SPC563Mxx - 32-bit Power Architecture® based MCU with up to 1.5 Mbyte Flash and 111 Kbyte RAM memories

Preface

Overview

The primary objective of this document is to define the functionality of the SPC563Mxx family of microcontrollers for use by software and hardware developers. The SPC563Mxx family is built on Power Architecture® technology and integrate technologies that are important for today's lower-end applications.

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

To locate any published errata or updates for this document, visit the *www.st.com*.

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the SPC563Mxx device. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the Power Architecture.

Chapter Organization and Device-Specific Information

This document includes chapters that describe:

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

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

References

In addition to this reference manual, the following documents provide additional information on the operation of the SPC563Mxx:

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

Contents

1 Introduction 45

1.1 The SPC563Mxx Microcontroller Family 45

1.2 SPC563Mxx Device Summary 45

1.3 SPC563Mxx Blocks 46

1.3.1 Block Diagram 46

1.3.2 Block Summary 47

1.4 SPC563Mxx Features 48

1.4.1 Feature List 48

1.5 SPC563Mxx feature details 57

1.5.1 e200z335 core 57

1.5.2 Crossbar 58

1.5.3 eDMA 59

1.5.4 Interrupt controller 59

1.5.5 FMPLL 60

1.5.6 Calibration EBI 61

1.5.7 SIU 62

1.5.8 ECSM 63

1.5.9 Flash 63

1.5.10 SRAM 64

1.5.11 BAM 64

1.5.12 eMIOS 64

1.5.13 eTPU2 65

1.5.14 eQADC 67

1.5.15 DSPI 69

1.5.16 eSCI 71

1.5.17 FlexCAN 72

1.5.18 System timers 73

1.5.19 Software Watchdog Timer (SWT) 74

1.5.20 Debug features 74

2 Memory Map 77

2.1 Introduction 77

2.2 Memory map 77

3 Signal Descriptions 85

3.1 Device Pin Assignments 85

3.1.1 LQFP144 Pinout (all 144-pin devices) 86

3.1.2 LQFP176 Pinout (all 176–pin devices) 87

3.2 External Signal Summary 88

3.3 Detailed Signal Descriptions 107

3.3.1 Reset / Configuration 107

3.3.2 General Purpose I/O (GPIO) 108

3.3.3 Calibration External Bus Interface (EBI) 108

3.3.4 Nexus Port Controller (NPC) 109

3.3.5 JTAG 110

3.3.6 FlexCAN 111

3.3.7 eSCI 111

3.3.8 DSPI 111

3.3.9 eQADC 112

3.3.10 eTPU 114

3.3.11 eMIOS 118

3.3.12 Clock Synthesizer 119

3.3.13 Power / Ground 119

3.4 I/O Power/Ground Segmentation 121

3.5 DSPI Connections to eTPU_A, eMIOS and SIU 122

3.5.1 DSPI_B Connectivity 122

3.5.2 DSPI_C Connectivity 123

4 Resets 126

4.1 Reset sources 126

4.2 Reset vector 126

4.3 Reset pins 127

4.3.1 RESET 127

4.3.2 RSTOUT 127

4.4 Clock quality monitor gating signal 127

4.5 Reset source descriptions 128

4.5.1 Power-on reset 130

4.5.2 External reset 130

4.5.3 Loss of lock 130

4.5.4 Loss of clock 131

4.5.5 Watchdog timer/debug reset . . . 131
4.5.6 Software watchdog timer reset . . . 131
4.5.7 Checkstop reset . . . 132
4.5.8 JTAG reset . . . 132
4.5.9 Software system reset . . . 132
4.5.10 Software external reset . . . 132
4.6 Reset registers in the SIU . . . 133
4.7 Reset configuration . . . 133
4.7.1 Reset configuration half word (RCHW) . . . 133
4.7.2 Reset configuration timing . . . 134
4.7.3 Reset weak pull up/down configuration . . . 135

5 Operating modes . . . 136
5.1 Overview . . . 136
5.2 Modes of operation . . . 136
5.2.1 Normal Mode . . . 136
5.2.2 Debug Mode . . . 136
5.2.3 Low power modes . . . 136
5.3 Modes and clock architecture . . . 137
5.3.1 Block diagrams . . . 137
5.3.2 Clock architecture . . . 142

6 e200z335 Core . . . 146
6.1 Introduction . . . 146
6.2 Features . . . 147

7 Direct Memory Access (DMA) . . . 148
7.1 Information specific to this device . . . 148
7.1.1 Device-specific features . . . 148
7.1.2 Channel assignments . . . 148
7.2 Introduction . . . 149
7.2.1 Overview . . . 150
7.2.2 Features . . . 151
7.3 Memory map/register definition . . . 157
7.3.1 Register descriptions . . . 159
7.4 Functional description . . . 184

7.4.1 DMA microarchitecture 184
7.4.2 DMA basic data flow 185
7.4.3 DMA performance 188
7.5 Initialization/application information 191
7.5.1 DMA initialization 191
7.5.2 DMA programming errors 192
7.5.3 DMA arbitration mode considerations 192
7.5.4 DMA transfer 194
7.5.5 TCD status 196
7.5.6 Channel linking 197
7.5.7 Dynamic programming 198
7.5.8 Hardware request release timing 199

8 Multi-Layer AHB Crossbar Switch (XBAR) 200
8.1 Introduction 200
8.1.1 Overview 200
8.1.2 Features 201
8.1.3 Limitations 201
8.1.4 General operation 201
8.2 XBAR registers 202
8.2.1 Register summary 202
8.2.2 XBAR register descriptions 203
8.2.3 Coherency 207
8.3 Function 207
8.3.1 Arbitration 208
8.3.2 Priority assignment 209

9 Peripheral Bridge (PBRIDGE) 210
9.1 Introduction 210
9.2 PBRIDGE features 210
9.3 PBRIDGE block diagram 210
9.4 PBRIDGE signal description 211
9.5 PBRIDGE functional description 211
9.5.1 Read cycles 211
9.5.2 Write cycles 211
9.6 PBRIDGE registers 211

10 C90LC Flash Memory 212
10.1 Overview 212
10.2 Platform flash memory controller (PFLASH_LCA) 213
10.2.1 Overview 213
10.2.2 Features 214
10.2.3 Detailed description 214
10.2.4 Wait-state emulation 219
10.3 Flash memory block (C90LC) 220
10.3.1 Flash block overview 220
10.3.2 LC flash features 222
10.3.3 Programming considerations 222
10.4 Memory maps 232
10.4.1 Overview memory map 232
10.4.2 Flash memory space map 233
10.4.3 Test block 234
10.4.4 Flash control and configuration registers map 236
10.5 Register descriptions 237
10.5.1 Flash control and configuration registers 238

11 General-Purpose Static RAM (SRAM) 269
11.1 Overview 269
11.2 Features 269
11.3 Modes of operation 269
11.3.1 Normal (Functional) Mode 269
11.3.2 Standby Mode 269
11.4 Block diagram 269
11.5 External signal description 270
11.6 Functional description 270
11.6.1 Access timing 270
11.7 Module memory map 271
11.8 Register descriptions 271

12 External Bus Interface for Calibration Bus 272
12.1 Information specific to this device 272
12.1.1 Device-specific features 272

12.1.2 Unsupported features . . . 272
12.1.3 Device-specific register information . . . 272
12.2 Introduction . . . 273
12.2.1 Overview . . . 273
12.2.2 Features . . . 273
12.2.3 Modes of operation . . . 273
12.3 External signal description . . . 275
12.3.1 Overview . . . 275
12.3.2 Detailed signal descriptions . . . 276
12.3.3 Signal output buffer enable logic by mode . . . 279
12.4 Memory map/register definition . . . 279
12.4.1 Register descriptions . . . 280
12.5 Functional description . . . 289
12.5.1 External bus interface features . . . 289
12.5.2 External bus operations . . . 294
12.6 Initialization/Application information . . . 334
12.6.1 Booting from external memory . . . 334
12.6.2 Running with SDR (single data rate) burst memories . . . 334
12.6.3 Running with asynchronous memories . . . 335
12.6.4 Connecting an MCU to multiple memories . . . 337
12.6.5 EBI operation with reduced pinout MCUs . . . 338

13 Interrupt Controller (INTC) . . . 341
13.0.1 Device-specific features . . . 341
13.0.2 Device-specific register information . . . 341
13.1 Introduction . . . 341
13.1.1 Module overview . . . 341
13.1.2 Block diagram . . . 342
13.1.3 Features . . . 342
13.2 Modes of operation . . . 343
13.2.1 Normal mode . . . 343
13.2.2 Debug mode . . . 344
13.2.3 Stop mode . . . 344
13.2.4 Factory test mode . . . 344
13.3 External signal description . . . 344
13.4 Memory map/register definition . . . 344

13.4.1 Memory map . . . 344
13.4.2 Register information . . . 345
13.4.3 INTC Module Configuration Register (INTC_MCR) . . . 345
13.4.4 INTC Current Priority Register (INTC_CPR) . . . 346
13.4.5 INTC Interrupt Acknowledge Register (INTC_IACKR) . . . 347
13.4.6 INTC End of Interrupt Register (INTC_EOIR) . . . 348
13.4.7 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7) . . . 349
13.4.8 INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511) 350
13.5 Functional description . . . 352
13.5.1 Interrupt request sources . . . 352
13.5.2 Priority management . . . 353
13.5.3 Handshaking with processor . . . 354
13.5.4 Reserved spaces in memory map . . . 357
13.6 Initialization/application information . . . 358
13.6.1 Initialization flow . . . 358
13.6.2 Interrupt exception handler . . . 358
13.6.3 ISR, RTOS, and task hierarchy . . . 360
13.6.4 Order of execution . . . 360
13.6.5 Priority ceiling protocol . . . 362
13.6.6 Selecting priorities according to request rates and deadlines . . . 362
13.6.7 Software setable interrupt requests . . . 363
13.6.8 Lowering priority within an ISR . . . 364
13.6.9 Negating an interrupt request outside of its ISR . . . 364
13.6.10 Examining LIFO contents . . . 365

14 Interrupts . . . 366
14.1 Introduction . . . 366
14.2 Interrupt vectors . . . 366
14.2.1 External input . . . 366
14.2.2 Critical input . . . 368
14.3 Interrupt summary . . . 368

15 System Integration Unit (SIU) . . . 386
15.1 Overview . . . 386
15.2 Features . . . 386
15.3 Modes of operation . . . 386

15.3.1 Normal Mode . . . 386
15.3.2 Debug Mode . . . 387
15.4 Block diagram . . . 387
15.5 Signal description . . . 389
15.6 Detailed signal descriptions . . . 389
15.6.1 RESET — Reset Input . . . 389
15.6.2 RSTOUT — Reset Output . . . 389
15.6.3 GPIO[0:213] — General Purpose I/O Pins . . . 390
15.6.4 BOOTCFG1 (BOOTCFG1_IRQ[3]_ETRIG[3]_GPIO[212]) — Boot Configuration Pin . . . 390
15.6.5 WKPCFG (WKPCFG_NMI_DSPI_B_SOUT_GPIO[213]) — I/O Pin Weak Pull Up Reset Configuration Pin . . . 390
15.6.6 IRQ[0:15] — External Interrupt Request Input Pins . . . 390
15.7 Memory map . . . 390
15.8 Register descriptions . . . 392
15.8.1 MCU ID Register 2 (SIU_MIDR2) . . . 392
15.8.2 MCU ID Register (SIU_MIDR) . . . 394
15.8.3 Reset Status Register (SIU_RSR) . . . 396
15.8.4 System Reset Control Register (SIU_SRCR) . . . 397
15.8.5 External Interrupt Status Register (SIU_EISR) . . . 399
15.8.6 DMA/Interrupt Request Enable Register (SIU_DIRER) . . . 399
15.8.7 DMA/Interrupt Request Select Register (SIU_DIRSR) . . . 400
15.8.8 Overrun Status Register (SIU_OSR) . . . 401
15.8.9 Overrun Request Enable Register (SIU_ORER) . . . 402
15.8.10 IRQ Rising-Edge Event Enable Register (SIU_IREER) . . . 403
15.8.11 External IRQ Falling-Edge Event Enable Register (SIU_IFEER) . . . 403
15.8.12 External IRQ Digital Filter Register (SIU_IDFR) . . . 404
15.8.13 Pad Configuration Registers (SIU_PCR) . . . 405
15.8.14 GPIO Pin Data Output Registers (SIU_GPDO83_86 – SIU_GPDO230_232) . . . 449
15.8.15 GPO Data Output Registers (SIU_GPDO350 – SIU_GPDO413) . . . 450
15.8.16 GPIO Pin Data Input Registers (SIU_GPDI83_86 – SIU_GPDI_232) . 452
15.8.17 eQADC Trigger Input Select Register (SIU_ETISR) . . . 453
15.8.18 External IRQ Input Select Register (SIU_EIISR) . . . 455
15.8.19 DSPI Input Select Register (SIU_DISR) . . . 458
15.8.20 IMUX Select Register 3 (SIU_ISEL3) . . . 460
15.8.21 IMUX Select Register 8 (SIU_ISEL8) . . . 466

15.8.22 IMUX Select Register 9 (SIU_ISEL9) 468
15.8.23 Chip Configuration Register (SIU_CCR) 470
15.8.24 External Clock Control Register (SIU_ECCR) 471
15.8.25 Compare A High Register (SIU_CARH) 472
15.8.26 Compare A Low Register (SIU_CARL) 473
15.8.27 Compare B High Register (SIU_CBRH) 473
15.8.28 Compare B Low Register (SIU_CBRL) 473
15.8.29 System Clock Register (SIU_SYSDIV) 474
15.8.30 Halt Register (SIU_HLT) 475
15.8.31 Halt Acknowledge Register (SIU_HLTACK) 477
15.9 Functional description 480
15.9.1 System configuration 480
15.9.2 Reset control 480
15.9.3 External interrupt 480
15.9.4 GPIO operation 482
15.9.5 Internal multiplexing 482

16 Frequency-modulated phase-locked loop (FMPLL) 484
16.1 Information specific to this device 484
16.1.1 Device-specific features 484
16.1.2 Device-specific register field reset values 484
16.2 Introduction 484
16.2.1 Overview 484
16.2.2 Features 485
16.2.3 Modes of operation 486
16.3 External signal description 487
16.3.1 Detailed signal descriptions 487
16.4 Memory map and register definition 488
16.4.1 Memory map 488
16.4.2 Register descriptions 488
16.5 Functional description 498
16.5.1 Input clock frequency 498
16.5.2 Clock configuration 499
16.5.3 Lock detection 500
16.5.4 Loss-of-clock detection 500
16.5.5 Frequency modulation 504

17 Error Correction Status Module (ECSM) 506
17.1 Overview 506
17.2 Module memory map 506
17.3 Register descriptions 507
17.3.1 ECC registers 507

18 System Timer Module (STM) 523
18.1 Introduction 523
18.1.1 Overview 523
18.1.2 Features 523
18.1.3 Modes of operation 523
18.2 External signal description 523
18.3 Memory map and register definition 523
18.3.1 Memory map 523
18.3.2 Register descriptions 524
18.4 Functional description 527

19 Software Watchdog Timer (SWT) 528
19.1 Information specific to this device 528
19.1.1 Device-specific features 528
19.1.2 Reset assertion 528
19.1.3 Default configuration 528
19.2 Introduction 529
19.2.1 Overview 529
19.2.2 Features 529
19.2.3 Modes of operation 529
19.3 External signal description 529
19.4 Memory map and register definition 529
19.4.1 Memory map 530
19.4.2 Register descriptions 530
19.5 Functional description 535

20 Boot Assist Module (BAM) 537
20.1 Overview 537
20.2 Features 537

20.3 Modes of operation 537
20.3.1 Normal Mode 537
20.3.2 Debug Mode 537
20.3.3 Internal Boot Mode 537
20.3.4 Serial Boot Mode 538
20.3.5 Calibration Bus Boot Mode 538
20.4 Memory map 538
20.5 Functional description 538
20.5.1 BAM program flow chart 538
20.5.2 BAM program operation 539
20.5.3 Reset Configuration Half Word (RCHW) 542
20.5.4 Internal Boot Mode 543
20.5.5 Serial Boot Mode 546
20.5.6 Booting from the calibration bus 552

21 Configurable Enhanced Modular IO Subsystem (eMIOS200) 554
21.1 Introduction 554
21.1.1 Overview 555
21.1.2 Features 556
21.1.3 Modes of operation 557
21.2 External signal description 557
21.2.1 Overview 557
21.2.2 Detailed signal descriptions 557
21.3 Memory map/register definition 558
21.3.1 Memory map 558
21.3.2 Register descriptions 559
21.4 Functional description 573
21.4.1 Unified Channel (UC) 576
21.4.2 IP Bus Interface Unit (BIU) 619
21.4.3 STAC Bus Client submodule (REDC) 619
21.4.4 Global Clock Prescaler Submodule (GCP) 619
21.5 Initialization/Application Information 620
21.5.1 Considerations 620
21.5.2 Application information 620

22 Enhanced Time Processing Unit (eTPU2) 624

22.1 Introduction . . . 624
22.1.1 eTPU2 implementation . . . 624
22.2 Block diagram . . . 625
22.3 eTPU2 operation overview . . . 626
22.3.1 eTPU2 engine . . . 627
22.3.2 Host interface . . . 628
22.3.3 Shared data memory (SDM) . . . 629
22.3.4 Task scheduler . . . 630
22.3.5 Microengine . . . 631
22.3.6 Debug interface . . . 631
22.4 Features . . . 631
22.5 Modes of operation . . . 634
22.5.1 User Configuration Mode . . . 634
22.5.2 User Mode . . . 634
22.5.3 Debug Mode . . . 634
22.5.4 Module Disable Mode . . . 634
22.6 eTPU2 mode selection . . . 635
22.7 External signal description . . . 635
22.8 eTPU2 external signal description . . . 635
22.8.1 Output and input channel signals . . . 635
22.9 Memory map . . . 635
22.10 Register descriptions . . . 636
22.10.1 System configuration registers . . . 639
22.10.2 Time base registers . . . 647
22.10.3 Engine related registers . . . 653
22.10.4 Global channel registers . . . 655
22.10.5 Channel configuration and control registers . . . 661
22.11 Initialization and application information . . . 665

23 Enhanced Queued Analog-to-Digital Converter (eQADC) . . . 666

23.1 Introduction . . . 666
23.1.1 Module overview . . . 666
23.1.2 Block diagram . . . 666
23.1.3 Features . . . 669
23.2 Modes of operation . . . 670
23.2.1 Normal Mode . . . 670

23.2.2 Debug Mode 670
23.2.3 Stop Mode 671
23.3 External signal description 672
23.3.1 Overview 672
23.3.2 Detailed signal descriptions 675
23.4 Memory map/register definition 678
23.4.1 eQADC memory map 678
23.4.2 eQADC register descriptions 682
23.4.3 On-chip ADC registers 711
23.5 Functional description 725
23.5.1 Overview 725
23.5.2 Data flow in eQADC 726
23.5.3 Command/Result queues 749
23.5.4 eQADC command FIFOs 749
23.5.5 eQADC result FIFOs 778
23.5.6 On-chip ADC configuration and control 782
23.5.7 Internal/External multiplexing 793
23.5.8 eQADC DMA/Interrupt request 800
23.5.9 eQADC synchronous serial interface (SSI) sub-block 803
23.5.10 eQADC parallel side interface (PSI) sub-block 808
23.5.11 Analog sub-block 811
23.6 Initialization/Application information 814
23.6.1 Multiple queues control setup example 814
23.6.2 eQADC/DMAC interface 819
23.6.3 Sending immediate command setup example 821
23.6.4 Modifying queues 822
23.6.5 CQueue and RQueues usage 822
23.6.6 ADC result calibration 823
23.6.7 eQADC versus QADC 826

24 Decimation Filter 830
24.1 Information specific to this device 830
24.1.1 Device-specific features 830
24.2 Introduction 830
24.2.1 Overview 830
24.2.2 Features 832

24.2.3 Modes of operation . . . 832
24.3 External signal description . . . 833
24.3.1 Decimation trigger signal . . . 833
24.4 Memory map and register definition . . . 833
24.4.1 Decimation filter device memory map . . . 833
24.4.2 Decimation filter register descriptions . . . 835
24.4.3 Decimation filter memory map for parallel side interface . . . 846
24.4.4 PSI register description . . . 846
24.5 Functional description . . . 848
24.5.1 Overview . . . 848
24.5.2 Parallel Side Interface (PSI) description . . . 849
24.5.3 Input buffer description . . . 849
24.5.4 Output buffer description . . . 850
24.5.5 Bypass configuration description . . . 852
24.5.6 IIR and FIR filter . . . 852
24.5.7 Filter prefill control description . . . 855
24.5.8 Timestamp data transmission . . . 856
24.5.9 Flush Command description . . . 856
24.5.10 Soft-Reset command description . . . 856
24.5.11 Interrupts requests description . . . 857
24.5.12 DMA requests description . . . 858
24.5.13 Freeze Mode description . . . 859
24.5.14 Enhanced Debug Monitor description . . . 859
24.6 Initialization information . . . 859
24.6.1 Initialization procedure . . . 860
24.7 Application information . . . 860
24.7.1 eQADC IP as the master block . . . 860
24.8 Filter example simulation . . . 861
24.8.1 Coefficients calculation . . . 861
24.8.2 Input data calculation . . . 862
24.8.3 Filter results . . . 862

25 Deserial Serial Peripheral Interface (DSPI) . . . 864

25.1 Introduction . . . 864
25.1.1 Overview . . . 864
25.1.2 Features . . . 865

25.1.3 DSPI configurations 866
25.1.4 Modes of operation 868
25.2 External signal description 868
25.2.1 Overview 868
25.2.2 Detailed signal description 869
25.3 Memory map and register definition 870
25.3.1 Memory map 870
25.3.2 Register descriptions 871
25.4 Functional description 895
25.4.1 Start and Stop of DSPI Transfers 896
25.4.2 Serial Peripheral Interface (SPI) Configuration 897
25.4.3 Deserial Serial Interface (DSI) Configuration 899
25.4.4 Combined Serial Interface (CSI) Configuration 905
25.4.5 DSPI Baud Rate and Clock Delay Generation 906
25.4.6 Transfer Formats 909
25.4.7 Continuous Serial Communications Clock 919
25.4.8 Slave mode operation constraints 921
25.4.9 Timed Serial Bus (TSB) 922
25.4.10 Interleaved TSB (ITSB) Mode 925
25.4.11 Differences between TSB and ITSB modes 925
25.4.12 Parity Generation and Check 927
25.4.13 Interrupts/DMA Requests 928
25.4.14 Power saving features 930
25.5 Initialization/Application information 931
25.5.1 How to Manage DSPI Queues 931
25.5.2 Switching Master and Slave Mode 931
25.5.3 Baud rate settings 931
25.5.4 Delay settings 932
25.5.5 DSPI Compatibility with the QSPI of the MPC500 MCUs 933
25.5.6 Calculation of FIFO Pointer Addresses 933

26 Enhanced Serial Communication Interface (FMPLL) 936

26.1 Information specific to this device 936
26.1.1 Device-specific features 936
26.2 Introduction 936
26.2.1 Bibliography 936

26.2.2 Acronyms and abbreviations . . . 936
26.2.3 Glossary . . . 937
26.2.4 Overview . . . 937
26.2.5 Features . . . 939
26.2.6 Modes of operation . . . 940
26.3 External signal description . . . 940
26.3.1 Detailed signal descriptions . . . 940
26.4 Memory map and register definition . . . 940
26.4.1 Memory map . . . 940
26.4.2 Register descriptions . . . 941
26.5 Functional description . . . 954
26.5.1 Module control . . . 955
26.5.2 Frame formats . . . 955
26.5.3 Baud rate and clock generation . . . 958
26.5.4 Baud rate tolerance . . . 960
26.5.5 SCI mode . . . 963
26.5.6 LIN mode . . . 978
26.5.7 Interrupts . . . 985
26.6 Application information . . . 986
26.6.1 SCI data frames separated by preamble . . . 986

27 FlexCAN Module . . . 987
27.1 Information specific to this device . . . 987
27.1.1 Device-specific features . . . 987
27.2 Introduction . . . 987
27.2.1 Overview . . . 988
27.2.2 FlexCAN module features . . . 989
27.2.3 Modes of operation . . . 990
27.3 External signal description . . . 991
27.3.1 Overview . . . 991
27.3.2 Signal descriptions . . . 991
27.4 Memory map/register definition . . . 991
27.4.1 FlexCAN memory mapping . . . 991
27.4.2 Message buffer structure . . . 993
27.4.3 Rx FIFO structure . . . 996
27.4.4 Register descriptions . . . 998

27.5 Functional description 1016
27.5.1 Overview 1016
27.5.2 Transmit process 1017
27.5.3 Arbitration process 1017
27.5.4 Receive process 1018
27.5.5 Matching process 1020
27.5.6 Data coherence 1021
27.5.7 Rx FIFO 1023
27.5.8 CAN protocol related features 1025
27.5.9 Modes of operation details 1029
27.5.10 Interrupts 1032
27.5.11 Bus interface 1033
27.6 Initialization/application information 1033
27.6.1 FlexCAN initialization sequence 1033
27.6.2 FlexCAN addressing and RAM size configurations 1034

28 Periodic Interrupt Timer (PIT_RTI) 1035
28.1 Introduction 1035
28.1.1 Overview 1035
28.1.2 Features 1036
28.2 Modes of operation 1036
28.3 Signal description 1036
28.4 Memory map and register description 1036
28.4.1 Memory map 1036
28.4.2 Register descriptions 1037
28.5 Functional description 1040
28.5.1 General 1040
28.5.2 Interrupts 1042
28.6 Initialization and application information 1042
28.6.1 Example configuration 1042

29 Power Management Controller (PMC) 1044
29.1 Overview 1044
29.1.1 **Block diagram 1046**
29.2 External signal description 1047
29.2.1 Detailed signal descriptions 1047

29.3 Memory map/register definition 1048
29.3.1 Module Configuration Register (PMC_MCR) 1048
29.3.2 Trimming Register (PMC_TRIMR) 1050
29.3.3 Status Register (PMC_SR) 1053
29.4 Functional description 1056
29.4.1 Bandgap 1057
29.4.2 5 V LVI 1057
29.4.3 3.3 V internal voltage regulator 1058
29.4.4 3.3 V LVI 1059
29.4.5 1.2 V voltage regulator controller 1059
29.4.6 1.2 V LVI 1060
29.4.7 Resets and interrupts 1060
29.4.8 Soft-start (for 1.2 V and 3.3 V regulators) 1064
29.5 Electrical characteristics 1064

30 JTAG Controller (JTAGC) 1065
30.1 Information specific to this device 1065
30.1.1 Device-specific parameters 1065
30.1.2 Device identification register parameters 1065
30.1.3 JTAG instructions 1065
30.1.4 Unavailable instructions 1066
30.1.5 Auxiliary TAP controller instructions 1066
30.2 Introduction 1066
30.2.1 Overview 1067
30.2.2 Features 1067
30.2.3 Modes of operation 1067
30.3 External signal description 1068
30.3.1 Overview 1068
30.3.2 Detailed signal descriptions 1068
30.4 Register definition 1069
30.4.1 Register descriptions 1069
30.5 Functional description 1071
30.5.1 JTAGC reset configuration 1071
30.5.2 IEEE 1149.1-2001 (JTAG) test access port 1071
30.5.3 TAP controller state machine 1072
30.5.4 JTAGC block instructions 1074

30.5.5 Boundary scan 1076
30.6 Initialization/Application information 1076

31 Nexus Port Controller (NPC) 1078
31.1 Information specific to this device 1078
31.1.1 Parameter values 1078
31.1.2 Unavailable features 1078
31.1.3 Available features 1078
31.1.4 Nexus clients 1078
31.2 Introduction 1079
31.2.1 Overview 1079
31.2.2 Features 1080
31.2.3 Modes of operation 1080
31.3 External signal description 1081
31.3.1 Overview 1081
31.3.2 Detailed signal descriptions 1081
31.4 Register definition 1082
31.4.1 Register descriptions 1083
31.5 Functional description 1087
31.5.1 NPC reset configuration 1087
31.5.2 Auxiliary output port 1087
31.5.3 IEEE 1149.1-2001 (JTAG) TAP 1090
31.5.4 Nexus JTAG port sharing 1094
31.5.5 MCKO and ipg_sync_mcko 1094
31.5.6 EVTO sharing 1094
31.5.7 Nexus reset control 1094
31.5.8 System clock locked indication 1094
31.6 Initialization/application information 1095
31.6.1 Accessing NPC tool-mapped registers 1095

32 Temperature sensor 1096
32.1 Overview 1096
32.2 Detailed description 1096
32.3 Temperature formula 1097
32.3.1 T_{LOW} and T_{HIGH} 1098
32.3.2 $T_{TSENS_CODE}(T_{LOW})$ and $T_{TSENS_CODE}(T_{HIGH})$ 1098

32.3.3 $V_{BG_CODE}(T_{LOW})$ 1099
32.3.4 Temperature sensor voltage ($V_{TENS}(T)$) 1099
32.3.5 Bandgap reference voltage ($V_{BG_CODE}(T)$) 1099
32.3.6 Registers 1099

33 Revision history 1102

List of tables

Table 1. SPC563Mxx family device summary . . . 45
Table 2. SPC563Mxx Block Summary . . . 47
Table 3. Device memory map . . . 77
Table 4. Detailed device memory map . . . 79
Table 5. SPC563Mxx family device memory space . . . 83
Table 6. SPC563Mxx family device flash memory map . . . 83
Table 7. SPC563Mxx family device SRAM memory map . . . 84
Table 8. SPC563Mx signal properties . . . 88
Table 9. Pad types . . . 107
Table 10. SPC563Mx Power/Ground Segmentation . . . 121
Table 11. DSPI_B I/O Mapping . . . 122
Table 12. DSPI_C I/O Mapping . . . 124
Table 13. Timing for reset sources . . . 127
Table 14. Reset configuration half word sources . . . 133
Table 15. Reset Configuration Half Word (RCHW) field descriptions . . . 134
Table 16. MDIS support . . . 143
Table 17. DMA channel assignments . . . 148
Table 18. DMA 32-bit memory map . . . 158
Table 19. DMA_CR field descriptions . . . 160
Table 20. DMA_ESR field descriptions . . . 162
Table 21. DMA_ERQRL field descriptions . . . 164
Table 22. DMA_EEIRL field descriptions . . . 165
Table 23. DMA_SERQR field descriptions . . . 165
Table 24. DMA_CERQR field descriptions . . . 166
Table 25. DMA_SEEIR field descriptions . . . 166
Table 26. DMA_CEEIR field descriptions . . . 167
Table 27. DMA_CIRQR field descriptions . . . 167
Table 28. DMA_CER field descriptions . . . 168
Table 29. DMA_SSBR field descriptions . . . 168
Table 30. DDMA_CDSBR field descriptions . . . 169
Table 31. DMA_IRQRL field descriptions . . . 170
Table 32. DMA_ERL field descriptions . . . 171
Table 33. DMA_CPRn field descriptions . . . 172
Table 34. TCDn 32-bit memory structure . . . 173
Table 35. TCDn Word 0 (TCDn.saddr) field description . . . 173
Table 36. TCDn Word 1 (TCDn.{smod,ssize,dmod,dsize,soff}) field descriptions . . . 174
Table 37. TCDn Word 2 (TCDn.nbytes) field description . . . 176
Table 38. TCDn Word 3 (TCDn.slast) field descriptions . . . 176
Table 39. TCDn Word 4 (TCDn.daddr) field description . . . 177
Table 40. TCDn Word 5 (TCDn.{doff,citer}) field descriptions . . . 178
Table 41. TCDn Word 6 (TCDn.dlast_sga) field description . . . 180
Table 42. TCDn Word 7 (TCDn.{biter, control/status}) field descriptions . . . 181
Table 43. DMA peak transfer rates [MB/s] . . . 189
Table 44. DMA peak request rate [MReq/s] . . . 190
Table 45. Master / Slave connections . . . 200
Table 46. XBAR register configuration summary . . . 202
Table 47. XBAR Master Priority Register field descriptions . . . 204
Table 48. XBAR Slave General Purpose Control Register field descriptions . . . 206

