBE	Output Buffer Enable This bit enables the output buffer of the pad in case the pad is in GPIO mode. 1: Output Buffer of the pad is enabled when PA = 00. 0: Output Buffer of the pad is disabled when PA = 00.
IBE	Input Buffer Enable This bit enables the input buffer of the pad. 1: Input Buffer of the pad is enabled. 0: Input Buffer of the pad is disabled.
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. 1: Open drain enable signal is asserted for the pad. 0: Open drain enable signal is negated for the pad.
SRC	Slew Rate Control SRC = 0 Slowest configuration SRC = 1 Fastest configuration

Table 735. PCR0 Field Descriptions (continued)

Field	Description
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. 1: Weak pull device enable signal is asserted for the pad. 0: Weak pull device enable signal is negated 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. 1: The pull up enabled. 0: The pull down enabled.

48.5.2.9 Pad Selection for Multiplexed Inputs (PSMI0_3–PSMI40_43[q])

These registers define pads as input to peripheral functions.

Figure 854 and *Table 736* present the structure of the PSMI0_3 register. The structure of the other 10 PSMI registers is similar, with numbers modified appropriately.

Figure 854. Pad Selection for Multiplexed Inputs Register (PSMI0_3)

Address: Base + 0x0500 - 0x0528 (11 registers) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PADSEL0				0	0	0	0	PADSEL1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PADSEL2				0	0	0	0	PADSEL3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 736. PSMI0_3 field descriptions

Field	Description
PADSEL0–3	Pad Selection Bits Each PADSEL field selects the pad currently used for a certain input function (see *Table 12*). Example: In Function: LIN0 RXD PSMI32_35 PADSEL0: 00: B[3] 01: B[7]

q. PSMI43 does not exist; the register name includes the number 43 due to the sequential naming convention of these registers.

48.5.2.10 GPIO Pad Data Output Registers (GPDO)

These registers can be used to set or clear a single GPIO pad with a byte access.

Figure 855. Port GPIO Pad Data Output register 0–3 (GPDO0_3)

Address: Base + 0x0600–0x0668 (27 registers) Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDO[2]	0	0	0	0	0	0	0	PDO[3]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 737. 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. 1: Logic high value is driven on the corresponding GPIO pad when the pad is configured as an output 0: Logic low value is driven on the corresponding GPIO pad when the pad is configured as an output

Example 10 Accessing GPDO ports

```
Check if GPIO exists in pin muxing table (Section 4.4, Pin muxing)
Write Pad Data Output Register for A[0] which is GPIO[0].
   32 Bit write to address 0x0600 + 0x0000
   ==> register after write 0x--------_--------_--------_-------<GPIO[0]>


Write Pad Data Output Register for A[4] which is GPIO[4].
   32 Bit write to address 0x0600 + 0x0004
   ==> register after write 0x--------_--------_--------_-------<GPIO[4]>


Write Pad Data Output Register for G[2] which is GPIO[89].
   32 Bit write to address 0x0600 + 0x0016
   ==> register after write 0x--------_--------_-------<GPIO[89]>_--------
     8 Bit write to address  0x0600 + 0x0017
```

```
Address for 32 Bit accesses =  0x0600 + [hex [ [GPIO Number / 4]**  ]
** Returns the integer portion of the division
```

48.5.2.11 GPIO Pad Data Input Registers (GPDI)

These registers can be used to read the GPIO pad data with a byte access.

Figure 856. Port GPIO Pad Data Input register 0–3 (GPDI0_3–GPDI104_107))

Address: Base + 0x0800–0x0868 (27 registers) Access: User read

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDI [2]	0	0	0	0	0	0	0	PDI [3]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 738. GPDO0_3–GPDI104_107 Field Descriptions

Field	Description
PDI[x]	Pad Data In This bit stores the value of the external GPIO pad associated with this register. 1: The value of the data in signal for the corresponding GPIO pad is logic high 0: The value of the data in signal for the corresponding GPIO pad is logic low

Example 11 Accessing GPDI ports

```
Check if GPIO exists in pin muxing table (Section 4.4, Pin muxing)
Read Pad Data Input Register for A[0] which is GPIO[0].
   32 Bit read from address 0x0600 + 0x0000
   ==> return value 0x--------_--------_--------_-------<GPIO[0]>

Write Pad Data Input Register for A[4] which is GPIO[4].
   32 Bit read from address 0x0600 + 0x0004
   ==> return value 0x--------_--------_--------_-------<GPIO[4]>


Write Pad Data Input Register for G[2] which is GPIO[89].
   32 Bit read from address 0x0600 + 0x0016
   ==> return value 0x--------_--------_-------<GPIO[89]>_--------
     8 Bit read from address   0x0600 + 0x0017
```

```
Address for 32 Bit accesses =  0x0800 + [hex [GPIO Number / 4]** ]
** Returns the integer portion of the division
```

48.5.2.12 Parallel GPIO Pad Data Out Register (PGPDO0–PGPDO3)

These registers are used to set or clear the respective pads of the device. See *Section 4.5, Mapping of ports to PGPDO/I registers*.

Figure 857. Parallel GPIO Pad Data Out Register (PGPDO0–PGPDO3)

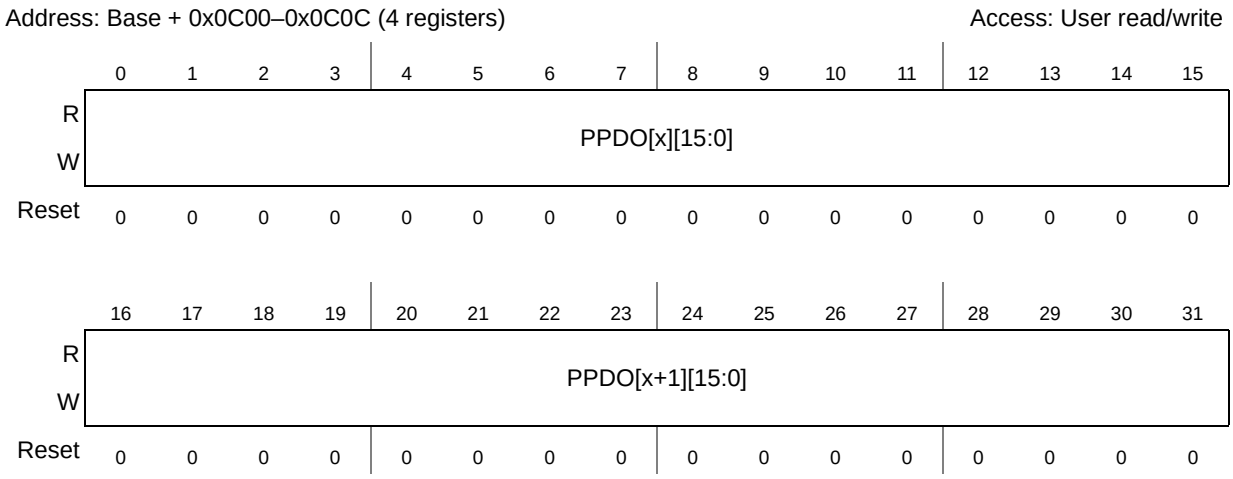


Table 739. PGPDO0–PGPDO3 Field Descriptions

Field	Description
PPDO[x]	Parallel Pad Data Out Write or read the data register that stores the value to be driven on the pad in output mode. Accesses to this register location are coherent with accesses to the bit-wise GPIO Pad Data Output Registers (GPDO). The x and bit index define which PPDO register bit is equivalent to which PDO register bit according to the following equation: PPDO[x][y] = PDO[(x*16)+y]

Note: The PGPDO registers access the same physical resource as the PDO and MPGPDO address locations. Some examples of the mapping:

PPDO[0][0] = PDO[0]

PPDO[2][0] = PDO[32]

48.5.2.13 Parallel GPIO Pad Data In Register (PGPDI0–PGPDI3)

These registers hold the synchronized input value from the pads. See *Section 4.5, Mapping of ports to PGPDO/I registers*.

Figure 858. Parallel GPIO Pad Data In Register (PGPDI0–PGPDI3)

Address: Base + 0x0C40–0x0C4C (4 registers) Access: User read-only

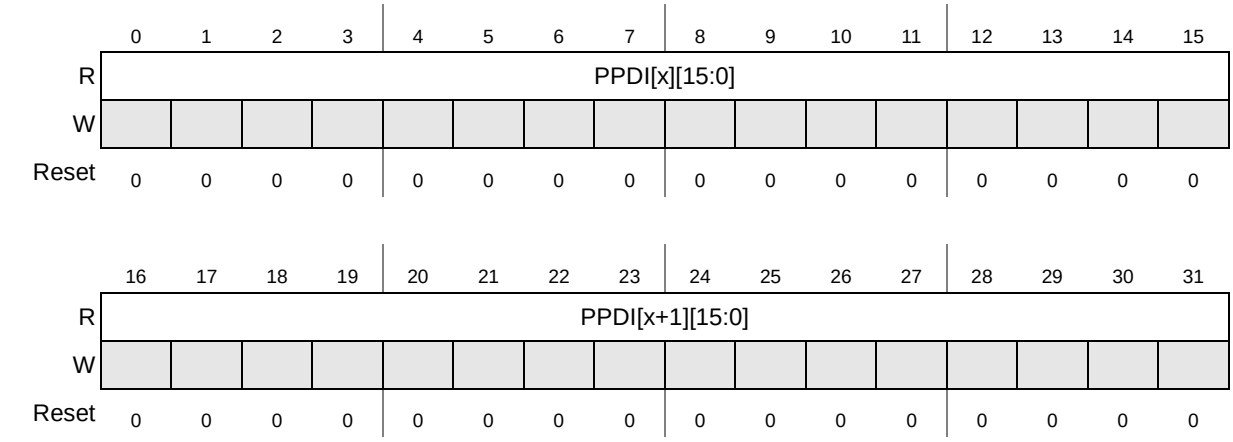


Table 740. PGPDI0_3 Field Descriptions

Field	Description
PPDI[x]	Parallel Pad Data In Read the current pad value. Accesses to this register location are coherent with accesses to the bit-wise GPIO Pad Data Input Registers (GPDI). The x and bit index define which PPDI register bit is equivalent to which PDI register bit according to the following equation: PPDI[x][y] = PDI[(x*16)+y]

48.5.2.14 Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO6)

This register can be used to selectively modify the pad values associated to PPDO[x][15:0]. The MPGPDO[x] register may only be accessed with 32-bit writes. 8-bit or 16-bit writes will not modify any bits in the register and cause a transfer error response by the module. Read accesses will return 0.

Figure 859. Masked Parallel GPIO Pad Data Out Register (MPGPDO0)

Address: Base + 0x0C80–0x0C98 (7 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	MASK[x][15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Table 741. MPGPDO0_6 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. 1: The associated bit value in the MPPDO[x] field is written 0: The associated bit value in the MPPDO[x] field is ignored
MPPDO[x] [15:0]	Masked Parallel Pad Data Out Write the data register that stores the value to be driven on the pad in output mode. Accesses to this register location are coherent with accesses to the bit-wise GPIO Pad Data Output Registers (GPDO). 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]

48.5.2.15 Interrupt Filter Maximum Counter Register (IFMC0–IFMC31)

These registers are used to configure the filter counter associated with each digital glitch filter.

Figure 860. Interrupt Filter Maximum Counter Register (IFMC0–IFMC31)

Address: Base + 0x1000–0x107C (32 registers) Access: User read/write

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

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

Table 742. IFMC0_31 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).

48.5.2.16 Interrupt Filter Clock Prescaler Register (IFCPR)

This register is used to configure a clock prescaler that is used to select the clock for all digital filter counters in the SIUL.

Figure 861. Interrupt Filter Clock Prescaler Register (IFCPR)

Address: Base + 0x1080 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	IFCP[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 743. IFCPR Field Descriptions

Field	Description
IFPC [3:0]	Interrupt Filter Clock Prescaler setting Prescaled Filter Clock Period = T(IRC) x (IFCP + 1) T(IRC) is the internal oscillator period. IFCP can be 0 to 15

48.6 Functional Description

48.6.1 General

This section provides a functional description of the System Integration Unit Lite.

48.6.2 Pad control

The SIUL controls the configuration and electrical characteristic of the device pads. It provides a consistent interface for all pads, both on a by-port and a by-bit basis. The SIUL allows you to configure each pad as either a General Purpose Input Output pad (GPIO) or as one or more alternate functions (input or output). The pad configuration registers (PCR*n*, see *Section 48.5.2.8, Pad Configuration Registers (PCR0–PCR132)*) allow software control

of the static electrical characteristics of external pins with a single write. These PCRs are used to configure the following pad features:

- Open drain output enable
- Input hysteresis enable
- Slew rate control
- Pull control
- Pad assignment
- Control of analog path switches
- Safe mode configuration

48.6.3 General purpose input and output pads (GPIO)

The SIUL allows each pad to be configured as either a General Purpose Input Output pad (GPIO), and as one or more alternate functions (input or output), the function of which is normally determined by the peripheral that will use the pad.

The SIUL manages GPIO pads organized as ports that can be accessed for data reads and writes as 32-bit, 16-bit or 8-bit.

As shown in *Figure 862*, all port accesses are identical with each read or write being performed only at a different location to access a different port width.

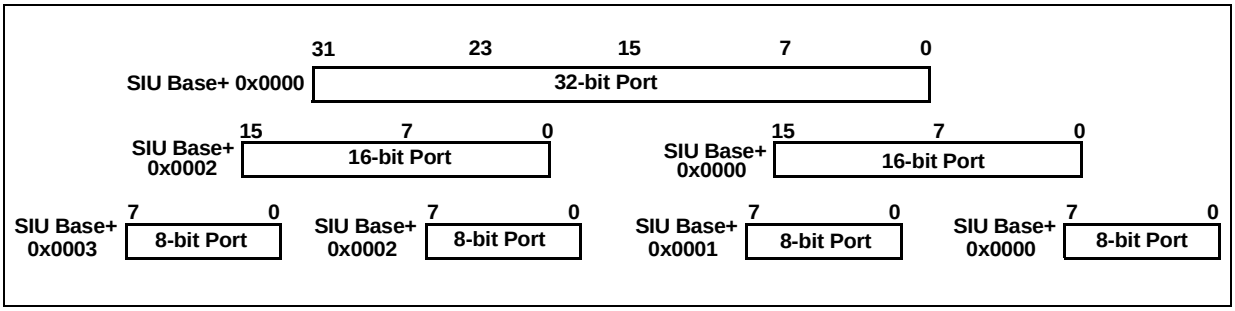


Figure 862. Data port width configurations for different port width accesses

This implementation requires that the registers are arranged in such a way as to support this range of port widths without having to split reads or writes into multiple accesses.

The SIUL has separate data input (GPDI*n_n*, see *Section 48.5.2.11, GPIO Pad Data Input Registers (GPDI)*) and data output (GPDO*n_n*, see *Section 48.5.2.10, GPIO Pad Data Output Registers (GPDO)*) registers for all pads, allowing the possibility of reading back an input or output value of a pad directly. This allows you to validate the pad configuration rather than confirming the value that was written to the data register by accessing the data input registers.

The data output registers support both read and write operations. The data input registers support read access only.

When the pad is configured to use one of its alternate functions, the data input value reflect the respective value of the pad. If a write operation is performed to the data output register for a pad configured as an alternate function (non GPIO), this write will not be reflected by the pad value until reconfigured to GPIO.

The allocation of what input function is connected to the pin is defined by the PSMI registers (PCR*n*, see *Section 48.5.2.8, Pad Configuration Registers (PCR0–PCR132)*).

48.6.4 External interrupts

The SIUL supports 32 external interrupts, EIRQ0-EIRQ31.

The SIUL supports four interrupt vectors to the interrupt controller. Each vector interrupt has eight external interrupts combined together with the presence of flag generating an interrupt for that vector if enabled. All of the external interrupt pads within a single group have equal priority.

Refer to *Figure 863* for an overview of the External Interrupt implementation:

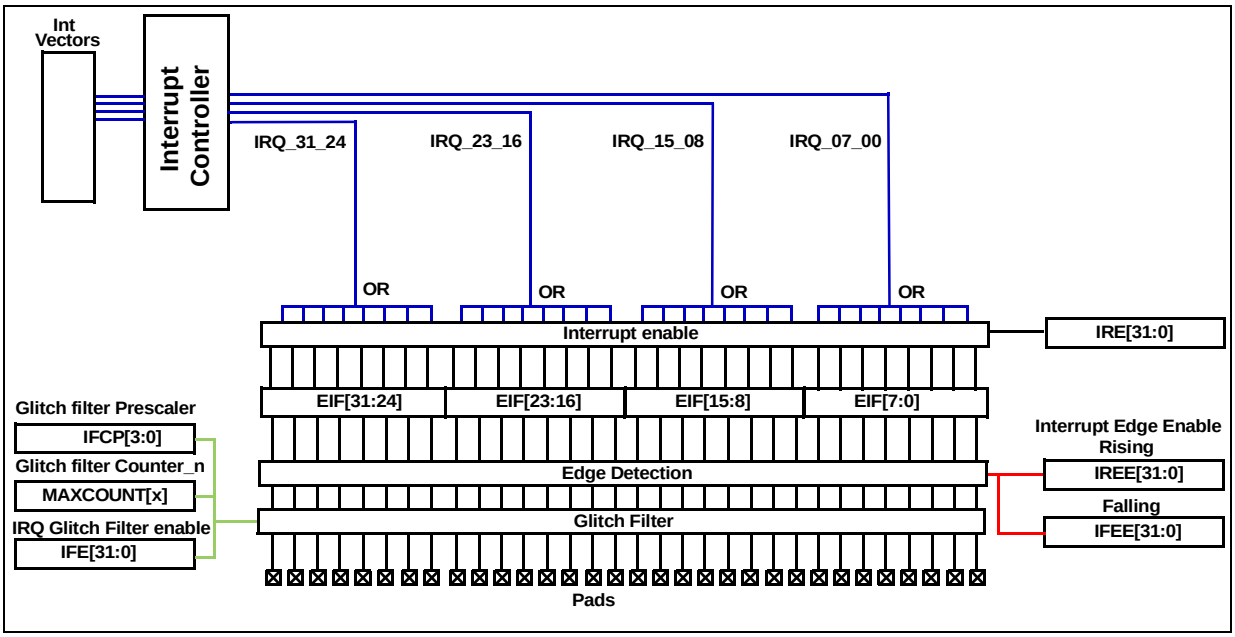


Figure 863. External Interrupt pad diagram

48.6.4.1 External interrupt management

Each interrupt can be enabled or disabled independently. This can be performed using the Interrupt Request Enable Register (IRER - *Section 48.5.2.4, Interrupt Request Enable Register (IRER)*). A pad defined as an external interrupt can be configured to recognize interrupts with an active rising edge, an active falling edge or both edges being active. A setting of having both edge events disabled is reserved and should not be configured.

The active EIRQ edge is controlled through the configuration of the registers IREER and IFEER.

Each external interrupt supports an individual flag which is held in the Flag register (ISR - *Section 48.5.2.3, Interrupt Status Flag Register (ISR)*). This register is a write-1-to-clear register type, preventing inadvertent overwriting of other flags in the same register.

48.7 Pin muxing

For pin muxing, see *Chapter 4: Signal Description.*

49 System Status and Configuration Module (SSCM)

49.1 Introduction

49.1.1 Overview

The System Status and Configuration Module (SSCM), shown in *Figure 864*, provides central device functionality.