Table 49. Flash access disable logic . . . 216
Table 50. Flash modify operations . . . 223
Table 51. Flash overview map . . . 232
Table 52. Flash memory storage address map . . . 233
Table 53. Test block locations . . . 234
Table 54. Test block allocation . . . 235
Table 55. Flash memory register addresses . . . 236
Table 56. BIUCR field descriptions . . . 239
Table 57. BIUAPR field descriptions . . . 242
Table 58. BIUCR2 field descriptions . . . 243
Table 59. PFCR3 field descriptions . . . 244
Table 60. MCR bits set/clear priority levels . . . 246
Table 61. MCR field descriptions . . . 247
Table 62. LMLR field descriptions . . . 251
Table 63. HLR field descriptions . . . 253
Table 64. SLMLR field descriptions . . . 255
Table 65. LMSR field descriptions . . . 257
Table 66. HSR field descriptions . . . 258
Table 67. AR field descriptions . . . 259
Table 68. AR content priority list . . . 259
Table 69. UT0 field descriptions . . . 260
Table 70. UT1 field descriptions . . . 263
Table 71. UT2 field descriptions . . . 263
Table 72. UMISR0 field descriptions . . . 264
Table 73. UMISR1 field descriptions . . . 265
Table 74. UMISR2 field descriptions . . . 266
Table 75. UMISR3 field descriptions . . . 267
Table 76. UMISR4 field descriptions . . . 268
Table 77. Wait states during RAM access . . . 271
Table 78. SRAM memory map . . . 271
Table 79. Register bit values . . . 273
Table 80. Signal properties . . . 275
Table 81. TSIZ[0:1] encoding . . . 278
Table 82. Signal output buffer enable logic by mode . . . 279
Table 83. EBI address map . . . 279
Table 84. EBI_MCR field descriptions . . . 281
Table 85. SIZE encoding . . . 283
Table 86. EBI_TESR field descriptions . . . 284
Table 87. EBI_BMCR field descriptions . . . 284
Table 88. EBI_BR0–EBI_BR3, EBI_CAL_BR0–3 field descriptions . . . 285
Table 89. EOE values . . . 287
Table 90. SBL, BL values . . . 287
Table 91. EBI_OR0–EBI_OR3, EBI_CAL_OR0–3 field descriptions . . . 289
Table 92. Default attributes for non-chip-select transfers . . . 291
Table 93. Write/byte enable signals function . . . 293
Table 94. Wrap bursts order . . . 308
Table 95. Small access cases . . . 313
Table 96. Examples of 4-word burst addresses . . . 315
Table 97. Examples of 2-word burst addresses . . . 318
Table 98. Transaction sizes supported by EBI . . . 318
Table 99. Data bus requirements for read cycles . . . 321
Table 100. Data bus contents for write cycles . . . 321

Table 101. Termination Signals Protocol . . . 323
Table 102. Misalignment cases supported by a 64-bit AMBA EBI (internal bus) . . . 328
Table 103. Misalignment cases supported by a 64-bit AMBA EBI (external bus). . . . 329
Table 104. Misalignment cases supported by a 32-bit AMBA EBI (internal bus) . . . 331
Table 105. Misalignment cases supported by a 32-bit AMBA EBI (external bus). . . . 332
Table 106. INTC memory map . . . 344
Table 107. Registers and bits inaccessible on single-processor devices . . . 345
Table 108. INTC_MCR field descriptions . . . 346
Table 109. INTC_CPR field descriptions . . . 347
Table 110. INTC_IACKR field descriptions . . . 348
Table 111. INTC_SSCIRx field descriptions . . . 350
Table 112. INTC_PSRx field descriptions . . . 351
Table 113. Processor interrupt handshaking compatibility . . . 357
Table 114. Order of ISR execution example . . . 361
Table 115. Interrupt summary . . . 369
Table 116. SIU signal properties . . . 389
Table 117. SIU address map . . . 390
Table 118. SIU_MIDR2 field description . . . 393
Table 119. Flash memory size . . . 393
Table 120. Flash memory size detailed . . . 394
Table 121. Flash size values by part number . . . 394
Table 122. SIU_MIDR field descriptions . . . 395
Table 123. Memory size core dependency . . . 395
Table 124. CSP and PKG[0–4] . . . 395
Table 125. SIU_RSR field descriptions . . . 396
Table 126. SIU_SRCR field descriptions . . . 398
Table 127. SIU_EISR field descriptions . . . 399
Table 128. SIU_DIRER field descriptions . . . 400
Table 129. SIU_DIRSR field descriptions . . . 401
Table 130. SIU_OSR field descriptions . . . 402
Table 131. SIU_ORER field descriptions . . . 402
Table 132. SIU_IREER field descriptions . . . 403
Table 133. SIU_IFEER field descriptions . . . 404
Table 134. SIU_IDFR field descriptions . . . 405
Table 135. SIU_PCR field descriptions . . . 406
Table 136. PCR215 PA field definition . . . 436
Table 137. PCR216 PA field definition . . . 437
Table 138. PCR217 PA field definition . . . 437
Table 139. PCR218 PA field definition . . . 438
Table 140. SIU_PCR350 – SIU_PCR381 DSPI muxing . . . 446
Table 141. SIU_PCR382 – SIU_PCR389 DSPI muxing . . . 448
Table 142. SIU_PCR390 – SIU_PCR413 DSPI muxing . . . 449
Table 143. SIU_GPDO*x* Register field descriptions . . . 450
Table 144. SIU_GPDO*x* Register field descriptions . . . 452
Table 145. SIU_GPDI*x* Register field descriptions . . . 453
Table 146. SIU_ETISR field descriptions . . . 455
Table 147. SIU_EIISR field descriptions . . . 456
Table 148. SIU_DISR field descriptions . . . 459
Table 149. SIU_ISEL3 Register Field Descriptions . . . 461
Table 150. SIU_ISEL8 Register field descriptions . . . 467
Table 151. SIU_ISEL9 Register field descriptions . . . 469
Table 152. SIU_CCR Register field descriptions . . . 470

Table 153. SIU_ECCR Register field descriptions . . . 472
Table 154. SIU_SYSDIV Register field descriptions . . . 474
Table 155. SIU_HLT field descriptions . . . 475
Table 156. SIU_HLTACK field descriptions . . . 478
Table 157. Register field reset values . . . 484
Table 158. Clock mode selection . . . 486
Table 159. Signal properties . . . 487
Table 160. IFMPLL detailed signal descriptions . . . 488
Table 161. FMPLL memory map . . . 488
Table 162. SYNCR field descriptions . . . 489
Table 163. SYNSR field descriptions . . . 492
Table 164. ESYNCR1 field descriptions . . . 494
Table 165. ESYNCR2 field descriptions . . . 496
Table 166. SYNFMMR field descriptions . . . 498
Table 167. Input clock frequency at the predivider input . . . 499
Table 168. Loss-of-clock monitoring . . . 501
Table 169. Loss-of-clock reset . . . 502
Table 170. Loss-of-clock interrupt request . . . 503
Table 171. ECSM 32-bit memory map . . . 506
Table 172. ECSM_ECR field descriptions . . . 508
Table 173. ECSM_ESR field descriptions . . . 509
Table 174. ECSM_EEGR field descriptions . . . 510
Table 175. ECSM_FEAR field descriptions . . . 513
Table 176. ECSM_FEMR field descriptions . . . 514
Table 177. ECSM_FEAT field descriptions . . . 514
Table 178. ECSM_FEDR field descriptions . . . 516
Table 179. ECSM_REAR field descriptions . . . 516
Table 180. ECSM_PRESR field descriptions . . . 517
Table 181. RAM syndrome mapping for single-bit correctable errors . . . 517
Table 182. ECSM_REMR field descriptions . . . 520
Table 183. ECSM_REAT field descriptions . . . 521
Table 184. ECSM_REDR field descriptions . . . 522
Table 185. STM memory map . . . 523
Table 186. STM_CR field descriptions . . . 525
Table 187. STM_CNT field descriptions . . . 525
Table 188. STM_CCR*n* field descriptions . . . 526
Table 189. STM_CIR*n* field descriptions . . . 526
Table 190. STM_CMP*n* field descriptions . . . 527
Table 191. SWT memory map . . . 530
Table 192. SWT_CR field descriptions . . . 531
Table 193. SWT_IR field descriptions . . . 532
Table 194. SWT_TO field descriptions . . . 533
Table 195. SWT_WN field descriptions . . . 533
Table 196. SWT_SR field descriptions . . . 534
Table 197. SWT_CO field descriptions . . . 534
Table 198. SWT_SR field descriptions . . . 535
Table 199. BAM memory map . . . 538
Table 200. MMU configuration for internal flash boot . . . 540
Table 201. Boot modes . . . 540
Table 202. RCHW field descriptions . . . 542
Table 203. Watchdog timeouts . . . 543
Table 204. Possible RCHW locations in the internal flash . . . 544

Table 205. CAN/eSCI pin configuration for CAN/eSCI fixed baud rate boot modes. 546
Table 206. Serial Boot Mode - baud rate & watchdog summary 547
Table 207. Lookup table for CAN bit timing 551
Table 208. Maximum and minimum detectable baud rates 552
Table 209. MMU Configuration for Calibration Bus Boot and Serial Boot modes. 552
Table 210. Calibration bus/EBI register settings. 553
Table 211. External signals 557
Table 212. eMIOS200 memory map. 558
Table 213. Unified Channel memory map 559
Table 214. EMIOS_MCR field descriptions 560
Table 215. EMIOS_GFR field descriptions. 562
Table 216. EMIOS_OUDR field descriptions 562
Table 217. EMIOS_UCDIS field descriptions 563
Table 218. EMIOS_CADR[n], EMIOS_CBDR[n] and EMIOS_ALTA[n] values assignment. 564
Table 219. EMIOS_CCR[n] field descriptions. 566
Table 220. EMIOS_CSR[n] field descriptions. 572
Table 221. eTPU high-level memory map 635
Table 222. Detailed memory map. 636
Table 223. ETPU_MCR field descriptions 639
Table 224. ETPU_CDCR field descriptions 642
Table 225. ETPU_MISCCMPR field descriptions. 643
Table 226. ETPU_SCMOFFDATAR field descriptions. 644
Table 227. ETPU_ECR field descriptions. 644
Table 228. CDFC[0:1] configuration—Channel digital filter control 646
Table 229. ETB[0:4] configuration—Entry table base address options 647
Table 230. ETPU_TBCR field descriptions. 648
Table 231. AM - angle mode selection 648
Table 232. TCR2CTL configuration—TCR2 clock sources 649
Table 233. TCRCF configuration—TCRCLK filter clock/mode 650
Table 234. TCR1 clock source 650
Table 235. ETPU_TB1R field descriptions 651
Table 236. ETPU_TB2R field descriptions 651
Table 237. ETPU_REDCR field descriptions 652
Table 238. ETPU_WDTR field descriptions 654
Table 239. ETPU_IDLE field descriptions 654
Table 240. ETPU_CISR field descriptions 655
Table 241. ETPU_CDTRSR field descriptions 656
Table 242. ETPU_CIOSR field descriptions. 657
Table 243. ETPU_CDTROSR field descriptions. 657
Table 244. ETPU_CIER field descriptions 658
Table 245. ETPU_CDTRER field descriptions 658
Table 246. ETPU_WDSR field descriptions 659
Table 247. ETPU_CPSSR field descriptions 660
Table 248. ETPU_CSSR field descriptions 661
Table 249. Channel registers structure. 661
Table 250. eTPU channel register map 661
Table 251. CnCR field descriptions 662
Table 252. CnSCR field descriptions 664
Table 253. CxHSRR field descriptions 665
Table 254. External signals 672
Table 255. eQADC memory map 678
Table 256. EQADC_MCR field descriptions. 682

Table 257. EQADC_NMSFR field descriptions . . . 684
Table 258. EQADC_ETDFR field descriptions . . . 684
Table 259. Minimum required time to valid ETRIG . . . 685
Table 260. EQADC_CFPRx field descriptions . . . 686
Table 261. EQADC_RFPRx field descriptions . . . 687
Table 262. EQADC_CFCR*x* field descriptions . . . 688
Table 263. EQADC_IDCR*x* field descriptions . . . 692
Table 264. EQADC_FISRx field descriptions . . . 694
Table 265. EQADC_CFTCRx field descriptions . . . 699
Table 266. EQADC_CFSSRx field descriptions . . . 701
Table 267. EQADC_CFSR field descriptions . . . 702
Table 268. Current CFIFO status . . . 703
Table 269. EQADC_SSICR field descriptions . . . 704
Table 270. EQADC_SSIRDR field descriptions . . . 705
Table 271. EQADC_REDLCCR field descriptions . . . 706
Table 272. EQADC_CFxRw field descriptions . . . 708
Table 273. EQADC_CF0ERw field descriptions . . . 709
Table 274. EQADC_RFxRw field descriptions . . . 711
Table 275. On-chip ADC memory map . . . 712
Table 276. ADC0/1_CR field descriptions . . . 715
Table 277. System clock divide factor for ADC clock . . . 717
Table 278. ADC_TSCR field descriptions . . . 718
Table 279. Clock divide factor for time stamp . . . 718
Table 280. ADC_TBCR field descriptions . . . 719
Table 281. ADC0/1_GCCR field descriptions . . . 720
Table 282. ADC0/1_OCCR field descriptions . . . 721
Table 283. ADC_ACR1–8 field descriptions . . . 722
Table 284. ADC0/1_AGRx field descriptions . . . 724
Table 285. ADC0/1_AORx field descriptions . . . 724
Table 286. ADC_PUDCRx field descriptions . . . 725
Table 287. Conversion command format for standard configuration – field descriptions . . . 731
Table 288. Conversion command format for alternate configurations – field descriptions . . . 733
Table 289. Write configuration command format for on-chip ADC operation – field descriptions . . . 737
Table 290. Read configuration command format for on-chip ADC operation – field descriptions . . . 739
Table 291. ADC result format when FMT = 0 (Right Justified Unsigned) – field descriptions . . . 742
Table 292. Correspondence between analog voltages and digital values . . . 742
Table 293. Command message format for external device operation – field descriptions . . . 744
Table 294. Result message format for external device operation – field descriptions . . . 746
Table 295. Null message format for external device operation – field descriptions . . . 748
Table 296. CFIFO scan trigger mode - command transfer start/stop summary . . . 767
Table 297. Command FIFO status switching condition . . . 769
Table 298. ADC clock configuration example (system clock frequency = 120 MHz) . . . 784
Table 299. Gain calibration constant format field descriptions . . . 790
Table 300. Binary and decimal representations of the gain constant . . . 790
Table 301. ADC0/1_EMUX bit combinations . . . 794
Table 302. Non-multiplexed channel assignments . . . 794
Table 303. Multiplexed channel assignments . . . 795
Table 304. Encoding of MA pins . . . 798
Table 305. eQADC FIFO interrupt summary . . . 800
Table 306. eQADC FIFO DMA summary . . . 801
Table 307. PSI Input and Output Data Buses Content . . . 809
Table 308. Application of each CQueue . . . 814

Table 309. Example of CQueue commands . . . 818
Table 310. Calibration example . . . 825
Table 311. Terminology comparison between QADC and eQADC . . . 828
Table 312. Usage comparison between QADC and eQADC system . . . 829
Table 313. Operation mode selection . . . 832
Table 314. Block memory map . . . 834
Table 315. DECFILTER_MCR field descriptions . . . 836
Table 316. FTYPE[1:0] configuration—Filter type selection . . . 839
Table 317. ISEL/MIXM definition – read/write to input/output buffers . . . 839
Table 318. DECFILTER_MSR field descriptions . . . 840
Table 319. DECFILTER_IB field descriptions . . . 843
Table 320. DECFILTER_OB field descriptions . . . 844
Table 321. DECFILTER_COEFn field descriptions . . . 845
Table 322. DECFILTER_TAPn field descriptions . . . 845
Table 323. DECFILTER_EDID field descriptions . . . 846
Table 324. Parallel side interface memory map for decimation filter data exchange . . . 846
Table 325. DECFILTER_IOB field descriptions . . . 847
Table 326. M_CTRL[1:0] field configuration . . . 848
Table 327. Coefficient values given by SPW digital filter design tool . . . 861
Table 328. Coefficient values for decimation filter . . . 862
Table 329. Signal properties . . . 869
Table 330. DSPI memory map . . . 870
Table 331. DSPI_MCR field descriptions . . . 872
Table 332. DSPI_TCR field descriptions . . . 874
Table 333. DSPI_CTAR*n* field descriptions in master mode . . . 875
Table 334. DSPI SCK duty cycle . . . 878
Table 335. Delay scaler encoding . . . 879
Table 336. DSPI baud rate scaler . . . 879
Table 337. DSPI_CTAR*0*, DSPI_CTAR*1* field descriptions in slave mode . . . 879
Table 338. DSPI_SR field descriptions . . . 880
Table 339. DSPI_RSER field descriptions . . . 883
Table 340. DSPI_PUSHR field descriptions in master mode . . . 885
Table 341. DSPI_PUSHR field descriptions in slave mode . . . 887
Table 342. DSPI_POPR field descriptions . . . 887
Table 343. DSPI_TXFR*n* field descriptions . . . 888
Table 344. DSPI_RXFR*n* field descriptions . . . 888
Table 345. DSPI_DSICR field descriptions . . . 889
Table 346. DSPI_SDR field descriptions . . . 892
Table 347. DSPI_ASDR field descriptions . . . 892
Table 348. DSPI_COMPR field descriptions . . . 893
Table 349. DSPI_DDR field descriptions . . . 893
Table 350. DSPI_DSICR1 field descriptions . . . 894
Table 351. DSI data transfer initiation control . . . 902
Table 352. Baud Rate computation example . . . 907
Table 353. PCS to SCK Delay computation example . . . 907
Table 354. After SCK Delay computation example . . . 907
Table 355. Delay after Transfer computation example . . . 908
Table 356. Peripheral Chip Select Strobe Assert computation example . . . 909
Table 357. Peripheral Chip Select Strobe Negate computation example . . . 909
Table 358. DSPI slave mode system constraints . . . 922
Table 359. Differences between TSB and ITSB modes . . . 925
Table 360. Interrupt and DMA Request Conditions . . . 928

Table 361. Baud rate values (bps) . . . 932
Table 362. Delay values . . . 933
Table 363. SPC563Mxx eSCI signal names . . . 936
Table 364. Acronyms and abbreviations . . . 936
Table 365. Glossary . . . 937
Table 366. Block memory map . . . 940
Table 367. Register conventions . . . 941
Table 368. SCI_CR1 field descriptions . . . 942
Table 369. Receive source mode selection . . . 943
Table 370. SCI_CR2 field descriptions . . . 944
Table 371. SCI_DR field descriptions . . . 946
Table 372. SCI_SR field descriptions . . . 947
Table 373. SCI_LCR field descriptions . . . 950
Table 374. SCI_LTR field descriptions . . . 952
Table 375. SCI_LRR field descriptions . . . 953
Table 376. SCI_LPR field descriptions . . . 954
Table 377. Supported data frame formats for RX and TX . . . 955
Table 378. Supported data frame formats for RX only . . . 955
Table 379. Supported break character formats . . . 957
Table 380. Supported idle character formats . . . 958
Table 381. Baud rates error example (MCLK = 10.2 MHz) . . . 959
Table 382. Faster receiver maximum tolerance . . . 961
Table 383. Slower receiver maximum tolerance . . . 962
Table 384. Transmitter states . . . 964
Table 385. Transmitter application transitions . . . 964
Table 386. Transmitter module transitions . . . 964
Table 387. Transmit source priority . . . 965
Table 388. Receiver states . . . 967
Table 389. Receiver application transition . . . 968
Table 390. Receiver module transition . . . 968
Table 391. Start bit verification result . . . 972
Table 392. Start bit noise detection . . . 973
Table 393. Data bit sampling . . . 974
Table 394. Stop bit verification . . . 976
Table 395. eSCI interrupt flags and enable bits . . . 985
Table 396. FlexCAN signals . . . 991
Table 397. Module memory map . . . 992
Table 398. Message buffer MB0 memory mapping . . . 993
Table 399. Message buffer field descriptions . . . 994
Table 400. Message buffer code for Rx buffers . . . 995
Table 401. Message buffer code for Tx buffers . . . 995
Table 402. Rx FIFO field descriptions . . . 997
Table 403. CAN_MCR field descriptions . . . 999
Table 404. IDAM coding . . . 1003
Table 405. CAN_CR field descriptions . . . 1004
Table 406. CAN_RXGMASK field descriptions . . . 1008
Table 407. CAN_ESR field descriptions . . . 1010
Table 408. CAN_IMRH field descriptions . . . 1013
Table 409. CAN_IMRL field descriptions . . . 1014
Table 410. CAN_IFRH field descriptions . . . 1014
Table 411. CAN_IFRL field descriptions . . . 1015
Table 412. CAN_RXIMR0–CAN_RXIMR63 field descriptions . . . 1016

Table 413. Recommended FEN and BCC settings . . . 1024
Table 414. Time segment syntax . . . 1028
Table 415. CAN standard compliant bit time segment settings . . . 1028
Table 416. Minimum ratio between peripheral clock frequency and CAN bit rate . . . 1029
Table 417. Wake-up from doze mode . . . 1031
Table 418. Wake-up from stop mode . . . 1032
Table 419. PIT_RTI memory map . . . 1036
Table 420. Timer Channel n / RTI Channel . . . 1037
Table 421. PITMCR field descriptions . . . 1037
Table 422. LDVAL field descriptions . . . 1038
Table 423. CVAL field descriptions . . . 1039
Table 424. TCTRL field descriptions . . . 1040
Table 425. TFLG field descriptions . . . 1040
Table 426. PMC ADC monitor channels . . . 1044
Table 427. Power management controller external signals (maximum ratings) . . . 1047
Table 428. Power management controller memory map . . . 1048
Table 429. PMC_MCR field descriptions . . . 1049
Table 430. PMC_TRIMR field descriptions . . . 1051
Table 431. PMC_SR field descriptions . . . 1054
Table 432. Non-Volatile User Options Register (NVUSRO) field descriptions . . . 1059
Table 433. Device-specific parameters . . . 1065
Table 434. Device identification register parameters . . . 1065
Table 435. Device JTAG ID . . . 1065
Table 436. Device-specific auxiliary TAP controller instructions . . . 1066
Table 437. JTAG signal properties . . . 1068
Table 438. Device Identification Register field descriptions . . . 1070
Table 439. CENSOR_CTRL field descriptions . . . 1071
Table 440. JTAG Instructions . . . 1074
Table 441. Device parameter values . . . 1078
Table 442. NPC signal properties . . . 1081
Table 443. NPC registers . . . 1083
Table 444. DID field descriptions . . . 1084
Table 445. PCR field descriptions . . . 1085
Table 446. MCKO_DIV values . . . 1086
Table 447. NPC reset configuration options . . . 1087
Table 448. NPC output messages . . . 1088
Table 449. Implemented instructions . . . 1090
Table 450. Loading NEXUS-ENABLE instruction . . . 1092
Table 451. Write to a 32-bit Nexus client register . . . 1093
Table 452. Temperature Calculation Constants Register 0 field descriptions . . . 1100
Table 453. Temperature Calculation Constants Register 1 field descriptions . . . 1100
Table 454. Document revision history . . . 1102

List of figures

Figure 1. SPC563Mxx Block Diagram (SPC563M64) . . . 47
Figure 2. 144-pin LQFP Pinout (top view; all 144-pin devices) . . . 86
Figure 3. 176-pin LQFP Pinout (SPC563M64; top view) . . . 87
Figure 4. DSPI_B Connectivity . . . 122
Figure 5. DSPI_C Connectivity . . . 124
Figure 6. External reset flow diagram . . . 128
Figure 7. Internal reset flow diagram . . . 129
Figure 8. Reset Configuration Half Word (RCHW) . . . 134
Figure 9. Reset Configuration Timing . . . 135
Figure 10. FMPLL bypass mode with crystal reference . . . 138
Figure 11. FMPLL bypass mode with external reference . . . 139
Figure 12. FMPLL normal mode with crystal reference . . . 140
Figure 13. FMPLL normal mode with external reference . . . 141
Figure 14. MDIS and halt clock gating . . . 143
Figure 15. DMA block diagram . . . 150
Figure 16. DMA Control Register (DMA_CR) . . . 160
Figure 17. DMA Error Status Register (DMA_ESR) . . . 162
Figure 18. DMA Enable Request (DMA_ERQRL) Registers . . . 164
Figure 19. DMA Enable Error Interrupt (DMA_EEIRL) Register . . . 165
Figure 20. DMA Set Enable Request Register (DMA_SERQR) . . . 165
Figure 21. DMA Clear Enable Request Register (DMA_CERQR) . . . 166
Figure 22. DMA Set Enable Error Interrupt Register (DMA_SEEIR) . . . 166
Figure 23. DMA Clear Enable Error Interrupt Register (DMA_CEEIR) . . . 167
Figure 24. DMA Clear Interrupt Request Register (DMA_CIRQR) . . . 167
Figure 25. DMA Clear Error Register (DMA_CER) . . . 168
Figure 26. DMA Set START Bit Register (DMA_SSBR) . . . 168
Figure 27. DMA Clear DONE Status Bit Register (DMA_CDSBR) . . . 169
Figure 28. DMA Interrupt Request Register (DMA_IRQRL) . . . 170
Figure 29. DMA Error (DMA_ERL) Registers . . . 171
Figure 30. DMA Channel n Priority (DMA_CPRn) Register . . . 172
Figure 31. TCDn Word 0 (TCDn.saddr) field . . . 173
Figure 32. TCDn Word 1 (TCDn.{soff,smod,ssize,dmod,dsize}) fields . . . 174
Figure 33. TCDn Word 2 (TCDn.nbytes) field . . . 175
Figure 34. TCDn Word 3 (TCDn.slast) field . . . 176
Figure 35. TCDn Word 4 (TCDn.daddr) field . . . 177
Figure 36. TCDn Word 5 (TCDn.{citer,doff}) fields . . . 177
Figure 37. TCDn Word 6 (TCDn.dlast_sga) field . . . 179
Figure 38. TCDn Word 7 (TCDn.{biter,control/status}) fields . . . 180
Figure 39. DMA operation, part 1 . . . 186
Figure 40. DMA operation, part 2 . . . 187
Figure 41. DMA operation, part 3 . . . 188
Figure 42. ipd_req hardware handshake . . . 199
Figure 43. XBAR device-specific block diagram . . . 200
Figure 44. Key to register fields . . . 203
Figure 45. Master Priority Register (XBAR_MPRn) . . . 204
Figure 46. Slave General Purpose Control Register (XBAR_SGPCRn) . . . 206
Figure 47. PBRIDGE interface . . . 210
Figure 48. SPC563Mxx flash block diagram . . . 212

Figure 49. LC flash array organization . . . 221
Figure 50. Bus Interface Unit Configuration Register (BIUCR) . . . 238
Figure 51. Bus Interface Unit Access Protection Register (BIUAPR) . . . 241
Figure 52. Bus Interface Unit Configuration Register 2 (BIUCR2) . . . 243
Figure 53. PFLASH Configuration Register 3 (PFCR3) . . . 244
Figure 54. Module Configuration Register (MCR) . . . 246
Figure 55. Low/Mid Address Space Block Locking Register (LMLR) . . . 251
Figure 56. High Address Space Block Locking Register (HLR) . . . 253
Figure 57. Secondary Low/Mid Address Space Block Lock Register (SLMLR) . . . 255
Figure 58. Low/Mid Address Space Block Select Register (LMSR) . . . 257
Figure 59. High Address Space Block Select Register (HSR) . . . 258
Figure 60. Address Register (AR) . . . 259
Figure 61. User Test 0 (UT0) Register . . . 260
Figure 62. User Test 1 (UT1) Register . . . 262
Figure 63. User Test 2 (UT2) Register . . . 263
Figure 64. User Multiple Input Signature Register 0 (UMISR0) Register . . . 264
Figure 65. User Multiple Input Signature Register 1 (UMISR1) Register . . . 265
Figure 66. User Multiple Input Signature Register 2 (UMISR2) Register . . . 266
Figure 67. User Multiple Input Signature Register 3 (UMISR3) Register . . . 267
Figure 68. User Multiple Input Signature Register 4 (UMISR4) Register . . . 268
Figure 69. SRAM block diagram . . . 270
Figure 70. EBI Module Configuration Register (EBI_MCR) . . . 281
Figure 71. EBI Transfer Error Status Register (EBI_TESR) . . . 283
Figure 72. EBI Bus Monitor Control Register (EBI_BMCR) . . . 284
Figure 73. EBI Base Registers (EBI_BR0–EBI_BR3, EBI_CAL_BR0–3) . . . 285
Figure 74. EBI Option Registers (EBI_OR0–EBI_OR3, EBI_CAL_OR0–3) . . . 288
Figure 75. Bank base address & match structure . . . 290
Figure 76. Basic transfer protocol . . . 294
Figure 77. Basic flow diagram of a single beat read cycle . . . 296
Figure 78. Single beat 32-bit read cycle, CS access, zero wait states . . . 297
Figure 79. Single beat 32-bit read cycle, CS access, one wait state . . . 297
Figure 80. Single beat 32-bit read cycle, non-CS access, zero wait states . . . 298
Figure 81. Single beat 32-bit read cycle, CS access, zero wait states (EOE = 0b01, 0b10) . . . 298
Figure 82. Basic flow diagram of a single beat write cycle . . . 299
Figure 83. Single beat 32-bit write cycle, CS access, zero wait states . . . 300
Figure 84. Single beat 32-bit write cycle, CS access, one wait state . . . 300
Figure 85. Single beat 32-bit write cycle, non-CS access, zero wait states . . . 301
Figure 86. Single beat 32-bit write cycle, CS access, zero wait states, LWRN = 1 . . . 301
Figure 87. Back-to-back 32-bit reads to the same CS bank . . . 302
Figure 88. Back-to-back 32-bit reads to different CS banks . . . 303
Figure 89. Write after read to the same CS bank . . . 304
Figure 90. Back-to-back 32-bit writes to the same CS bank . . . 305
Figure 91. Read after write to the same CS bank . . . 306
Figure 92. Read after write to the same CS bank (EBI_BR[GCSN] = 1) . . . 307
Figure 93. Basic flow diagram of a burst read cycle . . . 310
Figure 94. Burst 32-bit read cycle, zero wait states . . . 311
Figure 95. Burst 32-bit read cycle, one initial wait state . . . 311
Figure 96. Burst 32-bit read cycle, one wait state between beats, TBDIP = 0 . . . 312
Figure 97. Burst 32-bit read cycle, one wait state between beats, TBDIP = 1 . . . 312
Figure 98. Single beat 32-bit write cycle, 16-bit port size, basic timing . . . 314
Figure 99. 32-byte write cycle with external TA, basic timing . . . 315
Figure 100. 32-byte read with back-to-back 4-word bursts to 32-bit port, zero wait states . . . 316

Figure 101. Single beat 64-bit read cycle, 16-bit port size, basic timing 317
Figure 102. 32-byte read with 4 back-to-back 2-word bursts to 32-bit port, zero wait states (SBL = 1) 318
Figure 103. Internal operand representation 320
Figure 104. Interface to different port size devices 320
Figure 105. Termination signals protocol timing diagram 323
Figure 106. External master back-to-back writes to same CS bank 324
Figure 107. 32-bit read from MCU with DBM = 1 325
Figure 108. 32-bit write to MCU with DBM = 1 325
Figure 109. Back-to-back 32-bit reads to CS, CAL_CS banks 327
Figure 110. Small access (32-bit read to 16-bit port) on address/data multiplexed bus 333
Figure 111. MCU connected to SDR burst memory 334
Figure 112. MCU connected to asynchronous memory 335
Figure 113. Read operation to asynchronous memory, three initial wait states 336
Figure 114. Write operation to asynchronous memory, three initial wait states 337
Figure 115. MCU connected to multiple memories 338
Figure 116. Address/data multiplexing with both 16-bit and 32-bit memories in single master system 340
Figure 117. INTC block diagram 342
Figure 118. INTC Module Configuration Register (INTC_MCR) 346
Figure 119. INTC Current Priority Register (INTC_CPR) 347
Figure 120. INTC Interrupt Acknowledge Register (INTC_IACKR) 348
Figure 121. INTC End Of Interrupt Register (INTC_EOIR) 349
Figure 122. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR0_3) 349
Figure 123. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR0_3) 350
Figure 124. INTC Priority Select Register 0–3 (INTC_PSR0_3) 351
Figure 125. INTC Priority Select Register 508 - 511 (INTC_PSR508_511) 351
Figure 126. Software vector mode handshaking timing diagram 356
Figure 127. Hardware vector mode handshaking timing diagram 357
Figure 128. Software Vector Mode Interrupt Exception Handler Address Calculation 367
Figure 129. Hardware Vector Mode Interrupt Exception Handler Address Calculation 368
Figure 130. SIU block diagram 388
Figure 131. MCU ID Register 2 (SIU_MIDR2) 392
Figure 132. MCU ID Register (SIU_MIDR) 394
Figure 133. Reset Status Register (SIU_RSR) 396
Figure 134. System Reset Control Register (SIU_SRCR) 398
Figure 135. External IRQ Status Register (SIU_EISR) 399
Figure 136. DMA/Interrupt Request Enable Register (SIU_DIRER) 400
Figure 137. DMA/Interrupt Request Select Register (SIU_DIRSR) 401
Figure 138. Overrun Status Register (SIU_OSR) 401
Figure 139. Overrun Request Enable Register (SIU_ORER) 402
Figure 140. IRQ Rising-Edge Event Enable Register (SIU_IREER) 403
Figure 141. External IRQ Falling-Edge Event Enable Register (SIU_IFEER) 404
Figure 142. IRQ Digital Filter Register (SIU_IDFR) 405
Figure 143. Sample PCR map 406
Figure 144. CAN_A_TX_SCI_A_TX_GPIO[83] Pad Configuration Register (SIU_PCR83) 409
Figure 145. CAN_A_TX_SCI_A_TX_GPIO[84] Pad Configuration Register (SIU_PCR84) 409
Figure 146. CAN_C_TX_GPIO[87] Pad Configuration Register (SIU_PCR87) 410
Figure 147. CAN_C_RX_GPIO[88] Pad Configuration Register (SIU_PCR88) 410
Figure 148. SCI_A_TX_eMIOS[13]_GPIO[89] Pad Configuration Register (SIU_PCR89) 410
Figure 149. SCI_A_RX_eMIOS[15]_GPIO[90] Pad Configuration Register (SIU_PCR90) 411
Figure 150. SCI_B_TX_GPIO[91] Pad Configuration Register (SIU_PCR91) 411
Figure 151. SCI_B_RX_GPIO[92] Pad Configuration Register (SIU_PCR92) 412

Figure 152. GPIO[98:99] Pad Configuration Registers (SIU_PCR98 – SIU_PCR99) 412
Figure 153. DSPI_B_SCK_DSPI_C_PCS[1]_GPIO[102] Pad Configuration Register (SIU_PCR102) 412
Figure 154. DSPI_B_SIN_DSPI_C_PCS[2]_GPIO[103] Pad Configuration Register (SIU_PCR103). 413
Figure 155. DSPI_B_SOUT_DSPI_C_PCS[5]_GPIO[104] Pad Configuration Register (SIU_PCR104). 413
Figure 156. DSPI_B_PCS[0]_GPIO[105] Pad Configuration Register (SIU_PCR105) 414
Figure 157. DSPI_B_PCS[1]_GPIO[106] Pad Configuration Register (SIU_PCR106) 414
Figure 158. DSPI_B_PCS[2]_DSPI_C_SOUT_GPIO[107] Pad Configuration Register (SIU_PCR107). 415
Figure 159. DSPI_B_PCS[3]_DSPI_C_SIN_GPIO[108] Pad Configuration Register (SIU_PCR108). 415
Figure 160. DSPI_B_PCS[4]_DSPI_C_SCK_GPIO[109] Pad Configuration Register (SIU_PCR109) 416
Figure 161. DSPI_B_PCS[5]_DSPI_C_PCS[0]_GPIO[110] Pad Configuration Register (SIU_PCR110) 416
Figure 162. eTPU_A[0]_eTPU_A[12]_eTPU_A[19]_GPIO[114] Pad Configuration Register (SIU_PCR114) 417
Figure 163. eTPU_A[1:4]_eTPU_A[13:16]_GPIO[115:118] Pad Configuration Register (SIU_PCR115 – SIU_PCR118) 417
Figure 164. eTPU_A[5]_eTPU_A[17]_DSPI_B_SCK_LVDS−_GPIO[119] Pad Configuration Register (SIU_PCR119) 418
Figure 165. eTPU_A[6]_eTPU_A[18]_DSPI_B_SCK_LVDS+_GPIO[120] Pad Configuration Register (SIU_PCR120) 418
Figure 166. eTPU_A[7]_eTPU_A[19]_DSPI_B_SOUT_LVDS−_eTPU_A[6]_GPIO[121] Pad Configuration Register (SIU_PCR121). 419
Figure 167. eTPU_A[8]_eTPU_A[20]_DSPI_B_SOUT_LVDS+_GPIO[122] Pad Configuration Register (SIU_PCR122) 419
Figure 168. eTPU_A[9:11]_eTPU_A[21:23]_GPIO[123:125] Pad Configuration Register (SIU_PCR123 – SIU_PCR125) 420
Figure 169. eTPU_A[12]_DSPI_B_PCS[1]_GPIO[126] Pad Configuration Register (SIU_PCR126) . 420
Figure 170. eTPU_A[13]_DSPI_B_PCS[3]_GPIO[127] Pad Configuration Register (SIU_PCR127) . 421
Figure 171. eTPU_A[14]_DSPI_B_PCS[4]_eTPU_A[9]_GPIO[128] Pad Configuration Register (SIU_PCR128) 421
Figure 172. eTPU_A[15]_DSPI_B_PCS[5]_GPIO[129] Pad Configuration Register (SIU_PCR129) . 422
Figure 173. eTPU_A[16:19]_GPIO[130:133] Pad Configuration Registers (SIU_PCR130 – SIU_PCR133) 422
Figure 174. eTPU_A[20:21]_IRQ[8:9]_GPIO[134:135] Pad Configuration Register (SIU_PCR134 – SIU_PCR135) 423
Figure 175. eTPU_A[22]_IRQ[10]_eTPU_A[17]_GPIO[136] Pad Configuration Register (SIU_PCR136) 423
Figure 176. eTPU_A[23]_IRQ[11]_eTPU_A[21]_GPIO[137] Pad Configuration Register (SIU_PCR137) 424
Figure 177. eTPU_A[24]_IRQ[12]_DSPI_C_SCK_LVDS−_GPIO[138] Pad Configuration Register (SIU_PCR138) 424
Figure 178. eTPU_A[25]_IRQ[13]_DSPI_C_SCK_LVDS+_GPIO[139] Pad Configuration Register (SIU_PCR139) 425
Figure 179. eTPU_A[26]_IRQ[14]_DSPI_C_SOUT_LVDS−_GPIO[140] Pad Configuration Register (SIU_PCR140) 425
Figure 180. eTPU_A[27]_IRQ[15]_DSPI_C_SOUT_LVDS+_DSPI_B_SOUT_GPIO[141] Pad Configuration Register (SIU_PCR141). 426
Figure 181. eTPU_A[28:29]_DSPI_C_PCS[1:2]_GPIO[142:143] Pad Configuration Register (SIU_PCR142 – SIU_PCR143) 426
Figure 182. eTPU_A[30]_DSPI_C_PCS[3]_eTPU_A[11]_GPIO[144] Pad Configuration Register (SIU_PCR144) 427

Figure 183. eTPU_A[31]_DSPI_C_PCS[4]_eTPU_A[13]_GPIO[145] Pad Configuration Register (SIU_PCR145) . . . 427
Figure 184. eMIOS[0]_eTPU_A[0]_eTPU_A[25]_GPIO[179] Pad Configuration Register (SIU_PCR179) . . . 428
Figure 185. eMIOS[1]_eTPU_A[1]_GPIO[180] Pad Configuration Register (SIU_PCR180). . . . 428
Figure 186. eMIOS[2]_eTPU_A[2]_GPIO[181] Pad Configuration Register (SIU_PCR181). . . . 429
Figure 187. eMIOS[4]_eTPU_A[4]_GPIO[183] Pad Configuration Register (SIU_PCR183). . . . 429
Figure 188. eMIOS[8]_eTPU_A[8]_SCI_B_TX_GPIO[187] Pad Configuration Register (SIU_PCR187) . . . 430
Figure 189. eMIOS[9]_eTPU_A[9]_SCI_B_RX_GPIO[188] Pad Configuration Register (SIU_PCR188) . . . 430
Figure 190. eMIOS[10:11]_GPIO[189:190] Pad Configuration Register (SIU_PCR189 – SIU_PCR190). . . . 431
Figure 191. eMIOS[12]_DSPI_C_SOUT_eTPU_A[27]_GPIO[191] Pad Configuration Register (SIU_PCR191) . . . 431
Figure 192. eMIOS[13]_GPIO[192] Pad Configuration Register (SIU_PCR192). . . . 432
Figure 193. eMIOS[14]_IRQ[0]_eTPU_A[29]_GPIO[193] Pad Configuration Register (SIU_PCR193)432
Figure 194. eMIOS[15]_IRQ[1]_GPIO[194] Pad Configuration Register (SIU_PCR194) . . . 433
Figure 195. eMIOS[23]_GPIO[202] Pad Configuration Register (SIU_PCR202). . . . 433
Figure 196. GPIO[206:207] Pad Configuration Registers (SIU_PCR206 – SIU_PCR207) . . . 434
Figure 197. PLLREF_IRQ[4]_ETRIG[2]_GPIO[208] Pad Configuration Register (SIU_PCR208). . . . 434
Figure 198. BOOTCFG1_IRQ[3]_ETRIG[3]_GPIO[212] Pad Configuration Register (SIU_PCR212). 435
Figure 199. WKPCFG_NMI_DSPI_B_SOUT_GPIO[213] Pad Configuration Register (SIU_PCR213)435
Figure 200. AN[12]_MA[0]_eTPU_A[19]_SDS Pad Configuration Register (SIU_PCR215) . . . 436
Figure 201. AN[13]_MA[1]_eTPU_A[21]_SDO Pad Configuration Register (SIU_PCR216). . . . 436
Figure 202. AN[14]_MA[2]_eTPU_A[27]_SDI Pad Configuration Register (SIU_PCR217). . . . 437
Figure 203. AN[15]_FCK_eTPU_A[29] Pad Configuration Register (SIU_PCR218). . . . 438
Figure 204. MCKO_GPIO[219] Pad Configuration Register (SIU_PCR219). . . . 438
Figure 205. MDO[0]_eTPU_A[13]_GPIO[220] Pad Configuration Register (SIU_PCR220) . . . 439
Figure 206. MDO[1]_eTPU_A[19]_GPIO[221] Pad Configuration Register (SIU_PCR221) . . . 439
Figure 207. MDO[2]_eTPU_A[21]_GPIO[222] Pad Configuration Register (SIU_PCR222) . . . 440
Figure 208. MDO[3]_eTPU_A[25]_GPIO[223] Pad Configuration Register (SIU_PCR223) . . . 440
Figure 209. MSEO[0]_eTPU_A[27]_GPIO[224] Pad Configuration Register (SIU_PCR224) . . . 441
Figure 210. MSEO[1]_eTPU_A[29]_GPIO[225] Pad Configuration Register (SIU_PCR225) . . . 441
Figure 211. EVTO_eTPU_A[4]_GPIO[227] Pad Configuration Register (SIU_PCR227) . . . 442
Figure 212. TDO_eMIOS[6]_GPIO[228] Pad Configuration Register (SIU_PCR228). . . . 442
Figure 213. CLKOUT Pad Configuration Register (SIU_PCR229) . . . 442
Figure 214. RSTOUT Pad Configuration Register (SIU_PCR230) . . . 443
Figure 215. EVTI_eTPU_A[2]_GPIO[231] Pad Configuration Register (SIU_PCR231) . . . 443
Figure 216. TDI_eMIOS[5]_GPIO[232] Pad Configuration Register (SIU_PCR232). . . . 443
Figure 217. CAL_CS[0] Pad Configuration Register (SIU_PCR336). . . . 444
Figure 218. CAL_CS[2:3]_CAL_ADDR[10:11] Pad Configuration Registers (SIU_PCR338 – SIU_PCR339) . . . 444
Figure 219. CAL_ADDR[12:15] Pad Configuration Register (SIU_PCR340). . . . 444
Figure 220. CAL_DATA[0:15] Pad Configuration Register (SIU_PCR341) . . . 445
Figure 221. CAL_RD_WR, CAL_WE[0:1]/BE[0:1] and CAL_OE Pad Configuration Register (SIU_PCR342) . . . 445
Figure 222. CAL_TS_ALE Pad Configuration Register (SIU_PCR343) . . . 445
Figure 223. ALT_MCKO and ALT_EVTO Pad Configuration Register (SIU_PCR344). . . . 446
Figure 224. CAL_ADDR[16:27]_ALT_MDO[0:11], CAL_ADDR[28:29]_CAL_MSEO[0:1] and CAL_ADDR[30]_ALT_EVTI Pad Configuration Register (SIU_PCR345). . . . 446
Figure 225. Pad Configuration Register 350 – 381 (SIU_PCR350 – SIU_PCR381) . . . 446

Figure 226. Pad Configuration Register 382 – 389 (SIU_PCR382 – SIU_PCR389) 448
Figure 227. Pad Configuration Register 390 – 413 (SIU_PCR390 – SIU_PCR413) 448
Figure 228. GPIO Pin Data Out Register 83 – 86 (SIU_GPDO83 – SIU_GPDO86) 450
Figure 229. GPIO Pin Data Out Register 230 – 232 (SIU_GPDO230 – SIU_GPDO232) 450
Figure 230. GPIO Pin Data Out Register 350 – 353 (SIU_GPDO350 – SIU_GPDO353) 451
Figure 231. GPIO Pin Data Out Register 410 – 413 (SIU_GPDO410 – SIU_GPDO413) 451
Figure 232. GPIO Pin Data In Register 83 – 86 (SIU_GPDI83 – SIU_GPDI86) 452
Figure 233. GPIO Pin Data In Register 230 – 232 (SIU_GPDI230 – SIU_GPDI232) 453
Figure 234. Trigger selection for eQADC CFIFO queue 5 454
Figure 235. eQADC Trigger Input Select Register (SIU_ETISR) 454
Figure 236. External IRQ Input Select Register (SIU_EIISR) 456
Figure 237. DSPI Input Select Register (SIU_DISR) 459
Figure 238. IMUX Select Register 3 (SIU_ISEL3) 461
Figure 239. IMUX Select Register 8 (SIU_ISEL8) 467
Figure 240. IMUX Select Register 9 (SIU_ISEL9) 468
Figure 241. Chip Configuration Register (SIU_CCR) 470
Figure 242. External Clock Control Register (SIU_ECCR) 471
Figure 243. Compare A High Register (SIU_CARH) 472
Figure 244. Compare A Low Register (SIU_CARL) 473
Figure 245. Compare B High Register (SIU_CBRH) 473
Figure 246. Compare B Low Register (SIU_CBRL) 474
Figure 247. System Clock Register (SIU_SYSDIV) 474
Figure 248. Halt Register (SIU_HLT) 475
Figure 249. Halt Acknowledge Register (SIU_HLTACK) 478
Figure 250. SIU DMA/Interrupt request diagram 481
Figure 251. eQADC external trigger input multiplexing 482
Figure 252. SIU external interrupt input multiplexing 483
Figure 253. FMPLL block diagram 485
Figure 254. Synthesizer Control Register (SYNCR) 489
Figure 255. Synthesizer Status Register (SYNSR) 491
Figure 256. Enhanced Synthesizer Control Register 1 (ESYNCR1) 494
Figure 257. Enhanced Synthesizer Control Register 2 (ESYNCR2) 496
Figure 258. Synthesizer FM Modulation Register (SYNFMMR) 497
Figure 259. Clock quality monitor 501
Figure 260. Triangular frequency modulation 504
Figure 261. ECC Configuration Register (ECSM_ECR) 507
Figure 262. ECC Status Register (ECSM_ESR) 509
Figure 263. ECC Error Generation Register (ECSM_EEGR) 510
Figure 264. Flash ECC Address Register (ECSM_FEAR) 513
Figure 265. Flash ECC Master Number Register (ECSM_FEMR) 514
Figure 266. Flash ECC Attributes (ECSM_FEAT) Register 514
Figure 267. Flash ECC Data Register (ECSM_FEDR) 515
Figure 268. RAM ECC Address Register (ECSM_REAR) 516
Figure 269. RAM ECC Syndrome Register (ECSM_PRESR) 517
Figure 270. RAM ECC Master Number Register (ECSM_REMR) 520
Figure 271. RAM ECC Attributes (ECSM_REAT) Register 521
Figure 272. RAM ECC Data Register (ECSM_REDR) 522
Figure 273. Key to Register Fields 524
Figure 274. STM Control Register (STM_CR) 525
Figure 275. STM Count Register (STM_CNT) 525
Figure 276. STM Channel Control Register (STM_CCRn) 526
Figure 277. STM Channel Interrupt Register (STM_CIRn) 526

Figure 278. STM Channel Compare Register (STM_CMPn) . . . 527
Figure 279. SWT Control Register (SWT_CR) . . . 530
Figure 280. SWT Interrupt Register (SWT_IR) . . . 532
Figure 281. SWT Timeout Register (SWT_TO) . . . 533
Figure 282. SWT Window Register (SWT_WN) . . . 533
Figure 283. SWT Service Register (SWT_SR) . . . 533
Figure 284. SWT Counter Output Register (SWT_CO) . . . 534
Figure 285. SWT Service Register (SWT_SR) . . . 534
Figure 286. BAM program flow chart . . . 539
Figure 287. Censorship Word . . . 541
Figure 288. Serial Boot Flash Password . . . 541
Figure 289. Reset Configuration Half Word . . . 542
Figure 290. Reset boot vector . . . 543
Figure 291. Enabling JTAG/Nexus port access on a censored device . . . 545
Figure 292. CAN bit timing . . . 548
Figure 293. Start Address, VLE Bit and Download Size in Bytes . . . 549
Figure 294. eMIOS200 block diagram . . . 555
Figure 295. eMIOS200 Module Configuration Register (EMIOS_MCR) . . . 560
Figure 296. eMIOS200 Global FLAG Register (EMIOS_GFR) . . . 562
Figure 297. eMIOS200 Output Update Disable Register (EMIOS_OUDR) . . . 562
Figure 298. eMIOS200 Enable Channel Register (EMIOS_UCDIS) . . . 563
Figure 299. eMIOS200 UC A Register (EMIOS_CADR[n]) . . . 563
Figure 300. eMIOS200 UC B register (EMIOS_CBDR[n]) . . . 564
Figure 301. eMIOS200 UC Counter Register (EMIOS_CCNTR[n]) . . . 565
Figure 302. eMIOS200 UC Control Register (EMIOS_CCR[n]) . . . 566
Figure 303. eMIOS200 UC Status Register (EMIOS_CSR[n]) . . . 572
Figure 304. eMIOS200 UC Alternate A register (EMIOS_ALTA[n]) . . . 573
Figure 305. eMIOS200 Full Channel Configuration using Unified Channels only . . . 575
Figure 306. eMIOS200 example configuration . . . 576
Figure 307. Unified Channel block diagram . . . 577
Figure 308. Unified Channel Control and Datapath Block Diagrams . . . 578
Figure 309. Single Action Input Capture with rising edge triggering example . . . 579
Figure 310. Single Action Input Capture with both edges triggering example . . . 580
Figure 311. SAOC example with EDPOL value being transferred to the output flip-flop . . . 580
Figure 312. SAOC example toggling the output flip-flop . . . 581
Figure 313. SAOC example with flag behavior . . . 581
Figure 314. Input Pulse Width Measurement example . . . 582
Figure 315. B1 and A1 updates at EMIOS_CADR[n] and EMIOS_CBDR[n] reads . . . 582
Figure 316. Input Period Measurement example . . . 583
Figure 317. A1 and B1 updates at EMIOS_CADR[n] and EMIOS_CBDR[n] reads . . . 584
Figure 318. Double Action Output Compare with FLAG set on the second match . . . 585
Figure 319. Double Action Output Compare with FLAG set on both matches . . . 585
Figure 320. DAOC with transfer disabling example . . . 586
Figure 321. Pulse/Edge Accumulation continuous mode example . . . 587
Figure 322. Pulse/Edge Accumulation single-shot mode example . . . 588
Figure 323. Pulse/Edge Counting continuous mode example . . . 589
Figure 324. Pulse/Edge Counting single-shot mode example . . . 589
Figure 325. Quadrature Decode mode example with count & direction encoder . . . 590
Figure 326. Quadrature Decode mode example with phase_a & phase_B encoder . . . 590
Figure 327. Windowed Programmable Time Accumulation example . . . 591
Figure 328. Modulus Counter up mode example . . . 593
Figure 329. Modulus Counter up/down mode example . . . 593

Figure 330. Modulus Counter Buffered (MCB) Up Count mode . . . 594
Figure 331. Modulus Counter Buffered (MCB) Up/Down Mode . . . 595
Figure 332. MCB Mode A1 Register Update in Up Counter Mode . . . 595
Figure 333. MCB Mode A1 Register Update in Up/Down Counter Mode . . . 596
Figure 334. OPWFM with immediate update . . . 597
Figure 335. OPWFM with next period update . . . 597
Figure 336. OPWFMB A1 and B1 match to Output Register Delay . . . 598
Figure 337. OPWFMB Mode with A1 = 0 (0% duty cycle) . . . 599
Figure 338. OPWFMB A1 and B1 Registers Update and Flags . . . 600
Figure 339. OPWFMB Mode with Active Output Disable . . . 601
Figure 340. OPWFMB Mode from 100% to 0% Duty Cycle . . . 602
Figure 341. OPWMC with leading dead time insertion . . . 603
Figure 342. OPWMC with trailing dead time insertion . . . 604
Figure 343. OPWMCB A1 and B1 registers load . . . 605
Figure 344. OPWMCB with Lead Dead Time Insertion . . . 606
Figure 345. OPWMCB with Trail Dead Time Insertion . . . 607
Figure 346. OPWMCB with 100% Duty Cycle (A1=4 and B1=3) . . . 608
Figure 347. PWM with immediate update . . . 610
Figure 348. PWM with next period update . . . 610
Figure 349. OPWMB Mode Matches and Flags . . . 611
Figure 350. OPWMB Mode with 0% Duty Cycle . . . 612
Figure 351. OPWMB Mode with Active Output Disable . . . 613
Figure 352. OPWMB Mode from 100% to 0% Duty Cycle . . . 613
Figure 353. OPWMT example . . . 616
Figure 354. OPWMT with 0% Duty Cycle . . . 616
Figure 355. OPWMT with 100% Duty Cycle . . . 617
Figure 356. Input Programmable Filter Submodule Diagram . . . 617
Figure 357. Input Programmable Filter Example . . . 618
Figure 358. REDC block diagram . . . 619
Figure 359. Timing diagram for the Red STAC bus and REDC output . . . 619
Figure 360. Time base period when running in the fastest prescaler ratio . . . 621
Figure 361. Time base generation with external clock and clear on match start . . . 622
Figure 362. Time base generation with internal clock and clear on match start . . . 622
Figure 363. Time base generation with clear on match end . . . 622
Figure 364. eTPU block diagram . . . 625
Figure 365. eTPU engine block diagram . . . 626
Figure 366. SDM PSE area write . . . 629
Figure 367. PSE accesses . . . 630
Figure 368. eTPU Module Configuration Register (ETPU_MCR) . . . 639
Figure 369. eTPU Coherent Dual-Parameter Controller Register (ETPU_CDCR) . . . 642
Figure 370. eTPU MISC Compare Register (ETPU_MISCCMPR) . . . 643
Figure 371. eTPU SCM Off-Range Data Register (ETPU_SCMOFFDATAR) . . . 644
Figure 372. eTPU Engine Configuration Register (ETPU_ECR) . . . 644
Figure 373. eTPU Time Base Configuration Register (ETPU_TBCR) . . . 648
Figure 374. eTPU Time Base 1 (TCR1) Visibility Register (ETPU_TB1R) . . . 651
Figure 375. eTPU Time Base 2 (TCR2) Visibility Register (ETPU_TB2R) . . . 651
Figure 376. STAC Bus Configuration Register (ETPU_REDCR) . . . 652
Figure 377. eTPU Watchdog Timer Register (ETPU_WDTR) . . . 653
Figure 378. eTPU Idle Register (ETPU_IDLE) . . . 654
Figure 379. eTPU Channel Interrupt Status Register (ETPU_CISR) . . . 655
Figure 380. eTPU Channel Data Transfer Request Status Register (ETPU_CDTRSR) . . . 656
Figure 381. eTPU Channel Interrupt Overflow Status Register (ETPU_CIOSR) . . . 656

Figure 382. eTPU Channel Data Transfer Request Overflow Status Register (ETPU_CDTROSR) . . 657
Figure 383. eTPU Channel Interrupt Enable Register (ETPU_CIER) 658
Figure 384. eTPU Channel Data Transfer Request Enable Register (ETPU_CDTRER) 658
Figure 385. ETPU_WDSR 659
Figure 386. eTPU Channel Pending Service Status Register (ETPU_CPSSR) 660
Figure 387. eTPU Channel Service Status Register (ETPU_CSSR). 660
Figure 388. eTPU Channel n Configuration Register (ETPU_CnCR) 662
Figure 389. eTPU Channel n Status Control Register (ETPU_CnSCR) 663
Figure 390. eTPU Channel x Host Service Request Register (ETPU_CxHSRR) 665
Figure 391. eQADC block diagram 667
Figure 392. eQADC Module Configuration Register (EQADC_MCR) 682
Figure 393. eQADC Null Message Send Format Register (EQADC_NMSFR) 683
Figure 394. eQADC External Trigger Digital Filter Register (EQADC_ETDFR) 684
Figure 395. eQADC CFIFO Push Register x (EQADC_CFPRx) 686
Figure 396. eQADC RFIFO Pop Register x (EQADC_RFPRx) 686
Figure 397. eQADC CFIFO Control Register 0 (EQADC_CFCR0) 687
Figure 398. eQADC CFIFO Control Register 1 (EQADC_CFCR1) 688
Figure 399. eQADC CFIFO Control Register 2 (EQADC_CFCR2) 688
Figure 400. eQADC Interrupt and DMA Control Register 0 (EQADC_IDCR0) 691
Figure 401. eQADC Interrupt and DMA Control Register 1 (EQADC_IDCR1) 691
Figure 402. eQADC Interrupt and DMA Control Register 2 (EQADC_IDCR2) 691
Figure 403. eQADC FIFO and Interrupt Status Register x (EQADC_FISRx) 694
Figure 404. eQADC CFIFO Transfer Counter Register 0 (EQADC_CFTCR0) 698
Figure 405. eQADC CFIFO Transfer Counter Register 1 (EQADC_CFTCR1) 698
Figure 406. eQADC CFIFO Transfer Counter Register 2 (EQADC_CFTCR2) 699
Figure 407. eQADC CFIFO Status Snapshot Register 0 (EQADC_CFSSR0). 700
Figure 408. eQADC CFIFO Status Snapshot Register 1 (EQADC_CFSSR1). 700
Figure 409. eQADC CFIFO Status Snapshot Register 2 (EQADC_CFSSR2). 701
Figure 410. eQADC CFIFO Status Register (EQADC_CFSR) 702
Figure 411. eQADC SSI Control Register (EQADC_SSICR). 703
Figure 412. eQADC SSI Receive Data Register (EQADC_SSIRDR) 705
Figure 413. eQADC STAC Client Configuration Register (EQADC_REDLCCR) 705
Figure 414. eQADC CFIFO0 Registers (EQADC_CF0Rw) (w = 0,..., 3) 706
Figure 415. eQADC CFIFO1 Registers (EQADC_CF1Rw) (w = 0,..., 3) 707
Figure 416. eQADC CFIFO2 Registers (EQADC_CF2Rw) (w = 0,..., 3) 707
Figure 417. eQADC CFIFO3 Registers (EQADC_CF3Rw) (w = 0,..., 3) 707
Figure 418. eQADC CFIFO4 Registers (EQADC_CF4Rw) (w = 0,...,3) 708
Figure 419. eQADC CFIFO5 Registers (EQADC_CF5Rw) (w = 0,..., 3) 708
Figure 420. eQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w = 0,..., 3) 709
Figure 421. eQADC RFIFO0 Registers (EQADC_RF0Rw) (w = 0,..., 3) 709
Figure 422. eQADC RFIFO1 Registers (EQADC_RF1Rw) (w = 0,..., 3) 710
Figure 423. eQADC RFIFO2 Registers (EQADC_RF2Rw) (w = 0,..., 3) 710
Figure 424. eQADC RFIFO3 Registers (EQADC_RF3Rw) (w = 0,..., 3) 710
Figure 425. eQADC RFIFO4 Registers (EQADC_RF4Rw) (w = 0,..., 3) 711
Figure 426. eQADC RFIFO5 Registers (EQADC_RF5Rw) (w = 0,..., 3) 711
Figure 427. ADC0/1 Control Registers (ADC0/1_CR) 715
Figure 428. ADC Time Stamp Control Register (ADC_TSCR) 718
Figure 429. ADC Time Base Counter Register (ADC_TBCR). 719
Figure 430. ADC0/1 Gain Calibration Constant Registers (ADC0/1_GCCR) 720
Figure 431. ADC0/1 Offset Calibration Constant Registers (ADC0/1_OCCR) 720
Figure 432. Alternate Configuration 1–8 Control Registers (ADC_ACR1–8). 721
Figure 433. ADC0/1 Alternate x Gain Register (ADC0/1_AGRx, x = 1–2). 723

Figure 434. ADC0/1 Alternate x Offset Registers (ADC0/1_AORx, x = 1–2) . . . 724
Figure 435. ADC Pull Up/Down Control Register x (ADC_PUDCRx, x = 0–7) . . . 725
Figure 436. Command flow during eQADC operation . . . 727
Figure 437. Result flow during eQADC operation . . . 728
Figure 438. Conversion command format for standard configuration . . . 730
Figure 439. Conversion command format for alternate configurations . . . 733
Figure 440. Write configuration command format for on-chip ADC operation . . . 737
Figure 441. Read configuration command format for on-chip ADC operation . . . 738
Figure 442. ADC result format when FMT = 1 (Right Justified Signed) . . . 741
Figure 443. ADC result format when FMT = 0 (Right Justified Unsigned) . . . 741
Figure 444. Command message format for external device operation . . . 744
Figure 445. Result message format for external device operation . . . 745
Figure 446. Null message send format for external device operation . . . 747
Figure 447. Null message receive format for external device operation . . . 747
Figure 448. CFIFO diagram . . . 751
Figure 449. CFIFO entry pointer example . . . 752
Figure 450. Advance trigger and repeat trigger relationship block diagram . . . 754
Figure 451. Timing diagram of typical use of advance trigger and repeat trigger . . . 755
Figure 452. CFIFO0 in streaming mode diagram . . . 756
Figure 453. CFIFO0 in streaming mode entry pointer example . . . 757
Figure 454. CFIFO prioritization logic . . . 761
Figure 455. ETRIG event propagation example . . . 763
Figure 456. State machine of CFIFO status . . . 768
Figure 457. Trigger overrun on level-trigger mode CFIFOs . . . 772
Figure 458. Command sequence examples . . . 773
Figure 459. External CBuffer status detection at command sequence transfer start . . . 775
Figure 460. Non-coherency event when different CFIFOs use the same CBuffer . . . 776
Figure 461. Non-coherency event when different CFIFOs are using different external CBuffers . . . 777
Figure 462. Non-coherency detection when transfers from a command sequence are interrupted . . . 778
Figure 463. RFIFO diagram . . . 780
Figure 464. RFIFO entry pointer example . . . 781
Figure 465. ADC0/1 clock generation . . . 783
Figure 466. REDLC block diagram . . . 787
Figure 467. Timing diagram for the STAC bus and REDLC output . . . 787
Figure 468. MAC unit diagram . . . 789
Figure 469. Gain calibration constant format . . . 790
Figure 470. On-chip ADC control scheme . . . 792
Figure 471. Overlapping consecutive conversion commands . . . 793
Figure 472. Example of external multiplexing . . . 799
Figure 473. eQADC DMA and interrupt requests . . . 802
Figure 474. eQADC synchronous serial interface block diagram . . . 803
Figure 475. Full duplex pin connection . . . 804
Figure 476. Synchronous serial interface protocol timing . . . 806
Figure 477. Slave driving the MSB and consecutive bits in a data transmission . . . 807
Figure 478. eQADC parallel side interface block diagram . . . 808
Figure 479. PSI Input and Output Data Buses Content . . . 809
Figure 480. RSD ADC block diagram . . . 811
Figure 481. RSD stage block diagram . . . 812
Figure 482. RSD stage transfer function . . . 813
Figure 483. RSD adder . . . 814
Figure 484. Example of a CQueue configuring the on-chip ADCs/external device . . . 816
Figure 485. CQueue/CFIFO interface . . . 820