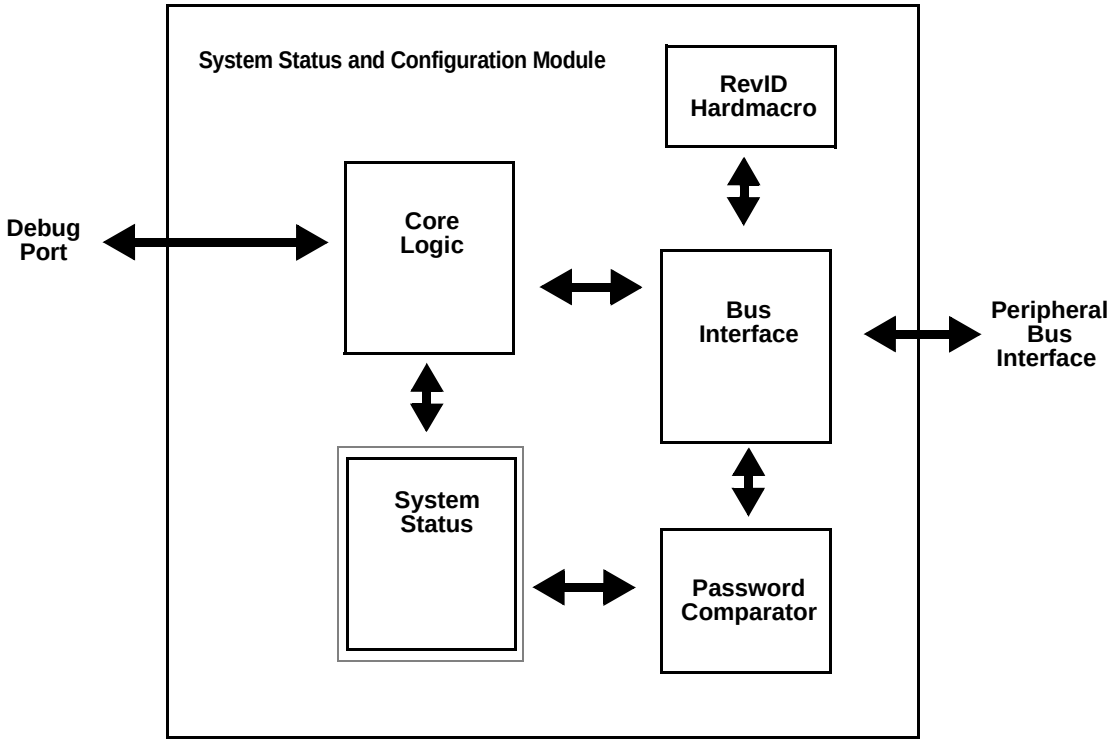


Figure 864. SSCM block diagram

49.1.2 Features

The SSCM includes these distinctive features:

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

49.1.3 Modes of operation

The SSCM operates identically in all system modes.

49.2 External signal description

The SSCM has no external pins.

49.3 Memory map/register definition

This section provides a detailed description of all memory-mapped registers in the SSCM.

Table 744 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 744. Module memory map

Address	Register	Size	Access	Mode (1)
Base + 0x0000	System Status (STATUS)	16 bits	R/W	A
Base + 0x0002	System Memory and ID (MEMCONFIG)	16 bits	R	A
Base + 0x0004	Reserved	16 bits	Reads/Writes have no effect.	A
Base + 0x0006	Error Configuration (ERROR)	16 bits	R/W	A
Base + 0x0008	Debug Status Port (DEBUGPORT)	16 bits	R/W	A
Base + 0x000A	Reserved	16 bits	Reads/Writes have no effect.	A
Base + 0x0014	Reserved	32 bits	Reads/Writes have no effect.	A
Base + 0x0018	DPM Boot Register	32 bits	R	A
Base + 0x001C	DPM Boot Key Register	32 bits	R	A
Base + 0x0020	User Option Status Register	32 bits	R	A
Base + 0x0024	SSCM Control Register	32 bits	R/W	A
Base + 0x0028 to Base + 0x3FFF	Reserved	See Note (2)		

1. **U** = User Mode, **S** = Supervisor Mode, **T** = Test Mode, **V** = DFV Mode, **A** = All (No restrictions)
2. If enabled at the SoC level, accessing these register addresses will cause bus aborts.

All registers are accessible via 8-bit, 16-bit or 32-bit accesses unless otherwise noted. 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 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.

49.3.1 Register descriptions

The following registers are available in the SSCM. Those bits that are shaded out are reserved for future use. To optimize future compatibility, these bits should be masked out during any read/write operations to avoid conflict with future revisions.

49.3.1.1 System Status Register (STATUS)

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

Figure 865. Status (STATUS) Register

Address: Base + 0x0000 Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LSM	CER	0	NXEN1	NXEN	PUB	SEC	0	BMODE			VLE	ABD	0	0	0
W		clear														
RESET:	0/1(1)	0	0	0	0	0	0	0	0/1	0/1	0/1	1	0	0	0	0

= Reserved

1. Reset value depends on the associated option bit.

Table 745. STATUS Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed
WRITE	Allowed	Allowed	Allowed

Table 746. STATUS field descriptions

Field	Description
LSM	Lock Step Mode. This field indicates how the two processor cores of the device are used. Lock Step Mode (LSM) is used to increase safety, Decoupled Parallel Mode (DPM) is used to increase performance. 1 Device is in Lock Step Mode (LSM) 0 Device is in Decoupled Parallel Mode (DPM) *Note: This field is used only to see what mode (LSM or DPM) the chip is in. To configure the chip to run in one of these modes, see Section 5.4, Selecting LSM or DPM.*
CER	Configuration Error. This field indicates that the SSCM has detected a configuration error during bootup. 1 Device configuration is not correct 0 No configuration problem detected by the SSCM
NXEN1	Processor 1 Nexus enabled.
NXEN	Processor 0 Nexus enabled.

Table 746. STATUS field descriptions (continued)

Field	Description
PUB	Public Serial Access Status. This bit indicates whether serial boot mode with public password is allowed. 1Serial boot mode with public password is allowed 0Serial boot mode with private Flash password is allowed, provided the key hasn't been swallowed
SEC	Security Status. This bit reflects the current security state of the Flash. 1The Flash is secured 0The Flash is not secured
VLE	Variable Length Instruction Mode. When booting from Flash, this field indicates that the code stored there is using the VLE instruction set. The value of this field is determined by the RCHW field of the Flash boot sector. 1 Main Flash contains VLE code 0 Main Flash contains standard PPC code
BMODE	Device Boot Mode. 000 Scan of both serial interfaces (FlexCAN and LINFlex) with autobaud 001 CAN Serial Boot Loader 010 SCI Serial Boot Loader 011 Single Chip 100 Expanded Chip This field is only updated during reset.
ABD	Autobaud. Indicates that autobaud detection is active when in SCI or CAN serial boot loader mode. No meaning in other modes. *Note: Autobaud functionality is not supported on this chip.*

49.3.1.2 System Memory and ID Register (MEMCONFIG)

The System Memory Configuration register is a read-only register that reflects the memory configuration of the system. It also contains the JTAG ID.

Figure 866. System Memory and ID (MEMCONFIG) Register

Address Base + 0x0002 Access: Read Only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	JPIN										IVLD	MREV				0
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(2)	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0

= Reserved

1. Reset value is SOC-specific.
2. Reset value depends on boot mode and security status (see SOC documentation).

Table 747. MEMCONFIG Field Descriptions

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

Table 748. MEMCONFIG Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed (also reads STATUS register)
WRITE	Not Allowed	Not Allowed	Not Allowed

49.3.1.3 Error Configuration (ERROR)

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

Figure 867. Error Configuration (ERROR) Register

Address : Base + 0x0006 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

= Reserved

Table 749. ERROR Field Descriptions

Field	Description
PAE	Peripheral Bus Abort Enable. This bit enables bus aborts on any access to a peripheral slot that is not used on the device. This feature is intended to aid in debugging when developing application code. 1 Illegal accesses to non-existing peripherals produce a Prefetch or Data Abort exception 0 Illegal accesses to non-existing peripherals do not produce a Prefetch or Data Abort exception
RAE	Register Bus Abort Enable. This bit enables bus aborts on illegal accesses to off-platform peripherals. Illegal accesses are defined as reads or writes to reserved addresses within the address space for a particular peripheral. This feature is intended to aid in debugging when developing application code. 1 Illegal accesses to peripherals produce a Prefetch or Data Abort exception 0 Illegal accesses to peripherals do not produce a Prefetch or Data Abort exception *Note: Transfers to Peripheral Bus resources may be aborted even before they reach the Peripheral Bus (i.e. at the PBRIDGE level). In this case, the **PER_ABORT** and **REG_ABORT** register bits will have no effect on the abort.*

Table 750. ERROR Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed
WRITE	Allowed	Allowed	Not Allowed

49.3.1.4 Debug Status Port Register (DEBUGPORT)

The Debug Status Port register is used to (optionally) provide debug data on a set of pins. Consult the SOC guide for this information.

Figure 868. Debug Status Port (DEBUGPORT) Register

Address: Base + 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	DEBUG_MODE		
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Reserved for future use

Table 751. 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 752* describes the functionality of the Debug Status Port in each mode.

Table 752. Debug Status Port Modes

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

1. All signals are active high, unless otherwise noted

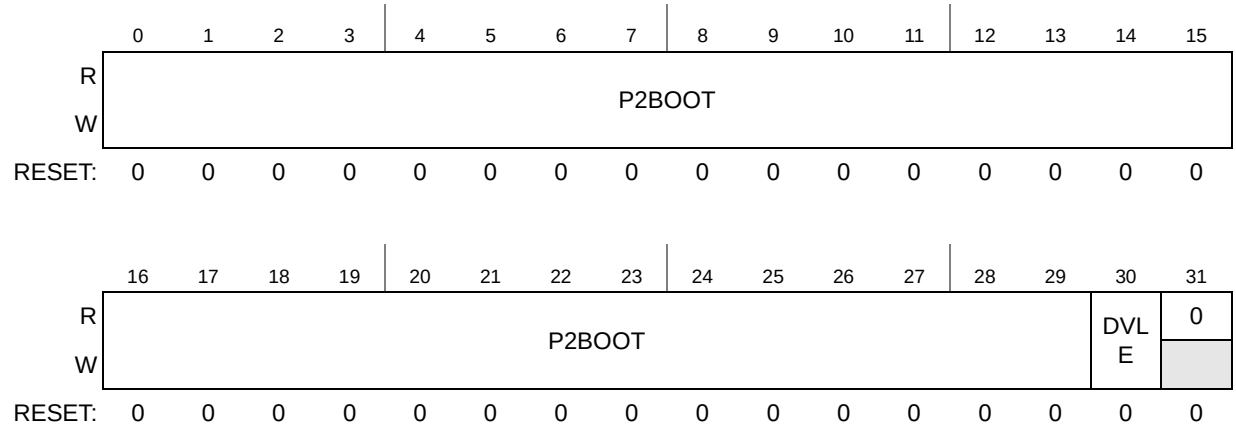
Table 753. DEBUGPORT Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Allowed
WRITE	Allowed	Allowed	Not Allowed

49.3.1.5 DPM Boot Register (DPMBOOT)

Figure 869. DPM Boot (DPMBOOT) Register

Address: Base + 0x0018 Access: Read/Write



= Writes have no effect on this bit

Table 754. DPMBOOT Field Descriptions

Field	Description
P2BOOT	Determines the location from which the 2nd processor will boot, once the main processor releases it from reset. This field is only used if the device is operating in DPM mode.
DVLE	Determines whether the 2nd processor will start executing VLE mode (1=VLE mode, 0=BookE mode). This field is only used if the device is operating in DPM mode.

49.3.1.6 Boot Key Register (DPMKEY)

Figure 870. Boot Key Register (DPMKEY) Register

Address: Base + 0x001C Access: Read/Write

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

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

= Writes have no effect on this bit

Table 755. DPMKEY Field Descriptions

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

49.3.1.7 User Option Status Register (UOPS)

Figure 871. User Option Status (UOPS) Register

Address: Base + 0x0020 Access: Read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UOPT															
W																
RESET:	Varies depending on the contents of the shadow block															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	UOPT															
W																
RESET:	Varies depending on the contents of the shadow block															

= Writes have no effect on this bit

Table 756. UOPS field descriptions

Field	Description
UOPT	Shows the values read from the User Option Bits location in the flash memory (see *Section 24.1.7, User option bits*).

49.3.1.8 SSCM Control Register (SCTR)

Figure 872. SSCM Control (SCTR) Register

Address: Base + 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	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	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	TFE	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Writes have no effect on this bit

Table 757. SCTR Field Descriptions

Field	Description
TFE	Test Flash Enable - setting this bit will map the TestFlash array to offset 0 of the Flash address space. This bit can only be set one time. Once it has been set and cleared it will remain cleared until the next device reset. Note that it is not possible to access the main flash memory array while TFE is set.

49.4 Functional description

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.

49.5 Initialization/application information

49.5.1 Reset

The reset state of each individual bit is shown within the Register Description section (see *Section 49.3.1, Register descriptions*).

50 System Timer Module (STM)

50.1 Introduction

50.1.1 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).

50.1.2 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

50.1.3 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.

50.2 External signal description

The STM does not have any external interface signals.

50.3 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.

50.3.1 Memory map

The STM memory map is shown in *Table 758*.

Table 758. STM memory map

Address Offset	Register Name	Register Description	Size (bits)	Access	Location
0x0000	STM_CR	STM Control Register	32	R/W	*on page 1388*
0x0004	STM_CNT	STM Counter Value	32	R/W	*on page 1388*
0x0008		Reserved	32	R/W	

Table 758. STM memory map (continued)

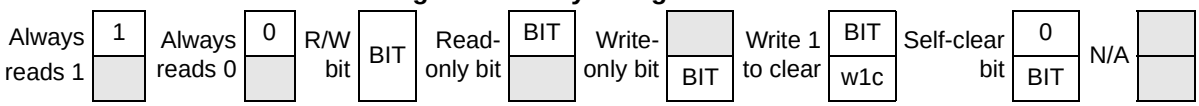
Address Offset	Register Name	Register Description	Size (bits)	Access	Location
0x000C		Reserved	32	R/W	
0x0010	STM_CCR0	STM Channel 0 Control Register	32	R/W	*on page 1389*
0x0014	STM_CIR0	STM Channel 0 Interrupt Register	32	R/W	*on page 1390*
0x0018	STM_CMP0	STM Channel 0 Compare Register	32	R/W	*on page 1390*
0x001C		Reserved	32	R/W	
0x0020	STM_CCR1	STM Channel 1 Control Register	32	R/W	*on page 1389*
0x0024	STM_CIR1	STM Channel 1 Interrupt Register	32	R/W	*on page 1390*
0x0028	STM_CMP1	STM Channel 1 Compare Register	32	R/W	*on page 1390*
0x002C		Reserved	32	R/W	
0x0030	STM_CCR2	STM Channel 2 Control Register	32	R/W	*on page 1389*
0x0034	STM_CIR2	STM Channel 2 Interrupt Register	32	R/W	*on page 1390*
0x0038	STM_CMP2	STM Channel 2 Compare Register	32	R/W	*on page 1390*
0x003C		Reserved	32	R/W	
0x0040	STM_CCR3	STM Channel 3 Control Register	32	R/W	*on page 1389*
0x0044	STM_CIR3	STM Channel 3 Interrupt Register	32	R/W	*on page 1390*
0x0048	STM_CMP3	STM Channel 3 Compare Register	32	R/W	*on page 1390*
0x004C - 0x3FFF		Reserved			

50.3.2 Register descriptions

The following sections detail the individual registers within the STM programming model.

Figure 873 shows the conventions used in the register figures.

Figure 873. Key to register fields



50.3.2.1 STM Control Register (STM_CR)

The STM Control Register (STM_CR) includes the prescale value, freeze control and timer enable bits.

Figure 874. STM Control Register (STM_CR)

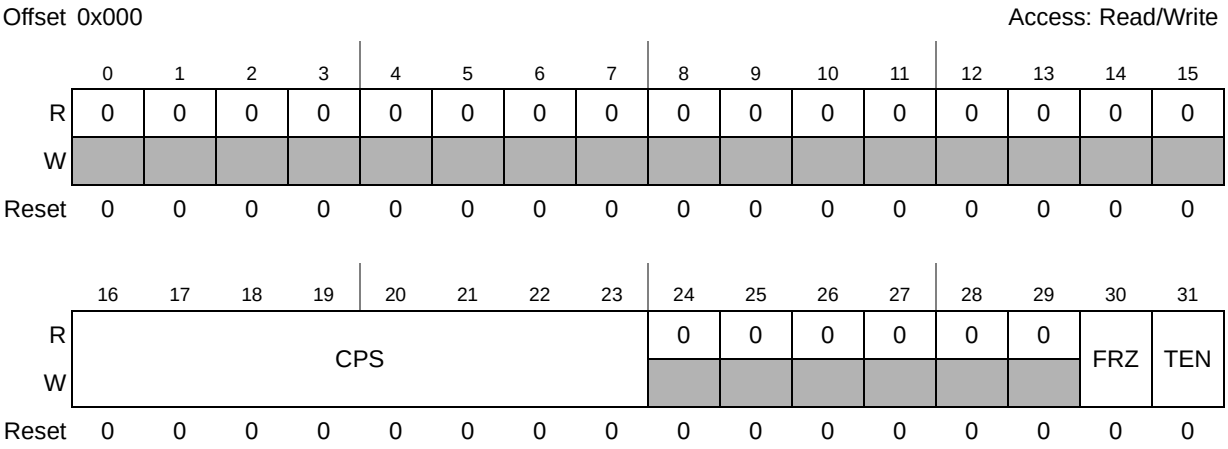


Table 759. STM_CR Field Descriptions

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.

50.3.2.2 STM Count Register (STM_CNT)

The STM Count Register (STM_CNT) holds the timer count value.

Figure 875. STM Count Register (STM_CNT)

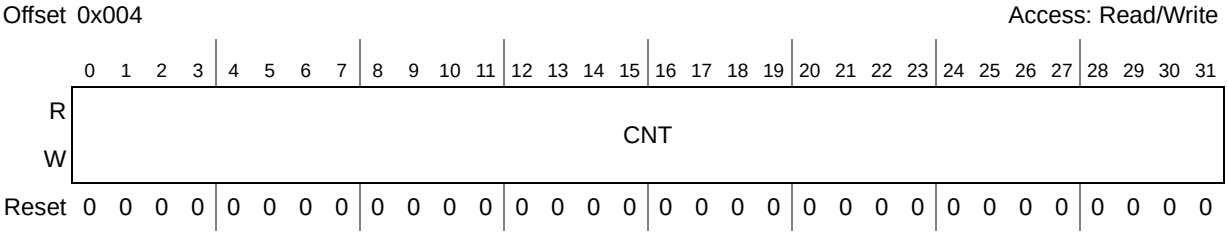


Table 760. STM_CNT Field Descriptions

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.

50.3.2.3 STM Channel Control Register (STM_CCRn)

The STM Channel Control Register (STM_CCRn) has the enable bit for channel n of the timer.

Figure 876. STM Channel Control Register (STM_CCRn)

Offset 0x10+0x10*n Access: Read/Write

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

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

Table 761. STM_CCRn Field Descriptions

Field	Description
CEN	Channel Enable. 0 = The channel is disabled. 1 = The channel is enabled.

50.3.2.4 STM Channel Interrupt Register (STM_CIRn)

The STM Channel Interrupt Register (STM_CIRn) has the interrupt flag for channel n of the timer.

Figure 877. STM Channel Interrupt Register (STM_CIRn)

Offset 0x14+0x10*n Access: Read/Write

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

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

Table 762. STM_CIRn Field Descriptions

Field	Description
CIF	Channel Interrupt Flag 0 = No interrupt request. 1 = Interrupt request due to a match on the channel.

50.3.2.5 STM Channel Compare Register (STM_CMPn)

The STM channel compare register (STM_CMPn) holds the compare value for channel n.

Figure 878. STM Channel Compare Register (STM_CMPn)

Offset 0x18+0x10*n Access: Read/Write

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

Table 763. STM_CMPn Register 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.

50.4 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.

51 Temperature Sensor (TSENS)

51.1 Introduction

The TSENS module provides an analog voltage that is proportional to the internal temperature of the SPC56ELxx. This voltage can be sampled by the ADC and converted to an actual temperature by using factory-programmed calibration constants.

Figure 879 shows a block diagram of the TSENS module. The module operates on the principles of a basic silicon bandgap sensor that measures the difference in base-emitter voltages of multiple transistors, which can be related to the absolute temperature of the device.