Figure 486. RQueue/RFIFO interface . . . 821
Figure 487. eQADC command and result queues . . . 823
Figure 488. Quantization error reduction during calibration . . . 826
Figure 489. QADC overview . . . 827
Figure 490. eQADC system overview . . . 827
Figure 491. Decimation filter block diagram . . . 831
Figure 492. Decimation Filter Module Configuration Register (DECFILTER_MCR) . . . 835
Figure 493. Decimation Filter Module Status Register (DECFILTER_MSR) . . . 840
Figure 494. Decimation Filter Interface Input Buffer Register (DECFILTER_IB) . . . 843
Figure 495. Decimation Filter Interface Output Buffer Register (DECFILTER_OB) . . . 844
Figure 496. Decimation Filter Coefficient n Register (DECFILTER_COEFn) . . . 844
Figure 497. Decimation Filter TAPn Register (DECFILTER_TAPn) . . . 845
Figure 498. Decimation Filter Interface Enhanced Input Buffer Register (DECFILTER_EDID) . . . 846
Figure 499. Decimation Filter Interface Input/Output Buffers Register (DECFILTER_IOB) . . . 847
Figure 500. 1 × 4 poles IIR filter functional diagram . . . 852
Figure 501. Fourth order IIR filter implementation block diagram . . . 853
Figure 502. Filter configuration paths (FIR or 1 × 4 poles IIR) . . . 854
Figure 503. Convergent rounding methodology . . . 855
Figure 504. Decimation filter/eQADC interface . . . 861
Figure 505. DSPI block diagram . . . 864
Figure 506. DSPI with Queues and DMA . . . 867
Figure 507. DSPI Module Configuration Register (DSPI_MCR) . . . 871
Figure 508. DSPI Transfer Count Register (DSPI_TCR) . . . 874
Figure 509. DSPI Clock and Transfer Attributes Register 0–7 (DSPI_CTAR0–DSPI_CTAR7)—master mode . . . 875
Figure 510. DSPI Clock and Transfer Attributes Register 0, 1 (DSPI_CTAR0, DSPI_CTAR1)—slave mode . . . 875
Figure 511. DSPI Status Register (DSPI_SR) . . . 880
Figure 512. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER) . . . 883
Figure 513. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode . . . 885
Figure 514. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in slave mode . . . 886
Figure 515. DSPI POP RX FIFO Register (DSPI_POPR) . . . 887
Figure 516. DSPI Transmit FIFO Register 0–15 (DSPI_TXFR0–DSPI_TXFR15) . . . 888
Figure 517. DSPI Receive FIFO Registers 0–15 (DSPI_RXFR0–DSPI_RXFR15) . . . 888
Figure 518. DSPI DSI Configuration Register (DSPI_DSICR) . . . 889
Figure 519. DSPI DSI Serialization Data Register (DSPI_SDR) . . . 891
Figure 520. DSPI DSI Alternate Serialization Data Register (DSPI_ASDR) . . . 892
Figure 521. DSPI DSI Transmit Comparison Register (DSPI_COMPR) . . . 892
Figure 522. DSPI Deserialization Data Register (DSPI_DDR) . . . 893
Figure 523. DSPI DSI Configuration Register 1 (DSPI_DSICR1) . . . 894
Figure 524. SPI and DSI serial protocol overview . . . 896
Figure 525. DSI serialization diagram . . . 901
Figure 526. DSI deserialization diagram . . . 902
Figure 527. Example of Parallel Chaining of DSPIs . . . 904
Figure 528. Example of Serial Chaining of DSPIs . . . 905
Figure 529. Example of System using DSPI in CSI Configuration . . . 906
Figure 530. Communications Clock Prescalers and Scalers . . . 907
Figure 531. Peripheral Chip Select Strobe Timing . . . 908
Figure 532. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 0, FMSZ = 8) . . . 910
Figure 533. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 1, FMSZ = 8) . . . 911
Figure 534. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/4$) . . . 913
Figure 535. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/2$) . . . 913

Figure 536. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/3$) . . . 914
Figure 537. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/2$) . . . 915
Figure 538. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/3$) . . . 915
Figure 539. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/4$) . . . 916
Figure 540. Example of Non-Continuous Format (CPHA = 1, CONT = 0) . . . 917
Figure 541. Example of Continuous Transfer (CPHA = 1, CONT = 1) . . . 917
Figure 542. Example of fast continuous selection format . . . 919
Figure 543. Continuous SCK Timing Diagram (CONT = 0) . . . 920
Figure 544. Continuous SCK Timing Diagram (CONT = 1) . . . 921
Figure 545. DSPI usage in the TSB configuration . . . 923
Figure 546. TSB Downstream Frames . . . 923
Figure 547. TSB Data Frame Format for MSC Dual Receiver Operation . . . 924
Figure 548. Sample MSC downstream transmission using ITSB mode . . . 927
Figure 549. TX FIFO Pointers and Counter . . . 934
Figure 550. eSCI block diagram . . . 938
Figure 551. SCI control register 1 (SCI_CR1) . . . 941
Figure 552. SCI control register 2 (SCI_CR2) . . . 944
Figure 553. SCI data register (SCI_DR) . . . 946
Figure 554. Status register (SCI_SR) . . . 947
Figure 555. LIN control register (SCI_LCR) . . . 949
Figure 556. LIN transmit register (SCI_LTR) . . . 952
Figure 557. LIN receive register (SCI_LRR) . . . 953
Figure 558. LIN polynomial register (SCI_LPR) . . . 954
Figure 559. LIN byte field format . . . 956
Figure 560. SCI frame formats (8 payload bits) . . . 956
Figure 561. SCI frame formats (9 payload bits) . . . 956
Figure 562. SCI frame formats (2 stop bits) . . . 956
Figure 563. Inverted SCI frame formats . . . 957
Figure 564. LIN break symbol format . . . 957
Figure 565. SCI break character formats . . . 957
Figure 566. SCI idle character formats . . . 958
Figure 567. Faster receiver . . . 960
Figure 568. Slower receiver . . . 961
Figure 569. Transmitter state diagram . . . 963
Figure 570. DMA controlled SCI data frame generation . . . 966
Figure 571. Receiver state diagram . . . 967
Figure 572. Dual wire mode . . . 968
Figure 573. Single wire mode . . . 969
Figure 574. Loop mode . . . 969
Figure 575. DMA controlled SCI data frame reception . . . 971
Figure 576. Start bit sampling and strobing . . . 972
Figure 577. Data and stop bit sampling and strobing . . . 974
Figure 578. Data bit synchronization (right shifted edges) . . . 975
Figure 579. Data bit synchronization (left shifted edges) . . . 975
Figure 580. Idle-line wakeup format . . . 977
Figure 581. Address-mark wakeup format . . . 977
Figure 582. Standard LIN frame format . . . 979
Figure 583. CRC Enhanced LIN frame format . . . 979
Figure 584. DMA controlled LIN TX frame generation . . . 980
Figure 585. DMA controlled LIN RX frame generation and reception . . . 982
Figure 586. Fast bit error detection on a LIN bus . . . 983
Figure 587. Timing diagram fast bit error detection . . . 983

Figure 588. FlexCAN block diagram . . . 988
Figure 589. Message buffer structure . . . 993
Figure 590. Rx FIFO structure . . . 997
Figure 591. ID Table 0–7 . . . 997
Figure 592. Module Configuration Register (CAN_MCR) . . . 999
Figure 593. Control Register (CAN_CR) . . . 1003
Figure 594. Free Running Timer (CAN_TIMER) . . . 1007
Figure 595. Rx Global Mask Register (CAN_RXGMASK) . . . 1007
Figure 596. Error Counter Register (CAN_ECR) . . . 1010
Figure 597. Error and Status Register (CAN_ESR) . . . 1010
Figure 598. Interrupt Masks 2 Register (CAN_IMRH) . . . 1013
Figure 599. Interrupt Masks 1 Register (CAN_IMRL) . . . 1013
Figure 600. Interrupt Flags 2 Register (CAN_IFRH) . . . 1014
Figure 601. Interrupt Flags 1 Register (CAN_IFRL) . . . 1015
Figure 602. Rx Individual Mask Registers (CAN_RXIMR0–CAN_RXIMR63) . . . 1016
Figure 603. CAN engine clocking scheme . . . 1026
Figure 604. Segments within the bit time . . . 1027
Figure 605. Arbitration, match and move time windows . . . 1028
Figure 606. Block diagram of PIT_RTI . . . 1035
Figure 607. PIT Module Control Registers (PITMCR) . . . 1037
Figure 608. Timer Load Value Register (LDVAL) . . . 1038
Figure 609. Current Timer Value Register (CVAL) . . . 1039
Figure 610. Timer Control Register (TCTRL) . . . 1039
Figure 611. Timer Flag Register (TFLG) . . . 1040
Figure 612. Stopping and starting a timer . . . 1041
Figure 613. Modifying running timer period . . . 1041
Figure 614. Dynamically setting a new load value . . . 1041
Figure 615. Power management controller diagram . . . 1046
Figure 616. Bandgap reference block diagram . . . 1047
Figure 617. Module Configuration Register (PMC_MCR) . . . 1049
Figure 618. Trimming Register (PMC_TRIMR) . . . 1051
Figure 619. Status Register (PMC_SR) . . . 1054
Figure 620. Vreg 3.3 V power connection . . . 1058
Figure 621. Non-Volatile User Options Register (NVUSRO) - Array0 . . . 1059
Figure 622. POR rising and falling edges . . . 1061
Figure 623. POR - LVI relative rising and falling edges . . . 1061
Figure 624. JTAG STL (IEEE 1149.1) block diagram . . . 1066
Figure 625. 5-bit Instruction Register . . . 1069
Figure 626. Device Identification Register . . . 1070
Figure 627. CENSOR_CTRL Register . . . 1071
Figure 628. Shifting data through a register . . . 1072
Figure 629. IEEE 1149.1-2001 TAP controller finite state machine . . . 1073
Figure 630. Nexus Port Controller block diagram . . . 1079
Figure 631. 4-bit Instruction Register . . . 1083
Figure 632. Nexus Device ID Register (DID) . . . 1084
Figure 633. Port Configuration Register (PCR) . . . 1085
Figure 634. MSEO Transfers (for 2-bit MSEO) . . . 1088
Figure 635. Message field sizes . . . 1089
Figure 636. Transmission sequence of messages . . . 1089
Figure 637. Shifting data into register . . . 1090
Figure 638. IEEE 1149.1-2001 TAP controller state machine . . . 1091
Figure 639. NEXUS controller state machine . . . 1092

Figure 640. IEEE 1149.1 controller command input 1093
Figure 641. Calibration points 1097
Figure 642. Temperature formula 1098
Figure 643. Temperature Calculation Constants Register 0 1099
Figure 644. Temperature Calculation Constants Register 1 1100

1 Introduction

1.1 The SPC563Mxx Microcontroller Family

The SPC563Mxx is a family of system-on-chip devices that are built on Power Architecture technology and:

- Are 100 % user-mode compatible with the classic Power Architecture instruction set
- Contain enhancements that improve the architecture's fit in embedded applications
- Include additional instruction support for Digital Signal Processing (DSP)
- Integrate technologies, such as an enhanced time processor unit, enhanced queued analog-to-digital converter, Controller Area Network, and an enhanced modular input-output system, that are important for today's lower-end powertrain applications

1.2 SPC563Mxx Device Summary

Table 1 summarizes the SPC563Mxx family of microcontrollers.

Table 1. SPC563Mxx family device summary

Feature	SPC563M64	SPC563M60
Flash memory size (KB)	1536	1024
Total SRAM size (KB)	94	64
Standby SRAM size (KB)	32	32
Processor core	32-bit e200z335 with SPE and FPU support	32-bit e200z335 with SPE and FPU support
Core frequency (MHz)	64/80	40/64/80
Calibration bus width(1)	16 bits	16 bits
DMA (direct memory access) channels	32	32
eMIOS (enhanced modular input-output system) channels	16	16
eQADC (enhanced queued analog-to-digital converter) channels (on-chip)	Up to 34(2)	Up to 34(2)
eSCI (serial communication interface)	2	2
DSPI (deserial serial peripheral interface)	2	2
Microsecond Channel compatible interface	2	2
eTPU (enhanced time processor unit)	Yes	Yes
Channels	32	32
Code memory (KB)	14	14
Parameter RAM (KB)	3	3
FlexCAN (controller area network)(3)	2	2
FMPLL (frequency-modulated phase-locked loop)	Yes	Yes
INTC (interrupt controller) channels	364(4)	364(4)

Table 1. SPC563Mxx family device summary (continued)

Feature	SPC563M64	SPC563M60
JTAG controller	Yes	Yes
NDI (Nexus development interface) level	Class 2+	Class 2+
Non-maskable interrupt and critical interrupt	Yes	Yes
PIT (periodic interrupt timers)	5	5
Task monitor timer	4 channels	4 channels
Temperature sensor	Yes	Yes
Windowing software watchdog	Yes	Yes
Packages	LQFP144 LQFP176	LQFP144 LQFP176

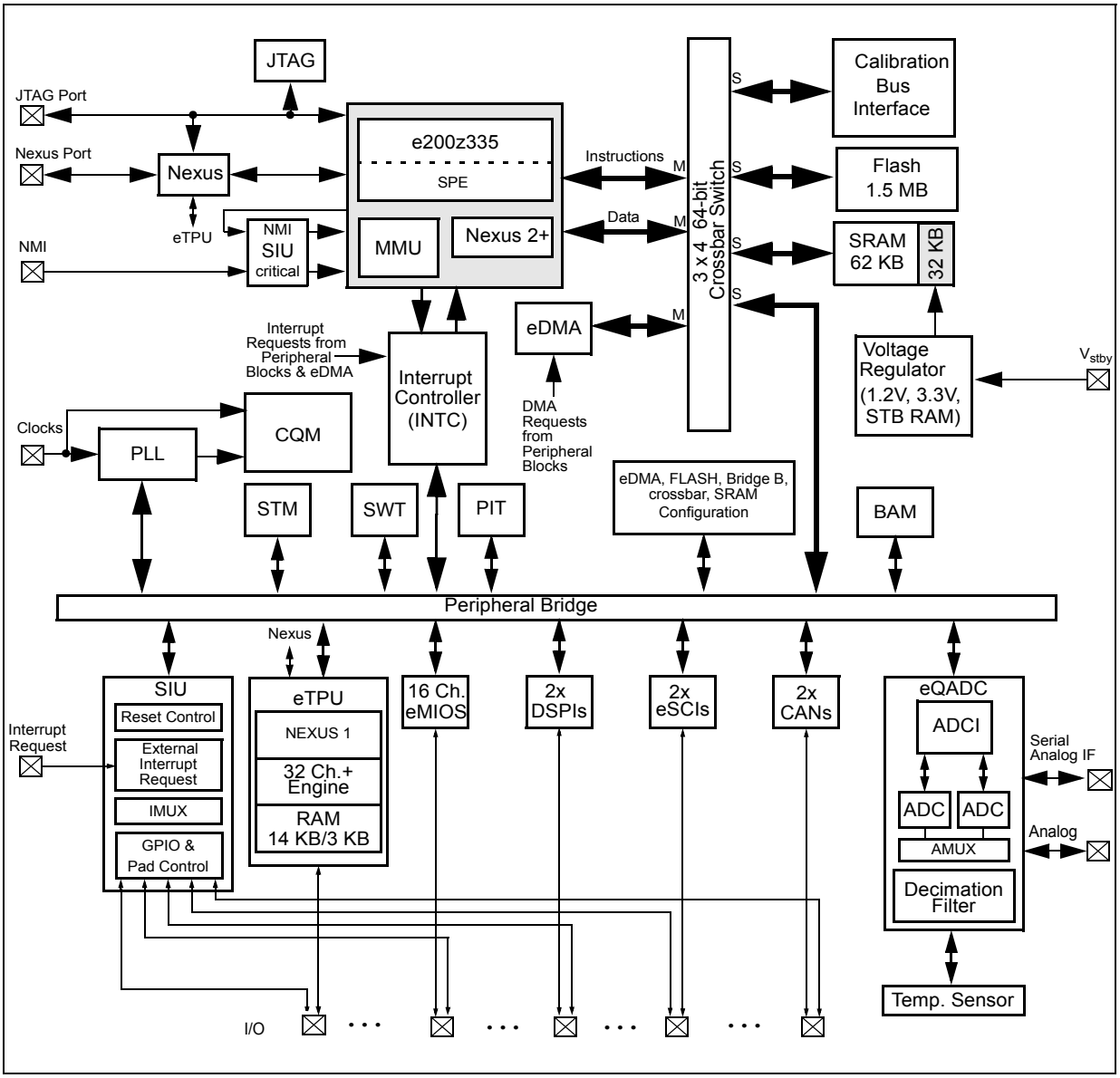
1. Calibration package only.
2. The 176-pin package has 34 input channels; 144-pin package has 32 input channels; 100-pin package has 23 input channels.
3. One FlexCAN module has 64 message buffers; the other has 32 message buffers.
4. 165 interrupt channels are reserved for compatibility with future devices. This device has 191 peripheral interrupt sources plus 8 software interrupts available to the user.

1.3 SPC563Mxx Blocks

1.3.1 Block Diagram

Figure 1 shows a top-level block diagram of the SPC563Mxx family.

Figure 1. SPC563Mxx Block Diagram (SPC563M64)



1.3.2 Block Summary

Table 2 summarizes the functions of the blocks present on the SPC563Mxx family.

Table 2. SPC563Mxx Block Summary

Block	Function
e200z335 core	Executes programs and interrupt handlers
Flash memory	Provides storage for program code, constants, and variables
RAM (random-access memory)	Provides storage for program code, constants, and variables
Calibration bus	Transfers data across the crossbar switch to/from peripherals attached to the calibration tool connector

Table 2. SPC563Mxx Block Summary (continued)

Block	Function
DMA (direct memory access)	Performs complex data movements with minimal intervention from the core
DSPI (deserial serial peripheral interface)	Provides a synchronous serial interface for communication with external devices
eMIOS (enhanced modular input-output system)	Provides the functionality to generate or measure events
eQADC (enhanced queued analog-to-digital converter)	Provides accurate and fast conversions for a wide range of applications
eSCI (serial communication interface)	Allows asynchronous serial communications with peripheral devices and other microcontroller units
eTPU (enhanced time processor unit) channels	Processes real-time input events, performs output waveform generation, and accesses shared data without host intervention
FlexCAN (controller area network)	Supports the standard CAN communications protocol
FMPLL (frequency-modulated phase-locked loop)	Generates high-speed system clocks and supports the programmable frequency modulation of these clocks
INTC (interrupt controller)	Provides priority-based preemptive scheduling of interrupt requests
JTAG controller	Provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode
NPC (Nexus port controller)	Provides real-time development support capabilities in compliance with the IEEE-ISTO 5001-2003 standard
PIT (periodic interrupt timer)	Produces periodic interrupts and triggers
SWT (software watchdog timer)	Provides protection from runaway code
STM (system timer module)	Timer providing a set of output compare events to support AUTOSAR (Automotive Open System Architecture) and operating system tasks
Temperature sensor	Provides the temperature of the device as an analog value

1.4 SPC563Mxx Features

1.4.1 Feature List

- Operating Parameters
  - Fully static operation, 0 MHz – 80 MHz (plus 2 % frequency modulation - 82 MHz)
  - –40 °C to 150 °C junction temperature operating range
  - Low power design
    - Less than 400 mW power dissipation (nominal)
    - Designed for dynamic power management of core and peripherals
    - Software controlled clock gating of peripherals
    - Low power stop mode, with all clocks stopped
  - Fabricated in 90 nm process
  - 1.2 V internal logic

  - Single power supply with 5.0 V $-10\ \% / +5\ \%$ (4.5 V to 5.25 V) with internal regulator to provide 3.3 V and 1.2 V for the core
  - Input and output pins with 5.0 V $-10\% / +5\%$ (4.5 V to 5.25 V) range
  - 35 %/65 % V_{DDE} CMOS switch levels (with hysteresis)
    - Selectable hysteresis
    - Selectable slew rate control
  - Nexus pins powered by 3.3 V supply
  - Designed with EMI reduction techniques
    - Phase-locked loop
    - Frequency modulation of system clock frequency
    - On-chip bypass capacitance
    - Selectable slew rate and drive strength
- High performance e200z335 core processor
  - 32-bit *Power Architecture Book E* programmer's model
  - Variable Length Encoding Enhancements
    - Allows Power Architecture instruction set to be optionally encoded in a mixed 16 and 32-bit instructions
    - Results in smaller code size
  - Single issue, 32-bit *Power Architecture technology* compliant CPU
  - In-order execution and retirement
  - Precise exception handling
  - Branch processing unit
    - Dedicated branch address calculation adder
    - Branch acceleration using Branch Lookahead Instruction Buffer
  - Load/store unit
    - One-cycle load latency
    - Fully pipelined
    - Big and Little Endian support
    - Misaligned access support
    - Zero load-to-use pipeline bubbles
  - Thirty-two 64-bit general purpose registers (GPRs)
  - Memory management unit (MMU) with 16-entry fully-associative translation look-aside buffer (TLB)
  - Separate instruction bus and load/store bus
  - Vectored interrupt support
  - Interrupt latency < 120 ns @ 80 MHz (measured from interrupt request to execution of first instruction of interrupt exception handler)
  - Non-maskable interrupt (NMI) input for handling external events that must produce an immediate response, e.g., power down detection. On this device, the NMI input is connected to the Critical Interrupt Input. (May not be recoverable)
  - Critical Interrupt input. For external interrupt sources that are higher priority than provided by the Interrupt Controller. (Always recoverable)
  - New 'Wait for Interrupt' instruction, to be used with new low power modes
  - Reservation instructions for implementing read-modify-write accesses
  - Signal processing extension (SPE) APU

    - Operating on all 32 GPRs that are all extended to 64 bits wide
    - Provides a full compliment of vector and scalar integer and floating point arithmetic operations (including integer vector MAC and MUL operations) (SIMD)
    - Provides rich array of extended 64-bit loads and stores to/from extended GPRs
    - Fully code compatible with e200z6 core
  - Floating point (FPU)
    - IEEE 754 compatible with software wrapper
    - Scalar single precision in hardware, double precision with software library
    - Conversion instructions between single precision floating point and fixed point
    - Fully code compatible with e200z6 core
  - Long cycle time instructions, except for guarded loads, do not increase interrupt latency
  - Extensive system development support through Nexus debug port
- Advanced microcontroller bus architecture (AMBA) crossbar switch (XBAR)
  - Three master ports, four slave ports
  - Masters: CPU Instruction bus; CPU Load/store bus (Nexus); eDMA
  - Slave: Flash; SRAM; Peripheral Bridge; calibration EBI
  - 32-bit internal address bus, 64-bit internal data bus
- Enhanced direct memory access (eDMA) controller
  - 32 channels support independent 8-bit, 16-bit, or 32-bit single value or block transfers
  - Supports variable sized queues and circular queues
  - Source and destination address registers are independently configured to post-increment or remain constant
  - Each transfer is initiated by a peripheral, CPU, or eDMA channel request
  - Each eDMA channel can optionally send an interrupt request to the CPU on completion of a single value or block transfer
- Interrupt controller (INTC)
  - 191 peripheral interrupt request sources
  - 8 software setable interrupt request sources
  - 9-bit vector
    - Unique vector for each interrupt request source
    - Provided by 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

- Frequency Modulating Phase-locked loop (FMPLL)
  - Reference clock pre-divider (PREDIV) for finer frequency synthesis resolution
  - Reduced frequency divider (RFD) for reducing the FMPLL output clock frequency without forcing the FMPLL to re-lock
  - System clock divider (SYSDIV) for reducing the system clock frequency in normal or bypass mode
  - Input clock frequency range from 4 MHz to 20 MHz before the pre-divider, and from 4 MHz to 16 MHz at the FMPLL input
  - Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz
  - VCO free-running frequency range from 25 MHz to 125 MHz
  - Four bypass modes: crystal or external reference with PLL on or off
  - Two normal modes: crystal or external reference
  - Programmable frequency modulation
    - Triangle wave modulation
    - Register programmable modulation frequency and depth
  - Lock detect circuitry reports when the FMPLL has achieved frequency lock and continuously monitors lock status to report loss of lock conditions
    - User-selectable ability to generate an interrupt request upon loss of lock
    - User-selectable ability to generate a system reset upon loss of lock
  - Clock quality monitor (CQM) module provides loss-of-clock detection for the FMPLL reference and output clocks
    - User-selectable ability to generate an interrupt request upon loss of clock
    - User-selectable ability to generate a system reset upon loss of clock
    - Backup clock (reference clock or FMPLL free-running) can be applied to the system in case of loss of clock
- Calibration bus interface (EBI)
  - Available only in the calibration package (496 CSP package)
  - 1.8 V to 3.3 V ± 10% I/O (1.6 V to 3.6 V)
  - Memory controller with support for various memory types
  - 16-bit data bus, up to 22-bit address bus
  - Selectable drive strength
  - Configurable bus speed modes
  - Bus monitor
  - Configurable wait states
- System integration unit (SIU)
  - Centralized GPIO control of 80 I/O pins
  - Centralized pad control on a per-pin basis
    - Pin function selection
    - Configurable weak pull-up or pull-down
    - Drive strength
    - Slew rate
    - Hysteresis
  - System reset monitoring and generation

  - External interrupt inputs, filtering and control
  - Critical Interrupt control
  - Non-Maskable Interrupt control
  - Internal multiplexer subblock (IMUX)
    - Allows flexible selection of eQADC trigger inputs (eTPU, eMIOS and external signals)
    - Allows selection of interrupt requests between external pins and DSPI
- Error correction status module (ECSM)
  - Configurable error-correcting codes (ECC) reporting
  - Single-bit error correction reporting
- On-chip flash memory
  - Up to 1.5 MB flash memory, accessed via a 64-bit wide bus interface
  - 16 KB shadow block
  - Fetch Accelerator
    - Provide single cycle flash access at 80 MHz
    - Quadruple 128-bit wide prefetch/burst buffers
    - Prefetch buffers can be configured to prefetch code or data or both
  - Censorship protection scheme to prevent flash content visibility
  - Flash divided into two independent arrays, allowing reading from one array while erasing/programming the other array (used for EEPROM emulation)
  - Memory block:
    - For SPC563M64: 18 blocks (4 × 16 KB, 2 × 32 KB, 2 × 64 KB, 10 × 128 KB)
    - For SPC563M60: 14 blocks (4 × 16 KB, 2 × 32 KB, 2 × 64 KB, 6 × 128 KB)
  - Hardware programming state machine
- On-chip static RAM
  - For SPC563M64: 94 KB general purpose RAM of which 32 KB are on standby power supply
  - For SPC563M60: 64 KB general purpose RAM of which 32 KB are on standby power supply
- Boot assist module (BAM)
  - Enables and manages the transition of MCU from reset to user code execution in the following configurations:
    - Execution from internal flash memory
    - Execution from external memory on the calibration bus
    - Download and execution of code via FlexCAN or eSCI
- Periodic interrupt timer (PIT)
  - 32-bit wide down counter with automatic reload
  - Four channels clocked by system clock
  - One channel clocked by crystal clock
  - Each channel can produce periodic software interrupt
  - Each channel can produce periodic triggers for eQADC queue triggering
  - One channel out of the five can be used as wake-up timer to wake device from low power stop mode

- System timer module (STM)
  - 32-bit up counter with 8-bit prescaler
  - Clocked from system clock
  - Four-channel timer compare hardware
  - Each channel can generate a unique interrupt request
  - Designed to address AUTOSAR task monitor function
- Software watchdog timer (SWT)
  - 32-bit timer
  - Clock by system clock or crystal clock
  - Can generate either system reset or non-maskable interrupt followed by system reset
  - Enabled out of reset
- Enhanced modular I/O system (eMIOS)
  - 16 timer channels (up to 14 channels in LQFP144)
  - 24-bit timer resolution
  - 3 selectable time bases plus shared time or angle counter bus from eTPU2
  - DMA and interrupt request support
  - Motor control capability
- Second-generation enhanced time processor unit (eTPU2)
  - Object-code compatible with eTPU—no changes are required to hardware or software if only eTPU features are used
  - Intelligent co-processor designed for timing control
  - High level tools, assembler and compiler available
  - 32 channels (each channel has dedicated I/O pin in all packages)
  - 24-bit timer resolution
  - 14 KB code memory and 3 KB data memory
  - Double match and capture on all channels
  - Angle clock hardware support
  - Shared time or angle counter bus with eMIOS
  - DMA and interrupt request support
  - Nexus Class 1 debug support
  - eTPU2 enhancements
    - Counters and channels can run at full system clock speed
    - Software watchdog
    - Real-time performance monitor
    - Instruction set enhancements for smaller more flexible code generation
    - Programmable channel mode for customization of channel operation
- Enhanced queued A/D converter (eQADC)
  - Two independent on-chip redundant signed digit (RSD) cyclic ADCs
  - 8-, 10-, and 12-bit resolution
    - Differential conversions

  - Targets up to 10-bit accuracy at 500 KSample/s (ADC_CLK = 7.5 MHz) and 8-bit accuracy at 1 MSample/s (ADC_CLK = 15 MHz) for differential conversions
  - Differential channels include variable gain amplifier (VGA) for improved dynamic range (×1; ×2; ×4)
  - Differential channels include programmable pull-up and pull-down resistors for biasing and sensor diagnostics (200 kΩ; 100 kΩ; low value of 5 kΩ)
  - Single-ended signal range from 0 to 5 V
  - Sample times of 2 (default), 8, 64 or 128 ADC clock cycles
  - Provides time stamp information when requested
  - Parallel interface to eQADC command FIFOs (CFIFOs) and result FIFOs (RFIFOs)
  - Supports both right-justified unsigned and signed formats for conversion results
  - Temperature sensor to enable measurement of die temperature
  - Ability to measure all power supply pins directly
- Automatic application of ADC calibration constants
  - Provision of reference voltages (25% VREF and 75% VREF) for ADC calibration purposes
- Up to 34[(a)] input channels available to the two on-chip ADCs
- Four pairs of differential analog input channels
- Full duplex synchronous serial interface to an external device
  - Has a free-running clock for use by the external device
  - Supports a 26-bit message length
  - Transmits a null message when there are no triggered CFIFOs with commands bound for external CBuffers, or when there are triggered CFIFOs with commands bound for external CBuffers but the external CBuffers are full
- Parallel Side Interface to communicate with an on-chip companion module
- Zero jitter triggering for queue 0. (Queue 0 trigger causes current conversion to be aborted and the queued conversions in the CBUFFER to be bypassed. Delay from Trigger to start of conversion is 13 system clocks + 1 ADC clock.)
- eQADC Result Streaming. Generation of a continuous stream of ADC conversion results from a single eQADC command word. Controlled by two different trigger signals; one to define the rate at which results are generated and the other to define the beginning and ending of the stream. Used to digitize waveforms during specific time/angle windows, e.g., engine knock sensor sampling.
- Angular Decimation. The ability of the eQADC to sample an analog waveform in the time domain, perform Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filtering also in the time domain, but to down sample the results in the angle domain. Resulting in a time domain filtered result at a given engine angle.
- Priority Based CFIFOs

a. 176-pin package has 34 input channels; 144-pin package has 32 input channels; 100-pin package has 23 input channels.

  - Supports six CFIFOs with fixed priority. The lower the CFIFO number, the higher its priority. When commands of distinct CFIFOs are bound for the same CBuffer, the higher priority CFIFO is always served first.
  - Supports software and several hardware trigger modes to arm a particular CFIFO
  - Generates interrupt when command coherency is not achieved
- External Hardware Triggers
  - Supports rising edge, falling edge, high level and low level triggers
  - Supports configurable digital filter
- Supports four external 8-to-1 muxes which can expand the input channel number from 34[b] to 59

- Two Deserial Serial Peripheral Interface modules (DSPI)
  - SPI
    - Full duplex communication ports with interrupt and DMA request support
    - Support for queues in RAM
    - 6 chip selects, expandable to 64 with external demultiplexers
    - Programmable frame size, baud rate, clock delay and clock phase on a per frame basis
    - Modified SPI mode for interfacing to peripherals with longer setup time requirements
    - LVDS option for output clock and data to allow higher speed communication
  - Deserial serial interface (DSI)
    - Pin reduction by hardware serialization and deserialization of eTPU, eMIOS channels and GPIO
    - 32 bits per DSPI module
    - Triggered transfer control and change in data transfer control (for reduced EMI)
    - Compatible with Microsecond Channel Version 1.0 downstream
- Two enhanced serial communication interface (eSCI) modules
  - UART mode provides NRZ format and half or full duplex interface
  - eSCI bit rate up to 1 Mbps
  - Advanced error detection, and optional parity generation and detection
  - Word length programmable as 8, 9, 12 or 13 bits
  - Separately enabled transmitter and receiver
  - LIN support
  - DMA support
  - Interrupt request support
  - Programmable clock source: system clock or oscillator clock
  - Support Microsecond Channel (Timed Serial Bus - TSB) upstream Version 1.0

b. 176-pin package.

- Two FlexCAN
  - One with 32 message buffers; the second with 64 message buffers
  - Full implementation of the CAN protocol specification, Version 2.0B
  - Programmable acceptance filters
  - Short latency time for high priority transmit messages
  - Arbitration scheme according to message ID or message buffer number
  - Listen only mode capabilities
  - Programmable clock source: system clock or oscillator clock
  - Message buffers may be configured as mailboxes or as FIFO
- Nexus port controller (NPC)
  - Per IEEE-ISTO 5001-2003
  - Real time development support for Power Architecture core and eTPU engine through Nexus class 2/1
  - Read and write access (Nexus class 3 feature that is supported on this device)
    - Run-time access of entire memory map
    - Calibration
  - Support for data value breakpoints / watchpoints
    - Run-time access of entire memory map
    - Calibration
    - Table constants calibrated using MMU and internal and external RAM
    - Scalar constants calibrated using cache line locking
  - Configured via the IEEE 1149.1 (JTAG) port
- IEEE 1149.1 JTAG controller (JTAGC)
  - IEEE 1149.1-2001 Test Access Port (TAP) interface
  - 5-bit instruction register that supports IEEE 1149.1-2001 defined instructions
  - 5-bit instruction register that supports additional public instructions
  - Three test data registers: a bypass register, a boundary scan register, and a device identification register
  - Censorship disable register. By writing the 64-bit serial boot password to this register, Censorship may be disabled until the next reset
  - TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry
- On-chip Voltage Regulator for single 5 V supply operation
  - On-chip regulator 5 V to 3.3 V for internal supplies
  - On-chip regulator controller 5 V to 1.2 V (with external bypass transistor) for core logic
- Low-power modes
  - SLOW Mode. Allows device to be run at very low speed (approximately 1 MHz), with modules (including the PLL) selectively disabled in software
  - STOP Mode. System clock stopped to all modules including the CPU. Wake-up timer used to restart the system clock after a predetermined time

1.5 SPC563Mxx feature details

1.5.1 e200z335 core

The e200z335 processor utilizes a four stage pipeline for instruction execution. The Instruction Fetch (stage 1), Instruction Decode/Register file Read/Effective Address Calculation (stage 2), Execute/Memory Access (stage 3), and Register Writeback (stage 4) stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions.

The integer execution unit consists of a 32-bit Arithmetic Unit (AU), a Logic Unit (LU), a 32-bit Barrel shifter (Shifter), a Mask-Insertion Unit (MIU), a Condition Register manipulation Unit (CRU), a Count-Leading-Zeros unit (CLZ), a 32×32 Hardware Multiplier array, result feed-forward hardware, and support hardware for division.

Most arithmetic and logical operations are executed in a single cycle with the exception of the divide instructions. A Count-Leading-Zeros unit operates in a single clock cycle. The Instruction Unit contains a PC incrementer and a dedicated Branch Address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching is performed to accelerate taken branches. Prefetched instructions are placed into an instruction buffer capable of holding six instructions.

Branches can also be decoded at the instruction buffer and branch target addresses calculated prior to the branch reaching the instruction decode stage, allowing the branch target to be prefetched early. When a branch is detected at the instruction buffer, a prediction may be made on whether the branch is taken or not. If the branch is predicted to be taken, a target fetch is initiated and its target instructions are placed in the instruction buffer following the branch instruction. Many branches take zero cycle to execute by using branch folding. Branches are folded out from the instruction execution pipe whenever possible. These include unconditional branches and conditional branches with condition codes that can be resolved early.

Conditional branches which are not taken and not folded execute in a single clock. Branches with successful target prefetching which are not folded have an effective execution time of one clock. All other taken branches have an execution time of two clocks. Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The Condition Register unit supports the condition register (CR) and condition register operations defined by the Power Architecture. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching. Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

The hardware floating-point unit utilizes the IEEE-754 single-precision floating-point format and supports single-precision floating-point operations in a pipelined fashion. The general purpose register file is used for source and destination operands, thus there is a unified

storage model for single-precision floating-point data types of 32 bits and the normal integer type. Single-cycle floating-point add, subtract, multiply, compare, and conversion operations are provided. Divide instructions are multi-cycle and are not pipelined.

The Signal Processing Extension (SPE) Auxiliary Processing Unit (APU) provides hardware SIMD operations and supports a full complement of dual integer arithmetic operation including Multiply Accumulate (MAC) and dual integer multiply (MUL) in a pipelined fashion. The general purpose register file is enhanced such that all 32 of the GPRs are extended to 64 bits wide and are used for source and destination operands, thus there is a unified storage model for 32×32 MAC operations which generate greater than 32-bit results.

The majority of both scalar and vector operations (including MAC and MUL) are executed in a single clock cycle. Both scalar and vector divides take multiple clocks. The SPE APU also provides extended load and store operations to support the transfer of data to and from the extended 64-bit GPRs.

The CPU includes support for Variable Length Encoding (VLE) instruction enhancements. This enables the classic Power Architecture instruction set to be represented by a modified instruction set made up from a mixture of 16- and 32-bit instructions. This results in a significantly smaller code size footprint without noticeably affecting performance. The Power Architecture instruction set and VLE instruction set are available concurrently. Regions of the memory map are designated as PPC or VLE using an additional configuration bit in each of Table Look-aside Buffers (TLB) entries in the MMU.

The CPU core is enhanced by the addition of two additional interrupt sources; Non-Maskable Interrupt and Critical Interrupt. These two sources are routed directly from package pins, via edge detection logic in the SIU to the CPU, bypassing completely the Interrupt Controller. Once the edge detection logic is programmed, it cannot be disabled, except by reset. The non-maskable Interrupt is, as the name suggests, completely un-maskable and when asserted will always result in the immediate execution of the respective interrupt service routine. The non-maskable interrupt is not guaranteed to be recoverable. The Critical Interrupt is very similar to the non-maskable interrupt, but it can be masked by other exceptional interrupts in the CPU and is guaranteed to be recoverable (code execution may be resumed from where it stopped).

The CPU core has an additional ‘Wait for Interrupt’ instruction that is used in conjunction with low power STOP mode. When Low Power Stop mode is selected, this instruction is executed to allow the system clock to be stopped. An external interrupt source or the system wake-up timer is used to restart the system clock and allow the CPU to service the interrupt.

1.5.2 Crossbar

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

The crossbar allows three concurrent transactions to occur from the master ports to any slave port; but each master must access a different slave. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions. Requesting masters are treated with equal priority and are granted access to a slave port in round-robin fashion, based upon the ID of the last master to be granted access.

The crossbar provides the following features:

- 3 master ports:
  - e200z335 core complex Instruction port
  - e200z335 core complex Load/Store port
  - eDMA
- 4 slave ports
  - FLASH
  - calibration bus
  - SRAM
  - Peripheral bridge A/B (eTPU2, eMIOS, SIU, DSPI, eSCI, FlexCAN, eQADC, BAM, decimation filter, PIT, STM and SWT)
- 32-bit internal address, 64-bit internal data paths

1.5.3 eDMA

The enhanced direct memory access (eDMA) controller is a second-generation module capable of performing complex data movements via 32 programmable channels, with minimal intervention from the host processor. The hardware micro architecture includes a DMA engine which performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels. This implementation is utilized to minimize the overall block size. The eDMA module provides the following features:

- All data movement via dual-address transfers: read from source, write to destination
- Programmable source and 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 activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
  - Peripheral-paced hardware requests (one per channel)
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
- 1 interrupt per channel, optionally asserted at completion of major iteration count
- Error termination interrupts are optionally enabled
- Support for scatter/gather DMA processing
- Channel transfers can be suspended by a higher priority channel

1.5.4 Interrupt controller

The INTC (interrupt controller) provides priority-based preemptive scheduling of interrupt requests, suitable for statically scheduled hard real-time systems. The INTC allows interrupt request servicing from up to 191 peripheral interrupt request sources, plus 165 sources reserved for compatibility with other family members).

For high priority interrupt requests, the time from the assertion of the interrupt request from the peripheral to when the processor is executing the interrupt service routine (ISR) has

been minimized. The INTC provides a unique vector for each interrupt request source for quick determination of which ISR needs to be executed. It also provides an ample number of priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. To allow the appropriate priorities for each source of interrupt request, the priority of each interrupt request is software configurable.

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

Multiple processors can assert interrupt requests to each other through software setable interrupt requests. These same software setable 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 asserts a software setable interrupt request to finish the servicing in a lower priority ISR. Therefore these software setable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

The INTC provides the following features:

- 356 peripheral interrupt request sources
- 8 software setable interrupt request sources
- 9-bit vector addresses
- 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
- Preemptive prioritized interrupt requests to processor
- ISR at a higher priority preempts executing 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 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

This device also includes a non-maskable interrupt (NMI) pin that bypasses the INTC and multiplexing logic.

1.5.5 FMPLL

The FMPLL allows the user to generate high speed system clocks from a 4 MHz to 20 MHz crystal oscillator or external clock generator. Further, the FMPLL supports programmable frequency modulation of the system clock. The PLL multiplication factor, output clock divider ratio are all software configurable. The PLL has the following major features:

- Input clock frequency from 4 MHz to 20 MHz
- Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz, resulting in system clock frequencies from 16 MHz to 80 MHz with granularity of 4 MHz or better
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the PLL to relock
- 3 modes of operation
  - Bypass mode with PLL off

  - Bypass mode with PLL running (default mode out of reset)
  - PLL normal mode
- Each of the three modes may be run with a crystal oscillator or an external clock reference
- Programmable frequency modulation
  - Modulation enabled/disabled through software
  - Triangle wave modulation up to 100 kHz modulation frequency
  - Programmable modulation depth (0% to 2% modulation depth)
  - Programmable modulation frequency dependent on reference frequency
- Lock detect circuitry reports when the PLL has achieved frequency lock and continuously monitors lock status to report loss of lock conditions
- Clock Quality Module
  - detects the quality of the crystal clock and cause interrupt request or system reset if error is detected
  - detects the quality of the PLL output clock. If an error is detected, causes a system reset or switches the system clock to the crystal clock and causes an interrupt request
- Programmable interrupt request or system reset on loss of lock

1.5.6 Calibration EBI

The Calibration EBI controls data transfer across the crossbar switch to/from memories or peripherals attached to the calibration tool connector in the calibration address space. The Calibration EBI is only available in the calibration tool. The Calibration EBI includes a memory controller that generates interface signals to support a variety of external memories. The Calibration EBI memory controller supports legacy flash, SRAM, and asynchronous memories. In addition, the calibration EBI supports up to three regions via chip selects (two chip selects are multiplexed with two address bits), along with programmed region-specific attributes. The calibration EBI supports the following features:

- 22-bit address bus (two most significant signals multiplexed with two chip selects)
- 16-bit data bus
- Multiplexed mode with addresses and data signals present on the data lines

Note: The calibration EBI must be configured in multiplexed mode when the extended Nexus trace is used on the calibration tool. This is because Nexus signals and address lines of the calibration bus share the same balls in the calibration package.

- Memory controller with support for various memory types:
  - Asynchronous/legacy flash and SRAM
- Bus monitor
  - User selectable
  - Programmable timeout period (with 8 external bus clock resolution)
- Configurable wait states (via chip selects)
- 3 chip-select (Cal_$\overline{CS}$[0], Cal_$\overline{CS}$[2:3]) signals (Multiplexed with 2 most significant address signals)
- 2 write/byte enable (WE[0:1]/BE[0:1]) signals

- Configurable bus speed modes
  - system frequency
  - 1/2 of system frequency
  - 1/4 of system frequency
- Optional automatic CLKOUT gating to save power and reduce EMI
- Selectable drive strengths; 10 pF, 20 pF, 30 pF, 50 pF

1.5.7 SIU

The SPC563Mxx SIU controls MCU reset configuration, pad configuration, external interrupt, general purpose I/O (GPIO), internal peripheral multiplexing, and the 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 reset controller performs reset monitoring of internal and external reset sources, and drives the RSTOUT pin. Communication between the SIU and the e200z335 CPU core is via the crossbar switch. The SIU provides the following features:

- System configuration
  - MCU reset configuration via external pins
  - Pad configuration control for each pad
  - Pad configuration control for virtual I/O via DSPI serialization
- System reset monitoring and generation
  - Power-on reset support
  - Reset status register provides last reset source to software
  - Glitch detection on reset input
  - Software controlled reset assertion
- External interrupt
  - 11 interrupt requests
  - Rising or falling edge event detection
  - Programmable digital filter for glitch rejection
  - Critical Interrupt request
  - Non-Maskable Interrupt request
- GPIO
  - GPIO function on 80 I/O pins
  - Virtual GPIO on 64 I/O pins via DSPI serialization (requires external deserialization device)
  - Dedicated input and output registers for setting each GPIO and Virtual GPIO pin
- Internal multiplexing
  - Allows serial and parallel chaining of DSPIs
  - Allows flexible selection of eQADC trigger inputs
  - Allows selection of interrupt requests between external pins and DSPI

1.5.8 ECSM

The error correction status module provides status information regarding platform memory errors reported by error-correcting codes.

1.5.9 Flash

Devices in the SPC563Mxx family provide up to 1.5 MB of programmable, non-volatile, flash memory. The non-volatile memory (NVM) can be used for instruction and/or data storage. The flash module includes a Fetch Accelerator, that optimizes the performance of the flash array to match the CPU architecture and provides single cycle random access to the flash @ 80 MHz. The flash module interfaces the system bus to a dedicated flash memory array controller. For CPU 'loads', DMA transfers and CPU instruction fetch, 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 a four-entry, 128-bit prefetch buffer and a prefetch controller which prefetches sequential lines of data from the flash array into the buffer. Prefetch buffer hits allow no-wait responses. Normal flash array accesses are registered and are forwarded to the system bus on the following cycle, incurring three wait-states. Prefetch operations may be automatically controlled, and are restricted to instruction fetch.

The flash memory provides the following features:

- Supports a 64-bit data bus for instruction fetch, CPU loads and DMA access. Byte, halfword, word and doubleword reads are supported. Only aligned word and doubleword writes are supported.
- Fetch Accelerator
  - Architected to optimize the performance of the flash with the CPU to provide single cycle random access to the flash up to 80 MHz system clock speed
  - Configurable read buffering and line prefetch support
  - Four line read buffers (128 bits wide) and a prefetch controller
- 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 one, 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
- Erase of selected block(s)
- Read page size of 128 bits (four words)
- ECC with single-bit correction, double-bit detection
- Program page size of 64 bits (two words)
- ECC single-bit error corrections are visible to software
- Minimum program size is two consecutive 32-bit words, aligned on a 0-modulo-8 byte address, due to ECC
- Embedded hardware program and erase algorithm
- Erase suspend

- Shadow information stored in non-volatile shadow block
- Independent program/erase of the shadow block

1.5.10 SRAM

The SPC563Mxx SRAM module provides a general-purpose up to 94 KB memory block. The SRAM controller includes these features:

- Supports read/write accesses mapped to the SRAM memory from any master
- 32 KB or 24 KB block powered by separate supply for standby operation
- Byte, halfword, word and doubleword addressable
- ECC performs single-bit correction, double-bit detection on 32-bit data element

1.5.11 BAM

The BAM (Boot Assist Module) is a block of read-only memory that is programmed once by ST and is identical for all SPC563Mxx MCUs. The BAM program is executed every time the MCU is powered-on or reset in normal mode. The BAM supports different modes of booting. They are:

- Booting from internal flash memory
- Serial boot loading (A program is downloaded into RAM via eSCI or the FlexCAN and then executed)
- Booting from external memory on calibration bus

The BAM also reads the reset configuration half word (RCHW) from internal flash memory and configures the SPC563Mxx hardware accordingly. The BAM provides the following features:

- Sets up MMU to cover all resources and mapping all physical address to logical addresses with minimum address translation
- Sets up the MMU to allow user boot code to execute as either Power Architecture code (default) or as VLE code
- Detection of user boot code
- Automatic switch to serial boot mode if internal flash is blank or invalid
- Supports user programmable 64-bit password protection for serial boot mode
- Supports serial bootloading via FlexCAN bus and eSCI using fixed baudrate protocol
- Supports serial bootloading via FlexCAN bus and eSCI with auto baud rate sensing
- Supports serial bootloading of either Power Architecture code (default) or VLE code
- Supports booting from calibration bus interface
- Supports censorship protection for internal flash memory
- Provides an option to enable the core watchdog timer
- Provides an option to disable the software watchdog timer

1.5.12 eMIOS

The eMIOS (Enhanced Modular Input Output System) module provides the functionality to generate or measuretime events. The channels on this module provide a range of operating modes including the capability to perform dual input capture or dual output compare as well as PWM output.

The eMIOS provides the following features:

- 16 channels (24-bit timer resolution)
- For compatibility with other family members selected channels and timebases are implemented:
  - Channels 0 to 6, 8 to 15, and 23
  - Timebases A, B and C
- Channels 1, 3, 5 and 6 support modes:
  - General Purpose Input/Output (GPIO)
  - Single Action Input Capture (SAIC)
  - Single Action Output Compare (SAOC)
- Channels 2, 4, 11 and 13 support all the modes above plus:
  - Output Pulse Width Modulation Buffered (OPWMB)
- Channels 0, 8, 9, 10, 12, 14, 15, 23 support all the modes above plus:
  - Input Period Measurement (IPM)
  - Input Pulse Width Measurement (IPWM)
  - Double Action Output Compare (set flag on both matches) (DAOC)
  - Modulus Counter Buffered (MCB)
  - Output Pulse Width and Frequency Modulation Buffered (OPWFMB)
- Three 24-bit wide counter buses
  - Counter bus A can be driven by channel 23 or by the eTPU2 and all channels can use it as a reference
  - Counter bus B is driven by channel 0 and channels 0 to 6 can use it as a reference
  - Counter bus C is driven by channel 8 and channels 8 to 15 can use it as a reference
- Shared time bases with the eTPU2 through the counter buses
- Synchronization among internal and external time bases

1.5.13 eTPU2

The eTPU2 is an enhanced co-processor designed for timing control. Operating in parallel with the host CPU, eTPU2 processes instructions and real-time input events, performs output waveform generation, and accesses shared data without host intervention. Consequently, for each timer event, the host CPU setup and service times are minimized or eliminated. A powerful timer subsystem is formed by combining the eTPU2 with its own instruction and data RAM. High-level assembler/compiler and documentation allows customers to develop their own functions on the eTPU2.

The eTPU2 includes these distinctive features:

- The Timer Counter (TCR1), channel logic and digital filters (both channel and the external timer clock input [TCRCLK]) now have an option to run at full system clock speed or system clock / 2.
- Channels support unordered transitions: transition 2 can now be detected before transition 1. Related to this enhancement, the transition detection latches (TDL1 and TDL2) can now be independently negated by microcode.
- A new User Programmable Channel Mode has been added: the blocking, enabling, service request and capture characteristics of this channel mode can be programmed via microcode.

- Microinstructions now provide an option to issue Interrupt and Data Transfer requests selected by channel. They can also be requested simultaneously at the same instruction.
- Channel Flags 0 and 1 can now be tested for branching, in addition to selecting the entry point.
- Channel digital filters can be bypassed.
- The Timer Counter (TCR1), channel logic and digital filters (both channel and the external timer clock input [TCRCLK]) now have an option to run at full system clock speed or system clock / 2.
- Channels support unordered transitions: transition 2 can now be detected before transition 1. Related to this enhancement, the transition detection latches (TDL1 and TDL2) can now be independently negated by microcode.
- A new User Programmable Channel Mode has been added: the blocking, enabling, service request and capture characteristics of this channel mode can be programmed via microcode.
- Microinstructions now provide an option to issue Interrupt and Data Transfer requests selected by channel. They can also be requested simultaneously at the same instruction.
- Channel Flags 0 and 1 can now be tested for branching, in addition to selecting the entry point.
- Channel digital filters can be bypassed.
- 32 channels, each channel is associated with one input and one output signal
  - Enhanced input digital filters on the input pins for improved noise immunity.
  - Identical, orthogonal channels: each channel can perform any time function. Each time function can be assigned to more than one channel at a given time, so each signal can have any functionality.
  - Each channel has an event mechanism which supports single and double action functionality in various combinations. It includes two 24-bit capture registers, two 24-bit match registers, 24-bit greater-equal and equal-only comparators
  - Input and output signal states visible from the host
- 2 independent 24-bit time bases for channel synchronization:
  - First time base clocked by system clock with programmable prescale division from 2 to 512 (in steps of 2), or by output of second time base prescaler
  - Second time base counter can work as a continuous angle counter, enabling angle based applications to match angle instead of time
  - Both time bases can be exported to the eMIOS timer module
  - Both time bases visible from the host
- Event-triggered microengine:
  - Fixed-length instruction execution in two-system-clock microcycle
  - 14 KB of code memory (SCM)
  - 3 KB of parameter (data) RAM (SPRAM)
  - Parallel execution of data memory, ALU, channel control and flow control sub-instructions in selected combinations
  - 32-bit microengine registers and 24-bit wide ALU, with 1 microcycle addition and subtraction, absolute value, bitwise logical operations on 24-bit, 16-bit, or byte

operands, single-bit manipulation, shift operations, sign extension and conditional execution
  - Additional 24-bit Multiply/MAC/Divide unit which supports all signed/unsigned Multiply/MAC combinations, and unsigned 24-bit divide. The MAC/Divide unit works in parallel with the regular microcode commands
- Resource sharing features support channel use of common channel registers, memory and microengine time:
  - Hardware scheduler works as a “task management” unit, dispatching event service routines by predefined, host-configured priority
  - Automatic channel context switch when a “task switch” occurs, i.e., one function thread ends and another begins to service a request from other channel: channel-specific registers, flags and parameter base address are automatically loaded for the next serviced channel
  - SPRAM shared between host CPU and eTPU2, supporting communication either between channels and host or inter-channel
  - Dual-parameter coherency hardware support allows atomic access to two parameters by host
- Test and development support features:
  - Nexus Class 1 debug, supporting single-step execution, arbitrary microinstruction execution, hardware breakpoints and watchpoints on several conditions
  - Software breakpoints
  - SCM continuous signature-check built-in self test (MISC — multiple input signature calculator), runs concurrently with eTPU2 normal operation
- System enhancements
  - Software watchdog with programmable timeout
  - Real-time performance information
- Channel enhancements
  - Channels 1 and 2 can optionally drive angle clock hardware
- Programming enhancements
  - Engine relative addressing mode

1.5.14 eQADC

The enhanced queued analog to digital converter (eQADC) block provides accurate and fast conversions for a wide range of applications. The eQADC provides a parallel interface to two on-chip analog to digital converters (ADC), and a single master to single slave serial interface to an off-chip external device. Both on-chip ADCs have access to all the analog channels.

The eQADC prioritizes and transfers commands from six command conversion command ‘queues’ to the on-chip ADCs or to the external device. The block can also receive data from the on-chip ADCs or from an off-chip external device into the six result queues, in parallel, independently of the command queues. The six command queues are prioritized with Queue_0 having the highest priority and Queue_5 the lowest. Queue_0 also has the added ability to bypass all buffering and queuing and abort a currently running conversion on either ADC and start a Queue_0 conversion. This means that Queue_0 will always have a deterministic time from trigger to start of conversion, irrespective of what tasks the ADCs were performing when the trigger occurred. The eQADC supports software and external hardware triggers from other blocks to initiate transfers of commands from the queues to the

on-chip ADCs or to the external device. It also monitors the fullness of command queues and result queues, and accordingly generates DMA or interrupt requests to control data movement between the queues and the system memory, which is external to the eQADC.

The ADCs also support features designed to allow the direct connection of high impedance acoustic sensors that might be used in a system for detecting engine knock. These features include differential inputs; integrated variable gain amplifiers for increasing the dynamic range; programmable pull-up and pull-down resistors for biasing and sensor diagnostics.

The eQADC also integrates a programmable decimation filter capable of taking in ADC conversion results at a high rate, passing them through a hardware low pass filter, then down-sampling the output of the filter and feeding the lower sample rate results to the result FIFOs. This allows the ADCs to sample the sensor at a rate high enough to avoid aliasing of out-of-band noise; while providing a reduced sample rate output to minimize the amount DSP processing bandwidth required to fully process the digitized waveform.

The eQADC provides the following features:

- Dual on-chip ADCs
  - 2 × 12-bit ADC resolution
  - Programmable resolution for increased conversion speed (12 bit, 10 bit, 8 bit)
    - 12-bit conversion time – 1 µs (1M sample/sec)
    - 10-bit conversion time – 867 ns (1.2M sample/second)
    - 8-bit conversion time – 733 ns (1.4M sample/second)
  - Up to 10-bit accuracy at 500 KSample/s and 9-bit accuracy at 1 MSample/s
  - Differential conversions
  - Single-ended signal range from 0 to 5 V
  - Variable gain amplifiers on differential inputs (×1, ×2, ×4)
  - Sample times of 2 (default), 8, 64 or 128 ADC clock cycles
  - Provides time stamp information when requested
  - Parallel interface to eQADC CFIFOs and RFIFOs
  - Supports both right-justified unsigned and signed formats for conversion results
- Up to 34[c] input channels (accessible by both ADCs)
- 23 additional internal channels for measuring control and monitoring voltages inside the device
  - Including Core voltage, I/O voltage, LVI voltages, etc.
- An internal bandgap reference to allow absolute voltage measurements
- 4 pairs of differential analog input channels
  - Programmable pull-up/pull-down resistors on each differential input for biasing and sensor diagnostic (200 kΩ, 100 kΩ, 5 kΩ)
- Silicon die temperature sensor
  - Provides temperature of silicon as an analog value
  - Read using an internal ADC analog channel
  - May be read with either ADC

c. 176-pin package has 34 input channels; 144-pin package has 32 input channels.

- Decimation Filter
  - Programmable decimation factor (2 to 16)
  - Selectable IIR or FIR filter
  - Up to 4th order IIR or 8th order FIR
  - Programmable coefficients
  - Saturated or non-saturated modes
  - Programmable Rounding (Convergent; Two's Complement; Truncated)
  - Pre-fill mode to pre-condition the filter before the sample window opens
- Full duplex synchronous serial interface to an external device
  - Free-running clock for use by an external device
  - Supports a 26-bit message length
- Priority based Queues
  - Supports six Queues with fixed priority. When commands of distinct Queues are bound for the same ADC, the higher priority Queue is always served first
  - Queue_0 can bypass all prioritization, buffering and abort current conversions to start a Queue_0 conversion a deterministic time after the queue trigger
  - Streaming mode operation of Queue_0 to execute some commands several times
  - Supports software and hardware trigger modes to arm a particular Queue
  - Generates interrupt when command coherency is not achieved
- External hardware triggers
  - Supports rising edge, falling edge, high level and low level triggers
  - Supports configurable digital filter
- Supports four external 8-to-1 muxes which can expand the input channels to 56 channels total

1.5.15 DSPI

The deserial serial peripheral interface (DSPI) block provides a synchronous serial interface for communication between the SPC563Mxx MCU and external devices. The DSPI supports pin count reduction through serialization and deserialization of eTPU and eMIOS channels and memory-mapped registers. The channels and register content are transmitted using a SPI-like protocol. This SPI-like protocol is completely configurable for baud rate, polarity and phase, frame length, chip select assertion, etc. Each bit in the frame may be configured to serialize either eTPU channels, eMIOS channels or GPIO signals. The DSPI can be configured to serialize data to an external device that supports the Microsecond Channel protocol. There are two identical DSPI blocks on the SPC563Mxx MCU. The DSPI output pins support 5 V logic levels or Low Voltage Differential Signalling (LVDS) according to the Microsecond Channel specification.

The DSPIs have three configurations:

- Serial peripheral interface (SPI) configuration where the DSPI operates as an up to 16-bit SPI with support for queues
- Enhanced deserial serial interface (DSI) configuration where DSPI serializes up to 32 bits with three possible sources per bit
  - eTPU, eMIOS, new virtual GPIO registers as possible bit source
  - Programmable inter-frame gap in continuous mode

  - Bit source selection allows microsecond channel downstream with command or data frames up to 32 bits
  - Microsecond channel dual receiver mode
- Combined serial interface (CSI) configuration where the DSPI operates in both SPI and DSI configurations interleaving DSI frames with SPI frames, giving priority to SPI frames

For queued operations, the SPI queues reside in system memory external to the DSPI. Data transfers between the memory and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software.

The DSPI supports these SPI features:

- Full-duplex, synchronous transfers
- Selectable LVDS Pads working at 40 MHz for SOUT and SCK pins
- Master and Slave Mode
- Buffered transmit operation using the TX FIFO with parameterized depth of 4 entries
- Buffered receive operation using the RX FIFO with parameterized depth of 4 entries
- TX and RX FIFOs can be disabled individually for low-latency updates to SPI queues
- Visibility into the TX and RX FIFOs for ease of debugging
- FIFO Bypass Mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis:
  - Parameterized number of transfer attribute registers (from two to eight)
  - Serial clock with programmable polarity and phase
  - Various programmable delays:
  - PCS to SCK delay
  - SCK to PCS delay
  - Delay between frames
  - Programmable serial frame size of 4 to 16 bits, expandable with software control
  - Continuously held chip select capability
- 6 Peripheral Chip Selects, expandable to 64 with external demultiplexer
- Deglitching support for up to 32 Peripheral Chip Selects with external demultiplexer
- DMA support for adding entries to TX FIFO and removing entries from RX FIFO:
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- 6 Interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - Attempt to transmit with an empty Transmit FIFO (TFUF)
  - RX FIFO is not empty (RFDF)
  - FIFO Underrun (slave only and SPI mode, the slave is asked to transfer data when the TxFIFO is empty)
  - FIFO Overrun (serial frame received while RX FIFO is full)
- Modified transfer formats for communication with slower peripheral devices

- Continuous Serial Communications Clock (SCK)
- Power savings via support for Stop Mode
- Enhanced DSI logic to implement a 32-bit Timed Serial Bus (TSB) configuration, supporting the Microsecond Channel downstream frame format

The DSPIs also support these features unique to the DSI and CSI configurations:

- 2 sources of the serialized data:
  - eTPU_A and eMIOS output channels
  - Memory-mapped register in the DSPI
- Destinations for the deserialized data:
  - eTPU_A and eMIOS input channels
  - SIU External Interrupt Request inputs
  - Memory-mapped register in the DSPI
- Deserialized data is provided as Parallel Output signals and as bits in a memory-mapped register
- Transfer initiation conditions:
  - Continuous
  - Edge sensitive hardware trigger
  - Change in data
- Pin serialization/deserialization with interleaved SPI frames for control and diagnostics
- Continuous serial communications clock
- Support for parallel and serial chaining of up to four DSPI blocks

1.5.16 eSCI

The enhanced serial communications interface (eSCI) allows asynchronous serial communications with peripheral devices and other MCUs. It includes special support to interface to Local Interconnect Network (LIN) slave devices. The eSCI block provides the following features:

- Full-duplex operation
- Standard mark/space non-return-to-zero (NRZ) format
- 13-bit baud rate selection
- Programmable 8-bit or 9-bit, data format
- Programmable 12-bit or 13-bit data format for Timed Serial Bus (TSB) configuration to support the Microsecond Channel upstream
- Automatic parity generation
- LIN support
  - Autonomous transmission of entire frames
  - Configurable to support all revisions of the LIN standard
  - Automatic parity bit generation
  - Double stop bit after bit error
  - 10- or 13-bit break support
- Separately enabled transmitter and receiver
- Programmable transmitter output parity

- 2 receiver wake up methods:
  - Idle line wake-up
  - Address mark wake-up
- Interrupt-driven operation with flags
- Receiver framing error detection
- Hardware parity checking
- 1/16 bit-time noise detection
- DMA support for both transmit and receive data
  - Global error bit stored with receive data in system RAM to allow post processing of errors

1.5.17 FlexCAN

The SPC563Mxx MCU contains two controller area network (FlexCAN) blocks. The FlexCAN module is a communication controller implementing the CAN protocol according to Bosch Specification version 2.0B. The CAN protocol was designed to be used primarily as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. FlexCAN module ‘A’ contains 64 message buffers (MB); FlexCAN module ‘C’ contains 32 message buffers.