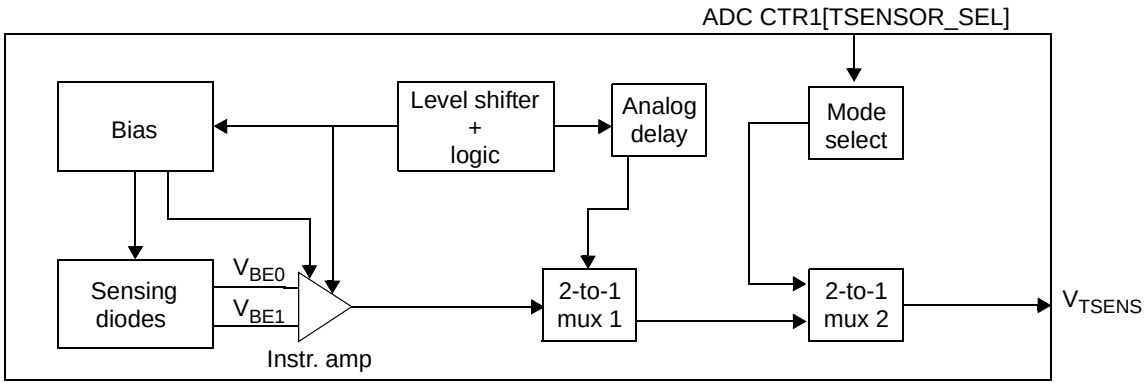


Figure 879. TSENS block diagram

51.2 Features

The TSENS has the following features:

- Temperature monitoring range: –40–150 °C
- Sensitivity: Approximately 5.14 mV/°C

51.3 Signals

The TSENS does not use any input signals external to the device.

The TSENS outputs one analog signal, V_{TSENS}. Depending on the TSENS mode of operation, V_{TSENS} is proportional or inversely proportional to the device temperature. See *Chapter 9: Analog-to-Digital Converter (ADC)*, for information on which ADC channels are connected to V_{TSENS}.

51.4 Modes of operation

The TSENS has two modes of operation:

- Proportional to absolute temperature (PTAT) — V_{TSENS} increases linearly with increasing temperature
- Complementary to absolute temperature (CTAT) — V_{TSENS} decreases linearly with increasing temperature

The mode of operation is controlled by the CTR1[TSENSOR_SEL] field in the ADC, as described in *Section 9.3.10.2, Conversion Timing Register 1 (CTR1)*.

51.5 Obtaining the device temperature using TSENS

In order to obtain the device temperature using TSENS, you must do the following:

- Extract the necessary TSENS calibration constants from the SPC56ELxx test flash memory (see *Section 51.5.1, TSENS calibration constants*)
- Measure V_{TSENS} in PTAT and CTAT modes
- Calculate the device temperature using the equations in *Section 51.5.2, Equations for converting TSENS voltage to device temperature*

The need for the measurements in two different modes is driven by the fact that although the TSENS output is linear in either mode, the intercept of the V_{TSENS}-T plot varies significantly based on the ADC reference voltage. Performing the two measurements allows the resulting equations to be independent of this reference voltage.

51.5.1 TSENS calibration constants

The equations needed to convert VTSENS to a device temperature depend on four calibration constants as described in *Table 764*. These constants are determined during factory testing of the SPC56ELxx and stored in the SPC56ELxx test flash memory as specified in *Section 24.1.8, Test flash memory*.

Table 764. TSENS calibration constants

Constant	Description
P_1	Code from the ADC converting V_{TSENS} in PTAT mode at 150 °C
P_2	Code from the ADC converting V_{TSENS} in PTAT mode at -40 °C
C_1	Code from the ADC converting V_{TSENS} in CTAT mode at 150 °C
C_2	Code from the ADC converting V_{TSENS} in CTAT mode at -40 °C

51.5.2 Equations for converting TSENS voltage to device temperature

In the equations below:

- P_n and C_n are the calibration constants described in *Section 51.5.1, TSENS calibration constants*
- T is the device temperature in °C
- P_x is the code from the ADC converting the VTSENS output in PTAT mode at a generic temperature T
- C_x is the code from the ADC converting the VTSENS output in CTAT mode at a generic temperature T
- T_2 = -40
- T_1 = 150

Define

$$A = P_x C_2 - P_2 C_x$$

$$B = C_x P_1 - P_x C_1$$

The temperature is calculated as

$$T = T_2 + \frac{(T_1 - T_2) \cdot A}{A + B}$$

52 Wakeup Unit (WKPU)

52.1 Introduction

52.1.1 Overview

The WKPU supports one external source that can cause non-maskable interrupt requests or wakeup events.

Figure 880 is a block diagram of the WKPU and its interfaces to other system components.

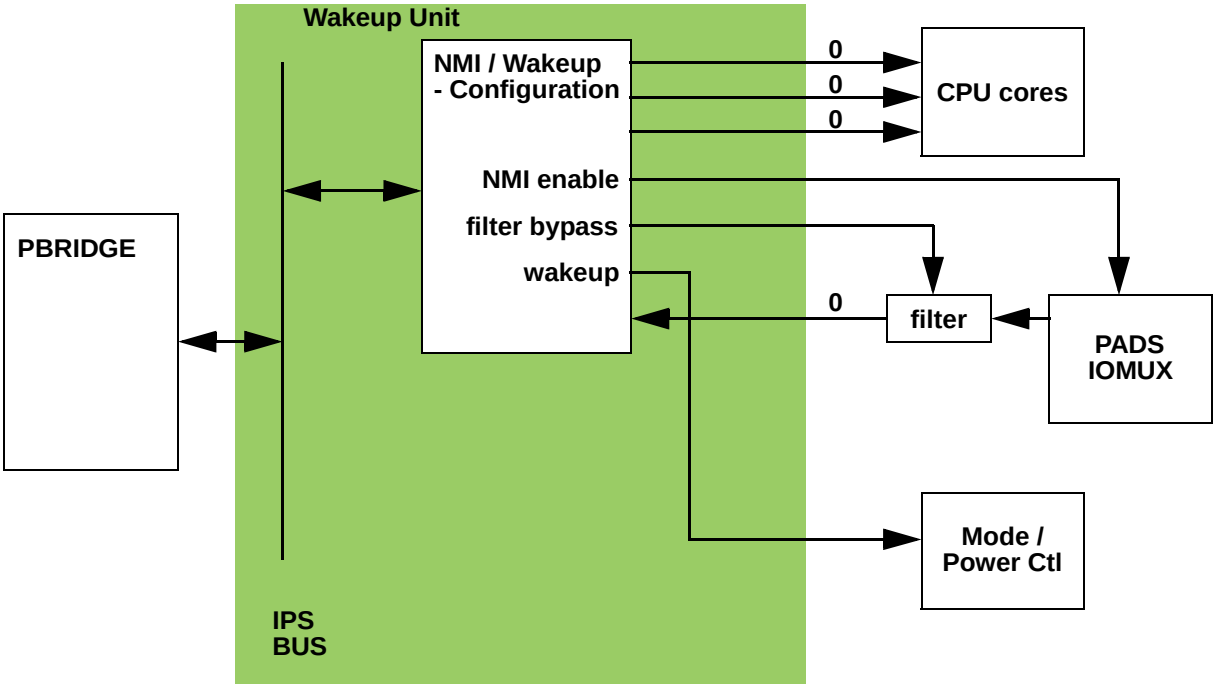


Figure 880. WKPU block diagram

52.1.2 Features

The WKPU supports:

- 1 NMI source
- 1 analog glitch filter
- Independent interrupt destination: non-maskable interrupt, critical interrupt, or machine check request
- Edge detection
- Cconfigurable system wakeup triggering from a detected NMI event

52.2 External signal description

The NMI input pin can be used as a non-maskable interrupt source in normal run mode or as a chip wakeup source during ***STOP0*** mode.

Note: Be aware that the Wakeup pins are enabled in ALL modes. Therefore, the Wakeup pins should be correctly terminated to ensure minimal current consumption. Any unused Wakeup signal input should be terminated by using an external pull-up or pull-down.

52.3 Memory map and register description

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

52.3.1 Memory map

Figure 765 gives an overview on the WKPU registers implemented.

Table 765. WKPU memory map

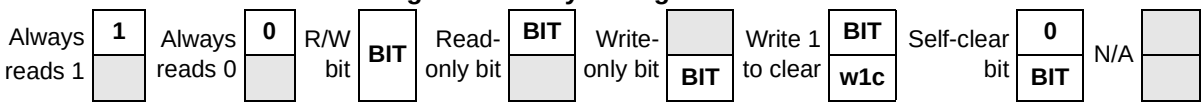
Address Offset	Use	Abbreviation	Size	Supported Access Sizes
0x0000	NMI Status Flag Register	NSR	32	32/16/8
0x0004 - 0x0007	Reserved			
0x0008	NMI Configuration Register	NCR	32	32/16/8
0x000C - 0x3FFF	Reserved			

Note: Reserved registers will read as 0, writes will have no effect. If supported and enabled by the chip, a transfer error will be issued when trying to access completely reserved register space.

52.3.2 Register descriptions

This section describes in address order all the WKPU 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.

Figure 881. Key to Register Fields



52.3.2.1 NMI Status Flag Register (NSR)

This register holds the non-maskable interrupt status flags.

Figure 882. NMI Status Flag Register (NSR)

Address: 0x0000 Access: User read/write (write 1 to clear)

	0	1	2	3	4	5	6	7
R	NIF0	NOVF0	0	0	0	0	0	0
W	w1c	w1c						
Reset	0	0	0	0	0	0	0	0

	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

Table 766. NSR Field Descriptions

Field	Description
NIF0	NMI Status Flag 0. This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (NREE0 or NFEE0 set), NIF0 causes an interrupt request. 1 An event as defined by NREE0 and NFEE0 has occurred 0 No event has occurred on the pad
NOVF0	NMI Overrun Status Flag 0. This flag can be cleared only by writing a 1. Writing a 0 has no effect. It will be a copy of the current NIF0 value whenever a NMI event occurs, thereby indicating to the software that a NMI occurred while the last one was not yet serviced. If enabled (NREE0 or NFEE0 set), NOVF0 causes an interrupt request. 1 An overrun has occurred on NMI pin 0 No overrun has occurred on NMI pin

52.3.2.2 NMI Configuration Register (NCR)

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

Figure 883. NMI Configuration Register (NCR)

Address: 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7
R	NLOCK0	NDSS0		NWRE0	0	NREE0	NFEE0	NFE0
W								
Reset	0	0	0	0	0	0	0	0

	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

Table 767. NCR Field Descriptions

Field	Description
NLOCK0	NMI Configuration Lock Register 0. 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 0. 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 0. 1 A set NIF0 bit or set NOVF0 bit causes a system wakeup request 0 System wakeup requests from the corresponding NIF0 bit are disabled
NREE0	NMI Rising-edge Events Enable 0. 1 Rising-edge event is enabled 0 Rising-edge event is disabled

Table 767. NCR Field Descriptions (continued)

Field	Description
NFEE0	NMI Falling-edge Events Enable 0. 1 Falling-edge event is enabled 0 Falling-edge event is disabled
NFE0	NMI Filter Enable 0. Enable analog glitch filter on the NMI pad input. 1 Filter is enabled 0 Filter is disabled

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

52.4 Functional description

52.4.1 General

This section provides a complete functional description of the WKPU.

52.4.2 Non-maskable interrupts

The WKPU supports one non-maskable interrupt.

The WKPU supports the generation of 3 types of interrupts for the NMI input pin. The WKPU supports the capturing of a second event before the interrupt is cleared, thus reducing the chance of losing an NMI event.

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.

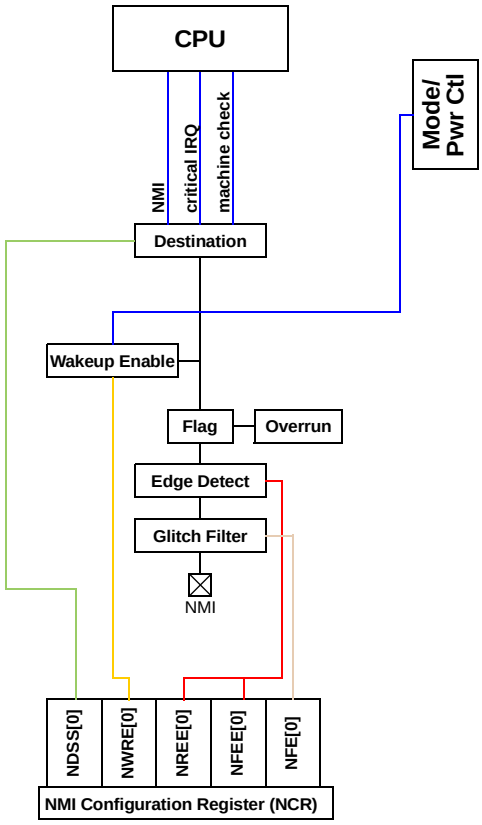


Figure 884. NMI pad diagram

52.4.2.1 NMI management

The NMI can be enabled or disabled as required by the application. This can be performed using the single NCR register laid out to contain all configuration bits in a single byte (see *Figure 883*). The NMI pin 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.

The NMI destination interrupt is controlled by the user through the configuration of the NDSS bits. See *Table 767* for details.

The NMI supports a status flag and an overrun flag which are located in the NSR register (see *Figure 882*). 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 (i.e. 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.

Appendix A Revision History

This appendix describes corrections to the *SPC56ELxx Microcontroller Reference Manual*. For convenience, the corrections are grouped by revision.

Table 768. Document revision history

Date	Revision	Document changes
19-Jan-2009	1	Formatting, spelling, grammar, and layout corrections.
10-Mar-2009	2	Chapter Introduction: Removed information about the devices with 384 KB and 512 KB Flash memory. In the device comparison table: – Changed the number of master/slave ports (was 4 × 3, is dependent on operating mode). – Changed “CRC” to “Cyclic Redundancy Checker (CRC) unit”. – Changed “KGD” to “Known Good Die (KGD)”. – Added information about the planned BGA package and modules available on it. Reduced the length of the feature summary. Deleted the text “0 wait state” from the “On-chip SRAM with ECC” section. In the “Platform Memory Access Time Summary” table, changed all data phase wait states to TBD. In the “System Clocks and Clock Generation” section: - Added a ÷8 option to the programmable output clock divider of system clock. - Changed “Supports frequency trimming by user application” to “Supports automated frequency trimming by hardware during device startup and by user application”. In the FCCU section: - Changed “collection of test results” to “collection of self-test results”. - Changed “external device” to “external/surrounding system”. In the BAM section: - Changed “that is programmed once” with “with hard-coded content”. Added “Triggering of device self-tests during reset phase of device boot” to the SSCM feature list. In the ADC section, changed “Safety built-in test” with “Built-in self-test features triggered by software”. In the CRC section: - Changed “CRC Unit” to “Cyclic Redundancy Checker (CRC) Unit”. - Changed “compute CRC on data” to “compute CRC signatures on data”. - Changed “each of them with different polynomial and seed” to “each with a selectable polynomial and seed”. In the “Junction temperature sensor” section, changed “Temperature alarm” to “Software temperature alarm”. In the PMU section: - Changed “bare die” to “Known Good Die”. - Changed “on-line testing feature” to “on-line self-testing feature”. - Replaced references to LVD with references to LVI. Replaced “Flash memory and RAM controllers” with “Flash memory controller and RAM controllers”. Replaced “the latest CPU from the e200 family” with “a CPU from the e200z4 family”.

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Mar-2009	2	- Deleted the text “0 wait state” from the “On-chip SRAM with ECC” section. - In the “Platform Memory Access Time Summary” table, changed all data phase wait states to TBD (per Oliver Bibel, these values will be available after synthesis). - In the “System Clocks and Clock Generation” section: - Added a ÷8 option to the programmable output clock divider of system clock. - Changed “Supports frequency trimming by user application” to “Supports automated frequency trimming by hardware during device startup and by user application”. - In the “FMPLL” section, changed “Modulation depth ±4% if centered or 0% to –8%” to “Modulation depth ±2% if centered or 0% to –4%”. - In the “DSPI” section, changed “As many as four chip select lines available, depending on package and pin multiplexing, enabling 16 external devices to be selected using external muxing from a single DSPI” to “Up to 8 chip select lines available, depending on package and pin multiplexing”. - In the FlexPWM section, added information that the 144-pin package has only one FlexPWM. - Modified the feature list in the “ADC” section. - Changed the second bullet of the “RCCU” section. - Deleted the accuracy specification in the “Junction Temperature Sensor” section. - Deleted the $\overline{\text{RDY}}$ pin in the “NPC” section. - In the “PMU” section, changed “bare die” to “Known Good Die” and changed the associated information. - Modified the features in the “BIST” section. Chapter Memory Map: Revised the memory map table. Chapter Signal Description: Added a pinout diagram and several signal tables, synchronizing the content between this document and the data sheet. In the “Supply Pins” table: - Deleted the trailing abbreviations (for example, “LV REGCOR0”) from many entries in the “Description” column. - Modified several ADC entries. - Deleted the entries for pins 50–59. Reset Generation Module: Replaced the entire chapter. Chapter Operating Modes: Revised the “Overview” section. Revised the description of the Lock Step Mode. In the “Lock Step Mode” section, changed “for which SIL 3 capability has been certified“ to “for which a SIL 3 capability certificate has been planned“. Changed the note for RUN0..3 modes (was “... software must ensure that the code will execute from RAM before it changes to this mode”, is “... software must ensure that the code will execute from RAM before it changes to a mode with powered-down instruction Flash memory”).

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Mar-2009	2	Chapter e200z4d Core: Replaced the entire chapter. Chapter Functional Safety: Moved the subsections of the “Redundancy” section to a new section, “Built-In Self-Test”. Deleted the bullet “NVM BIST for Flash memory”. Added the section “Software Triggered Self-Tests After Boot”. Updated the information on modules which require BIST runs. Removed the comment “(under discussion...” from the “External Measures” section. In the “BIST During Boot” section, deleted the sentence “It is expected that the device operates at most for 10 hours (maximum Trip Time) before being shut down and booted again.” In the “External Measures” section, deleted the “PWM Output ASIC” bullet item. Chapter Enhanced Direct Memory Access: Added information specific to this device. DMA Channel Mux: Added the DMA_MUX trigger inputs and DMA_MUX request source slot mapping tables. Chapter Peripheral Bridge: Replaced the chapter contents with new content (AIPS-Lite peripheral bridge). Chapter Flash Memory: Added information about the Flash-memory block and the platform Flash-memory controller. Chapter Interrupts: Replaced the interrupts table with a new version. Chapter System Integration Unit: Deleted the erroneous pad selection and pin muxing tables (correct versions will be added in a future revision of this document). Chapter Boot Assist Module: In the overview, changed “LINFlex” to “LINFlex-UART”. In the features, changed “initialized or invalid” to “initialized or holds no block having registered a valid application code”. In the “Entering boot modes” section, changed “FAB (Force Alternate Boot Mode) which” to “FAB (Force Alternate Boot Mode) pin which”. In the “Entering boot modes” section, changed “lowest Flash sector” to “first Flash-memory sector”. In the “Boot mode selection” figure, added the note “(planned)” to the FlexRay/autobaud scan node. Added the VLE bit to the RCHW. Changed the memory partitioning figure and Flash boot sector table. Deleted the entry for 100 (Expanded Chip) in the BMODE field description. Added “SWT is disabled in case of a serial boot mode or when entering static mode” to the “BAM resources” section. Deleted the footnote in the “Download and execute the new code” section.

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Mar-2009	2	Chapter Fault Collection Unit: Replaced the contents of the entire chapter with the correct content for the Fault Collection and Control Unit (FCCU). Chapter Power Management Unit: Added a placeholder chapter (content will be provided in a future revision). Chapter Self-test Control Unit: Added a placeholder chapter (content will be provided in a future revision). Sine Wave Generator: Added a placeholder chapter (content will be provided in a future revision). System Status and Configuration Module: Replaced the entire chapter. Chapter Analog-to-Digital Converter: In the “ADC sampling and conversion timing” section, replaced “$T_{conv} = (T_{sample} + T_{eval} + ndelay) \bullet T_{ck}$“ with “$T_{conv} = T_{sample} + T_{eval} + (ndelay \bullet T_{ck})$“. Periodic Interrupt Timer: Added missing register information. Chapter FlexRay: Replaced the entire chapter. Chapter LINFlex: Removed the earlier chapter contents because they did not describe the module present on this device. Chapter JTAG Controller: Added information specific to this device. Chapter Nexus Port Controller: Added information specific to this device. Appendix A Register Under Protection: Added information about the protected registers on this device.
23-Sep-2009	3	Preface: Added a table of acronyms and abbreviated terms. Chapter Introduction: Added the SEMA4 and SWG blocks to the block diagram. In the device-comparison table, added “+2% FM” to the “Execution speed” entry. In the feature summary, revised the temperature specifications to differentiate junction and ambient temperature. In the feature details: – Added information that the 144-pin package has one FlexPWM. – Changed “Internal reference clock (RC) oscillator” to “Internal RC oscillator”. Chapter Signal Description: Updated the signal diagrams and pin descriptions.