The FlexCAN module provides the following features:

- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Zero to eight bytes data length
  - Programmable bit rate up to 1 Mbit/s
- Content-related addressing
- 64 / 32 message buffers of zero to eight bytes data length
- Individual Rx Mask Register per message buffer
- Each message buffer configurable as Rx or Tx, all supporting standard and extended messages
- Includes 1056 / 544 bytes of embedded memory for message buffer storage
- Includes a 256-byte and a 128-byte memories for storing individual Rx mask registers
- Full featured Rx FIFO with storage capacity for six frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against 8 extended, 16 standard or 32 partial (8 bits) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN versions
- Programmable clock source to the CAN Protocol Interface, either system clock or oscillator clock
- Listen only mode capability
- Programmable loop-back mode supporting self-test operation
- 3 programmable Mask Registers
- Programmable transmit-first 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
- Warning interrupts when the Rx and Tx Error Counters reach 96
- Independent of the transmission medium (an external transceiver is assumed)
- Multi master concept
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages
- Low power mode, with programmable wake-up on bus activity

1.5.18 System timers

The system timers provide two distinct types of system timer:

- Periodic interrupts/triggers using the Periodic Interrupt Timer (PIT)
- Operating system task monitors using the System Timer Module (STM)

1.5.18.1 Periodic Interrupt Timer (PIT)

The PIT provides five independent timer channels, capable of producing periodic interrupts and periodic triggers. The PIT has no external input or output pins and is intended to be used to provide system ‘tick’ signals to the operating system, as well as periodic triggers for eQADC queues. Of the five channels in the PIT, four are clocked by the system clock, one is clocked by the crystal clock. This one channel is also referred to as Real Time Interrupt (RTI) and is used to wakeup the device from low power stop mode.

The following features are implemented in the PIT:

- 5 independent timer channels
- Each channel includes 32-bit wide down counter with automatic reload
- 4 channels clocked from system clock
- 1 channel clocked from crystal clock (wake-up timer)
- Wake-up timer remains active when System STOP mode is entered. Used to restart system clock after predefined timeout period
- Each channel can optionally generate an interrupt request or a trigger event (to trigger eQADC queues) when the timer reaches zero

1.5.18.2 System Timer Module (STM)

The System Timer Module (STM) is designed to implement the software task monitor as defined by AUTOSAR (see autosar web site). It consists of a single 32-bit counter, clocked by the system clock, and four independent timer comparators. These comparators produce a CPU interrupt when the timer exceeds the programmed value.

The following features are implemented in the STM:

- 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

1.5.19 Software Watchdog Timer (SWT)

The Software Watchdog Timer (SWT) is a second watchdog module to complement the standard Power Architecture watchdog integrated in the CPU core. The SWT is a 32-bit modulus counter, clocked by the system clock or the crystal clock, that can provide a system reset or interrupt request when the correct software key is not written within the required time window.

The following features are implemented:

- 32-bit modulus counter
- Clocked by system clock or crystal clock
- Optional programmable watchdog window mode
- Can optionally cause system reset or interrupt request on timeout
- Reset by writing a software key to memory mapped register
- Enabled out of reset
- Configuration is protected by a software key or a write-once register

1.5.20 Debug features

1.5.20.1 Nexus port controller

The NPC (Nexus Port Controller) block provides real-time development support capabilities for the SPC563Mxx Power Architecture-based MCU in compliance with the IEEE-ISTO 5001-2003 standard. This development support is supplied for MCUs without requiring external address and data pins for internal visibility. The NPC block is an integration of several individual Nexus blocks that are selected to provide the development support interface for SPC563Mxx. The NPC block interfaces to the host processor (e200z335), eTPU, and internal buses to provide development support as per the IEEE-ISTO 5001-2003 standard. The development support provided includes program trace and run-time access to the MCUs internal memory map and access to the Power Architecture and eTPU internal registers during halt. The Nexus interface also supports a JTAG only mode using only the JTAG pins. SPC563Mxx in the production LQFP144 supports a 3.3 V reduced (4-bit wide) Auxiliary port. These Nexus port pins can also be used as 5 V I/O signals to increase usable I/O count of the device. When using this Nexus port as IO, Nexus trace is still possible using calibration tool calibration. In the calibration tool calibration package, the full 12-bit Auxiliary port is available.

Note: In the calibration tool package, the full Nexus Auxiliary port shares balls with the addresses of the calibration bus. Therefore multiplexed address/data bus mode must be used for the calibration bus when using full width Nexus trace in calibration tool assembly.

The following features are implemented:

- 5-pin JTAG port (JCOMP, TDI, TDO, TMS, and TCK)
  - Always available in production package
  - Supports both JTAG Boundary Scan and debug modes
  - 3.3 V interface
  - Supports Nexus class 1 features
  - Supports Nexus class 3 read/write feature
- 9-pin Reduced Port interface in LQFP144 production package
  - Alternate function as IO
  - 5 V (in GPIO or alternate function mode), 3.3 V (in Nexus mode) interface
  - Auxiliary Output port
    - 1 MCKO (message clock out) pin
    - 4 MDO (message data out) pins
    - 2 $\overline{\text{MSEO}}$ (message start/end out) pins
    - 1 $\overline{\text{EVTO}}$ (event out) pin
  - Auxiliary input port
    - 1 $\overline{\text{EVTI}}$ (event in) pin
- 17-pin Full Port interface in calibration package used on calibration tool boards
  - 3.3 V interface
  - Auxiliary Output port
    - 1 MCKO (message clock out) pin
    - 4 (reduced port mode) or 12 (full port mode) MDO (message data out) pins; 8 extra full port pins shared with calibration bus
    - 2 $\overline{\text{MSEO}}$ (message start/end out) pins
    - 1 $\overline{\text{EVTO}}$ (event out) pin
  - Auxiliary input port
    - 1 $\overline{\text{EVTI}}$ (event in) pin
- Host processor (e200) development support features
  - IEEE-ISTO 5001-2003 standard class 2 compliant
  - 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 between the discontinuities. Thus, static code may be traced.
  - Watchpoint trigger enable of program trace messaging
  - Data Value Breakpoints (JTAG feature of the e200z335 core): allows CPU to be halted when the CPU writes a specific value to a memory location
    - 4 data value breakpoints
    - CPU only
    - Detects ‘equal’ and ‘not equal’
    - Byte, half word, word (naturally aligned)

Note: This feature is imprecise due to CPU pipelining.

  - Subset of Power Architecture software debug facilities with OnCE block (Nexus class 1 features)

- eTPU development support features
  - IEEE-ISTO 5001-2003 standard class 1 compliant for the eTPU
  - Nexus based breakpoint configuration and single step support (JTAG feature of the eTPU)
- Run-time access to the on-chip memory map via the Nexus read/write access protocol. This feature supports accesses for run-time internal visibility, calibration variable acquisition, calibration constant tuning, and external rapid prototyping for powertrain automotive development systems.
- All features are independently configurable and controllable via the IEEE 1149.1 I/O port
- Power-on-reset status indication during reset via MDO[0] in disabled and reset modes

1.5.20.2 JTAG

The JTAGC (JTAG Controller) 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. The JTAGC block is compliant with the IEEE 1149.1-2001 standard and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface 4 pins (TDI, TMS, TCK, and TDO)
- A 5-bit instruction register that supports the following IEEE 1149.1-2001 defined instructions:
  - BYPASS, IDCODE, EXTEST, SAMPLE, SAMPLE/PRELOAD, HIGHZ, CLAMP
- A 5-bit instruction register that supports the additional following public instructions:
  - ACCESS_AUX_TAP_NPC
  - ACCESS_AUX_TAP_ONCE
  - ACCESS_AUX_TAP_eTPU
  - ACCESS_CENSOR
- 3 test data registers to support JTAG Boundary Scan mode
  - Bypass register
  - Boundary scan register
  - Device identification register
- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry
- Censorship Inhibit Register
  - 64-bit Censorship password register
  - If the external tool writes a 64-bit password that matches the Serial Boot password stored in the internal flash shadow row, Censorship is disabled until the next system reset.

2 Memory Map

This chapter presents the memory map for this device.

2.1 Introduction

All addresses in the device, including those that are reserved, are identified in the tables. The addresses represent the physical addresses assigned to each IP block. Logical addresses are translated by the Memory Management Unit (MMU) into physical addresses.

Under software control of the MMU, the logical addresses allocated to IP blocks may be changed on a minimum of a 4 KB boundary.

2.2 Memory map

Table 3 shows the device memory map.

Table 3. Device memory map

Region	Start – End address
Flash Memory (1.5 MB)(1)	0x0000_0000 – 0x0017_FFFF
Reserved	0x0018_0000 – 0x00FF_BFFF
Flash Shadow Block	0x00FF_C000 – 0x00FF_FFFF
Emulation reMapping of Flash	0x0100_0000 – 0x1FFF_FFFF
Reserved	0x2000_0000 – 0x2FFF_FFFF
Calibration Memory Space	0x3000_0000 – 0x3FFF_FFFF
SRAM (94 KB)(2)	0x4000_0000 – 0x4001_77FF
Reserved	0x4001_7000 – 0xBFFF_FFFF
Reserved	0xC000_0000 – 0xC3EF_FFFF
Reserved for PBridge A	0xC3F0_0000 – 0xC3F0_3FFF
Reserved	0xC3F0_4000 – 0xC3F7_FFFF
PLL	0xC3F8_0000 – 0xC3F8_3FFF
EBI Configuration	0xC3F8_4000 – 0xC3F8_7FFF
Flash Configuration and Bank0, Array 0 registers	0xC3F8_8000 – 0xC3F8_BFFF
Reserved	0xC3F8_C000 – 0xC3F8_FFFF
SIU	0xC3F9_0000 – 0xC3F9_3FFF
Reserved	0xC3F9_4000 – 0xC3F9_FFFF
eMIOS	0xC3FA_0000 – 0xC3FA_3FFF
Reserved	0xC3FA_4000 – 0xC3FA_FFFF
Flash Bank1, Array1 Registers	0xC3FB_0000 – 0xC3FB_3FFF
Flash Bank1, Array2 Registers	0xC3FB_4000 – 0xC3FB_7FFF

Table 3. Device memory map (continued)

Region	Start – End address
Reserved	0xC3FB_8000 – 0xC3FB_BFFF
PMC	0xC3FB_C000 – 0xC3FB_FFFF
eTPU Registers	0xC3FC_0000 – 0xC3FC_3FFF
Reserved	0xC3FC_4000 – 0xC3FC_7FFF
eTPU Parameter RAM	0xC3FC_8000 – 0xC3FC_BFFF
eTPU Parameter RAM Mirror	0xC3FC_C000 – 0xC3FC_FFFF
eTPU Code RAM	0xC3FD_0000 – 0xC3FD_3FFF
Reserved	0xC3FD_4000 – 0xFBFF_FFFF
Reserved	0xFC00_0000 – 0xFFEF_FFFF
e200 Platform Peripherals (XBAR, SWT, STM, ECSM, eDMA and INTC)	0xFFF0_0000 – 0xFFF7_FFFF
eQADC	0xFFF8_0000 – 0xFFF8_3FFF
Reserved	0xFFF8_4000 – 0xFFF8_7FFF
Decimation filter A	0xFFF8_8000 – 0xFFF9_BFFF
Reserved	0xFFF8_C000 – 0xFFF9_3FFF
DSPI_B	0xFFF9_4000 – 0xFFF9_7FFF
DSPI_C	0xFFF9_8000 – 0xFFF9_BFFF
Reserved	0xFFF9_C000 – 0xFFF9_FFFF
Reserved	0xFFFA_0000 – 0xFFFA_FFFF
eSCI_A	0xFFFB_0000 – 0xFFFB_3FFF
eSCI_B	0xFFFB_4000 – 0xFFFB_7FFF
Reserved	0xFFFB_8000 – 0xFFFB_FFFF
FlexCAN_A	0xFFFC_0000 – 0xFFFC_3FFF
Reserved	0xFFFC_4000 – 0xFFFC_7FFF
FlexCAN_C	0xFFFC_8000 – 0xFFFC_9FFF
Reserved for FlexCAN_C (higher MSBs)	0xFFFC_A000 – 0xFFFC_FFFF
Temperature Sensor	0xFFFE_C000 – 0xFFFF_BFFF
Boot Assist Module	0xFFFF_C000 – 0xFFFF_FFFF

1. See *Table 5* for amount of flash memory and *Table 6* for the specific mapping.
2. See *Table 5* for the amount of SRAM and *Table 7* for the specific mapping.

Peripheral blocks may be redundantly mapped. The customer must use the MMU to prevent corruption.

The SPC563Mxx only has a single peripheral bridge, but to match the memory map of other devices, the peripherals are mapped to appear as if they are on two different peripheral bridges.

Table 4. Detailed device memory map

Address range[1]	Size (bytes)		Use
	Allocated	Used	
0x0000_0000 – 0x0017_FFFF	1.5 M	1.5 M	Flash Memory Array[2]
0x0018_0000 – 0x00FF_BFFF	14.5 M – 16 K	—	Reserved
0x00FF_C000 – 0x00FF_FFFF	16 K	16 K	Flash Memory Shadow Block
0x0100_0000 – 0x1FFF_FFFF	496 M	1.5 M	Emulation Remapping of Flash Memory Array
0x2000_0000 – 0x2FFF_FFFF	256 M	—	Reserved
0x3000_0000 – 0x3FFF_FFFF	256 M	—	Calibration Memory Space
0x4000_0000 – 0x4000_7FFF	32 K	32 K	SRAM Array, Standby Powered[3]
0x4000_8000 – 0x4001_77FF	62 K	62 K	SRAM Array[3]
0x4001_7800 – 0xBFFF_FFFF	2048 M – 94 K	—	Reserved
Bridge Peripherals A			
0xC000_0000 – 0xC3EF_FFFF	63 M	—	Reserved
0xC3F0_0000 – 0xC3F0_3FFF	16 K	—	Reserved for PBridge A
0xC3F0_4000 – 0xC3F7_FFFF	496 K	—	Reserved
0xC3F8_0000 – 0xC3F8_3FFF	16 K	28	FMPLL
0xC3F8_4000 – 0xC3F8_7FFF	16 K	48	External Bus Interface (EBI) Configuration
0xC3F8_8000 – 0xC3F8_BFFF	16 K	40	Platform Flash Configuration and Bank0, Array0 Registers
0xC3F8_C000 – 0xC3F8_FFFF	16 K	—	Reserved Data Flash
0xC3F9_0000 – 0xC3F9_3FFF	16 K	2.5 K	System Integration Unit (SIU)
0xC3F9_4000 – 0xC3F9_7FFF	16 K	—	Reserved
0xC3F9_8000 – 0xC3F9_BFFF	16 K	—	Reserved
0xC3F9_C000 – 0xC3F9_FFFF	16 K	—	Not allocated
0xC3FA_0000 – 0xC3FA_3FFF	16 K	128	Modular Timer System (eMIOS_A)
0xC3FA_4000 – 0xC3FA_7FFF	16 K	—	Reserved
0xC3FA_8000 – 0xC3FA_BFFF	16 K	—	Reserved
0xC3FA_C000 – 0xC3FA_FFFF	16 K	—	Not allocated
0xC3FB_0000 – 0xC3FB_3FFF	16 K	28	Flash bank1, Array1 Registers
0xC3FB_4000 – 0xC3FB_7FFF	16 K	28	Flash bank1, Array2 Registers
0xC3FB_8000 – 0xC3FB_BFFF	16 K	—	Reserved
0xC3FB_C000 – 0xC3FB_FFFF	16 K	12	PMC
0xC3FC_0000 – 0xC3FC_3FFF	16 K	3 K	Enhanced Time Processing Unit (eTPU) Registers
0xC3FC_4000 – 0xC3FC_7FFF	16 K	—	Not allocated
0xC3FC_8000 – 0xC3FC_BFFF	16 K	3 K	eTPU Parameter RAM

Table 4. Detailed device memory map (continued)

Address range[1]	Size (bytes)		Use
	Allocated	Used	
0xC3FC_C000 – 0xC3FC_FFFF	16 K	3 K	eTPU Parameter RAM mirror
0xC3FD_0000 – 0xC3FD_3FFF	16 K	14 K	eTPU Code RAM
0xC3FD_4000 – 0xC3FD_7FFF	16 K	—	Reserved
0xC3FD_8000 – 0xC3FD_BFFF	16 K	—	Reserved
0xC3FD_C000 – 0xC3FD_FFFF	16 K	—	Reserved
0xC3FE_0000 – 0xC3FE_3FFF	16 K	—	Reserved
0xC3FE_4000 – 0xC3FE_7FFF	16 K	—	Reserved
0xC3FE_8000 – 0xC3FE_BFFF	16 K	—	Reserved
0xC3FE_C000 – 0xC3FE_FFFF	16 K	—	Reserved
0xC3FF_0000 – 0xC3FF_3FFF	16 K	84	PIT/RTI
0xC3FF_4000 – 0xC3FF_7FFF	16 K	—	Reserved
0xC3FF_8000 – 0xC3FF_BFFF	16 K	—	Reserved
0xC3FF_C000 – 0xC3FF_BFFF	16 K	—	Not allocated
0xC400_0000 – 0xDFFF_FFFF	(512 M – 64 M)	—	Reserved
Bridge Peripherals B			
0xE000_0000 – 0xFFE7_FFFF	512 M – 1.5 M	—	Reserved
0xFFE8_0000 – 0xFFE8_3FFF	16 K	28	FMPLL_A
0xFFE8_4000 – 0xFFE8_7FFF	16 K	—	Reserved for EBI Configuration
0xFFE8_8000 – 0xFFE8_BFFF	16 K	40	Flash Bank0 (array0) IPS Registers, Platform Flash Configuration
0xFFE8_C000 – 0xFFE8_FFFF	16 K	—	Reserved for Data Flash A Configuration
0xFFE9_0000 – 0xFFE9_3FFF	16 K	2.5 K	SIU
0xFFE9_4000 – 0xFFE9_7FFF	16 K	—	Reserved
0xFFE9_8000 – 0xFFE9_BFFF	16 K	—	Not allocated
0xFFE9_C000 – 0xFFE9_FFFF	16 K	—	Not allocated
0xFFEA_0000 – 0xFFEA_3FFF	16 K	128	eMIOS_A
0xFFEA_4000 – 0xFFEA_7FFF	16 K	—	Reserved for EMIOS_B
0xFFEA_8000 – 0xFFEA_BFFF	16 K	—	Reserved for eMIOS_C
0xFFEA_C000 – 0xFFEA_FFFF	16 K	—	Not allocated
0xFFEB_0000 – 0xFFEB_3FFF	16 K	28	Flash Bank 1 (array1) Registers
0xFFEB_4000 – 0xFFEB_7FFF	16 K	28	Flash Bank 1 (array 2) Registers
0xFFEB_8000 – 0xFFEB_BFFF	16 K	—	Reserved for Code Flash D Configuration
0xFFEB_C000 – 0xFFEB_FFFF	16 K	12	PMU

Table 4. Detailed device memory map (continued)

Address range[1]	Size (bytes)		Use
	Allocated	Used	
0xFFEC_0000 – 0xFFEC_3FFF	16 K	3 K	eTPU
0xFFEC_4000 – 0xFFEC_7FFF	16 K	—	Not allocated
0xFFEC_8000 – 0xFFEC_BFFF	16 K	3 K	eTPU Parameter RAM
0xFFEC_C000 – 0xFFEC_FFFF	16 K	3 K	eTPU Parameter RAM Mirror
0xFFED_0000 – 0xFFED_3FFF	16 K	14 K	eTPU Parameter Code RAM
0xFFED_4000 – 0xFFED_7FFF	16 K	—	Reserved for eTPU Parameter Code RAM
0xFFED_8000 – 0xFFED_BFFF	16 K	—	Reserved for SSCM
0xFFED_C000 – 0xFFED_FFFF	16 K	—	Reserved for MC_ME
0xFFEE_0000 – 0xFFEE_3FFF	16 K	—	Reserved for MC_CGM
0xFFEE_4000 – 0xFFEE_7FFF	16 K	—	Reserved for MC_RGM
0xFFEE_8000 – 0xFFEE_BFFF	16 K	—	Reserved for MC_PCU
0xFFEE_C000 – 0xFFEE_FFFF	16 K	—	Reserved for RTC/API
0xFFEF_0000 – 0xFFEF_3FFF	16 K	84	PIT/RTI
0xFFEF_4000 – 0xFFEF_7FFF	16 K	—	Not allocated
0xFFEF_8000 – 0xFFEF_BFFF	16 K	—	Reserved for RNGA
0xFFEF_C000 – 0xFFEF_FFFF	16 K	—	Reserved for Process Monitor
0xFFF0_0000 – 0xFFF0_3FFF	16 K	—	Reserved for PBridge B
0xFFF0_4000 – 0xFFF0_7FFF	16 K	4 K	Crossbar (AXBS)
0xFFF0_8000 – 0xFFF0_BFFF	16 K	—	Not allocated
0xFFF0_C000 – 0xFFF0_FFFF	16 K	—	Not allocated
0xFFF1_0000 – 0xFFF1_3FFF	16 K	—	Reserved
0xFFF1_4000 – 0xFFF1_7FFF	16 K	—	Not allocated
0xFFF1_8000 – 0xFFF1_BFFF	16 K	—	Not allocated
0xFFF1_C000 – 0xFFF1_FFFF	16 K	—	Not allocated
0xFFF2_0000 – 0xFFF2_3FFF	16 K	—	Not allocated
0xFFF2_4000 – 0xFFF2_7FFF	16 K	—	Not allocated
0xFFF2_8000 – 0xFFF2_BFFF	16 K	—	Not allocated
0xFFF2_C000 – 0xFFF2_FFFF	16 K	—	Not allocated
0xFFF3_0000 – 0xFFF3_3FFF	16 K	—	Not allocated
0xFFF3_4000 – 0xFFF3_7FFF	16 K	—	Reserved
0xFFF3_8000 – 0xFFF3_BFFF	16 K	20	SWT
0xFFF3_C000 – 0xFFF3_FFFF	16 K	72	STM
0xFFF4_0000 – 0xFFF4_3FFF	16 K	128	ECSM

Table 4. Detailed device memory map (continued)

Address range[1]	Size (bytes)		Use
	Allocated	Used	
0xFFF4_4000 – 0xFFF4_7FFF	16 K	6 K	DMA Controller 2 (eDMA)
0xFFF4_8000 – 0xFFF4_BFFF	16 K	272	Interrupt Controller (INTC)
0xFFF4_C000 – 0xFFF4_FFFF	16 K	—	Reserved
0xFFF5_0000 – 0xFFF5_3FFF	16 K	—	Not allocated
0xFFF5_4000 – 0xFFF5_7FFF	16 K	—	Reserved
0xFFF5_8000 – 0xFFF5_BFFF	16 K	—	Not allocated
0xFFF5_C000 – 0xFFF5_FFFF	16 K	—	Reserved
0xFFF6_0000 – 0xFFF6_3FFF	16 K	—	Not allocated
0xFFF6_4000 – 0xFFF6_7FFF	16 K	—	Not allocated
0xFFF6_8000 – 0xFFF6_BFFF	16 K	—	Not allocated
0xFFF6_C000 – 0xFFF6_FFFF	16 K	—	Not allocated
0xFFF7_0000 – 0xFFF7_3FFF	16 K	—	Not allocated
0xFFF7_4000 – 0xFFF7_7FFF	16 K	—	Not allocated
0xFFF7_8000 – 0xFFF7_BFFF	16 K	—	Not allocated
0xFFF7_C000 – 0xFFF7_FFFF	16 K	—	Not allocated
0xFFF8_0000 – 0xFFF8_3FFF	16 K	292	Enhanced Queued Analog-to-Digital Converter (eQADC_A)
0xFFF8_4000 – 0xFFF8_7FFF	16 K	—	Reserved
0xFFF8_8000 – 0xFFF8_BFFF	16 K	4 K	Decimation Filter A
0xFFF8_C000 – 0xFFF8_FFFF	16 K	—	Reserved
0xFFF9_0000 – 0xFFF9_3FFF	16 K	—	Reserved
0xFFF9_4000 – 0xFFF9_7FFF	16 K	112	Deserial Serial Peripheral Interface (DSPI_B)
0xFFF9_8000 – 0xFFF9_BFFF	16 K	112	Deserial Serial Peripheral Interface (DSPI_C)
0xFFF9_C000 – 0xFFF9_FFFF	16 K	—	Reserved
0xFFFA_0000 – 0xFFFA_3FFF	16 K	—	Reserved
0xFFFA_4000 – 0xFFFA_7FFF	16 K	—	Reserved
0xFFFA_8000 – 0xFFFA_BFFF	16 K	—	Not allocated
0xFFFA_C000 – 0xFFFA_FFFF	16 K	—	Not allocated
0xFFFB_0000 – 0xFFFB_3FFF	16 K	44	Enhanced Serial Communications Interface (eSCI_A)
0xFFFB_4000 – 0xFFFB_7FFF	16 K	44	Enhanced Serial Communications Interface (eSCI_B)
0xFFFB_8000 – 0xFFFB_BFFF	16 K	—	Reserved
0xFFFB_C000 – 0xFFFB_FFFF	16 K	—	Reserved
0xFFFC_0000 – 0xFFFC_3FFF	16 K	1152	Controller Area Network (FlexCAN_A)

Table 4. Detailed device memory map (continued)

Address range[1]	Size (bytes)		Use
	Allocated	Used	
0xFFFC_4000 – 0xFFFC_7FFF	16 K	—	Reserved
0xFFFC_8000 – 0xFFFC_9FFF	8 K	576	Controller Area Network (FlexCAN_C)
0xFFFC_A000 – 0xFFFC_FFFF	8 K	—	Reserved for FlexCAN_C (higher MSBs)
0xFFFD_0000 – 0xFFFD_3FFF	16 K	—	Reserved
0xFFFD_4000 – 0xFFFD_7FFF	16 K	—	Reserved
0xFFFD_8000 – 0xFFFD_BFFF	16 K	—	Not allocated
0xFFFD_C000 – 0xFFFD_FFFF	16 K	—	Reserved
0xFFFE_0000 – 0xFFFE_3FFF	16 K	—	Reserved
0xFFFE_4000 – 0xFFFE_7FFF	16 K	—	Not allocated
0xFFFE_8000 – 0xFFFE_BFFF	16 K	—	Reserved
0xFFFE_C000 – 0xFFFE_FFFF	16 K	8	Temperature Sensor
0xFFFF_0000 – 0xFFFF_3FFF	16 K	—	Not allocated
0xFFFF_4000 – 0xFFFF_7FFF	16 K	—	Not allocated
0xFFFF_8000 – 0xFFFF_BFFF	16 K	—	Reserved
0xFFFF_C000 – 0xFFFF_FFFF	16 K	4 K	Boot Assist Module (BAM)

1. If allocated size > used size, then the base address for the block is the lowest address of the listed address range, unless noted otherwise.
2. See *Table 5* for amount of flash memory and *Table 6* for the specific mapping.
3. See *Table 5* for the amount of SRAM and *Table 7* for the specific mapping.

Table 5. SPC563Mxx family device memory space

Memory	SPC563M64	SPC563M60
Flash	1.5 MB	1 MB
SRAM	94 KB	64 KB

Table 6. SPC563Mxx family device flash memory map

Start address	End address	Description	SPC563M60	SPC563M64
0x0000_0000	0x0000_3FFF	16 KB flash	Available	Available
0x0000_4000	0x0000_7FFF	16 KB flash	Available	Available
0x0000_8000	0x0000_FFFF	32 KB flash	Available	Available
0x0001_0000	0x0001_7FFF	32 KB flash	Available	Available
0x0001_8000	0x000B_FFFF	16 KB flash	Available	Available
0x0001_C000	0x0001_FFFF	16 KB flash	Available	Available
0x0002_0000	0x0002_FFFF	64 KB flash	Available	Available

Table 6. SPC563Mxx family device flash memory map (continued)

Start address	End address	Description	SPC563M60	SPC563M64
0x0003_0000	0x0003_FFFF	64 KB flash	Available	Available
0x0004_0000	0x0005_FFFF	128 KB flash	Available	Available
0x0006_0000	0x0007_FFFF	128 KB flash	Available	Available
0x0008_0000	0x0009_FFFF	128 KB flash	Available	Available
0x000A_0000	0x000B_FFFF	128 KB flash	Available	Available
0x000C_0000	0x000D_FFFF	128 KB flash	Available	Available
0x000E_0000	0x000F_FFFF	128 KB flash	Available	Available
0x0010_0000	0x0011_FFFF	128 KB flash	Not available	Available
0x0012_0000	0x0013_FFFF	128 KB flash	Not available	Available
0x0014_0000	0x0015_FFFF	128 KB flash	Not available	Available
0x0016_0000	0x0017_FFFF	128 KB flash	Not available	Available
Reserved				
0x00FF_C000	0x00FF_FFFF	16 KB flash (shadow block)	Available	Available

Table 7. SPC563Mxx family device SRAM memory map

Start address	End address	Description	SPC563M60	SPC563M64
0x4000_0000	0x4000_7FFF	32 KB SRAM	Standby SRAM	Standby SRAM
0x4000_8000	0x4000_BFFF	16 KB SRAM	SRAM	SRAM
0x4000_C000	0x4000_FFFF	16 KB SRAM	SRAM	SRAM
0x4001_0000	0x4001_77FF	30 KB SRAM	Not available	SRAM

3 Signal Descriptions

This chapter describes signals that connect to package pins. It includes pinout diagrams, recommended system connections, and detailed discussions of signals.

3.1 Device Pin Assignments

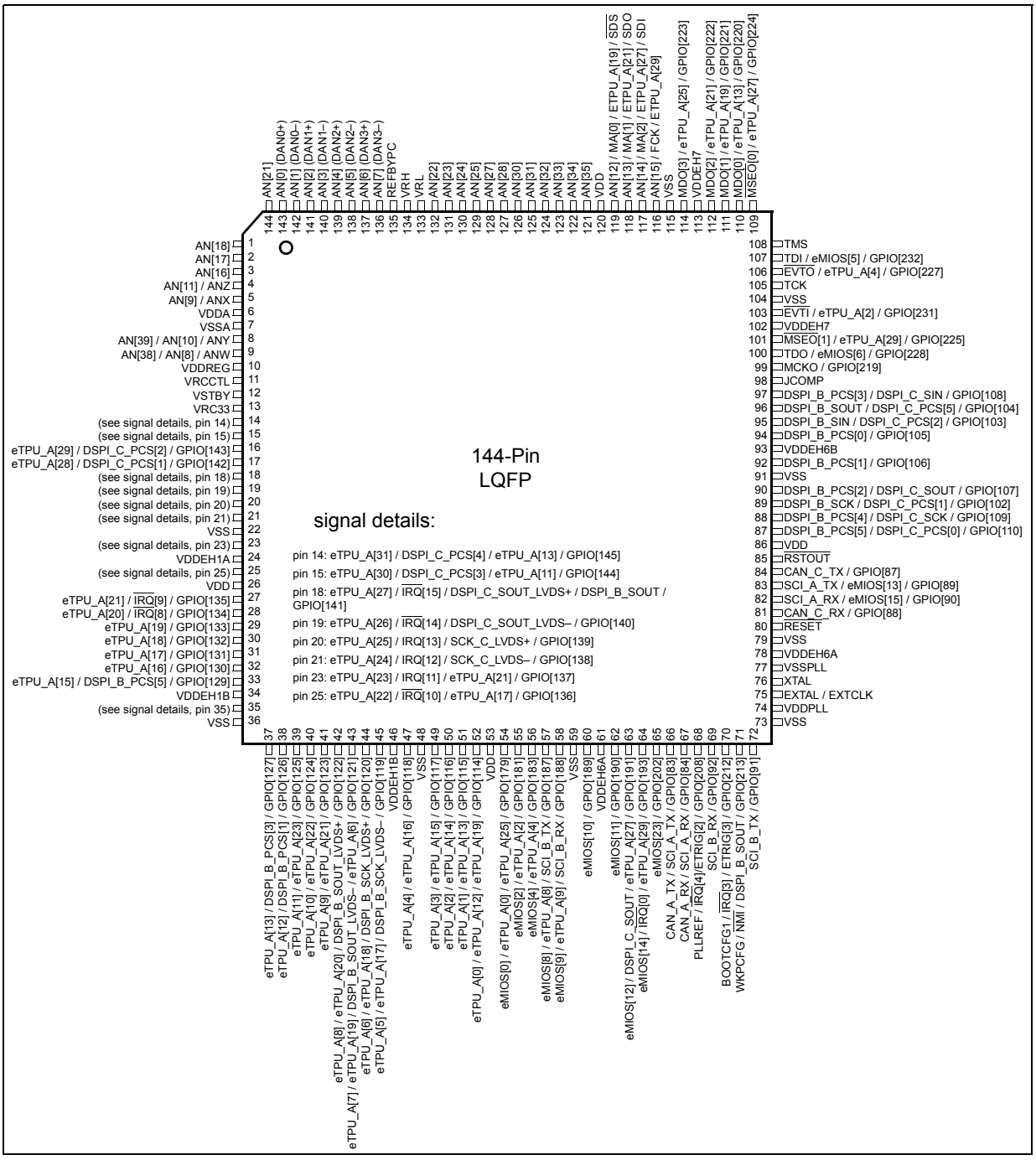
This section contains the pinouts for all production packages for the SPC563Mxx family of devices. Please note the following:

- 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 “NIC” have no internal connection.