Table 768. Document revision history (continued)

Date	Revision	Document changes
23-Sep-2009	3	Chapter Reset Generation Module: Replaced the entire chapter. Replaced the “Mode Entry Module (MC_ME)” section. Chapter e200z4d Core Complex Overview: Deleted the “Auxiliary interface for higher data input/output” entry from the “Nexus3+ Features” section. Chapter Semaphore Unit: Added a chapter describing this module. Chapter Enhanced Direct Memory Access: Changed the bit order in the register figures (was 31..0, is 0..31) to adhere to Power Architecture conventions. Chapter DMA Channel Mux: Replaced the contents of the following subsections: Introduction External signal description Memory map and register definition Functional description Initialization/application information Chapter Crossbar Switch: Updated the “Logical master IDs” table. Removed the fixed reset values for the MPR register and added the reset value to the “Information specific to this device” section. Added device-specific register and signal information to the “Information specific to this device” section. Chapter Flash Memory: Removed erroneous references to internal-only documentation. Chapter General-Purpose Static RAM: Added SRAM size to the introduction. Chapter Memory Protection Unit: Added a chapter describing this module. Chapter Interrupts: Added a footnote to IRQs 247 and 248 reminding the user that the SEMA4 unit is only available in DP Mode. Chapter System Integration Unit: Corrected the reset value of MIDR2[PARTNUM] (was 0x50, is 0x4C). Chapter Clock Generation Module: Replaced the entire chapter. Chapter Frequency-Modulated Phase-Locked Loop: Revised the block diagram.

Table 768. Document revision history (continued)

Date	Revision	Document changes
23-Sep-2009	3	Chapter Oscillators: Changed “Internal reference clock (RC) oscillator” to “Internal RC oscillator”. Chapter Error Correction Status Module: Removed all references to AXBS_lite. Removed all references to the ECSM register protection. Replaced the syndrome-mapping table in the RESR section. Modified the definition of the EEGR register. Chapter Boot Assist Module: Updated the feature list. Revised the “Boot Modes” section. Revised the “Memory Map” section. Revised the “Entering Boot Modes” section. Revised the “SPC56ELxx Flash memory partitioning and RCHW search” figure. Revised the “Single Chip boot mode” section. In the “Fields of SSCM STATUS register used by BAM” table, replaced the entry for BMODE = 0b000 (is “Reserved”). Added “PIT for time measurement when performing serial boot mode” to the “BAM Resources” list. Chapter Fault Collection and Control Unit: Changed the bit order in the register figures (was 31..0, is 0..31) to adhere to Power Architecture conventions. Deleted the “Pinout” section (the module contains no external signals). Deleted the “Static configuration” section. In the FCCU_CFG[FOP] field description, added a factor of 1024 to the equation for EOUT frequency. Chapter Voltage Regulators and Power Supplies: Removed this chapter (the correct information is now present in the PMU chapter). Chapter Power Management Unit: Added information about this module. Chapter Self-Test Control Unit: Added information about this module. Chapter System Status and Control Module: Removed information about test Flash registers and labeled the associated address offsets (0x0028–0x0034) as reserved. Added register abbreviations to their respective section headings. Deleted the footnotes in the register figures regarding register access. Deleted the implementation note from the SCTR[TFE] field description. Chapter CRC Unit: Deleted the “Pinout” section (the module contains no external signals). Changed the bit order in the register figures (was 31..0, is 0..31) to adhere to Power Architecture conventions.

Table 768. Document revision history (continued)

Date	Revision	Document changes
16-Oct-2009	3	Chapter Deserial Serial Peripheral Interface: Extensive modifications throughout the chapter. Chapter FlexRay Communications Controller: Added disclaimer footnote for protocol spec. Chapter LINFlex Controller: Added information about this module. Chapter Motor Control Pulse Width Modulator Module: Removed the “Customization” section. Chapter JTAG Controller: Removed the number of JCOMP bits from the device-specific information. Added the size of the CENSOR_CTRL register to the “Information specific to this device” section. Added clarifying text to the “Boundary Scan Register” section. Chapter Nexus Port Controller: Removed the duplicate “Device specific parameters” section. Added an entry for MCKO_DIV[2:0] = 2 (frequency = SYS_CLK/3) and a clarifying note to the “MCKO_DIV Values” table. Revised the “MCKO and ipg_sync_mcko” and “Nexus reset control” sections. Appendix A Register Under Protection: Replaced the entire table.
27-Oct-2009	4	Chapter Introduction Added information about the 257-ball BGA package. In the feature details, added information about the SWG. Chapter Signal Description Added information about the 257-ball BGA package Chapter Reset Generation Module Modifications throughout the chapter. Chapter Operating Modes Replaced the “Mode Entry Module (MC_ME)” section. Chapter Peripheral Bridge Replaced the entire chapter. Chapter Flash Memory Revised the C90FL Flash memory map. Added the “User option bits” section. Replaced the content for the PFLASH2P_LCA controller with content for the PFLASH2P_H7FB controller and revised the register memory map.

Table 768. Document revision history (continued)

Date	Revision	Document changes
27-Oct-2009	4	Chapter Memory Protection Unit In the “Features” section, changed “128 bits in size” to “128 bits per descriptor”. Added device-specific information throughout the register description section. In the MPU_EDRn field description, changed “logically anded with the access error indication” to “where the error occurred”. Deleted internal-only information. Chapter Interrupt Controller Extensive modifications throughout the chapter. Chapter Interrupts Replaced “Rsrvd EMIOS” with “Reserved”. Added interrupt sources 250-255. Chapter Clock Architecture Added this chapter. Chapter Clock Generation Module Modifications throughout the chapter. Chapter Power Control Unit Added information about this module. Chapter Error Correction Status Module In the RESR section, deleted the second “RAM Syndrome Mapping for Single-Bit Correctable Errors” table. Changed the definition of the MUDCR. Changed the reset value of the IMC register. Chapter Boot Assist Module Moved the RCHW section to the SSCM chapter. In the “Boot mode selection” figure, replaced “POR or Standby Recovery” with “POR”. Chapter Fault Collection and Control Unit Deleted the “Fault interface” section. Added information about the FCCU_CFG[CM] bit. Deleted the following unavailable registers: FCCU_EINOUT FCCU_NAFS FCCU_AFFS FCCU_NFFS FCCU_FAFS FCCU_SCFS

Table 768. Document revision history (continued)

Date	Revision	Document changes
27-Oct-2009	4	Chapter Self-Test Control Unit Changed “MCM” to “ECSM”. Chapter Sine Wave Generator Added information about this module. Chapter System Status and Configuration Module Deleted references to Standby mode. Moved the RCHW section from the BAM chapter to this chapter and revised it. Chapter Analog-to-digital converter Added the “Inter-module communication” section. Added the missing text for the presampling feature. Deleted the information about threshold registers (not implemented on this device). Removed redundant reset values in register figures. Chapter Deserial Serial Peripheral Interface Revised the module feature list. Removed content related to DSI and CSI configurations (unavailable on this device). Changed the number of FIFO registers (was 16, is 5). Changed the number of CTARs (was 8, is 4). Added a cross-reference to the CTU “Interaction with other peripherals” section. Revised the DSPI_SR[EOQF] field description. Chapter FlexRay Added a cross-reference to the CTU “Interaction with other peripherals” section. Chapter Cross Triggering Unit Added the “Interaction with other peripherals” section. Chapter LIN Controller Changed the abbreviation for this module from “LINFlex” to “LINFlex3”. Chapter Motor Control Pulse Width Modulator Module Added device-specific information to the beginning of this chapter. Chapter Enhanced Motor Control Timer Added device-specific information to the beginning of this chapter.

Table 768. Document revision history (continued)

Date	Revision	Document changes
27-Oct-2009	4	Chapter JTAG Controller Revised the device-specific header information. Removed the incorrect descriptions of the ACCESS_AUX_TAP instructions from the “JTAG instructions” table and added the correct instructions to the device-specific header. Chapter Nexus Port Controller Revised the device-specific header information. Chapter Register Under Protection Added information about the functionality of the register protection (REG_PROT) module. Converted the chapter from an appendix to a regular numbered chapter.
26-Apr-2010	5	Throughout Editorial changes and improvements. Rearranged the chapter order (alphabetical by module name). Preface Added TBD to the list of acronyms and abbreviated terms. Chapter Introduction Renamed the peripheral bridge to “PBRIDGE”. Changed “LINFlex” to “LINFlexD” to be consistent with the rest of the documentation. Revised the information for FlexRay. Revised the “Clock, reset, power, mode and test control module” section. Added information about the 100-pin package. Chapter Memory Map Added the text “All memory not listed in this table is reserved, and access to those locations may produce undesirable results” to the introduction. Added information about the test flash memory mapping. Added information about the shadow block. Aligned some module names to match those used in the rest of the document. Added a second table sorted by module name. Removed the entry for eDMA_1 in DP mode (does not exist in that mode). Revised the FLASH0 entry to show a PCTL number of 66. Chapter Signal Description Extensive revisions to signal descriptions and pin muxing information. Added information about the 100-pin package.

Table 768. Document revision history (continued)

Date	Revision	Document changes
26-Apr-2010	5	Chapter Operating Modes In the “Lock Step Mode (LSM)” section, changed “as its partner on a clock-by-clock basis” to “as its partner”. Added information about how to enter DPM. Chapter Device Boot Modes Added this chapter. Chapter Device Security Added this chapter. Chapter Analog-to-Digital Converter Revised the “Device-specific implementation” section. In the “Features” section, deleted information about MA signals (not present on this device). Revised the “ADC sampling and conversion timing” section. Deleted the “External decode signals delay” section (this feature is not supported on this device). In the “ADC sampling and conversion timing” section: – Added a cross-reference to the “Conversion timing” figure for the case of INPCMP = 0. – Deleted “INPLATCH and INPCMP configurations are limited when the system clock frequency is greater than 20 MHz.” Added the “DMA functionality” section. Added information about the ADC self test. Added a register memory map. Revised the PSCR section. Revised the information for the following registers: – CEOCFR – CIMR – CTR1 – CDR*n* Added information for the following registers: – WTISR – WTIMR – DMAE – DMAR0 – THRHLR*n* – PSCR – PSR0 – CTR0 – NCMR0 – AWORR0 All self-test registers Added information about the CWENR1 register.

Table 768. Document revision history (continued)

Date	Revision	Document changes
26-Apr-2010	5	In the THRLHRn field descriptions, added a cross-reference to the “Programmable analog watchdog” section. Changed the numbering in the fields of the following registers (were ascending, are descending): – CIMR0 – DMAR0 – PSR0 – JCMR0 – CWSEL0 – CWSEL1 Revised the CTR1[TSENSOR_SEL] field description. Added the missing “Presampling” section. Chapter Boot Assist Module Deleted the information about boot modes (it is now a separate chapter in this document). In the “Memory map” section, added a warning about using the BAM memory space. Revised the “Overview” section. In the “Executing BAM” section, changed “BAM is executed only on the following cases” to “BAM is executed only on one or both of the following cases”. Revised the “BAM software flow” section. Revised the “BAM resources” section. Revised the “Download 64-bit password and password check” section. Revised the “Download start address, VLE bit and code size” section. Revised the “Download data” section. Deleted the “Autobaud feature” section (not supported on this device). In the “Boot from UART - autobaud disabled” section, changed “LINFlex controller is configured to operate at a baud rate = system clock frequency/833” to “The LINFlexD controller is configured to operate at a baud rate of 19200”. Added the “Inhibiting BAM operation” section. Clock Architecture Revised the “System clock generation” figure. Added the system clock distribution figures. Clock Generation Module Deleted the first paragraph in the Introduction section.

Table 768. Document revision history (continued)

Date	Revision	Document changes
26-Apr-2010	5	Chapter Cross-Triggering Unit Revised the “Scheduler subunit” figure. Revised the TGSCR[MRS_SM] field description. Revised the following register descriptions: – THCR1 – THCR2 – CLRx – CTUEFR – CTUIFR – CTUIR Deleted the CTUPCR register. Added the following registers: – CTU_EXP_A – CTU_EXP_B – CTU_CNTRNG Chapter Deserial Serial Peripheral Interface In the Overview section, changed the number of DSPI modules (was 1, is 3). Removed “deserialization” from the “Overview” section. Removed “HT” from the block diagram. Removed the MTRIG signal (not supported on this device). In the “External Stop Mode” section, added “No register access is possible in this mode”. Revised the “Debug Mode” section. Revised the DSPI_MCR[DIS_TXF] and DSPI_MCR[DIS_RXF] field descriptions. Revised the DSPI_SR[TCF] field description. Revised the DSPI_RSER description. Revised the “Functional description” section. Chapter DMA Channel Mux Changed the module name (was “DMA Channel Mux”, is “eDMA Channel Mux”). Deleted the first footnote to the “DMA_MUX source slot mapping” table. Chapter e200z4d Core Complex Overview Revised the introductory material. Revised the “e200z4d block diagram” figure. Revised the “Integer Unit features” section. Renamed “System bus (core complex interface) features” to “Exernal core complex interface features”. In the footnote to the “Exceptions and conditions” table, changed “0xFFC” to “0b0”. Added information about the PVR, PIR, and SVR registers. In the “Microarchitecture summary” section, removed references to SPE (not supported on this device). Revised the “Availability of detailed documentation” section to reflect that the z4 core reference manual (CRM) is now available online.

Table 768. Document revision history (continued)

Date	Revision	Document changes
26-Apr-2010	5	Chapter Enhanced Direct Memory Access Extensive modifications throughout chapter. Chapter Enhanced Motor Control Timer Revised the entire chapter to reflect the features implemented on this device. Chapter Error Correction Status Module Revised the Features section. Changed the reset value of the PCT register (was 0xE012, is 0xE446). Removed the MIR register (not implemented on this device). In the “ECC Error Generation Register” section, added information about address bus inversions. Revised the “ECC Configuration Register” section. Chapter Fault Collection and Control Unit Replaced the following figures: – Dual-rail protocol (slow switching mode) – Dual-rail protocol (fast switching mode) – Time switching protocol – Bi-stable protocol Added correct reset values to the following registers: – FCCU_CF_CFG0..3 – FCCU_NCF_CFG0..3 – FCCU_CFS_CFG0..7 – FCCU_NCFS_CFG0..7 – FCCU_NCFE0..3 – FCCU_NCF_TOE0..3 In the “FSM description” section, changed “Following one of the following events” to “The FSM will leave the NORMAL state following one of these events”. Chapter Flash Memory Changed “C90FL Flash EEPROM Functional Description” to “C90FL flash memory functional description”. Added the “Array integrity self check” section. In the footnote to the “C90FL flash memory map” table, changed “shadow row” to “shadow block”. n the “Register memory map” table, changed the entry at address offset 0x28 (was BIU3, is reserved). Revised the ADR[ADDR] field description. Revised the PFCR0 field descriptions. Revised the PFLASH2P_H7Fb feature list. Deleted the “PFLASH2P_H7Fb Interface Signals” section. Revised the PFCR0 figure. In the “C90FL block overview”, changed the base address (was 0x03F8_8000, is 0xC3F8_8000).

Table 768. Document revision history (continued)

Date	Revision	Document changes
26-Apr-2010	5	Revised the “Read While Write (RWW)” section. In the “C90FL flash system block diagram” figure, changed PFLASH_C90FL to PFLASH2P_H7Fb. Added the BIU4 section. Changed the MCR[DONE] reset value (was 0/1, is 1). Revised the MCR field descriptions. In the “Register memory map” table, revised the entries for PFCR0, PFCR1, and PFAPR. In the PFLASH2P_H7Fb section, removed extraneous information throughout. Chapter FlexCAN Module Revised the chapter to reflect that FlexCAN on this device has 32 message buffers. Removed references to Doze mode (unsupported on this device). Chapter Flexible Motor Control Pulse Width Modulator Module In the “PWMA[n] and PWMB[n] - External PWM Pair” section, deleted the sentence “When not needed as an output, they can be used as inputs to the input capture circuitry.” Revised the “Deadtime Insertion and Fine Control Logic” figure. Revised the STS, INTEN, DMAEN, CVAL0CYC, and CVAL1CYC register definitions. Revised the “Interrupt summary” table. Chapter Frequency-Modulated Phase-Locked Loop Revised the “Features” section. Revised the MR[MOD_PERIOD] field description. Revised the note in the “Normal Mode with frequency modulation” section. Added the “Requirements” section and rearranged content to highlight that f_{VCO} must be in the range 256–512 MHz. Chapter General-Purpose Static RAM Changed the chapter title (was “General-Purpose Statis RAM”, is “Static RAM”). Chapter Interrupt Controller Revised the INTC_CPR_PRC0 section. Revised the INTC_EOIR_PRC0 section. Removed an erroneous reference to INTC_EOIR_PRC1 in the “End of Interrupt Exception Handler” section. Added step numbers to the “Order of ISR execution example” table. In the “Interrupt sources for INTC_0 and INTC_1” table, modified the “Not used” and “Reserved” entries to span the entire row. Merged the previous version’s “Interrupts” chapter with this chapter. Revised interrupt sources 27, 28, 29, 67, 87, 133, and 134.

Table 768. Document revision history (continued)

Date	Revision	Document changes
26-Apr-2010	5	Chapter JTAG Controller Revised the “Device-specific auxiliary TAP controller instructions” table. Added information about censorship mode. Revised the CENSOR_CTRL register section. Revised the “Enabling the TAP controller” section. Revised the “ENABLE_CENSOR_CTRL Instruction” section. Chapter LIN Controller Changed the module acronym (was LINFlex3, is LINFlexD). In the “Fractional baud rate generation” section, changed the note from “LFDIV must be greater than or equal to 1.0d” to “LFDIV must be greater than or equal to 1.5d, i.e. LINIBRR=1 and LINFBRR=8. Therefore, the maximum possible baudrate is fperiph_set_1_clk / 24.”. Chapter Memory Protection Unit In the Introduction section, deleted “pre-programmed”. Revised the “Features” section. In the “MPU memory map” table, removed the entries for the MPU_XEDR0 registers (they don’t exist). Revised the MPU_CESR section. Added the missing reset values to the MPU_EARn and MPU_EDRn register figures. In the “MPU Region Descriptor n (MPU_RGDn)” section, changed “16-byte” to “four 32-bit word”. Added the missing reset value to the MPU_RGDn.Word*m* figures. Added the missing reset value to the MPU_RGDAACn figure. Modified the MPU_RGDn.Word2 and MPU_RGDAACn descriptions. Revised the “MPU Region Descriptor n, Word 2 (MPU_RGDn.Word2)” section. In the “MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3)” section, changed “is a 128-bit entity” to “is a 4-word entity”. Removed superfluous content from the “Putting it all together and AHB error terminations” section. Chapter Motor Control Pulse Width Modulator Module Removed information about fractional delay and PWMA/PWMB capture capability (not implemented on this device). Chapter Multi-Layer AHB Crossbar Switch Changed the chapter title to “Crossbar Switch”. Changed “AXBS” to “XBAR”.

Table 768. Document revision history (continued)

Date	Revision	Document changes
26-Apr-2010	5	Chapter Nexus Port Controller In the “Information specific to this device” section: – Added a note that the MCKO division factor of 3 is not supported. – Added information about MCKO division factors for which Nexus DDR mode is supported. Revised the descriptions of Censored mode and Reset mode. Revised the “Censored Mode” section. Chapter Operating Modes In the “Protection of Mode Configuration Registers” section, added text explaining that PLL settings are protected if the external oscillator is not settled. Chapter Periodic Interrupt Timer Changed the PITMCR[MDIS] reset value (was 1, is 0). Chapter Peripheral Bridge Changed the module acronym (was AIPS-Lite, is PBRIDGE). In the “Memory map” section, changed “return zeros on reads” to “return undefined data on reads”. In the “PACRn field structure” figure, added a note about PACR0[SP] and PACR0[TP]. Added PACR18 to the module memory map. In the “Off-platform peripherals and OPACR numbers” table, changed “SIU” to “SIUL”. In the memory map, deleted MPROT8 and MPROT9 (not implemented on this device). In the MPROT section: – Revised the reset information. – Revised the “MPROT*n* field structure” figure. – Removed references to buffered writes (not supported on this device). In the “Off-platform peripherals and OPACR numbers” table: – Changed the OPACR number for FCCU (was 5, is 59). – Added an entry for FLASH0 (OPACR number 66). Chapter Power Management Unit In the “PMU blocks” table, changed the function for the HPREG2 from “Provides internal NPN...” to “Provides external NPN...”. In the “Memory map and register description” section, added the text “The PMU registers are mapped into the MC_PCU address space (base address: 0xC3FE_8080).” Chapter Register Protection Changed “AIPS” to “PBRIDGE”. In the “Overview” section, added information about the module base address. In the Features section, changed “Address mirror” to “Write to address mirror”. Revised the “Memory map and register definition” section.

Table 768. Document revision history (continued)

Date	Revision	Document changes
26-Apr-2010	5	Chapter Reset Generation Module In the “MC_RGM block diagram” figure, changed PA[4:2] to A[4:2]. Added “0” to all empty register fields. Revised the RGM_DES and RGM_DERD register definitions. Chapter Self-Test Control Unit In the “Memory map” section, added a note about preliminary functionality. Changed the reset value for STCU_CRCE (was 0x0000_0000, is 0xFFFF_FFFF). Chapter Semaphore Unit In the “Sample Gatelock Function” example, changed CP2_LOCK to CP1_LOCK. In the “Application information” section, deleted the sentence “The exact method for accessing the logical processor number varies by architecture. Chapter Software Watchdog Timer In the “Features” section, added the statement “On this device, the SWT is always driven by the IRSOSC.”. Revised the SWT_CR description. Added the reset value for the SWT_TO register. Chapter System Integration Unit Lite Revised the PSMI0_3–PSMI32_35[PADSEL0–3] section. Revised the MIDR1 and MIDR2 sections. Chapter System Status and Configuration Module Revised the MEMCONFIG register description. Revised the UOPS reset value. Removed information from the “Functional description” section that is duplicated elsewhere. Chapter System Timer Module Revised the STM_CIRn figure and field description to show the CIF bit as “w1c” (write 1 to clear). Chapter Temperature Sensor Added a chapter describing this module. Chapter Wakeup Unit Revised content throughout the chapter to indicate that the WKPU supports one NMI source and no external interrupt/wakeup sources. Added a note about pin termination to the “External signal description” section.

Table 768. Document revision history (continued)

Date	Revision	Document changes
24-Feb-2011	6	Throughout Editorial changes and improvements. Changed all WARNING labels to CAUTION labels. Chapter Introduction In the feature list: – Changed “As much as 128 KB on-chip SRAM” to “128 KB on-chip SRAM”. – Changed “CAN/UART/FlexRay Bootstrap loader” to “CAN/UART bootstrap loader”. In the feature details: – In the “On-chip SRAM with ECC” section, added information about required RAM wait states. – In the BAM and SSCM sections, changed “classic PowerPC Book E code” to “Power Architecture code”. – In the PIT section, deleted “32-bit counter for real time interrupt, clocked from main external oscillator” (not supported on this device). – In the flash-memory section, changed “16 KB Test” to “16 KB test sector” and revised the wait state information. – In the SRAM section, revised the wait state information. – In the STM section, added a footnote explaining the AUTOSAR acronym. Chapter Signal Description In the 144-pin pinout diagram: – Renamed pin 58 (was VDD_HV_ADV0_ADV1, is VDD_HV_ADV). – Renamed pin 59 (was VSS_HV_ADV0_ADV1, is VSS_HV_ADV). In the “LQFP144 pin function summary” table, for pin 39, changed $V_{SS_LV_COR}$ to $V_{DD_LV_COR}$. Added the “LQFP100 pin function summary” table. In the “Supply pins” table: – Changed the description for $V_{DD_LV_COR}$ (was “Voltage regulator supply voltage”, is “Core logic supply”). – Changed the description for $V_{DD_HV_PMU}$ (was “Core regulator supply”, is “Voltage regulator supply”). In the “Pin muxing” table: – In the “Pad speed” column headings, changed “SRC = 0” to “SRC = 1” and “SRC = 1” to “SRC = 0”. – For port B[6], changed the pad speed for SRC=0 (was M, is F). Chapter Operating Modes Revised the “Dual-core boot concepts” section. – Added content to the DPM section. Chapter Safety In the Monitoring section, added a bullet for the SMC clock. Revised the “Fault reaction” section. – In the “External measures” section, added an entry for the PWM output monitor.

Table 768. Document revision history (continued)

Date	Revision	Document changes
24-Feb-2011	6 (cont.)	Chapter Analog-to-Digital Converter In the Features section: – Changed “(PIT)” to “(CTU)”. – Revised the section to reflect the actual number and type of channels available on SPC56ELxx. In the “Inter-module communication” section, changed “Once selected, no mode change is possible unless a reset occurs” to “You can dynamically switch this mode without generating a reset or a power-down; in this case, however, you must not switch the mode while a conversion is in progress”. Revised the “Self testing” > “General operation” section. In the “DMA functionality” section, changed “DMA[0..3] registers” to “DMAR0 register”. In the “Power-down mode” section: – Added “After an exit from power-down mode, the first conversion can be started after 5 µs, otherwise the result can be affected by the improper setting of the ADC analog operating point.”. – Added a cross-reference to the PDEDR section. In the “CPU mode” section, changed “NCMR0-2” to “NCMR0”. Added content to the “CTU mode” section and revised the section to eliminate references to CTU trigger mode (not supported). In the “Watchdog timer” section, changed “NCMR0-2” to “NCMR0”. Revised the MCR definition. Revised the MSR definition. Revised the ISR definition. In the CEOCFR0 figure, changed the numbers in the field names (were 0..31, are 31..0). Revised the IMR definition. In the WTIMR figure, changed the address (was “Base + 0x0030”, is “Base + 0x0034”). Revised the PSCR figure to show correct data for reserved bits. In the STCR2[MSKST_EOC] field description, changed STSR2 to STSR1. Revised the STCR3[MSTEP] field description. In the CDRn figure, changed all fields to read-only. In the STCR2 figure, changed the address (was “Base + 0x340”, is “Base + 0x344”). Revised the MSR[REF_RANGE] field description. n the following register descriptions, marked bits 0–15 as reserved: – CEOCFR0 – CIMR0 – DMAR0 – PSR0 – NCMR0 – JCMR0 – AWORR0 Revised the “Conversion timing registers” section to describe which registers control which channels. In the STSR1 figure, changed fields ERR_C, ERR_RC, ERR_S2, ERR_S1, ERR_S0 from read/write to w1c.

Table 768. Document revision history (continued)

Date	Revision	Document changes
24-Feb-2011	6 (cont.)	Chapter Boot Assist Module In the Overview section: – Deleted the footnote about FlexRay support. – Changed “with and without autobaud” to “without autobaud”. In the “Boot from UART - autobaud disabled” and “Bootstrap with FlexCAN - autobaud disabled” sections, changed “driven by external oscillator” to “driven by the IRC (16 MHz)”. In the “BAM software flow” section: – Renamed the table (was “... register used by BAM”, is “... register read by BAM to detect the chosen boot mode”). – Removed references to ABD. In the “BAM resources” section, deleted the content pertaining to autobaud. In the “Download and execute the new code” section, deleted the optional step 0. In the “Download 64-bit password and password check” section, changed “puts MCU to standby mode” to “puts MCU to static mode”. In the “Download start address, VLE bit and code size” section, changed “VLE mode will be supported” to “Book E mode will be supported”. Chapter Clock Monitor Unit Revised the note in the “FMPLL clock monitor” section and added it to the end of the “Crystal clock monitor” section. Chapter Cross-Triggering Unit Revised the “CTU interaction with other peripherals” figure. Revised the FRx and FLx definitions. Added content to the CLRx section. Chapter Crossbar Switch In the Overview section, changed “and almost any data bus width” to “and a 64-bit data bus width”. Deleted the “XBAR block diagram” figure. Revised the “Logical master IDs” table. Deleted the information about AMPRx and ASGPCRx registers. Deleted the “XBAR master port block diagram” figure. Deleted the “XBAR slave port block diagram” figure. Chapter Deserial Serial Peripheral Interface In the Overview section, deleted “The DSPI supports SOC pin count reduction...”. Added a note to the DSPI_SR[EOQF] field description. Revised the “Continuous selection format” section. Removed parity information (parity is not supported on the DSPI on this chip). In the DSPI_MCR section, removed the DOZE field (is reserved). Revised the “Module Disable Mode” section to remove references to DOZE mode. Changed the DSPI_SR[TXRXS] field (was w1c, is read-only). In the “Continuous Serial Communications Clock” section, added the same note that appears in the “Continuous selection format” section.

Table 768. Document revision history (continued)

Date	Revision	Document changes
24-Feb-2011	6 (cont.)	Chapter eDMA Channel Mux In the “Modes of operation” section, changed the description of the Periodic Trigger Mode (was “... for channels 0–4”, is “... for channels 0–3”). In the “DMA_MUX trigger inputs” section, replaced “#NTRG” with 4. In the “DMA_MUX source slot mapping” section, deleted the “DMA acknowledge signal” column. Chapter Enhanced Direct Memory Access In the Features section, deleted the pseudocode examples. Renamed the DCHPRIn section (was “... 0, ..., {15, 31, 63}”, is “... 0–15”). In the “TCDn Word 1” table, revised the description for ssize[0:2] (was “101 32-byte”, is “101 32-byte burst (64-bit × 4)”). Deleted the third paragraph from “eDMA programming errors” section. Chapter Enhanced Motor Control Timer Revised the “External signal descriptions” section. Revised the STS[RCF] field description. In the ENBL[ENBL] field description, added “(ENBL[x] controls channel number x)”. In the DREQ1[DREQn] field description, changed “onto one of the four module” to “onto the module”. Deleted the Resets and Clocks sections. Chapter Fault Collection and Control Unit In the “Main features” section, changed “1 to 128” to “32”. In the “FCCU state diagram” figure, changed the condition for transitioning from FAULT to ALARM (was “any non-critical fault unrecovered”, is “(Critical fault recovered) AND (any non-critical fault unrecovered)”. Added the “Fault mapping” section. Revised the “Self-checking capabilities” section. Revised the “Reset interface” section. In the “WKUP/NMI interface” section, changed “single NMI output (nmi_b)” to “single NMI output”. In the FCCU_IRQ_STAT section, changed “external interrupt line “irq_misc_b” is asserted” to “external interrupt line is asserted”. In the FCCU_IRQ_EN section, changed “external interrupt line “irq_misc_b” is asserted” to “external interrupt is asserted”. Deleted the “Register map” section. In the “STCU interface” section, added clarifying cross-references. In the “NVM interface” figure, changed “nvm_cfg[10:0]” to “nvm_cfg[11:0]”. In the “Dual-rail protocol” section, added a note clarifying the phase information in the figures. In the FCCU_CFS_CFG0 and FCCU_CFS_CFG1 figures, revised bits 30–31 (were CFSC15, are CFSC0). In the FCCU_NCFS_CFG0..7 figure, revised bits 30–31 (were NCFSC15, are NCFSC0). Revised the FCCU_CFF and FCCU_NCFF field descriptions to reflect that the maximum number of injectable faults is 32 (0–31).

Table 768. Document revision history (continued)

Date	Revision	Document changes
24-Feb-2011	6 (cont.)	Chapter Flash Memory Removed references to “H7fb”. In the C90FL section, replaced the “Program” subsection. In the “User option bits” section, clarified how the user option bits are changed and verified. Revised the PFLASH2P “Register descriptions” section. Deleted the “Error termination” section. In the “Buffer invalidation” section, changed “non-zero value on haddr[28:24]” to “non-zero wait state value”. Revised the “Wait-state emulation” section. Added a warning and a note about how software running from flash memory should access registers. Added information about the NVPWD and NVSCI registers. In the “C90FL flash memory map” table, added the size of the flash memory shadow block (is 16 KB). In the “User option bits” table: – Changed the bit order (was 0..31, is 31..0). – Changed “Selects DPM” to “Selects single core/DPM”. In the PFCR0[B02_APC] field description, added ‘APC must equal RWSC”. Merged the memory map in the PFLASH2P section into the memory map in the C90FL section. Revised content throughout to eliminate references to banks. In the “Array integrity self check” section, added the note ”While Array Integrity is being executed, flash memory array accesses thru the BIU should not be requested”. Chapter FlexCAN In the MCR section, deleted the WAK_SRC field. In the “Rx Individual Mask Registers” figure, changed “0x0880–0x097F” to “0x0880–0x08FF”. Revised the “Stop Mode” section. In the CTRL[CLK_SRC] field description and the “Protocol Timing” section, added the note “In order to guarantee reliable operation, the selected CAN Protocol Interface (CPI) clock should not be faster as the the peripheral clock”. Chapter Flexible Motor Control Pulse Width Modulator Module In the CVAL0CYC section, changed “3 deep FIFO” to “4 deep FIFO”. In the CVAL1CYC section, changed “3 deep FIFO” to “4 deep FIFO”. Deleted the Resets and Clocks sections. Chapter Frequency-Modulated Phase-Locked Loop Revised the “Progressive clock switching” section. Chapter Interrupt Controller Revised the “Peripheral interrupt requests” section. Chapter Nexus Port Controller In the “Device parameter values” table, changed “1 in LSM, 2 in DPM” to “1”.

Table 768. Document revision history (continued)

Date	Revision	Document changes
24-Feb-2011	6 (cont.)	Chapter Oscillators In the “XOSC external oscillator” section, revised the notes in the OSC_CTL register figure. Chapter Periodic Interrupt Timer Removed references to RTI. In the CVAL figure, changed “Access: Read/Write” to “Access: Read-only”. Chapter Peripheral Bridge In the “Access support” section, changed “Aligned 32-bit” to “Aligned 64-bit”. In the “PBRIDGE memory map” table: – added MPROT4 to bit numbers 16–19. – Added MPROT5 to bit numbers 20–23. – Added OPACR62 (used by SWG). – Added OPACR66 (used by FLASH0). Added the “MPROT*n* field reset values” table. Chapter Register Protection Deleted the entry for DEAR. Chapter Reset Generation Module Added a footnote to all register figures to clarify the reset conditions. Deleted references to “comparator error”. In the RGM_FERD section, added “Some fields are read-only, with a POR value of 1 or 0, and the corresponding behavior cannot be changed.”. Revised the RGM_FERD[D_CMU12_FHL] field description. In the RGM_FESS section, changed the SS_SOFT field to read/write and added the required field description for SS_SOFT = 1. In the RGM_FESS[SS_SOFT] field description, added a description for a value of 1. Changed the SS_FCCU_SOFT field to read-only and added the note “(this value is not used on this chip)” to its field description. Revised the “IDLE Phase” section. Chapter Self-Test Control Unit Removed information about the CRC (not supported). Revised the STCU_MBCFMH figure to align to its field descriptions. In the “Reported errors” section, changed “STCU_MBSL flag register” to “STCU_MBSL and STCU_MBSH flag registers” and “STCU_MBEL flag register” to “STCU_MBEL and STCU_MBEH flag registers”. In the “STCU main features” section, changed “after a reset trigger event” to “after an STCU reset event”. In the “STCU main features” section, changed “Manages 5 LBISTs” to “Manages LBISTs”. In the “STCU main features” section, changed “Manages 35 MBISTs” to “Manages MBISTs”. Added a note to the “Boot sequence phase 1: Self test” figure. Added content at the end of the “The Safety Integrity Subsystem” section.

Table 768. Document revision history (continued)

Date	Revision	Document changes
24-Feb-2011	6 (cont.)	Chapter Sine Wave Generator Removed references to a 16-MHz input clock. Chapter Software Watchdog Timer In the SWT_CO section, added the caution “Do not access the SWT_CO register in Lock Step Mode (LSM)”. Chapter System Integration Unit Lite Revised content throughout the chapter to show the fact that this chip has 42 PSMI registers. In the MIDR1 and MIDR2 figures, revised the access type (was None, is “Read only”). In the MIDR1[PKG] field description, added the values for the 100-pin and 257-pin packages. Revised the IFMC0_31[MAXCNTx] field description. Chapter System Status and Configuration Module In the UOPS figure, changed the access type (was Read/Write, is “Read only”). Revised the UOPS section. Chapter Temperature Sensor In the “Equations for converting TSENS voltage to device temperature” section, changed the last equation (term R–R_2 was in the denominator, is in the numerator). Chapter Wakeup Unit Deleted the Preface section. In the NSR figure, changed NOVF to NOVF0.
10-Jun-2011	7	Internal release. Editorial changes and improvements. Chapter Preface Added information about how cut1 and cut2 differences are represented in this document. Chapter Introduction In the feature details: – In the PBRIDGE section, deleted “Write buffering for peripherals”. – In the CRC section, added information about the 8-bit polynomial implemented on cut2. Chapter Memory Map Added the “SRAM map for SPC56ELxx in LSM” and “SRAM map for SPC56ELxx in DPM” tables.

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Jun-2011	7 (cont.)	Chapter Signal Description In the 257-pin ballmap: – Added the definition of NC. – Added the $\overline{\text{RDY}}$ pin (K3) for cut2. In the “Pin function summary” tables, for the 257-pin package, added the $\overline{\text{RDY}}$ pin (K3) for cut2. In the 100-pin function summary table, changed RESET to $\overline{\text{RESET}}$. In the pin-muxing table: – In the “Input functions” column, for all ADC-related ports, changed GPIO to GPI. – For the 257-pin package, added the RDY pin (K3) for cut2. – Changed the pad speeds (were —/—, are M/S) for the following ports: C[15], D[0], D[3], and D[4] Added the “Mapping of ports to PGPDO/I registers” section. Chapter Operating Modes Revised the Overview section. Revised the DPM section. In the “Simplified boot process in DPM” figure, deleted “FCCU triggers NMI for both cores”. In the “Dual-core boot concepts” section, deleted “The system generates an NMI to both cores when the chip moves from DRUN to RUN0”. In the “Software setup” section, changed “Book E mode” to “Power Architecture mode”. Chapter Device Boot Modes In the “Hardware configuration” section, added the “Single Chip boot mode” section (formerly in the BAM chapter). Chapter Device Security In the “Securing the microcontroller” section, replaced references to NVSCI0 and NVSCI1 to a reference to NVSCI (this chip contains only one such register), and added the cut1 and cut2 definitions of this register. Chapter Functional Safety Revised the “BIST during boot” section. Chapter Analog-to-Digital Converter In the ISR figure, changed the REF_RANGE field from read-only to write-1-to-clear (w1c). In the “Normal conversion flow” figure, changed “Convert” to “Evaluate”. In the “Injected sample/conversion sequence” figure, changed “Convert” to “Evaluate”. Changed “INPSAMPLE” to “INPSAMP” throughout. In the “ADC sampling and conversion timing” section, revised the last equation. In the “Presampling sequence” figure, changed “Convert” to “Evaluate”. In the “Presampling sequence with PRECONV = 1” figure, changed “Convert” to “Evaluate”.