3.1.1 LQFP144 Pinout (all 144-pin devices)

Figure 2 shows the pinout for the 144-pin LQFP.

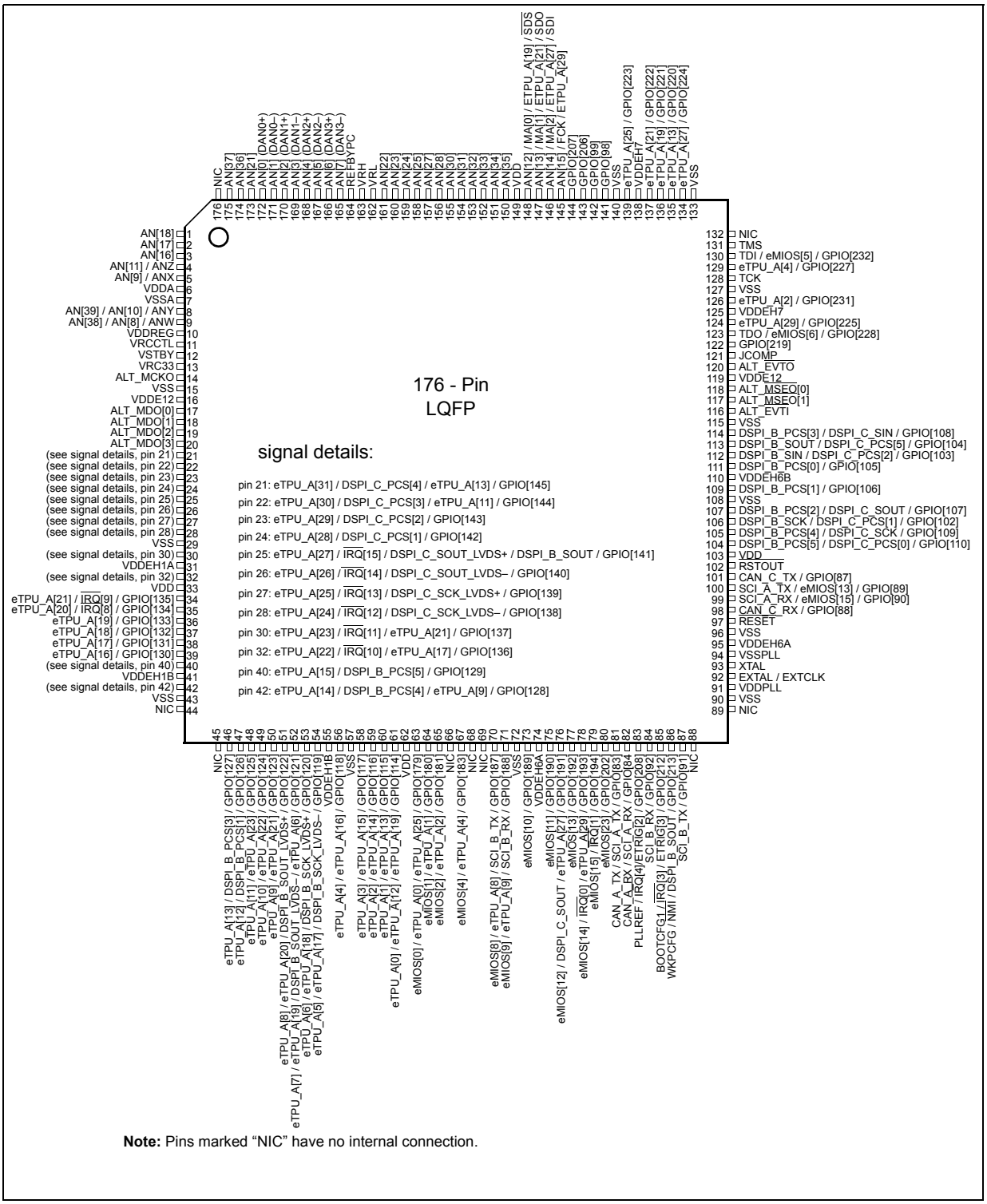
Figure 2. 144-pin LQFP Pinout (top view; all 144-pin devices)



3.1.2 LQFP176 Pinout (all 176–pin devices)

Figure 3 shows the 176-pin LQFP pinout for the SPC563M64 (1536 KB flash memory).

Figure 3. 176-pin LQFP Pinout (SPC563M64; top view)



Note: Pins marked “NIC” have no internal connection.

3.2 External Signal Summary

Table 8 gives a summary of the SPC563Mxx external signals and properties.

The SPC563Mxx pin names consist of the following items separated by underscores:

1. Primary function
2. Alternate functions (if applicable)
3. GPIO

For example, for the pin DSPI_B_SCK_DSPI_C_PCS[1]_GPIO[102], DSPI_B_SCK is the primary function and DSPI_C_PCS[1] is the alternate function.

Table 8. SPC563Mx signal properties

Name	Function(1)	Pad Config. Register (PCR)(2)	PCR PA Field(3)	I/O Type	Voltage(4) / Pad Type	Reset State(5)	Function / State After Reset(6)	Pin No.	
								LQFP 144	LQFP 176
Dedicated GPIO									
GPIO[98]	GPIO	PCR[98]	—	I/O	VDDEH7 Slow	– / Up	GPIO[98]/Up	—	141(7)
GPIO[99]	GPIO	PCR[99]	—	I/O	VDDEH7 Slow	– / Up	GPIO[99]/Up	—	142(7)
GPIO[206](8)	GPIO	PCR[206]	—	I/O	VDDEH7 Slow	– / Up	GPIO[206]/Up	—	143(7)
GPIO[207](8)	GPIO	PCR[207]	—	I/O	VDDEH7 Slow	– / Up	GPIO[207]/Up	—	144(7)
Reset / Configuration									
RESET	External Reset Input	—	—	I	VDDEH6a Slow	I / Up	$\overline{\text{RESET}}$ / Up	80	97
RSTOUT	External Reset Output	PCR[230]	—	O	VDDEH6a Slow	$\overline{\text{RSTOUT}}$/ Low	$\overline{\text{RSTOUT}}$/ High	85	102

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
PLLREF IRQ[4] ETRIG[2] GPIO[208]	FMPLL Mode Selection External Interrupt Request eQADC Trigger Input GPIO	PCR[208]	011 010 100 000	I I I I/O	VDDEH6a Slow	PLLREF / Up	– / Up	68	83
BOOTCFG1 IRQ[3] ETRIG[3] GPIO[212]	Boot Config. Input External Interrupt Request eQADC Trigger Input GPIO	PCR[212]	011 010 100 000	I I I I/O	VDDEH6a Slow	BOOTCFG1 / Down	– / Down	70	85
WKPCFG NMI DSPI_B_SOUT GPIO[213]	Weak Pull Config. Input Non-Maskable Interrupt DSPI_B Data Output GPIO	PCR[213]	01 11 10 00	I I O I/O	VDDEH6a Slow	WKPCFG / Up	– / Up	71	86
Calibration[9]									
CAL_ADDR[12:15][10]	Calibration Address Bus	PCR[340]	—	O	VDDE12 Fast	O / Low	CAL_ADDR / Low	—	—
CAL_ADDR[16][19] ALT_MDO[0][11]	Calibration Address Bus Nexus Msg Data Out	PCR[345]	—	O O	VDDE12[12] Fast	O / Low[13]	MDO / ALT_ADDR[11] / Low	—	17
CAL_ADDR[17][19] ALT_MDO[1][11]	Calibration Address Bus Nexus Msg Data Out	PCR[345]	—	O O	VDDE12[12] Fast	O / Low[13]	ALT_MDO / CAL_ADDR[11] / Low	—	18
CAL_ADDR[18][19] ALT_MDO[2][11]	Calibration Address Bus Nexus Msg Data Out	PCR[345]	—	O O	VDDE12[12] Fast	O / Low[13]	ALT_MDO / CAL_ADDR[11] / Low	—	19
CAL_ADDR[19][19] ALT_MDO[3][11]	Calibration Address Bus Nexus Msg Data Out	PCR[345]	—	O O	VDDE12[12] Fast	O / Low[13]	ALT_MDO / CAL_ADDR[11] / Low	—	20

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
CAL_ADDR[20:27] ALT_MDO[4:11]	Calibration Address Bus Nexus Msg Data Out	PCR[345]	—	O O	VDDE12[12] Fast	O / Low	ALT_MDO / CAL_ADDR[14] / Low	—	—
CAL_ADDR[28][19] ALT_$\overline{\text{MSEO}}$[0][11]	Calibration Address Bus Nexus Msg Start/End Out	PCR[345]	—	O O	VDDE12[12] Fast	O / Low[15]	ALT_$\overline{\text{MSEO}}$[14] / CAL_ADDR[15]/ Low	—	118
CAL_ADDR[29][19] ALT_$\overline{\text{MSEO}}$[1][11]	Calibration Address Bus Nexus Msg Start/End Out	PCR[345]	—	O O	VDDE12[12] Fast	O / Low[15]	ALT_$\overline{\text{MSEO}}$[14] / CAL_ADDR[15]/ Low	—	117
CAL_ADDR[30][19] ALT_$\overline{\text{EVTI}}$[11]	Calibration Address Bus Nexus Event In	PCR[345]	—	O I	VDDE12[12] Fast	—[16]	ALT_$\overline{\text{EVTI}}$ / CAL_ADDR[17]	—	116
ALT_$\overline{\text{EVTO}}$	Nexus Event Out	PCR[344]	—	O	VDDE12[12] Fast	O / Low	ALT_$\overline{\text{EVTO}}$ / High	—	120
ALT_MCKO	Nexus Msg Clock Out	PCR[344]	—	O	VDDE12[12] Fast	O / Low	ALT_MCKO / Enabled	—	14
NEXUSCFG[10]	Nexus/Calibration bus selector	—	—	I	VDDE12 Fast	I / Down	NEXUSCFG / Down	—	—
CAL_$\overline{\text{CS}}$[0][10]	Calibration Chip Selects	PCR[336]	—	O	VDDE12 Fast	O / High	CAL_$\overline{\text{CS}}$ / High	—	—
CAL_$\overline{\text{CS}}$[2][10] CAL_ADDR[10]	Calibration Chip Selects Calibration Address Bus	PCR[338]	11 10	O O	VDDE12 Fast	O / High	CAL_$\overline{\text{CS}}$ / High	—	—
CAL_$\overline{\text{CS}}$[3][10] CAL_ADDR[11]	Calibration Chip Selects Calibration Address Bus	PCR[339]	11 10	O O	VDDE12 Fast	O / High	CAL_$\overline{\text{CS}}$ / High	—	—
CAL_DATA[0:9][10]	Calibration Data Bus	PCR[341]		I/O	VDDE12 Fast	– / Up	– / Up	—	—
CAL_DATA[10:15][10]	Calibration Data Bus	PCR[341]		I/O	VDDE12 Fast	– / Up	– / Up	—	—

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
CAL_$\overline{OE}$[10]	Calibration Output Enable	PCR[342]	—	O	VDDE12 Fast	O / High	CAL_$\overline{OE}$ / High	—	—
CAL_RD_$\overline{WR}$[10]	Calibration Read/Write	PCR[342]	—	O	VDDE12 Fast	O / High	CAL_RD_$\overline{WR}$ /High	—	—
CAL_$\overline{TS}$_ALE[10]	Calibration Transfer Start Address Latch Enable	PCR[343]	TS=0b1 ALE=0b0	O O	VDDE12 Fast	O / High	CAL_$\overline{TS}$ / High	—	—
CAL_$\overline{WE}$_BE[0:1][10]	Calibration Write Enable Byte Enable	PCR[342]	—	O	VDDE12 Fast	O / High	CAL_$\overline{WE}$ / High	—	—
NEXUS[18]									
$\overline{EVTI}$[19] eTPU_A[2] GPIO[231]	Nexus Event In eTPU A Ch. GPIO	PCR[231]	01 10 00	I O I/O	VDDEH7 Multi-V	– / –	– / –	103	126
$\overline{EVTO}$ [19] eTPU_A[4] GPIO[227]	Nexus Event Out eTPU A Ch. GPIO	PCR[227]	01[20] 10 00	O O I/O	VDDEH7 Multi-V	I / Up	I / Up	106	129
MCKO[19] GPIO[219]	Nexus Msg Clock Out GPIO	PCR[219]	N/A[20] 00	O I/O	VDDEH7 Multi-V	– / –	– / –	99	122
MDO[0][19] eTPU_A[13] GPIO[220]	Nexus Msg Data Out eTPU A Ch. GPIO	PCR[220]	01 10 00	O O I/O	VDDEH7 Multi-V	– / –	– / –	110	135
MDO[1][19] eTPU_A[19] GPIO[221]	Nexus Msg Data Out eTPU A Ch. GPIO	PCR[221]	01[20] 10 00	O O I/O	VDDEH7 Multi-V	– / –	– / –	111	136
MDO[2][19] eTPU_A[21] GPIO[222]	Nexus Msg Data Out eTPU A Ch. GPIO	PCR[222]	01[20] 10 00	O O I/O	VDDEH7 Multi-V	– / –	– / –	112	137

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
MDO[3][19] eTPU_A[25] GPIO[223]	Nexus Msg Data Out eTPU A Ch. GPIO	PCR[223]	01[20] 10 00	O O I/O	VDDEH7 Multi-V	– / –	– / –	114	139
$\overline{\text{MSEO}}$[0][19] eTPU_A[27] GPIO[224]	Nexus Msg Start/End Out eTPU A Ch. GPIO	PCR[224]	01[20] 10 00	O O I/O	VDDEH7 Multi-V	– / –	– / –	109	134
$\overline{\text{MSEO}}$[1][19] eTPU_A[29] GPIO[225]	Nexus Msg Start/End Out eTPU A Ch. GPIO	PCR[225]	01[20] 10 00	O O I/O	VDDEH7 Multi-V	– / –	– / –	101	124
JTAG / TEST									
TCK	JTAG Test Clock Input	—	—	I	VDDEH7 Multi-V	TCK / Down	TCK / Down	105	128
TDI[21] eMIOS[5] GPIO[232]	JTAG Test Data Input eMIOS Ch. GPIO	PCR[232]	01[22] 10 00	I O I/O	VDDEH7 Multi-V	– / –	– / –	107	130
TDO[21] eMIOS[6] GPIO[228]	JTAG Test Data Output eMIOS Ch. GPIO	PCR[228]	01[22] 10 00	O O I/O	VDDEH7 Multi-V	– / –	– / –	100	123
TMS	JTAG Test Mode Select Input	—	—	I	VDDEH7 Multi-V	TMS / Up	TMS / Up	108	131
JCOMP	JTAG TAP Controller Enable	—	—	I	VDDEH7 Multi-V	JCOMP / Down	JCOMP / Down	98	121
CAN									
CAN_A_TX SCI_A_TX GPIO[83]	CAN_A Transmit eSCI_A Transmit GPIO	PCR[83]	01 10 00	O O I/O	VDDEH6a Slow	– / Up	– / Up[23]	66	81

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
CAN_A_RX SCI_A_RX GPIO[84]	CAN_A Receive eSCI_A Receive GPIO	PCR[84]	01 10 00	I I I/O	VDDEH6a Slow	– / Up	– / Up	67	82
CAN_C_TX GPIO[87]	CAN_C Transmit GPIO	PCR[87]	01 00	O I/O	VDDEH6a Medium	– / Up	– / Up	84	101
CAN_C_RX GPIO[88]	CAN_C Receive GPIO	PCR[88]	01 00	I I/O	VDDEH6a Slow	– / Up	– / Up	81	98
eSCI									
SCI_A_TX eMIOS[13] GPIO[89]	eSCI_A Transmit eMIOS Ch. GPIO	PCR[89]	01 10 00	O O I/O	VDDEH6a Slow	– / Up	– / Up	83	100
SCI_A_RX[24] eMIOS[15] GPIO[90]	eSCI_A Receive eMIOS Ch. GPIO	PCR[90]	01 10 00	I O I/O	VDDEH6a Slow	– / Up	– / Up	82	99
SCI_B_TX GPIO[91]	eSCI_B Transmit GPIO	PCR[91]	01 00	I/O I/O	VDDEH6a Slow	– / Up	– / Up	72	87
SCI_B_RX GPIO[92]	eSCI_B Receive GPIO	PCR[92]	01 00	I I/O	VDDEH6a Slow	– / Up	– / Up	69	84
DSPI									
DSPI_B_SCK DSPI_C_PCS[1] GPIO[102]	DSPI_B Clock DSPI_C Periph Chip Select GPIO	PCR[102]	01 10 00	I/O O I/O	VDDEH6b Medium	– / Up	– / Up	89	106
DSPI_B_SIN DSPI_C_PCS[2] GPIO[103]	DSPI_B Data Input DSPI_C Periph Chip Select GPIO	PCR[103]	01 10 00	I O I/O	VDDEH6b Medium	– / Up	– / Up	95	112

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
DSPI_B_SOUT DSPI_C_PCS[5] GPIO[104]	DSPI_B Data Output DSPI_C Periph Chip Select GPIO	PCR[104]	01 10 00	O O I/O	VDDEH6b Medium	– / Up	– / Up	96	113
DSPI_B_PCS[0] GPIO[105]	DSPI_B Periph Chip Select GPIO	PCR[105]	01 00	O I/O	VDDEH6b Medium	– / Up	– / Up	94	111
DSPI_B_PCS[1] GPIO[106]	DSPI_B Periph Chip Select GPIO	PCR[106]	01 00	O I/O	VDDEH6b Medium	– / Up	– / Up	92	109
DSPI_B_PCS[2] DSPI_C_SOUT GPIO[107]	DSPI_B Periph Chip Select DSPI_C Data Output GPIO	PCR[107]	01 10 00	O O I/O	VDDEH6b Medium	– / Up	– / Up	90	107
DSPI_B_PCS[3] DSPI_C_SIN GPIO[108]	DSPI_B Periph Chip Select DSPI_C Data Input GPIO	PCR[108]	01 10 00	O I I/O	VDDEH6b Medium	– / Up	– / Up	97	114
DSPI_B_PCS[4] DSPI_C_SCK GPIO[109]	DSPI_B Periph Chip Select DSPI_C Clock GPIO	PCR[109]	01 10 00	O I/O I/O	VDDEH6b Medium	– / Up	– / Up	88	105
DSPI_B_PCS[5] DSPI_C_PCS[0] GPIO[110]	DSPI_B Periph Chip Select DSPI_C Periph Chip Select GPIO	PCR[110]	01 10 00	O O I/O	VDDEH6b Medium	– / Up	– / Up	87	104
eQADC									
AN[0][25] DAN0+	Single Ended Analog Input Positive Terminal Diff. Input	—	—	I I	VDDA	I / –	AN[0] / –	143	172
AN[1][25] DAN0-	Single Ended Analog Input Negative Terminal Diff. Input	—	—	I I	VDDA	I / –	AN[1] / –	142	171
AN[2][25] DAN1+	Single Ended Analog Input Positive Terminal Diff. Input	—	—	I I	VDDA	I / –	AN[2] / –	141	170

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
AN[3][25] DAN1-	Single Ended Analog Input Negative Terminal Diff. Input	—	—	I I	VDDA	I / –	AN[3] / –	140	169
AN[4][25] DAN2+	Single Ended Analog Input Positive Terminal Diff. Input	—	—	I I	VDDA	I / –	AN[4] / –	139	168
AN[5][25] DAN2-	Single Ended Analog Input Negative Terminal Diff. Input	—	—	I I	VDDA	I / –	AN[5] / –	138	167
AN[6][25] DAN3+	Single Ended Analog Input Positive Terminal Diff. Input	—	—	I I	VDDA	I / –	AN[6] / –	137	166
AN[7][25] DAN3-	Single Ended Analog Input Negative Terminal Diff. Input	—	—	I I	VDDA	I / –	AN[7] / –	136	165
AN[8]	See AN[38]-AN[8]-ANW								
AN[9] ANX	Single Ended Analog Input External Multiplexed Analog Input	—	—	I I	VDDA	I / –	AN[9] / –	5	5
AN[10]	See AN[39]-AN[10]-ANY								
AN[11] ANZ	Single Ended Analog Input External Multiplexed Analog Input	—	—	I I	VDDA	I / –	AN[11] / –	4	4
AN[12] MA[0] ETPU_A[19] SDS	Single Ended Analog Input Mux Address ETPU_A Ch. eQADC Serial Data Strobe	PCR[215]	011 010 100 000	I O O O	VDDEH7	I / –	AN[12] / –	119	148
AN[13] MA[1] ETPU_A[21] SDO	Single Ended Analog Input Mux Address ETPU_A Ch. eQADC Serial Data Out	PCR[216]	011 010 100 000	I O O O	VDDEH7	I / –	AN[13] / –	118	147

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
AN[14] MA[2] ETPU_A[27] SDI	Single Ended Analog Input Mux Address ETPU_A Ch. eQADC Serial Data In	PCR[217]	011 010 100 000	I O O I	VDDEH7	I / –	AN[14] / –	117	146
AN[15] FCK ETPU_A[29]	Single Ended Analog Input eQADC Free Running Clock ETPU_A Ch.	PCR[218]	011 010 000	I O O	VDDEH7	I / –	AN[15] / –	116	145
AN[16]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	3	3
AN[17]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	2	2
AN[18]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	1	1
AN[21]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	144	173
AN[22]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	132	161
AN[23]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	131	160
AN[24]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	130	159
AN[25]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	129	158
AN[27]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	128	157
AN[28]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	127	156
AN[30]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	126	155
AN[31]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	125	154
AN[32]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	124	153
AN[33]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	123	152
AN[34]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	122	151
AN[35]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	121	150
AN[36]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	—	174[7]
AN[37]	Single Ended Analog Input	—	—	I	VDDA	I / –	AN[x] / –	—	175[7]

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
AN[38]-AN[8]-ANW	Single Ended Analog Input Multiplexed Analog Input	—	—	I	VDDA	I / –	AN[38] / –	9	9
AN[39]-AN[10]-ANY	Single Ended Analog Input Multiplexed Analog Input	—	—	I	VDDA	I / –	AN[39] / –	8	8
VRH	Voltage Reference High	—	—	I	VDDA	– / –	VRH	134	163
VRL	Voltage Reference Low	—	—	I	VSSA0	– / –	VRL	133	162
REFBYPC	Bypass Capacitor Input	—	—	I	VRL	– / –	REFBYPC	135	164
eTPU2									
eTPU_A[0] eTPU_A[12] eTPU_A[19] GPIO[114]	eTPU_A Ch. eTPU_A Ch. eTPU_A Ch. GPIO	PCR[114]	011 010 100 000	I/O O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	52	61
eTPU_A[1] eTPU_A[13] GPIO[115]	eTPU_A Ch. eTPU_A Ch. GPIO	PCR[115]	01 10 00	I/O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	51	60
eTPU_A[2] eTPU_A[14] GPIO[116]	eTPU_A Ch. eTPU_A Ch. GPIO	PCR[116]	01 10 00	I/O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	50	59
eTPU_A[3] eTPU_A[15] GPIO[117]	eTPU_A Ch. eTPU_A Ch. GPIO	PCR[117]	01 10 00	I/O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	49	58
eTPU_A[4] eTPU_A[16] GPIO[118]	eTPU_A Ch. eTPU_A Ch. GPIO	PCR[118]	01 10 00	I/O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	47	56

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
eTPU_A[5] eTPU_A[17] DSPI_B_SCK_LVDS- GPIO[119]	eTPU_A Ch. eTPU_A Ch. DSPI_B CLOCK LVDS- GPIO	PCR[119]	001 010 100 000	I/O O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	45	54
eTPU_A[6] eTPU_A[18] DSPI_B_SCK_LVDS+ GPIO[120]	eTPU_A Ch. eTPU_A Ch. DSPI_B Clock LVDS+ GPIO	PCR[120]	001 010 100 000	I/O O O I/O	VDDEH1b Medium	– / WKPCFG	– / WKPCFG	44	53
eTPU_A[7] eTPU_A[19] DSPI_B_SOUT_LVDS- eTPU_A[6] GPIO[121]	eTPU_A Ch. eTPU_A Ch. DSPI_B Data Output LVDS- eTPU_A Ch. GPIO	PCR[121]	0001 0010 0100 1000 0000	I/O O O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	43	52
eTPU_A[8] eTPU_A[20] DSPI_B_SOUT_LVDS+ GPIO[122]	eTPU_A Ch. eTPU_A Ch. DSPI_B Data Output LVDS+ GPIO	PCR[122]	001 010 100 000	I/O O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	42	51
eTPU_A[9] eTPU_A[21] GPIO[123]	eTPU_A Ch. eTPU_A Ch. GPIO	PCR[123]	01 10 00	I/O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	41	50
eTPU_A[10] eTPU_A[22] GPIO[124]	eTPU_A Ch. eTPU_A Ch. GPIO	PCR[124]	01 10 00	I/O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	40	49
eTPU_A[11] eTPU_A[23] GPIO[125]	eTPU_A Ch. eTPU_A Ch. GPIO	PCR[125]	01 10 00	I/O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	39	48

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
eTPU_A[12] DSPI_B_PCS[1] GPIO[126]	eTPU_A Ch. DSPI_B Periph Chip Select GPIO	PCR[126]	01 10 00	I/O O I/O	VDDEH1b Medium	– / WKPCFG	– / WKPCFG	38	47
eTPU_A[13] DSPI_B_PCS[3] GPIO[127]	eTPU_A Ch. DSPI_B Periph Chip Select GPIO	PCR[127]	01 10 00	I/O O I/O	VDDEH1b Medium	– / WKPCFG	– / WKPCFG	37	46
eTPU_A[14] DSPI_B_PCS[4] eTPU_A[9] GPIO[128]	eTPU_A Ch. DSPI_B Periph Chip Select eTPU_A Ch. GPIO	PCR[128]	001 010 100 000	I/O O O I/O	VDDEH1b Medium	– / WKPCFG	– / WKPCFG	35	42
eTPU_A[15] DSPI_B_PCS[5] GPIO[129]	eTPU_A Ch. DSPI_B Periph Chip Select GPIO	PCR[129]	01 10 00	I/O O I/O	VDDEH1b Medium	– / WKPCFG	– / WKPCFG	33	40
eTPU_A[16] GPIO[130]	eTPU_A Ch. GPIO	PCR[130]	01 00	I/O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	32	39
eTPU_A[17] GPIO[131]	eTPU_A Ch. GPIO	PCR[131]	01 00	I/O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	31	38
eTPU_A[18] GPIO[132]	eTPU_A Ch. GPIO	PCR[132]	01 00	I/O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	30	37
eTPU_A[19] GPIO[133]	eTPU_A Ch. GPIO	PCR[133]	01 00	I/O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	29	36
eTPU_A[20] $\overline{\text{IRQ}}$[8] GPIO[134]	eTPU_A Ch. External Interrupt Request GPIO	PCR[134]	01 10 00	I/O I I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	28	35
eTPU_A[21] $\overline{\text{IRQ}}$[9] GPIO[135]	eTPU_A Ch. External Interrupt Request GPIO	PCR[135]	01 10 00	I/O I I/O	VDDEH1a Slow	– / WKPCFG	– / WKPCFG	27	34

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
eTPU_A[22] IRQ[10] eTPU_A[17] GPIO[136]	eTPU_A Ch. External Interrupt Request eTPU_A Ch. External GPIO	PCR[136]	001 010 100 000	I/O I O I/O	VDDEH1a Slow	– / WKPCFG	– / WKPCFG	25	32
eTPU_A[23] IRQ[11] eTPU_A[21] GPIO[137]	eTPU_A Ch. External Interrupt Request eTPU_A Ch. External GPIO	PCR[137]	001 010 100 000	I/O I O I/O	VDDEH1a Slow	– / WKPCFG	– / WKPCFG	23	30
eTPU_A[24][26] IRQ[12] DSPI_C_SCK_LVDS- GPIO[138]	eTPU_A Ch. External Interrupt Request DSPI_C Clock LVDS- GPIO	PCR[138]	001 010 100 000	I/O I O I/O	VDDEH1a Slow	– / WKPCFG	– / WKPCFG	21	28
eTPU_A[25][26] IRQ[13] DSPI_C_SCK_LVDS+ GPIO[139]	eTPU_A Ch. External Interrupt Request DSPI _C Clock LVDS+ GPIO	PCR[139]	001 010 100 000	I/O I O I/O	VDDEH1a Medium	– / WKPCFG	– / WKPCFG	20	27
eTPU_A[26][26] IRQ[14] DSPI_C_SOUT_LVDS- GPIO[140]	eTPU_A Ch. External Interrupt Request DSPI_C Data Output LVDS- GPIO	PCR[140]	001 010 100 000	I/O I O I/O	VDDEH1a Slow	– / WKPCFG	– / WKPCFG	19	26
eTPU_A[27][26] IRQ[15] DSPI_C_SOUT_LVDS+ DSPI_B_SOUT GPIO[141]	eTPU_A Ch. External Interrupt Request DSPI_C Data Output LVDS+ DSPI_B Data Output GPIO	PCR[141]	0001 0010 0100 1000 0000	I/O I O O I/O	VDDEH1a Slow	– / WKPCFG	– / WKPCFG	18	25

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
eTPU_A[28][26] DSPI_C_PCS[1] GPIO[142]	eTPU_A Ch. (Input and Output) DSPI_C Periph Chip Select GPIO	PCR[142]	10 01 00	I/O O I/O	VDDEH1a Medium	– / WKPCFG	– / WKPCFG	17	24
eTPU_A[29][26] DSPI_C_PCS[2] GPIO[143]	eTPU_A Ch. (Input and Output) DSPI_C Periph Chip Select GPIO	PCR[143]	10 01 00	I/O O I/O	VDDEH1a Medium	– / WKPCFG	– / WKPCFG	16	23
eTPU_A[30] DSPI_C_PCS[3] eTPU_A[11] GPIO[144]	eTPU_A Ch. DSPI_C Periph Chip Select eTPU_A Ch. GPIO	PCR[144]	011 010 001 000	I/O O O I/O	VDDEH1a Medium	– / WKPCFG	– / WKPCFG	15	22
eTPU_A[31] DSPI_C_PCS[4] eTPU_A[13] GPIO[145]	eTPU_A Ch. DSPI_C Periph Chip Select eTPU_A Ch. GPIO	PCR[145]	011 010 001 000	I/O O O I/O	VDDEH1a Medium	– / WKPCFG	– / WKPCFG	14	21
eMIOS									
eMIOS[0] eTPU_A[0] eTPU_A[25][27] GPIO[179]	eMIOS Ch. eTPU_A Ch. eTPU_A Ch. GPIO	PCR[179]	001 010 100 000	I/O O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	54	63
eMIOS[1] eTPU_A[1] GPIO[180]	eMIOS Ch. eTPU_A Ch. GPIO	PCR[180]	01 10 00	I/O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	—	64[7]
eMIOS[2] eTPU_A[2] GPIO[181]	eMIOS Ch. eTPU_A Ch. GPIO	PCR[181]	01 10 00	I/O O I/O	VDDEH1b Slow	– / WKPCFG	– / WKPCFG	55	65