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Jun-2011	7 (cont.)	Chapter Boot Assist Module Added cut2 information, including: – Autobaud support – Test flash memory access Moved the “Single Chip boot mode” section to the “Device Boot Modes” section. Moved the second “Protocol” section to become a subsection of the “CAN Boot — Autobaud disabled” section. Chapter Clock Architecture Renamed the “System clock generation” section to “Clock generation”. Revised the “System clock generation” figure. Renamed the “System clock distribution” section to “Clock distribution” and revised its content. Chapter Clock Generation Module In the “MC_CGM memory map” table, changed the starting address of the last reserved entry (was 0xC3FE_0400, is 0xC3FE_03A4). Revised the following register figures (were 32-bit, are 8-bit): – CGM_OCDS_SC – CGM_SC_DC0 – CGM_AC0_DC0 – CGM_AC0_DC1 – CGM_AC1_DC0 – CGM_AC2_DC0 In the CGM_AC4_SC section, changed the text after the figure to state “This register is used to select the current clock source for the FMPLL1 reference clock”. In the CGM_SC_SS[SELSTAT] field description, changed the entry for 0010 (was reserved, is “4–40 MHz crystal osc.”). Chapter Clock Monitor Unit Changed occurrences of HFREFR to HFREFR_A to match the actual register and field names. Changed occurrences of LFREFR to LFREFR_A to match the actual register and field names.Revised the Overview section. Revised the clock names in the “Main features” section. Revised the “Functional description” section. Added the “Motor control clock monitor” and “FlexRay clock monitor” sections. In the CMU_CSR[CKSEL1] and CMU_CSR[RCDIV] field descriptions, revised the clock names. Revised the CMU_FDR[FD] field description. In the CMU_HFREFR_A[HFREF_A] field description, changed f_{RC} to F_{IRCOSC_CLK}. In the CMU_LFREFR_A[LFREF_A] field description, changed f_{RC} to F_{IRCOSC_CLK}. In the CMU_ISR field descriptions, revised the clock names and replaced “CK_FMPLL is ‘ON’ and the FMPLL locked as signaled by the MC_ME” with “The selected clock source is 'ON' (and locked in the case of FMPLL_0 and FMPLL_1)”. In the CMU_MDR[MD] field description, changed “IRC clock cycles” to “IRCOSC clock cycles”.

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Jun-2011	7 (cont.)	Chapter Cross-Triggering Unit In the “CLRx for ADC dual-conversion mode commands (CMS = 1)” section, changed bit 1 (was LC, is FC) and the associated field description. Chapter Crossbar Switch Added information to the “Register availability” section. Revised the “Logical master IDs” table. In the “XBAR slave port allocation” table, revised the entries for XBAR slave port S0. Chapter Cyclic Redundancy Checker Unit In the Introduction section, deleted the text “The number of contexts is a static parameter”. Deleted the duplicate register map. Added information about the 8-bit polynomial implemented on cut2. Added the CRC_OUTP_CHK register (cut2 only). Chapter Deserial Serial Peripheral Interface In the Features section, changed “Parameterized number of transfer attribute registers (2–4 registers)” to “4 transfer attribute registers”. Added a note to the “Slave Mode” section. Changed the reset value of DSPI_MCR[MDIS] for cut2 (was 0, is 1). Revised the information for the following registers to describe the implementation on cut1 and cut2 accurately: – DSPI_MCR – DSPI_HCR – DSPI_CTAR0 – DSPI_PUSHR – DSPI_SR – DSPI_RSER Changed the DSPI_SR[TXRXS] field from read-only to write-1-to-clear (w1c). Chapter Enhanced Motor Control Timer In the TAI[2:0] signal description, added the “eTimer module-to-module input signals” table and a cross-reference to the “CTU interaction with other peripherals” figure in the CTU chapter. In the “eTimer memory map” table, changed “CNHL goes from 0 to 7” to “CHNL goes from 0 to 5”. In the “eTimer memory map” table, added a footnote that the WDTOL and WDTOH registers are available only on eTimer_0. Added notes that the WDTOL and WDTOH registers are available only on eTimer_0. In the “Interrupt summary” table, changed the entries in the “Core interrupt” column.

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Jun-2011	7 (cont.)	Chapter Error Correction Status Module Editorial changes. Changed “AXBS” to “XBAR”. Revised the Introduction section. Added information for the PLAMC and PLASC registers. Renamed registers (and some associated fields) as follows: – Changed REV to PLREV – Changed IMC to IOPMC – Changed FEAR to PFEAR – Changed FEMR to PFEMR – Changed FEAT to PFEAT – Changed REAR to PREAR – Changed RESR to PRESR – Changed REMR to PREMR – Changed REAT to PREAT Changed “RAM” to “platform RAM”. Changed “Flash” to “platform flash memory”. In the PCT[PCT] field description, added the entryr “0xE446: e200z4d Power Architecture core”. In the MRSR section, changed the definition of bit 1 (was DIR, is OFPLR). In the MUDCR section, added a note clarifying the presence of MUDCR on ECSM_1 in DPM. Renamed the “ECC registers” section (is “Platform ECC registers” and added information about ECC reporting. Revised the ECR field descriptions. Revised the ESR section. Revised the note in the PRESR section. Changed “Platform XBAR” to “Platform Crossbar”. Revised the REDR definition (was one 32-bit register named REDR, is two 32-bit registers named PREDRL and PREDRH). Revised the FEDR definition (was one 32-bit register named FEDR, is two 32-bit registers named PFEDRL and PFEDRH). Added the text “If no flash memory ECC event is defined to be handled for this module, accesses to this register will terminate with an error” to the following register descriptions: – PFEAR – PFEMR – PFEAT – PFEDRL – PFEDRH Added the text “If no RAM ECC event is defined to be handled for this module, accesses to this register will terminate with an error” to the following register descriptions: – PREAR – PRESR – PREMR – PREAT – PREDRL – PREDRH

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Jun-2011	7 (cont.)	Chapter Fault Collection and Control Unit Editorial changes. Replaced EOUT with FCCU_F throughout the chapter to match the correct signal names. Replaced the “MC” abbreviation with the full abbreviation of the correct module. Editorial changes. Revised the Introduction section. In the “Main features” section: – Changed “External reaction (fault state): FCCU_F signaling” to “External reaction via fault-state dedicated system signals (FCCU_F pins)”. – Changed “Watchdog timer” to “Internal (to the FCCU) watchdog timer”. Deleted the “Standard features” section. In the “Block diagram” section: – Added a note about the differences between RCC and RCCU. – Changed “IPS system clock and RC oscillator clock” to “system clock and IRCOSC clock”. Revised the “Signal description” section to indicate that the FCCU does generate external signals (FCCU_F[0] and FCCU_F[1]). Changed “IPS bus interface” to “Register interface”. Revised the “Register interface” section. In the “Definitions” section, changed “resynchronized and latched” to “captured”. In the “FSM description” section, deleted “(irq_alarm_b)”. Revised the “Self-checking capabilities” section. In the “Reset interface” section, removed extraneous content and revised the “Reset sources” table. Revised the “Fault recovery” section. In the “NVM configuration” table, renamed the entries in the “Flash memory option bit locations” column. Revised the “WKUP/NMI” section. Revised the “STCU interface” section. Revised the “NVM interface” section. Revised the “FCCU_F interface” section. In the “FCCU mapping of critical faults” table, changed MCM to ECSM and replaced “rcc_out” with meaningful entries. In the “FCCU mapping of non-critical faults” table, changed MCM to ECSM. Replaced “safe clock” with “IRCOSC clock” throughout. Replaced ipg_clk_safe with IRCOSC throughout. In the FCCU_CFG[FOP] field description, changed RC_{16MHz} to F_{IRCOSC}. In the FCCU_CF_CFG0..3[CFCx] field description, deleted “(input fault fccu_cf[j])”. In the FCCU_NCF_CFG0..3[NCFCx] field description, deleted “(input fault fccu_ncf[j])”. In the FCCU_CFS0..3 section, deleted “(fccu_cf[j])” and “asserting a pulse on the fccu_cf_clr[x] signal”. In the FCCU_CFG_TO section, changed “RC oscillator clock” to “IRCOSC clock”. In the FCCU_CFG[FOP] field description, changed $EOUT_{freq}$ to F_{FCCU_F}. In the FCCU_STAT section, deleted “(ipg_clk)”. Revised the FCCU_CFF section. Revised the FCCU_NCFF section. In the FCCU_MCS section, changed “MC state” to “chip mode”.

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Jun-2011	7 (cont.)	Chapetr Flash Memory Renamed “C90FL flash memory functional description” to “C90FL flash memory functional description (user mode)” and revised the subsections under that section. Revised the “Flash memory block” section and subsections to indicate that only one NVSCI register is implemented, and added cut1 and cut2 definitions for this register. Revised the “Flash memory block” section and subsections to indicate that only one NVPWD register is implemented. Revised the “User option bits” table. Added the “Test flash memory” section. Chapter Flexible Motor Control Pulse Width Modulator Module In the Overview section, changed “one or more PWM submodules” to “4 PWM submodules” and “There may also be one or more fault channels” to “There are 4 fault channels”. In the “PWM block diagram” figure, deleted the FAULT4-7 block. Chapetr FlexRay Communication Controller Deleted the “FMPLL clocking” section. Chapter Frequency-Modulated Phase-Locked Loop In the Features section, changed “Progressive clock switching” to “Progressive clock switching (only on FMPLL_0 for the system clock)”. In the Recommendations section: – Changed “selected as system clock” to “selected as the source for an active clock on the chip”. – Changed “Use progressive clock switching” to “Use progressive clock switching for the system clock”. Chapter JTAG Controller Changed RDY_B to $\overline{\text{RDY}}$. Merged the information in the “Information specific to this device” section into the chapter body, and deleted the section. Added information about the TEST_CTRL register (supported on cut2 only). Revised the CENSOR_CTRL section to indicate that the register is 65 (not 64) bits wide. In the “JTAG Instructions” table, added entries for the NXSS modules (cut2 only).

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Jun-2011	7 (cont.)	Chapter Mode Entry Module In the “MC_ME memory map” table, added a footnote to the ME_PCTL0..143 registers. In the ME_GS[S_SYSCLK] and ME_<mode>_MC[SYSCLK] field descriptions, changed the entry for 0010 (was reserved, is “4–40 MHz crystal osc.”). Revised the “Protection of mode configuration registers” section. Editorial changes. In the “MC_ME memory map” table, added a footnote to the ME_PCTL0..143 registers. In the ME_GS[S_SYSCLK] and ME_<mode>_MC[SYSCLK] field descriptions, changed the entry for 0010 (was reserved, is “4–40 MHz crystal osc.”). Revised the “Protection of mode configuration registers” section (including adding cut-specific information). Revised the ME_ME, ME_IS, ME_IM, and ME_DMTS definitions to show the cut1 and cut2 configurations. In the ME_MCTL[TARGET_MODE] field description, changed the entry for 1111 (was reserved, is “reserved” for cut1 and “RESET (triggers a ‘destructive’ reset event)” for cut2). In the “Peripheral Control Registers” section, added a cut2-specific note. In the “RESET mode” section, replaced “ME_MCTL register is written with ‘0000’” with cut-specific text. In the “SAFE mode” and “STOP0 mode” sections, added cut2-specific notes. In the “Peripheral Clocks Disable” section, revised the caution to indicate that it applies to cut1 only and added a cut2-specific note. Added cut2-specific information to the “System clock switching” section. In the “MC_ME system clock selection overview” table, added a cut2-specific entry for “4-40 MHz crystal osc.”. In the “Current mode update” section, added a cut2-specific note. Chapter Nexus Crossbar Slave Port Data Trace Module (NXSS) [cut2 only] Added this chapter. Chapter Nexus Port Controller Changed EVTO_B to $\overline{\text{EVTO}}$. Revised the “Unavailable features” section to indicate the availability of MCKO clock gating on cut2. Added information about $\overline{\text{RDY}}$ pin support (cut2 only). Revised the “Nexus Double Data Rate mode” section to reflect the cut1 and cut2 implementation. Chapter Oscillators Revised clock names throughout the chapter. Added the “NVM interface” section. Chapter Peripheral Bridge Changed “Master Privilege Register” to “Master Protection Register”. Revised the “Memory map” section to include registers available on cut2 (MPROT8 and MPROT9). Revised the “MPROTn field reset values” table to reflect the correct implementation on cut1 and cut2.

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Jun-2011	7 (cont.)	Chapter Register Protection In the “SPC56ELxx register protection” table: – Revised some abbreviations and register names to match the rest of the document. – Revised the MC_CGM information to reflect the cut1 and cut2 implementation. – Added an entry for PCR132 (cut2 only). Chapter Reset Generation Module Revised the “MC_RGM block diagram” figure. In the “MC_RGM memory map” table, changed the reset value for RGM_FESS[SS_FCCU_SOFT] (was 0, is 1). In the “Register descriptions” section, added the note “Some fields may be read-only, and their reset value of ‘1’ or ‘0’ and the corresponding behavior cannot be changed”. In the RGM_FESS[SS_FCCU_SOFT] field description, deleted the description for 0. Revised the “Boot mode capturing” section. Added the “Software ‘destructive’ reset” information for cut2. Revised the RGM_FES, RGM_DES, RGM_FERD, RGM_DERD, RGM_FESS, and RGM_FBRE descriptions to reflect cut1 and cut2 implementation. Added cut-specific information to the text in the “Register descriptions” section. Chapter Self-Test Control Unit Revised the register information to reflect the correct availability of registers and fields on cut1 and cut2. In the “Block diagram and components” section, in the entry for “Collector”, deleted the text “(unless a stay-in-reset fault is encountered)”. Revised the “STCU register map” table. Revised the Introduction section. In the “Boot sequence phase 1: Self test” figure, deleted the last sentence in step 7. In the “The Safety Integrity Subsystem” section, changed “aware of the successful MBIST and LBIST’ to “aware of the MBIST and LBIST status”. In the “Integrity SW operations” section, changed “the expected and actual values within the LBIST MISR registers do not indicate an error” to “all MBIST and LBIST finished successfully and no further error is flagged”. In the “No reported errors” section, changed “STCU_LBS bits” to “STCU_LBS and STCU_LBE bits”. In the “STCU main features” section, changed “read-only register interface” to “register interface”. In the “Block diagram and components” section, revised the description of the watchdog timers. In the “Memory map and register definition” section, replaced the note with updated content. Revised the “LBIST partitioning for cut2” table. Addd the “MBIST partitioning” section. Changed the STCU_LB_MISREL and STCU_LB_MISREH registers to be read-only (were read-write). Revised the “Register conventions” section. Revised the STCU_ERR section. Revised the STCU_LBS, STCU_LBE, STCU_MBSL, and STCU_MBSH field descriptions. Added notes to the STCU_MBEL and STCU_MBEH registers. Added the “Self-test bypass and MBIST-only mode” section.

Table 768. Document revision history (continued)

Date	Revision	Document changes
10-Jun-2011	7 (cont.)	Chapter Semaphore Unit In the table in the SEMA4_RSTGT[RSTGMS] field description, deleted the entry for MLB and revised the entry for FlexRay (was master ID 6, is master ID 3). In the table in the SEMA4_RSTNTF[RSTNMS] field description, deleted the entry for MLB and revised the entry for FlexRay (was master ID 6, is master ID 3). Chapter Static RAM Labeled the “Reset effects on SRAM accesses” as applying to cut1 only. Chapter System Integration Unit Lite In the “SIUL block diagram” figure, changed “108” to “Number of GPIO”. Revised the Features section to reflect the correct number of GPI / GPIO pins. In the signal descriptions, changed “GPIO[0:107]” to “GPIO”. In the register descriptions, changed PCR107 to PCRn. In the register descriptions, changed GPDO... to GPDO and GPDI... to GPDI. Changed the MIDR1[PARTNUM] field description to include the correct value (was 0101_0110_0100_0110, is 0101_0110_0100_0011). Revised the MIDR2 reset values. In the PSMI register description, added a note clarifying the non-existence of PSMI43. Added examples for accessing GPDI and GPDO ports. Chapter Temperature Sensor Revised the “TSENS calibration constants” to include a cross-reference to the actual test flash memory locations of the constants. Chapter Wakeup Unit In the “WKPU block diagram” figure, changed PLATFORM to “CPU cores”. In the Features section, changed “from all NMI sources” to “from a detected NMI event”. Revised the “External signal description” section. In the NSR[NOVF0] field description, changed “NMI input 0” to “NMI pin”. Chapter Revision History In the “Changes between revisions 5 and 6” table, added an entry for changes in the Preface.
06-Jul-2011	8	Editorial changes and improvements. Changed cut2 to cut2/3. Changed IEEE-ISTO 5001-2001 to IEEE-ISTO 5001-2003. Chapter Preface Revised UART entry in the list of acronyms and abbreviated terms Added cut3 silicon version in “Information about different silicon versions”section

Table 768. Document revision history (continued)

Date	Revision	Document changes
06-Jul-2011	8 (cont.)	Chapter Introduction In Feature Details: – In the ADC section, updated the modes . – in the On-chip flash memory with ECC section, changed shadow sector to shadow block – In On-chip flash memory with ECC and in on-chip SRAM with ECC section, updated the precise information of wait states. – In the “Peripheral bridge (PBRIDGE)“ section, changed “Master access right” to “Master access privilege level”. `Changed the "Periodic Interrupt Timer Module (PIT)" title to "Periodic Interrupt Timer (PIT Module)".` Updated to correct the consistency of the Flow A content section and the Flow B Rev. History section." Chapter Signal Description In System_Pins_144_257: – Added Note regarding Open Drain Enable after table . – Added description to RESET pin In the Pin_Muxing_144_257: – Added Note about Open Drain after the table . – Change all entries of column 'Weak pull config during reset’ to ' - ' , except for PCR[2], PCR[3], PCR[4] and PCR[21] – Editorial Changes Chapter Operating Modes Changed ‘shadow sector’ to ‘shadow block’ in ‘Selecting LSM or DPM section’ Chapter Device Boot Modes Updated the text in Reset Configuration Half Word Section Replaced the complete text of “Single Chip Boot Mode” section Chapter Analog-to-Digital Converter In the “ADC sampling and conversion timing” section, revised the third equation Updated the field description of STAW0R, STAW3R and STAW4R Added external channels information in Introduction section Deleted “[gnd]” in CTU mode section Chapter Boot Assist Module Added a note in “Download start address, VLE bit and code size” section. Revised the Boot mode selection flowchart. Section “Shadow Flash Code Improvements and Extensions” is applicable only for cut2 and not on cut3 as exception. Added text after Note in Shadow Flash Code Improvements and Extensions section Replaced BookE code by Power Architecture Technology Added Note in section “Boot with Auto Baud feature [cut2/3 only]” Chapter Clock Generation Module Deleted the 32- bit register CGM_OCDS_SC (for cut1) and updated the title of 8 -bit register by removing cut2

Table 768. Document revision history (continued)

Date	Revision	Document changes
06-Jul-2011	8 (cont.)	Chapter Clock Monitor Unit Added the field description for CMU_CSR[CKSEL1] for cut1 and cut2/3 Chapter Cross-Triggering Unit Revised the CTU Interaction with other peripherals diagram Added Conversion time evaluate section Added functional description of CTU_EXP_A ,CTU_EXP_B and CTU_CNTRNG registers Chapter Crossbar Switch Added Reset values of bit 0 to 15 of Master Priority Register Deleted Notes “Thes reset value for this register is device dependent “ in Master Priority Register Figure Chapter Cyclic Redundancy Checker Unit Revised the fields of CRC_CFG register Deleted CRC_OUTP_CHK [cut2 only] register Chapter Deserial Serial Peripheral Interface Added “SLAVE_ “ prefix to all the fields in field description of DSPI_CTAR0, DSPI_CTAR1 in slave mode register (Table 149) Chapter Enhanced Motor Control Timer Revised STS[WDF] field description Added DREQn_EN bit in DREQ0/DREQ1 regsiter Incorporated ERRATA ERR003256 as follows: - Added Note after GATED-Count Mode Chapter Error Correction Status Module Revised the reset values of PLAMC register to 0x006F and PLASC register to 0x8085 Chapter Fault Collection and Control Unit Renamed the fields in bit 20 -31of FCCU_CFG register by adding initials FCCU_CFG to the existing ones. Renamed “ipg_clk” to “system clock” in FCCU top-level diagram. Updated FCCU mapping for critical faults table and FCCU mapping for non-critical faults table. Added a Note after FCCU CF Status Register giving the information about the bit usage in FCCU_CFS 0..3 register. Added footnote in FCCU Mapping of non-critical faults table telling about the bypass feature

Table 768. Document revision history (continued)

Date	Revision	Document changes
06-Jul-2011	8 (cont.)	Chapter Flash Memory Revised additional wait states in Additional wait-state encoding table. Added a note in Wait-State emulation section Swapped the fields of NVSCI register for SPC56ELxx cut1 and SPC56ELxx cut2/3 Added NVPWD1 register Added PFCR0 settings for different frequencies table in PFCR0 section. Deleted "The read state is active when STOP=0 in the MCR (User mode read)" from C90 Read and Write section. Incorporated ERRATA ERR001433 as follows: - Added text in UM0 register description Updated the Word name column and Function column in Test Flash Information table Chapter FlexCAN Changed "_INT" bits in ESR register to w1c type from read-write. Incorporated ERRATA ERR002360 as follows: - Added Precautions When Using Global Mask and Individual Mask Registers section after RxFIFO section - Deleted text after RXGMASK section that was added in previous draft to incorporate errata Chapter Flexible Motor Control Pulse Width Modulator Module Updated the reset value of FSTS register to 0X0F0F Updated Core Interrupt column in Interrupt Summary Table Incorporated errata ERR003388 as follows: - Added a paragraph in Note of Value Register1 Chapter FlexRay Communication Controller Changed the reset value of bit field "BANK" of ECC Error Report Address Register from 111 to 000 Incorporated errata ERR002421 as follows: - Added text in Application Transitions section (Section 26.6.6.2.3.1 and 26.6.6.3.1.1) Incorporated errata ERR002423 as follows: - Updated Channel Assignment Description Table Incorporated errata ERR001322 as follows: - Updated the field description of EIE of FR_EERICR Incorporated errata ERR001364 as follows: - Added Note after Slot Status Registers. Incorporated errata ERR001369 as follows: - Added footnote to SBCF_EF and ILSA_EF field of FR_CHIERFR Chapter Frequency-Modulated Phase-Locked Loop Changed field name in MR register from FM_EN to SSCG_EN Marked 'unlock_once' field in CR to Reserved Chapter Interrupt Controller Revised the source signal column in "Interrupt sources for INTC_0 and INTC_1" table.

Table 768. Document revision history (continued)

Date	Revision	Document changes
06-Jul-2011	8 (cont.)	Chapter JTAG Controller Added 11-14 bits to TEST_CTRL register Updated the reset values of PRN, DC and PIN of Device Identification Register for cut1, cut2 , cut3 Chapter Lin Controller In DMATXE and DMARXE registers , changed bits 16 to 30 as readonly Chapter Mode Entry Module Revised Note in Peripheral Control Registers as general and not cut-specific information . Updated the RESET bits of ME_ME register for cut1 and cut2/3 Changed cut2 to cut2/3 accross the chapter Chapter Nexus Crossbar Slave Port Data Trace Module (NXSS) [cut2/3 only] Updated the read address and write address of DC2 in Register Available in NXSS Programmer Model Chapter Nexus Port Controller Removed “and ipg_sync_mcko” from section MCKO Removed “ Number of blocks that inputan EVTO” and its value from Device Parameter Value Added Note in TCK- Test Clock Input section Deleted bit 23 NEXCFG from PCR Incorporated errata ERR003116 as follows: - Added footnote to TCK in NPC Signal Properties table - Added text in section TCK- Test Clock Input Chapter Power Management Unit Revised the field description CTB of PMUCTRL_STATUS register. Changed reset value of PMUCTRL_FAULT register. Added Table Note in NCF[15],NCF[16], NCF[17] in Fault Assertion Conditions Table Updated Cv1v2 values in Board-level schematic example Chapter Register Protection Deleted duplicate FlexRay registers from Register Protection Table. Chapter Reset Generation Module In the “MC_RGM memory map” table, changed the reset value for RGM_FESS[SS_FCCU_SOFT] (was1, is 0) Chapter Sine Wave Generator Added footnote to NRO field of the register SWG_CTRL “For cut1, this field must be set to 1 by software before using the SWG. This field is reserved for cut2.

Table 768. Document revision history (continued)

Date	Revision	Document changes
06-Jul-2011	8 (cont.)	Chapter System Integration Unit Lite In IFMC0-IFMC31 register description, corrected the formula of Filter Period Changed the number of PCR registers from PCR (107) to PCR (132) and also updated address offset Added Reset values of Major_ Mask and Minor Mask for MIDR 1 for cut1,cut2 and cut3 Changed the reset value of IBE from '1' to '0' and added footnote to this reset value
19-Mar-2012	9	Throughout Editorial changes and improvements. Changed cut2 to cut2/3. Changed IEEE-ISTO 5001-2001 to IEEE-ISTO 5001-2003. Changed XTALIN to XTAL and XTALOUT to EXTAL. Chapter Preface In the section *1.4Information about different device versions ("cuts")*: • Added the cut3 silicon version. • Added the mask identification corresponding to each cut. In the section *1.5Acronyms and abbreviations*, removed RTL from the table. Chapter Introduction In *Section 2.1, The SPC56ELxx microcontroller*, added footnote to ISO26262 ASILD text that all functional safety statements are under condition that the requirements in the Safety application Guide are followed. In *Table 1 (SPC56ELxx device summary)*: • Changed "Code/data flash" to "Flash". • Updated the eTimer footnote to reflect eTimer_2 external accessability. In *Section 2.5.7, Platform flash memory controller*, removed the bullet detailing Data Flash. In *Section 2.5.14, Frequency-Modulated Phase-Locked Loop (FMPLL)*, changed "Reduced frequency divider (RFD)" to "Output divider (ODF)". In *Section 2.5.28, Serial communication interface module (LINFlexD)*: • Changed configurable break duration of "up to 36-bit tmes" to be "up to 50-bit times". Changed configurable word length of "16-bit" to be "16-bit or 17-bit".

Table 768. Document revision history (continued)

Date	Revision	Document changes
19-Mar-2012	9 (cont.)	Chapter Signal Description In *Figure 4 (LFBGA257 ballmap)*, updated the VDD and VSS pin names (using shortened names to fit inside the cells) to match those in *Table 9 (LFBGA257 pin function summary)*: • Changed VSS to VSS_HV_IO for pins A1, A2, A16, A17, B1, B2, B9, B17, C3, C15, D9, H2, N2, P9, R3, R15, T1, T17, U1, U2, U16, U17. • Changed VSS to VSS_LV_COR for pins D5, G7, G8, G9. G10, G11, P6, P13. • Changed VSS to VSS_LV for pins H7, H8, H9, H10, H11, H14, J7, J8, J9, J10, J11, K7, K8, K9, K10, K11, L7, L8, L9, L10, L11. • Changed VSS to VSS_HV_FLA for pin J16. • Changed VSS to VSS_HV_OSC for pin P1. • Changed VSS to VSS_HV_ADV for pin U10. • Changed VDD_HV to VDD_HV_IO for pins A3, A9, B16, C1, D8, D14, G2, M2, P10, P14, T2, T16. • Changed VDD_HV to VDD_HV_REG_2 for pins C7, C8. • Changed VDD_LV to VDD_LV_COR for pins D6, F6, F7, F8, F9, F10, F11, F12, G6, G12, P5, P12. • Changed VDD_HV to VDD_HV_REG_1 for pins H15, J15. • Changed VDD_HV to VDD_HV_FLA for pin H16. • Changed VDD_HV to VDD_HV_REG_0 for pins J3, J4. • Changed VDD_HV to VDD_HV_OSC for pin M1. • Changed VDD_HV to VDD_HV_ADV for pin U9. • Changed VDD_HV to VDD_HV_PMU for pin U14. • Changed VREFP_HV_AD0 to VDD_HV_ADR0. • Changed VREFP_HV_AD1 to VDD_HV_ADR1. • Changed VREFN_HV_AD0 to VSS_HV_ADR0. • Changed VREFN_HV_AD1 to VSS_HV_ADR1. In *Table 8 (LQFP144 pin function summary)* and *Table 9 (LFBGA257 pin function summary)*, changed EVTI from output function to input function. In *Section 4.3, System pins*, added footnote stating that RESET is valid only when the 3.3V supply is stable. In *Section 4.3, System pins*, changed the direction for EXTAL from “Output Only” to “Input/Output”, and added a note outlining the conditions for each case. In *Table 12 (Pin muxing)*, changed “LINFlex” to “LINFlexD” for pins B2, B3, B7, and B13. Chapter Operating Modes In *Section 5.2, Lock Step Mode (LSM)*, added reset initialization information to the end of the section.

Table 768. Document revision history (continued)

Date	Revision	Document changes
19-Mar-2012	9 (cont.)	Chapter Analog-to-Digital Converter In *Section 9.1, Introduction*: • Replaced the initial paragraph with new content. • Added a note that channel 9 on ADC_0/1 is reserved for factory test. In *Section 9.2, Features*, added the list of features and a footnote to the end of the section. In *Section 9.3.2.2, Main Status Register (MSR)* and *Table 60 (Test channel activation)*, removed the references to CTU trigger mode. In *Section 9.3.3.1, Interrupt Status Register (ISR)*, added text explaining the internal routing of interrupt signals. In *Section 9.3.5, Watchdog Threshold Interrupt Mask Register (WTIMR)*, added text explaining the internal routing of interrupt signals. In *Section 9.3.9.1, Presampling Control Register (PSCR)*, added the PREVAL1 field to the graphic and descriptions. In *Table 42 (STCR1 field descriptions)*, added the valid self-test values to the field description entries for INPSAMP_C, INPSAMP_RC, and INPSAMP_S. In *Section 9.3.17.5, Self Test Status Register 1 (STSR1)*, added text explaining the internal routing of interrupt signals In *Section 9.4.2.3, Normal conversion operating modes*: • In *Example 1*, added a sentence at the end regarding additional chain execution after NSTART is cleared. • Added a sentence at the end of the section regarding clearing of ISR.EOC and CEOCFR.EOCCHx. Updated the mode information in *Section 9.4.2.4, Injected channel conversion*. In *Section 9.4.3, Analog clock generator and conversion timings*, deleted the description regarding the internal divider. In *Section 9.4.4, ADC sampling and conversion timing*, updated the text and equations with regard to conversion and sampling times and clock definitions. In *Section 9.4.5, Presampling*, added note regarding open fault detection. Changed “lies in the range” to “does not lie in the range” in the Algorithm S STEP1 discussion in *Section 9.4.11.5, Self test analog watchdog*. Chapter Boot Assist Module In *Section 10.4.5, Boot with Autobaud feature [cut2/3 only]*, added a note that autobaud is supported only on cut3 devices that have uncensored flash. Chapter Clock Architecture In *Figure 69 (System clock generation)*, added notes explaining PLL clock outputs.

Table 768. Document revision history (continued)

Date	Revision	Document changes
19-Mar-2012	9 (cont.)	Chapter Cross-Triggering Unit In *Section 14.4.1, Interaction with other peripherals*, added the description of the input signals. In *Section 14.4.2, Trigger events features*, added a bullet regarding the minimum time between triggers. In *Section 14.5.2, ADC commands list format*: • Added the sentence “The CTU FIFO order is non-deterministic.” • Added text noting that FIFO order may be reversed under certain conditions. In *Section 14.5.2, ADC commands list format*, *Table 107 (CLRx (CMS = 0) field descriptions)*, and *Table 108 (CLRx (CMS = 1) field descriptions)*, changed “ADC unit A” to “ADC unit 0” and “ADC unit B” to “ADC unit 1”. In *Section 14.5.3, ADC results* and *Table 94 (CTU interrupts)* footnotes, changed FIFO 1/2/3/4 to FIFIO 0/1/2/3 for consistency. In *Section 14.5.3, ADC results*, updated the definitions of “Unsigned right-justified” and “Signed left-justified” to reflect 12-bit resolution. Removed all content related to the MDIS bit in *Section 14.7, Power safety mode*. In *Section 14.8.1, DMA support*: • Changed “reaches the threshold value” to “exceeds the threshold value”. • Added restriction that threshold value must be less than the number of FIFO entries. In *Section 14.10, Memory map*: • Removed the introductory text regarding 8-, 16-, and 32-bit register accesses. • The format of the register figures was modified for consistency. In *Table 99 (TGSISR Field descriptions)*: • Changed “eTimers2” to “eTimers1 ETIMER1_IN”. • Changed “eTimers1” to “eTimers0 ETIMER0_IN”. • For I5_FE, changed “even” to “odd”. • For I5_RE, changed “even” to “odd”. • For I1_FE, changed “odd” to “even”. • For I1_RE, changed “odd” to “even”. In *Figure 113 (Trigger handler control register 1 (THCR1))* and *Figure 114 (Trigger handler control register 2 (THCR2))*, changed all Tn_T4E and Tn_T3E bits to reserved bits. In *Figure 116 (Commands list register x (x = 1,...,24) (CMS=1))*, changed FC to LC. In *Table 108 (CLRx (CMS = 1) field descriptions)*, changed FC and its description to LC and LC description. In *Section 14.10.9.1, CLRx for ADC single-conversion mode commands (CMS = 0)*: • Added the ST bit to the register figure. • Added the ST (Self-Test) entry to the field description table. In *Figure 115 (Commands list register x (x = 1,...,24) (CMS=0))*, *Figure 116 (Commands list register x (x = 1,...,24) (CMS=1))*, and *Figure 117 (Commands list register x (x = 1,...,24) (ST1 = 0))*, added the following footnote: “This bit may be called FC or LC depending on the device, but the functionality is identical“. In *Section 14.10.9.2, CLRx for ADC dual-conversion mode commands (CMS = 1)*, changed reserved bit 6 to be part of the CH_B field. In *Section 14.10.9.3, CLRx for self-test commands (ST1 = 0)*, changed the RFU (Reserved for Future Use) bit to be simply reserved and always 0. In *Table 109 (CLRx (ST1 = 0) field descriptions)*, corrected the table title “CMS = 1” to be “ST1 = 0“. Added *Table 110 (CLRx addresses)*. In *Figure 118 (Cross triggering unit error flag register (CTUEFR))*, added the CS bit. In *Table 111 (CTUEFR field description)*: • Added CS and ERR_CMP entry and description. • Added value descriptions to all table entries.

Table 768. Document revision history (continued)

Date	Revision	Document changes
19-Mar-2012	9 (cont.)	Chapter Cross-Triggering Unit (continued) In *Section 14.10.11, Cross triggering unit interrupt flag register (CTUIFR)*, updated the register figure and field descriptions table. In *Section 14.10.12, Cross triggering unit interrupt/DMA register (CTUIR)*, updated the field descriptions table entries. Added clarifying text to the COTR entry in *Table 114 (COTR field description)*. Added clarifying text to all entries in *Table 115 (CTUCR field descriptions)*. Changed field name from “N” to “FILTERVALUE” in *Section 14.10.15, Cross triggering unit digital filter (CTUDF)*. In *Section 14.10.20, FIFO control register (FCR)*, updated the field descriptions table entries. In *Section 14.10.21, FIFO threshold (FTH)*: • Changed register graphic name from “FIFO control register (FCR)” to “FIFO threshold (FTH)”. • Added clarifying text to the FIFO 0-3 descriptions. In *Section 14.10.22, FIFO status register (FST)*, updated the field descriptions table entries. Chapter Crossbar Switch In *Section 15.1.4, Logical master IDs*, added a note that Logical Master IDs 4 and 5 are not used. Chapter eDMA Channel Mux In *Table 173 (DMA_MUX source slot mapping)*, changed “LINFlex” to “LINFlexD” for sources 22 through 27. Chapter Enhanced Motor Control Timer In *Section 21.1, Introduction*, udpated the footnote to indicate that eTimer_2 is only available internally on the 144 LQFP package. In *Section 21.3.2, TAI[2:0] - Timer Auxiliary Inputs*, added a cross-reference to the description in the CTU chapter. In *Figure 237 (Control Register 3 (CTRL3))*: • Changed C2FCNT[2:0] to C2FCNT[1:0]. • Changed C1FCNT[2:0] to C1FCNT[1:0]. In *Section 21.4.3.12, Interrupt and DMA Enable Register (INTDMA)*, removed channel 6 from the RCFIE bit description. In *Figure 242 (Compare and Capture Control Register (CCCTRL))*: • Changed CFWM[1:0] to CFWM. • Changed the CFWM bit to be read/write. Updated the content in *Section 21.5.2.15, MODULO COUNTING Mode*. Chapter Error Correction Status Module In *Table 236 (MUDCR field descriptions)*, referred the user to the device data sheet for wait-state information relative to clock frequency. In *Section 22.4.2.10, ECC Status Register (ESR)*, added a note regarding functional reset assertion and the ECSM_ECCRNCR_IRQ and ECSM_ECCFNCR_IRQ interrupts. In *Table 239 (ESR field descriptions)*, changed “PPR1BC“ to “PR1BC“.

Table 768. Document revision history (continued)

Date	Revision	Document changes
19-Mar-2012	9 (cont.)	Chapter Fault Collection and Control Unit In *Table 252 (FCCU memory map)*: • Added the entry for the FCCU_EINOUT register. • Added the FCCU SC Freeze Status Register (FCCU_SCFS). In *Section 23.6.8, FCCU CF Status Register (FCCU_CF_S0..3)*, changed FCCU_CFS0_3 to FCCU_CF_S0_3 In *Section 23.6.10, FCCU NCF Status Register (FCCU_NCF_S0..3)*, changed FCCU_NCFS0_3 to FCCU_NCF_S0_3 In *Section 23.6.12, FCCU NCF Enable Register (FCCU_NCF_E0..3)*, changed FCCU_NCFE0_3 to FCCU_NCF_E0_3 Added *Section 23.6.16, FCCU IO Control Register (FCCU_EINOUT)*. Added *Section 23.6.18, FCCU SC Freeze Status Register (FCCU_SCFS)*, including the associated figure and table. In *Section 23.6.21, FCCU IRQ Status Register (FCCU_IRQ_STAT)*, updated the field name “NMI_STATUS” to be “NMI_STAT” in the register figure. In *Section 23.7.3, Self-checking capabilities*, added sentence regarding clearing of NOK to the end of the section. In *Section 23.7.10, FCCU_F interface*, changed “In case of inverted polarity (FCCU_CFG.FOP = 1b)" to "In case of inverted polarity (FCCU_CFG.PS = 1b)”. Chapter Flash Memory Moved the register definition section to appear before the functional description section. The order is now *Section 24.1.5, C90FL memory map and register definition* followed by *Section 24.1.6, C90FL flash memory functional description (user mode)*. In *Table 286 (C90FL flash memory map)*, changed location 0x00F0_3E08 from “BIU3” to “Reserved”. In *Table 287 (Register memory map)*: • Moved reserved space for “FLASH_REGS_BASE + 0x20” to its correct place in the map. • Added reserved locations 0x28 and 0x30-0x38. In *Figure 327 (LMS Register)*, updated the LSEL field and the adjacent Reserved field. In *Table 293 (LMS Field Descriptions)*: • Updated the field description table bit numbering for the LSEL field and the adjacent Reserved field. • Added detailed information on each bit to the MSEL and LSEL field descriptions. • Added information that bits:[5:0] correspond to LAS blocks L5-L0 in the LSEL field description. In *Figure 328 (HBS Register)*, updated the HSEL field and the adjacent Reserved field. In *Table 294 (HBS Field Descriptions)*: • Corrected the field description table bit numbering for the HSEL field and the adjacent Reserved field. • Added detailed information on each bit to the HSEL field description. In *Figure 330 (UT0 Register)*, updated the EA bit 7 to reflect the reset values for cut1, cut2 and cut 3. In *Table 296 (UT0 Field Descriptions)*, changed UT0 bit 7 function from Reserved to EA for Cut2/Cut3. In *Table 309 (Test flash information)*: • Corrected the constant names from ‘P2’ and ‘C2’ to ‘P1’ and ‘C1’ respectively in Function column for Address 0x0008 and 0x000C. • Added more description to addresses 0x0010 to 0x0054. Updated the text in *Section 24.2.2, Registers*.

Table 768. Document revision history (continued)

Date	Revision	Document changes
19-Mar-2012	9 (cont.)	Chapter Flash Memory (continued) In *Section 24.2.2.1, Platform Flash Configuration Register 0 (PFCR0)*: • Corrected the name of field “B0_P0_BCFG“ to be “B02_P0_BCFG“ in the register figure. Updated the value of RAM Wait State from ‘0’ to ‘1’ for frequency <=120 MHz in *Table 311 (PFLASH Configuration Register 0 (PFCR0) settings for different frequencies)* Chapter FlexCAN Deleted text from the Note in *Section 25.1.2, FlexCAN module features*. In *Section 25.3.4.2, Control Register (CTRL)*, added a note that freeze mode is required prior to entering STOP mode. In *Section 25.4.8.4, Protocol Timing*, added text that states the units for Information Processing Time (IPT) to the note at the end of the section. Chapter Flexible Motor Control Pulse Width Modulator Module In *Section 26.1.1, Overview*, added a cross-reference to the description in the CTU chapter. Removed all references to the unimplemented FRACx register. Sections affected: • *Section 26.3.4.4, Reload errors*; • *Section 26.4.3.12, Status Register (STS)*, in the RUF field; • *Section 26.4.3.14, DMA Enable Register (DMAEN)*, in the VALDE field; • *Section 26.4.4.5, Master Control Register (MCTRL)*, in the LDOK field. In *Section 26.4.3.12, Status Register (STS)*, added text to the CMPF field description identifying that bits[5:0] correspond to VAL5-VAL0. In *Section 26.4.3.13, Interrupt Enable Register (INTEN)*, added text to the CMPIE field description identifying that bits[5:0] correspond to VAL5-VAL0. In *Section 26.4.3.15, Output Trigger Control Register (TCTRL)*, added text to the OUT_TRIG_EN field description identifying that bits[5:0] correspond to VAL5-VAL0. In *Section 26.4.3.16, Fault Disable Mapping Register (DISMAP)*, added text identifying that bits[3:0] in each register field correspond to FAULT3-FAULT0. In *Section 26.4.3.17, Deadtime Count Registers (DTCNT0, DTCNT1)*, updated the following text: • Changed “12-bit values” to “11-bit values”. • Changed “default value of 0x0FFF” to “default value of 0x07FF“. • Changed “deadtime of 4095” to “deadtime of 2047”. In *Section 26.4.4.1, Output Enable Register (OUTEN)*, *Section 26.4.4.2, Mask Register (MASK)*, and *Section 26.4.5.2, Fault Status Register (FSTS)*, added a noted stating that “Field[3:0] refers to subumodule 3,2,1,0”. In *Section 26.4.4.5, Master Control Register (MCTRL)*, added description identifying which submodule corresponds to each bit in the register fields. In *Section 26.4.5.1, Fault Control Register (FCTRL)*, added text identifying that bits[3:0] in each register field correspond to FAULT3-FAULT0. Chapter Frequency-Modulated Phase-Locked Loop In *Section 28.3, Features*, changed “Reduced frequency divider (RFD)“ to “Output divider (ODF)“. In *Section 28.7, Requirements*, added “(External oscillator) and IRCOSC (Internal RC oscillator)“ to the sentence that addresses maximum system frequency.

Table 768. Document revision history (continued)

Date	Revision	Document changes
19-Mar-2012	9 (cont.)	Chapter Interrupt Controller In *Table 480 (Interrupt sources for INTC_0 and INTC_1)*, changed “LINFlex” to “LINFlexD” throughout the table. Chapter JTAG Controller In *Section 30.3.1.3, Device Identification Register*, added the major and minor maskset IDs for cut3.1. Chapter LIN Controller In *Section 31.4, LINFlexD and software intervention*, changed “kbps” to “Kbit/s”. Added *Section 31.7.1.5, Overrun*, which addresses the case in which the message buffer is overfilled in Master mode. In *Section 31.7.2.2, Data reception (transceiver as subscriber)*, changed “clearing” to “setting” in the filter configuration sentence. Replaced *Section 31.7.2.7, Overrun* with updated content. In *Section 31.7.3, Slave mode with identifier filtering*, changed “IFCR7” to “ICFR15”. In *Section 31.9.4, UART receiver*, changed “WL1” to “WL[1]”. In *Table 490 (BDRM access in UART mode)*, changed “WL1” to “WL[1]” and changed “WL0” to “WL[0]” in the table footnote. In *Table 491 (UART receiver scenarios)*, changed “UARTSR[FE]“ to “UARTSR[FEF]“. In *Figure 639 (LIN timeout control status register (LINTCSR))*, removed the footnote from the TOCE bit. In *Table 502 (LINTCSR field descriptions)*, added “and software cannot modify it” to the TOCE bit initialization mode description. In *Section 31.10.8, LIN output compare register (LINOCR)*, changed “LINTCSR[LTOM] = 1“ to “LINTCSR[LTOM] = 0“ in the register footnote. In *Section 31.10.13, LIN control register 2 (LINCR2)*, WURQ, DDRQ, DTRQ, ABRQ and HTRQ field are changed from “w1c” to “rw and cleared by hardware” (added footnote). In *Section 31.10.14, Buffer identifier register (BIDR)*, DIR and CCS fields are changed from “w1c” to “rw”. In *Section 31.10.17, Identifier filter enable register (IFER)*: • Changed the FACT field from 8 bits to 16 bits. • Removed the “IFER[FACT] configuration“ table. In *Figure 650 (Identifier filter match index (IFMI))*, changed the IFMI field from 4 bits to 5 bits. In *Figure 651 (Identifier filter mode register (IFMR))*, changed the IFM field from 4 bits to 8 bits. Added *Table 516 (IFMR[IFM] configuration)*, that explains how to set the filters. In *Section 31.12.1.1, LIN timeout mode*, changed “If the LINTCSR[LTOM]=1” to “If the LINTCSR[LTOM]=0”. In *Section 31.12.1.2, Output compare mode*, changed “the LTOM bit must be reset” to “the LTOM bit must be set”. Chapter Periodic Interrupt Timer In *Section 37.5.1, Example Configuration*, changed “PITCTRL“ to “PITMCR“. Chapter Peripheral Bridge In *Section 38.4.2, Memory map*, changed the MPROT4 and MPROT5 fields to “Reserved”. Chapter Power Control Unit In *Table 607 (MC_PCU Memory Map)*, updated the memory map offsets.

Table 768. Document revision history (continued)

Date	Revision	Document changes
19-Mar-2012	9 (cont.)	Chapter Power Management Unit Updated *Section 40.3, High-power regulators*. Corrected the reserved addresses in *Table 611 (PMU memory map)*. Chapter Reset Generation Module In *Section 42.3.1.1, Functional Event Status Register (RGM_FES)*, updated the infromation for the F_CORE bit. In *Section 42.3.1.2, Destructive Event Status Register (RGM_DES)*, replaced the NOTE regarding F_POR flag behavior. In *Figure 794 (Functional Event Reset Disable Register (RGM_FERD))*, added a footnote that certain bits are read-only. Chapter Self-Test Control Unit In *Section 43.5, LBIST partitioning*, updated the LBIST partitioning table for cut2/3. In *Section 43.7, Self-test bypass and MBIST-only mode*, added ShadowFlash Hex code for BYPASS and MBIST mode in Self-test bypass and MBIST-only mode. Chapter Sine Wave Generator In *Section 45.2, Features*, changed peak-to-peak amplitude from “0.47 - 2.26V“ to “0.426 - 2.063V“. Renamed table by adding “(approximate)” to its name; this is now *Table 714 (Sine wave amplitude (approximate) as a function of SWG_CTRL[IOAMPL])*. Updated all values in *Table 714 (Sine wave amplitude (approximate) as a function of SWG_CTRL[IOAMPL])*. Chapter SRAM In *Section 47.1, Introduction*, added feature bullet regarding SRAM wait-states. In *Section 47.6, Initialization and Application Information*, added note regarding default SRAM wait state setting. Chapter System Integration Unit Lite In *Table 727 (SIUL memory map)*, changed the PCR ending address from 0x116 to 0x148, and changed the subsequent beginning reserved address from 0x118 to 0x150. In *Table 728 (MIDR1 Field Descriptions)*, added the major and minor maskset IDs for cut3.1. In *Section 48.5.2.8, Pad Configuration Registers (PCR0–PCR132)*, added footnotes to the register figure detailing the reset values of the PA, OBE and SRC fields. Chapter System Status and Configuration Register Deleted the Password Comparison Registers (PWCMPH and PWCMPL). In *Table 746 (STATUS field descriptions)*, updated the description of the BMODE field. Chapter Temperature Sensor In *Table 764 (TSENS calibration constants)*, updated all descriptions. In *Section 51.5.2, Equations for converting TSENS voltage to device temperature*, updated the equations for device temperature calculation.

Table 768. Document revision history (continued)

Date	Revision	Document changes
19-Mar-2012	9 (cont.)	Appendix In revision history for Rev. 8, revised the following entries : • Introduction (deleted the Flow content A and Flow B) • Signal Description (deleted editorial changes) • Fault Collection and Control Unit (added the reset value of FCCU_CFG.FOP to 0x3F) • Mode Entry Module (deleted cut2/3 accross the chapter) • Power Management Unit (deleted the Board Level Schematic Example section) • Reset Generation Module (renamed the bit1 of RGM_DES register figure for cut2/3 from F_SOFT_DESC to F_SOFT_DEST) • Sine Wave Generator (changed cut2 to cut2/3 and added closing quote mark) • System Integration Unit Lite (added footnotes in Pad Configuration Register) • Temperature Sensor (updated temperature in TSENS calibration constants section and Equations for converting TSENS voltage to device temperature)
05-Jul-2013	10	Throughout Editorial changes and improvements. Changed AIPS to PBRIDGE. Chapter Introduction In *Section 2.5.6, On-chip SRAM with ECC*, updated wait state frecquencies from “<=120 MHz” to the range “> 80 MHz and <=120 MHz”. In *Section 2.5.30, FlexPWM*, modified text to update the number of FlexPWM that is implemented in the device. Chapter Memory Map In *Table 3*, SPC56ELxx memory map, ordered by start address and *Table 4*, SPC56ELxx memory map, ordered by module name, updated shadow block size of Flash memory from 1024 KB to 16 KB. Chapter Signal Description In *Table 8*, LQFP144 pin function summary, updated FCCU_F[0] and FCCU_F[1] as output functions only. Chapter Device Boot Modes In *Section 6.3, Boot-sector search*, added note regarding Boot-sector search. Chapter Functional Safety In *Section 8.3.1, BIST during boot*, updated information to add and clarifiy external reset (long) and no stay in reset.

Table 768. Document revision history (continued)

Date	Revision	Document changes
05-Jul-2013	10 (cont.)	Chapter Analog-to-Digital Converter In *Section 9.3.10.1, Conversion Timing Register 0 (CTR0)* and *Section 9.3.10.2, Conversion Timing Register 1 (CTR1)*, updated description for CTR0 and CTR1 respectively. In *Section 9.4.2.3, Normal conversion operating modes*, editorial change done. In *Section 9.4.4, ADC sampling and conversion timing*, updated the text and equations with regard to conversion and sampling times and clock definitions. In *Section 9.4.8, Interrupts*, updated text from, “The watchdog interrupt source sets two pending bits WDGxH and WDGxL in the WTISR for each of the four channels being monitored” to “The watchdog interrupt source sets two pending bits WDGxH and WDGxL in the WTISR for each of the channels being monitored”. In *Section 9.4.11.2, CPU mode*, updated the text for the scan mode sequence. In *Figure 11*, Main Status Register (MSR) and *Table 19*, MSR field descriptions, added SELF_TEST_S field to ADC MSR register. In *Table 18*, MCR field descriptions, updated ADCLKSEL field desription, changed system clock freqency instances to motor control clock frequency. In *Table 25*, WTIMR field descriptions, updated description of MSKWDGnH field. In *Table 29*, PSCR field descriptions updated description for PREVAL1 and PREVAL0. In *Table 30*, Presampling voltages, updated column header to “PSCR[PREVAL0/1] voltage label”. In *Table 62*, CTU-CLRx-ALG1, ALG0 meaning, updated Algorithm C; ALG0:0. In *Figure 25*, Conversion Timing Register 1 (CTR1): – updated CTR1[31] as part of the INPSAMP field. – added OFFSHIFT to 18 and 19 bit fields. In *Table 34*, CTR1 field descriptions: – updated INPSAMP field description. – added OFFSHIFT field and its description. Chapter Boot Assist Module In *Section 10.4.7, Inhibiting BAM operation*, updated the error bit from “PAE“ to “RAE“ in the SSCM to inhibit BAM operation. Chapter Clock Architecture In *Table 70*, Clock distribution: – updated frequency of FlexRay (CHI). – added information about FlexRay clock divider to the footnote. Chapter Clock Generation Module In *Figure 86*, MC_CGM auxiliary clock 1 generation overview added divider and MUX. In *Figure 84*, MC_CGM system clock generation overview, added 4–40 MHz crystal osc. in the figure. Chapter Clock Monitor Unit In *Table 87*, CMU memory map, updated register CMU_FDISP to CMU_FDR. In *Figure 91*, Control status register (CMU_CSR) and *Table 88*, CMU_CSR field descriptions, updated flag CME_A to CME.

Table 768. Document revision history (continued)

Date	Revision	Document changes
05-Jul-2013	10 (cont.)	Chapter Cross-Triggering Unit In *Table 97*, FIFO registers and *Figure 127*, FIFO DMA control register (FDCR), updated reset value of FDCR to 0x0F00. In *Table 99*, TGSISR Field descriptions updated description of flags: – I5_FE, I5_RE – I1_FE – I1RE. In *Figure 113*, Trigger handler control register 1 (THCR1), and *Figure 114*, Trigger handler control register 2 (THCR2) added Tn_T3E and Tn_T4E bit in the missing fields. Chapter Cyclic Redundancy Checker Unit In *Section 16.6, Functional description*, added note regarding the support of CRC-32 (Ethernet) and CRC-16 (CCITT) industry standards. Chapter Deserial Serial Peripheral Interface In *Figure 161*, DSPI Status Register (DSPI_SR) for cut2/3, updated TXRXS flag value to read only. – In *Section 17.3.2.1, DSPI Module Configuration Register (DSPI_MCR)*, updated DSPI_MCR to DSPIx_MCR; where x =0,1,2. Chapter Enhanced Motor Control Timer In *Section 21.4.3.6, Hold Register (HOLD)*, added information about HOLD register behavior when reading CCCTRL register. In *Section 21.4.3.16, Input Filter Register (FILT)*, added information regarding filtering done by FILT register. In *Section 21.4.4, Watchdog Timer Registers*, added information regarding watchdog function of the eTimer. In *Section 21.4.5.2, DMA Request Select Registers (DREQ0, DREQ1)*, editorial changes done. In *Section 21.7, DMA*, added information regarding eTimer capture DMA. In *Figure 238*, Status Register (STS), modified flags from read only to write enabled. In *Figure 246*, Channel Enable Register (ENBL), added channel information to the ENBL bit field. Chapter Error Correction Status Module In *Table 236*, MUDCR field descriptions, updated information on SRAM wait-states.

Table 768. Document revision history (continued)

Date	Revision	Document changes
05-Jul-2013	10 (cont.)	Chapter Fault Collection and Control Unit In *Section 23.6.2, FCCU CTRL Key Register (FCCU_CTRLK)*,*Section 23.6.10, FCCU NCF Status Register (FCCU_NCF_S0..3)*, and *Section 23.6.11, FCCU NCF Key Register (FCCU_NCFK)*, added note about the squence of operations for respective registers. In *Section 23.6.21, FCCU IRQ Status Register (FCCU_IRQ_STAT)*, added information, that in case CFG_TO_STAT, the respective enable bit of the FCCU_IRQ_EN register is also set. In *Figure 277*, FCCU top-level diagram, removed "MC_RGM" from the arrows coming out of RCC0 and RCC1. In *Figure 280*, FCCU Configuration Register (FCCU_CFG), added note about default values for the configuration register. In *Figure 284*, FCCU CFS Configuration Register 1 (FCCU_CFS_CFG1): – replaced the bit fields, CFSC0 … CFSC15 by CFSC16 … CFSC31. – replaced TCK Pull reference from Down to Up . In *Table 255*, FCCU_CFG field descriptions, updated RCCE1 and RCCE0 notes. In *Table 256*, FCCU_CF_CFG0..3 field descriptions, updated field desription for NCFTOEx when NCFEx is set. In *Table 284*, FCCU mapping of critical faults: – updated CF [20] values for Set/Clear Injection, NMI, and Safe mode request fields. – added note about CF[20] not supported in MPC5643L. Chapter Flash Memory In *Section 24.2.2.2, Platform Flash Access Protection Register (PFAPR)*, updated address of the shadow region in PFAPR register from0x203E00 to 0xF03E00. In *Table 311*, PFLASH Configuration Register 0 (PFCR0) settings for different frequencies, updated lowest recommended Flash Wait State Frequency. Chapter FlexCAN Throughout the chapter, removed information related to wake up feature in FlexCAN In *Section 25.3.1, FlexCAN memory mapping*, deleted non applicable note referring to specific MCU documentation. In *Section 25.4.6.1, Transmission Abort Mechanism*, added note about abort request to a TxMB. Chapter Flexible Motor Control Pulse Width Modulator Module In *Section 26.4.3.6, Value Register 1 (VAL1)*, updated note regarding PWMX output. In *Figure 387*, Output Logic Section, added MASKA and MASKB blocks. Chapter Frequency-Modulated Phase-Locked Loop In *Table 475*, MR Field Descriptions, added information about feedback divider to the fref definition. In *Section 28.6.2, Progressive clock switching*, added information about handling large transient current by “repeatedly decrementing or incrementing the ODF field by one, number of times until the desired frequency is obtained”.

Table 768. Document revision history (continued)

Date	Revision	Document changes
05-Jul-2013	10 (cont.)	Chapter Interrupt Controller In *Table 477*, INTC memory map, updated INTC_EOIR_PRC0 register access to R/W. In *Table 480*, Interrupt sources for INTC_0 and INTC_1, modified the column "Source signal" for the SIU External signals. Chapter JTAG Controller In *Table 481*, JTAG Signal Properties, replaced TCK Pull reference from Down to Up. Chapter LIN Controller In *Table 533*, LINFlexD interrupt control, added UART Timeout Error. In *Figure 635*, LIN status register (LINSR): – updated reset value LIN status register (LINSR). – updated RPS flag to read only. In *Figure 637*, UART mode control register (UARTCR), added note reference to the RFBM flag. In *Figure 638*, UART mode status register (UARTSR), updated RPS flag to read only. Chapter Mode Entry Module In *Section 33.3.2.7, Debug Mode Transition Status Register (ME_DMTS)* and *Section 33.3.2.8, RESET Mode Configuration Register (ME_RESET_MC)*, updated access rights of the registers to read only in user, supervisor, and test modes. In *Table 550*, Global Status Register (ME_GS) field descriptions, removed flag S_SSCLK1 as it is not implemented in real silicon. Chapter Nexus Port Controller In *Table 578*, NPC Signal Properties, replaced TCK Pull reference from Down to Up. Chapter Oscillators In *Table 591*, OSC_CTL field descriptions, replaced table title from RC_CTL to OSC_CTL. Chapter Periodic Interrupt Timer In *Section 37.1.1, Overview*, added note about PIT/DMA frequency requirement. In *Section 37.5.1, Example Configuration*, updated reference of the register PITCTRL to PITMCR. Chapter Peripheral Bridge In *Table 600*, PBRIDGE memory map, enhanced readability of the table fields. Chapter Power Management Unit In *Section 40.3, High-power regulators*, removed reference statement about internal CCB. Chapter Register Protection In *Section 41.5.2, Writing C code using the register protection scheme*, updated information about REG_PROT macros.

Table 768. Document revision history (continued)

Date	Revision	Document changes
05-Jul-2013	10 (cont.)	Chapter Reset Generation Module In *Figure 800*, MC_RGM State Machine, added note regarding SSCM reading the reset vector. Added *Section 42.4.6, Application Information*. Chapter Self-Test Control Unit In *Figure 808*, The STCU within the Safety Integrity Subsystem, removed Power-on reset event and updated destructive and external reset events from the description of STCU Reset Event. Chapter Sine Wave Generator In *Figure 835*, SWG status register (SWG_STAT), updated reset value of SWG status register (SWG_STAT). Chapter Software Watchdog Timer In *Table 717*, SWT_CR Field Descriptions added note regarding Logical Master IDs. Chapter Temperature Sensor In *Table 764 (TSENS calibration constants)*, updated all descriptions. In *Section 51.5.2, Equations for converting TSENS voltage to device temperature*, updated the equations for device temperature calculation.
18-Sep-2013	11	Updated Disclaimer.
02-Jan-2014	12	Updated “BLOCK NAME” variable in Chapter 12, Chapter 33, Chapter 39, Chapter 42 and Chapter 43.
07-Feb-2014	13	Updated *Figure 611* and *Table 728*.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2014 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com