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
eMIOS[4] eTPU_A[4] GPIO[183]	eMIOS Ch. eTPU_A Ch. GPIO	PCR[183]	01 10 00	I/O O I/O	VDDEH6a Slow	– / WKPCFG	– / WKPCFG	56	67
eMIOS[8] eTPU_A[8][28] SCI_B_TX GPIO[187]	eMIOS Ch. eTPU_A Ch. eSCI_B Transmit GPIO	PCR[187]	001 010 100 000	I/O O O I/O	VDDEH6a Slow	– / WKPCFG	– / WKPCFG	57	70
eMIOS[9] eTPU_A[9][28] SCI_B_RX GPIO[188]	eMIOS Ch. eTPU_A Ch. eSCI_B Receive GPIO	PCR[188]	001 010 100 000	I/O O I I/O	VDDEH6a Slow	– / WKPCFG	– / WKPCFG	58	71
eMIOS[10] GPIO[189]	eMIOS Ch. GPIO	PCR[189]	01 00	I/O I/O	VDDEH6a Slow	– / WKPCFG	– / WKPCFG	60	73
eMIOS[11] GPIO[190]	eMIOS Ch. GPIO	PCR[190]	01 00	I/O I/O	VDDEH6a Slow	– / WKPCFG	– / WKPCFG	62	75
eMIOS[12] DSPI_C_SOUT eTPU_A[27] GPIO[191]	eMIOS Ch. DSPI C Data Output eTPU_A Ch. GPIO	PCR[191]	001 010 100 000	I/O O O I/O	VDDEH6a Medium	– / WKPCFG	– / WKPCFG	63	76
eMIOS[13] GPIO[192]	eMIOS Ch. GPIO	PCR[192]	01 00	I/O I/O	VDDEH6a	– / WKPCFG	– / WKPCFG	—	77[7]
eMIOS[14] $\overline{\text{IRQ}}$[0] eTPU_A[29] GPIO[193]	eMIOS Ch. External Interrupt Request eTPU_A Ch. GPIO	PCR[193]	001 010 100 000	O I O I/O	VDDEH6a Slow	– / WKPCFG	– / WKPCFG	64	78
eMIOS[15] $\overline{\text{IRQ}}$[1] GPIO[194]	eMIOS Ch. External Interrupt Request GPIO	PCR[194]	01 10 00	O I I/O	VDDEH6a Slow	– / WKPCFG	– / WKPCFG	—	79[7]

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
eMIOS[23] GPIO[202]	eMIOS Ch. GPIO	PCR[202]	01 00	I/O I/O	VDDEH6a Slow	– / WKPCFG	– / WKPCFG	65	80
Clock Synthesizer									
XTAL	Crystal Oscillator Output	—	—	O	VDDEH6a	O / –	XTAL[29] / –	76	93
EXTAL EXTCLK	Crystal Oscillator Input External Clock Input	—	—	I	VDDEH6a	I / –	EXTAL[30]/ –	75	92
CLKOUT	System Clock Output	PCR[229]	—	O	VDDE5 Fast	CLKOUT / Enabled	CLKOUT / Enabled	—	—
Power / Ground									
VDDPLL	PLL Supply Voltage	—	—	I	VDDPLL (1.2V)	I / –	—	74	91
VSSPLL[31]	PLL Ground	—	—	I	VSSPLL	I / –	—	77	94
VSTBY	Power Supply for Standby RAM	—	—	I	VSTBY	I / –	—	12	12
VRC33	3.3V Voltage Regulator Bypass Capacitor	—	—	O	VRC33	O / –	—	13	13
VRCCTL	Voltage Regulator Control Output	—	—	O	NA	O / –	—	11	11
VDDA[32]	Analog Power Input for eQADC	—	—	I	VDDA (5.0 V)	I / –	—	6	6
VDDA0	Analog Power Input for eQADC	—	—	I	VDDA	I / –	—	—	—
VSSA0	Analog Ground Input for eQADC	—	—	I	VSSA	I / –	—	—	—
VDDA1	Analog Power Input for eQADC	—	—	I	VDDA	I / –	—	—	—

Table 8. SPC563Mx signal properties (continued)

Name	Function[1]	Pad Config. Register (PCR)[2]	PCR PA Field[3]	I/O Type	Voltage[4] / Pad Type	Reset State[5]	Function / State After Reset[6]	Pin No.	
								LQFP 144	LQFP 176
VSSA1	Analog Ground Input for eQADC	—	—	I	VSSA	I / –	—	—	—
VSSA[33]	Analog Ground Input for eQADC	—	—	I	VSSA	I / –	—	7	7
VDDREG	Voltage Regulator Supply	—	—	I	VDDREG (5.0 V)	I / –	—	10	10
VDD	Internal Logic Supply Input	—	—	I	VDD (1.2 V)	I / –	—	26, 53, 86, 120	33, 62, 103, 149
VSS	Ground	—	—	–	VSS0	I / –	—	22, 36, 48, 59, 73, 79, 91, 104, 115	15, 29, 43, 57, 72, 90, 96, 108, 115[7], 127, 133, 140
VDDEH1A[34] VDDEH1B[34]	I/O Supply Input	—	—	I	VDDEH1[35] (3.3V – 5.0V)	I / –	—	24, 34, 46	31, 41 55
VDDEH6a[36], [37] VDDEH6b[37]	I/O Supply Input	—	—	I	VDDEH6 (3.3V – 5.0V)	I / –	—	78, 93, 61	95, 110, 74
VDDEH6	I/O Supply Input	—	—	I	VDDEH6	I / –	—	—	—
VDDEH7	I/O Supply Input	—	—	I	VDDEH7[38] (3.3V – 5.0V)	I / –	—	102, 113	125, 138

1. For each pin in the table, each line in the Function column is a separate function of the pin. For all I/O pins the selection of primary pin function or secondary function or GPIO is done in the SIU except where explicitly noted.
2. Values in this column refer to registers in the System Integration Unit (SIU). The actual register name is “SIU_PCR” suffixed by the PCR number. For example, PCR[190] refers to the SIU register named SIU_PCR190.
3. The Pad Configuration Register (PCR) PA field is used by software to select pin function.

4. The VDDE and VDDEH supply inputs are broken into segments. Each segment of slow I/O pins (VDDEH) may have a separate supply in the 3.3 V to 5.0 V range (– 10 %/+5 %). Each segment of fast I/O (VDDE) may have a separate supply in the 1.8 V to 3.3 V range (+/– 10 %).
5. Terminology is O — output, I — input, Up — weak pull up enabled, Down — weak pull down enabled, Low — output driven low, High — output driven high. A dash for the function in this column denotes that both the input and output buffer are turned off.
6. Function after reset of GPI is general purpose input. A dash for the function in this column denotes that both the input and output buffer are turned off.
7. Not available on 1 MB version of 176-pin package.
8. The GPIO functions on GPIO[206] and GPIO[207] can be selected as trigger functions in the SIU for the ADC by making the proper selections in the SIU_ETISR and SIU_ISEL3 registers in the SIU.
9. Some signals in this section are available only on calibration package.
10. These pins are only available in the 496 CSP/MAPBGA calibration/development package.
11. On the calibration package, the Nexus function on this pin is enabled when the NEXUSCFG pin is high and Nexus is configured to full port mode. On the 176-pin and 208-pin packages, the Nexus function on this pin is enabled permanently. Do not connect the Nexus MDO or MSEO pins directly to a power supply or ground.
12. In the calibration package, the I/O segment containing this pin is called VDDE12.
13. When configured as Nexus (calibration package with NEXUSCFG=1), and JCOMP is asserted during reset, MDO[0] is driven high until the crystal oscillator becomes stable, at which time it is then negated.
14. The function of this pin is Nexus when NEXUSCFG is high.
15. High when the pin is configured to Nexus, low otherwise.
16. O/Low for the calibration with NEXUSCFG=0; I/Up otherwise.
17. ALT_ADDR/Low for the calibration package with NEXUSCFG=0; $\overline{\text{EVTI}}$/Up otherwise.
18. In 176-pin packages, the Nexus function is disabled and the pin/ball has the secondary function
19. This signal is not available in the 176-pin package.
20. The primary function is not selected via the PA field when the pin is a Nexus signal. Instead, it is activated by the Nexus controller.
21. TDI and TDO are required for JTAG operation.
22. The primary function is not selected via the PA field when the pin is a JTAG signal. Instead, it is activated by the JTAG controller.
23. The function and state of the CAN_A and eSCI_A pins after execution of the BAM program is determined by the BOOTCFG1 pin.
24. Connect an external 10K pull-up resistor to the SCI_A_RX pin to ensure that the pin is driven high during CAN serial boot.
25. For pins AN[0:7], during and just after POR negates, internal pull resistors can be enabled, resulting in as much as 4 mA of current draw. The pull resistors are disabled when the system clock propagates through the device.
26. ETPUA[24:29] are input and output. The input muxing is controlled by SIU_ISEL8 register.
27. eTPU_A[25] is an output only function.
28. Only the output channels of eTPU[8:9] are connected to pins.
29. The function after reset of the XTAL pin is determined by the value of the signal on the PLLCFG[1] pin. When bypass mode is chosen XTAL has no function and should be grounded.
30. The function after reset of the EXTAL_EXTCLK pin is determined by the value of the signal on the PLLCFG[1] pin. If the EXTCLK function is chosen, the valid operating voltage for the pin is 1.62 V to 3.6 V. If the EXTAL function is chosen, the valid operating voltage is 3.3 V.
31. VSSPLL and VSSREG are connected to the same pin.
32. This pin is shared by two pads: VDDA_AN, using pad_vdde_hv, and VDDA_DIG, using pad_vdde_int_hv.

33. This pin is shared by two pads: VSSA_AN, using pad_vsse_hv, and VSSA_DIG, using pad_vsse_int_hv.
34. VDDEH1A, VDDEH1B, and VDDEH1AB are shorted together in all production packages. The separation of the signal names is present to support legacy naming, however they should be considered as the same signal in this document.
35. LVDS pins will not work at 3.3 V.
36. The VDDEH6 segment may be powered from 3.0 V to 5.0 V for mux address or SSI functions, but must meet the VDDA specifications of 4.5 V to 5.25 V for analog input function.
37. VDDEH6A and VDDEH6B are shorted together in all production packages. The separation of the signal names is present to support legacy naming, however they should be considered as the same signal in this document.
38. If using JTAG or Nexus, the I/O segment that contains the JTAG and Nexus pins must be powered by a 5 V supply. The 3.3 V Nexus/JTAG signals are derived from the 5 volt power supply.

Table 9. Pad types

Pad Type	Name	Supply Voltage
Slow	pad_ssr_hv	3.0 V – 5.25 V
Medium	pad_msr_hv	3.0 V – 5.25 V
Fast	pad_fc	3.0 V – 3.6 V
MultiV	pad_multv_hv	3.0 V – 5.25 V (high swing mode) 4.5 V – 5.25 V (low swing mode)
Analog	pad_ae_hv	0.0 – 5.25 V
LVDS	pad_lo_lv	—

3.3 Detailed Signal Descriptions

3.3.1 Reset / Configuration

3.3.1.1 $\overline{\text{RESET}}$ — External Reset Input

The $\overline{\text{RESET}}$ input is asserted by an external device to reset the all modules of this device. The $\overline{\text{RESET}}$ pin should be asserted during a power-on reset. See *Chapter 4: Resets* for more details.

3.3.1.2 PLLREF_IRQ[4]_ETRIG[2]_GPIO[208] — FMPLL Mode Selection / External Interrupt Request / eQADC Trigger Input / GPIO

PLLREF_IRQ[4]_ETRIG[2]_GPIO[208] is used during reset to configure the operating mode of the FMPLL. It has to be set to the desired value soon after power-on reset and kept stable during the whole reset cycle. After reset is negated, this pin is used for one of the alternate functions. The alternate function is an external interrupt request input. The second alternate function is the external trigger input for the eQADC.

3.3.1.3 BOOTCFG[1]_IRQ[3]_ETRIG[3]_GPIO[212] — Reset Configuration / External Interrupt Request / eQADC Trigger Input / GPIO

BOOTCFG[1]_IRQ[3]_ETRIG[3]_GPIO[212] are sampled on the negation of the $\overline{\text{RSTOUT}}$ pin. The values are used by the BAM program to determine the boot configuration of this device. The alternate function is an external interrupt request input. The second alternate function is the external trigger input for the eQADC.

3.3.1.4 WKPCFG_NMI_DSPI_B_SOUT_GPIO[213] — Weak Pull Configuration / GPIO

WKPCFG_NMI_DSPI_B_SOUT_GPIO[213] determines whether specified eTPU and eMIOS pins are connected to a weak pull up or weak pull down during and immediately after reset. The primary alternate function is the Non-Maskable Interrupt; the second is the DSPI Serial Data Output.

3.3.2 General Purpose I/O (GPIO)

3.3.2.1 GPIO[98:99] / GPIO[206:207]

These are General Purpose I/O pins.

3.3.3 Calibration External Bus Interface (EBI)

3.3.3.1 CAL_ADDR[12:15] — Calibration Address

CAL_ADDR[12:15] are the calibration address signals.

3.3.3.2 CAL_ADDR[16:27]_ALT_MDO[0:11] — Calibration Addr / Nexus Message Data Out

CAL_ADDR[16:27]_ALT_MDO[0:11] are the calibration address signals. The alternate function are nexus message data outputs.

3.3.3.3 CAL_ADDR[28:29]_ALT_$\overline{\text{MSEO}}$[0:1] — Calibration Address / Nexus Message Start/End Out

CAL_ADDR[28:29]_ALT_$\overline{\text{MSEO}}$[0:1] are the calibration address signals. The alternate function are Nexus message start/end out.

3.3.3.4 CAL_ADDR[30]_ALT_$\overline{\text{EVTI}}$ — Calibration Address / Nexus Event In

CAL_ADDR[30]_ALT_$\overline{\text{EVTI}}$ is the calibration address signal. The alternate function is Nexus event in.

3.3.3.5 CAL_$\overline{\text{CS}}$[2:3]_CAL_ADDR[10:11] — Calibration Chip Selects / Calibration Address

CAL_$\overline{\text{CS}}$[2:3]_CAL_ADDR[10:11] are the calibration chip select output signals. The alternate functions are calibration address signals.

3.3.3.6 CAL_$\overline{\text{CS}}$[0] — Calibration Chip Select

CAL_$\overline{\text{CS}}$[0] is the calibration chip select output signal.

3.3.3.7 CAL_DATA[0:15] — Calibration Data

CAL_DATA[0:15] are the calibration data signals.

3.3.3.8 CAL_$\overline{\text{OE}}$ — Calibration Output Enable

CAL_$\overline{\text{OE}}$ indicates that the calibration interface is ready to accept read data.

3.3.3.9 CAL_RD_$\overline{\text{WR}}$ — Calibration Read/Write

CAL_RD_$\overline{\text{WR}}$ indicates whether a calibration bus transfer is a read or write operation.

3.3.3.10 CAL_$\overline{\text{TS}}$_ALE — Calibration Transfer Start / Address Latch Enable

The Calibration Transfer Start signal CAL_$\overline{\text{TS}}$ is asserted by this device to indicate the start of a transfer. The Address Latch Enable (ALE) signal is used to demultiplex the address

from data bus. It is asserted while the least significant 16 bits of the address are present in the multiplexed address/data bus.

3.3.3.11 CAL_WE[0:1]_BE[0:1] — Calibration Write/Byte Enable

CAL_WE[0:1]_BE[0:1] specify which data pins contain valid data for a calibration bus transfer.

3.3.3.12 ALT_EVTO— Nexus Event Out

ALT_EVTO is an output that provides timing to a development tool for a single watchpoint or breakpoint occurrence.

3.3.3.13 ALT_MCKO— Nexus Message Clock Out

ALT_MCKO is a free running clock output to the development tools which is used for timing of the MDO and MSEO signals.

3.3.4 Nexus Port Controller (NPC)

3.3.4.1 NEXUSCFG — Nexus Configuration

NEXUSCFG is an input pin to select which function (nexus or cal_addr) is assigned to pad_cal_addr outputs.

3.3.4.2 EVTI_eTPU_A[2]_GPIO[231] — Nexus Event In / eTPU_A Channel (Output Only) / GPIO

EVTI is an input that is read on the negation of TRST to enable or disable the Nexus Debug port. After reset, the EVTI pin is used to initiate program and data trace synchronization messages or generate a breakpoint. The alternate functions are output channel for eTPU_A[2] module and GPIO[231].

3.3.4.3 EVTO _eTPU_A[4]_GPIO[227] — Nexus Event Out / eTPU_A Channel (Output Only) / GPIO

EVTO is an output that provides timing to a development tool for a single watchpoint or breakpoint occurrence. The alternate functions are output channel for eTPU_A[4] module and GPIO[227].

3.3.4.4 MCKO_GPIO[219] — Nexus Message Clock Out / GPIO

MCKO is a free running clock output to the development tools which is used for timing of the MDO and MSEO signals. The alternate functions is GPIO[219], when package QFP is selected.

3.3.4.5 MDO[0]_eTPU_A[13]_GPIO[220] — Nexus Message Data Out / eTPU_A Channel (Output Only) / GPIO

This a trace message output to the development tools. This pin also indicates the status of the crystal oscillator clock following a power-on reset, when MDO[0] is driven high until the crystal oscillator clock achieves stability and then is negated. The alternate functions are output channel for eTPU_A[13] module and GPIO[220].

3.3.4.6 MDO[1]_eTPU_A[19]_GPIO[221] — Nexus Message Data Out / eTPU_A Channel (Output Only) / GPIO

This is the trace message output to the development tools. The alternate functions are output channel for eTPU_A[19] module and GPIO[221].

3.3.4.7 MDO[2]_eTPU_A[21]_GPIO[222] — Nexus Message Data Out / eTPU_A Channel (Output Only) / GPIO

This is the trace message output to the development tools. The alternate functions are output channel for eTPU_A[21] module and GPIO[222].

3.3.4.8 MDO[3]_eTPU_A[25]_GPIO[223] — Nexus Message Data Out / eTPU_A Channel (Output Only) / GPIO

This is the trace message output to the development tools. The alternate functions are output channel for eTPU_A[25] module and GPIO[223].

3.3.4.9 $\overline{\text{MSEO}}$[0]_eTPU_A[27]_GPIO[224] — Nexus Message Start/End Out / eTPU_A Channel (Output Only) / GPIO

This is the output that indicate when messages start and end on the MDO pins. The alternate functions are output channel for eTPU_A[27] module and GPIO[224].

3.3.4.10 $\overline{\text{MSEO}}$[1]_eTPU_A[29]_GPIO[225] — Nexus Message Start/End Out / eTPU_A Channel (Output Only) / GPIO

This is the output that indicates when messages start and end on the MDO pins. The alternate functions are output channel for eTPU_A[29] module and GPIO[225].

3.3.5 JTAG

3.3.5.1 TCK — JTAG Test Clock Input

TCK provides the clock input for the on-chip test logic.

3.3.5.2 TDI_eMIOS[5]_GPIO[232] — JTAG Test Data Input

TDI provides the serial test instruction and data input for the on-chip test logic. The alternate functions are output channel for eMIOS[5] module and GPIO[232].

3.3.5.3 TDO_eMIOS[6]_GPIO[228] — JTAG Test Data Output

TDO provides the serial test data output for the on-chip test logic. The alternate functions are output channel for eMIOS[6] module and GPIO[228].

3.3.5.4 TMS — JTAG Test Mode Select Input

TMS controls test mode operations for the on-chip test logic.

3.3.5.5 JCOMP — JTAG Compliance Input

The JCOMP pin is used to enable the JTAG TAP controller.

3.3.6 FlexCAN

3.3.6.1 CAN_A_TX_SCI_A_TX_GPIO[83] — CAN_A Transmit / eSCI_A Transmit / GPIO

CAN_A_TX_SCI_A_TX_GPIO[83] is the transmit pin for the FlexCAN A module. The alternate function is the transmit pin for the eSCI A module.

3.3.6.2 CAN_A_RX_SCI_A_RX_GPIO[84] — CAN_A Receive / eSCI_A Receive / GPIO

CAN_A_RX_SCI_A_RX_GPIO[84] is the receive pin for the FlexCAN A module. The alternate function is the receive pin for the eSCI A module.

3.3.6.3 CAN_C_TX_GPIO[87] — CAN_C Transmit / - /GPIO

CAN_C_TX_GPIO[87] is the transmit pin for the FlexCAN C module. The first alternate function is not implemented.

3.3.6.4 CAN_C_RX_GPIO[88] — CAN_C Receive / - / GPIO

CAN_C_RX_GPIO[88] is the receive pin for the FlexCAN C module. The first alternate function is not implemented.

3.3.7 eSCI

3.3.7.1 SCI_A_TX_eMIOS[13]_GPIO[89] — eSCI_A Transmit / eMIOS Channel / GPIO

SCI_A_TX_eMIOS[13]_GPIO[89] is the transmit pin for the eSCI A module. Its alternate function is eMIOS[13] channel output pin.

3.3.7.2 SCI_A_RX_eMIOS[15]_GPIO[90] — eSCI_A Receive / eMIOS Channel / GPIO

SCI_A_RX_eMIOS[15]_GPIO[90] is the receive pin for the eSCI A module. Its alternate function is eMIOS[15] channel input and output pin.

3.3.7.3 SCI_B_TX_GPIO[91] — eSCI_B Transmit / GPIO

SCI_B_TX_GPIO[91] is the transmit pin for the eSCI B module.

3.3.7.4 SCI_B_RX_GPIO[92] — eSCI_B Transmit / - / GPIO

SCI_B_RX_GPIO[92] is the transmit pin for the eSCI B module. Its first alternate function is not implemented.

3.3.8 DSPI

3.3.8.1 DSPI_B_SCK_DSPI_C_PCS[1]_GPIO[102] — DSPI_B Clock / GPIO

DSPI_B_SCK_DSPI_C_PCS[1]_GPIO[102] is the SPI clock pin for the DSPI B module. The alternate function is a peripheral chip select output pin for the DSPI C module.

3.3.8.2 DSPI_B_SIN_DSPI_C_PCS[2]_GPIO[103] — DSPI_B Data Input / GPIO

DSPI_B_SIN_DSPI_C_PCS[2]_GPIO[103] is the data input pin for the DSPI B module. The alternate function is a peripheral chip select output pin for the DSPI C module.

3.3.8.3 DSPI_B_SOUT_DSPI_C_PCS[5]_GPIO[104] — DSPI_B Data Output /GPIO

DSPI_B_SOUT_DSPI_C_PCS[5]_GPIO[104] is the data output pin for the DSPI B module. The alternate function is a peripheral chip select output pin for the DSPI C module.

3.3.8.4 DSPI_B_PCS[0]_GPIO[105] — DSPI_B Chip Select / GPIO

DSPI_B_PCS[0]_GPIO[105] is a peripheral chip select output pin (slave select input pin for slave operation) for the DSPI B module. Its first alternate function is not implemented.

3.3.8.5 DSPI_B_PCS[1]_GPIO[106] — DSPI_B Chip Select / - / GPIO

DSPI_B_PCS[1]_GPIO[106] is a peripheral chip select output pin for the DSPI B module. Its first alternate function is not implemented.

3.3.8.6 DSPI_B_PCS[2]_DSPI_C_SOUT_GPIO[107] — DSPI_B Chip Select/GPIO

DSPI_B_PCS[2]_DSPI_C_SOUT_GPIO[107] is a peripheral chip select output pin for the DSPI B module. The alternate function is a data output pin for the DSPI C module.

3.3.8.7 DSPI_B_PCS[3]_DSPI_C_SIN_GPIO[108] — DSPI_B Chip Select/GPIO

DSPI_B_PCS[3]_DSPI_C_SIN_GPIO[108] is a peripheral chip select output pin for the DSPI B module. The alternate function is a data input pin for the DSPI C module.

3.3.8.8 DSPI_B_PCS[4]_DSPI_C_SCK_GPIO[109] — DSPI_B Chip Select/GPIO

DSPI_B_PCS[4]_DSPI_C_SCK_GPIO[109] is a peripheral chip select output pin for the DSPI B module. The alternate function is a clock output pin for the DSPI C module.

3.3.8.9 DSPI_B_PCS[5]_DSPI_C_PCS[0]_GPIO[110] — DSPI_B Chip Select/GPIO

DSPI_B_PCS[5]_DSPI_C_PCS[0]_GPIO[110] is a peripheral chip select output pin for the DSPI B module. The alternate function is a peripheral chip select output pin for the DSPI C module.

3.3.9 eQADC

3.3.9.1 AN[0]_DAN0+ — Analog Input / Differential Analog Input Positive Terminal

AN[0] is a single ended analog input pin. DAN0+ is the positive terminal input of the differential analog input DAN0.

3.3.9.2 AN[1]_DAN0- — Analog Input / Differential Analog Input Negative Terminal

AN[1] is a single ended analog input pin. DAN0- is the negative terminal input of the differential analog input DAN0.

3.3.9.3 AN[2]_DAN1+ — Analog Input / Differential Analog Input Positive Terminal

AN[2] is a single ended analog input pin. DAN1+ is the positive terminal input of the differential analog input DAN1.

3.3.9.4 AN[3]_DAN1- — Analog Input / Differential Analog Input Negative Terminal

AN[3] is a single ended analog input pin. DAN1- is the negative terminal input of the differential analog input DAN1.

3.3.9.5 AN[4]_DAN2+ — Analog Input / Differential Analog Input Positive Terminal

AN[4] is a single ended analog input pin. DAN2+ is the positive terminal input of the differential analog input DAN2.

3.3.9.6 AN[5]_DAN2- — Analog Input / Differential Analog Input Negative Terminal

AN[5] is a single ended analog input pin. DAN2- is the negative terminal input of the differential analog input DAN2.

3.3.9.7 AN[6]_DAN3+ — Analog Input / Differential Analog Input Positive Terminal

AN[6] is a single ended analog input pin. DAN3+ is the positive terminal input of the differential analog input DAN3.

3.3.9.8 AN[7]_DAN3- — Analog Input / Differential Analog Input Negative Terminal

AN[7] is a single ended analog input pin. DAN3- is the negative terminal input of the differential analog input DAN3.

3.3.9.9 AN[9]_ANX — Analog Input / External Multiplexed Analog Input

AN[9] is a single ended analog input pin. ANX is a single ended analog input to one of the on-chip ADCs in external multiplexed mode.

3.3.9.10 AN[11]_ANZ — Analog Input / External Multiplexed Analog Input

AN[11] is a single ended analog input pin. ANZ is a single ended analog input to one of the on-chip ADCs in external multiplexed mode.

3.3.9.11 AN[12]_MA[0]_eTPU_A[19]_$\overline{\text{SDS}}$ — Analog Input / MUX Address / eTPU_A Channel (Output Only) / Serial Data Strobe

AN[12]_MA[0]_eTPU_A[19]_$\overline{\text{SDS}}$ is a single ended analog input pin. The alternate function is a MUX address pin. The second alternate function is eTPU_A[19] channel output pin. The third alternate function is the serial data strobe for the eQADC SSI.

3.3.9.12 AN[13]_MA[1]_eTPU_A[21]_SDO — Analog Input / MUX Address / eTPU_A Channel (Output Only) / Serial Data Output

AN[13]_MA[1]_eTPU_A[21]_SDO is a single ended analog input pin. The alternate function is a MUX address pin. The second alternate function is eTPU_A[21] channel input/output pin. The third alternate function is the serial data output for the eQADC SSI.

3.3.9.13 AN[14]_MA[2]_eTPU_A[27]_SDI— Analog Input / MUX Address / eTPU_A Channel (Output Only) / Serial Data Input

AN[14]_MA[2]_eTPU_A[27]_SDI is a single ended analog input pin. The alternate function is a MUX address pin. The second alternate function is eTPU[27] channel output pin. The third alternate function is the serial data input for the eQADC SSI.

3.3.9.14 AN[15]_FCK_eTPU_A[29]— Analog Input / Free Running Clock / eTPU_A Channel (Output Only)

AN[15]_FCK_eTPU_A[29] is a single ended analog input pin. The first alternate function is the free running clock for the eQADC SSI. The second alternate function is the eTPU[29] channel output pin.

3.3.9.15 AN[16:18] — Analog Input

AN[16:18] are single ended analog input pins.

3.3.9.16 AN[21:25] — Analog Input

AN[21:25] are single ended analog input pins.

3.3.9.17 AN[27:28] — Analog Input

AN[27:28] are single ended analog input pins.

3.3.9.18 AN[30:37] — Analog Input

AN[30:37] are single ended analog input pins.

3.3.9.19 AN[38]_ANW_AN[8] — Analog Input / External Multiplexed Analog Input / Analog Input

AN[38] and AN[8] are single ended analog input pins. ANW is a single ended analog input to one of the on-chip ADCs in external multiplexed mode.

3.3.9.20 AN[39]_ANY_AN[10] — Analog Input / External Multiplexed Analog Input / Analog Input

AN[39] and AN[10] are single ended analog input pins. ANY is a single ended analog input to one of the on-chip ADCs in external multiplexed mode.

3.3.9.21 VRH — Voltage Reference High

VRH is the voltage reference high input pin for the eQADC.

3.3.9.22 VRL — Voltage Reference Low

VRL is the voltage reference low input pin for the eQADC.

3.3.9.23 REFBYPC — Bypass Capacitor

REFBYPC is the bypass capacitor input pin for the eQADC.

3.3.10 eTPU

3.3.10.1 eTPU_A[0]_eTPU_A[12]_eTPU_A[19]_GPIO[114] — eTPU_A Channel / eTPU_A Channel (Output Only) / eTPU_A Channel (Output Only) / GPIO

eTPU_A[0]_eTPU_A[12]_eTPU_A[19]_GPIO[114] is input/output channel pin for the eTPU_A module. The alternate function is the output channel pin for the eTPU_A.

3.3.10.2 eTPU_A[1:4]_eTPU_A[13:16]_GPIO[115:118] — eTPU_A Channel / eTPU_A Channel (Output Only) / GPIO

eTPU_A[1:4]_eTPU_A[13:16]_GPIO[115:118] are input/output channel pins for the eTPU_A module. The alternate functions are the output channel pins for the eTPU_A.

3.3.10.3 eTPU_A[5]_eTPU_A[17]_DSPI_B_SCK_LVDS-_GPIO[119] — eTPU_A Channel / eTPU_A Channel (Output Only) / DSPI_B_SCK_LVDS- / GPIO

eTPU_A[5]_eTPU_A[17]_DSPI_B_SCK_LVDS-_GPIO[119] is input/output channel pin for the eTPU_A module. The alternate function is the output channel pin for the eTPU_A, LVDS- output for DSPI B clock.

3.3.10.4 eTPU_A[6]_eTPU_A[18]_DSPI_B_SCK_LVDS+_GPIO[120] — eTPU_A Channel / eTPU_A Channel (Output Only) / DSPI_B_SCK_LVDS+ / GPIO

eTPU_A[6]_eTPU_A[18]_DSPI_B_SCK_LVDS+_GPIO[120] is input/output channel pin for the eTPU_A module. The alternate function is the output channel pin for the eTPU_A, LVDS+ output for DSPI B clock.

3.3.10.5 eTPU_A[7]_eTPU_A[19]_DSPI_B_SOUT_LVDS-_eTPU_A[6]_GPIO[121] — eTPU_A Channel / eTPU_A Channel (Output Only) / DSPI_B_SOUT_LVDS- / eTPU_A Channel /GPIO

eTPU_A[7]_eTPU_A[19]_DSPI_B_SOUT_LVDS-_eTPU_A[6]_GPIO[121] is input/output channel pin for the eTPU_A module. The alternate function is the output channel pin for the eTPU_A, LVDS- output for DSPI B chip select.

3.3.10.6 eTPU_A[8]_eTPU_A[20]_DSPI_B_SOUT_LVDS+_GPIO[122] — eTPU_A Channel / eTPU_A Channel (Output Only) / DSPI_B_SOUT LVDS+ / GPIO

eTPU_A[8]_eTPU_A[20]_DSPI_B_SOUT_LVDS+_GPIO[122] is input/output channel pin for the eTPU_A module. The alternate function is the output channel pin for the eTPU_A, LVDS+ output for DSPI B chip select.

3.3.10.7 eTPU_A[9:11]_eTPU_A[21:23]_GPIO[123:125] — eTPU_A Channel / eTPU_A Channel (Output Only) / GPIO

eTPU_A[9:11]_eTPU_A[21:23]_GPIO[123:125] are input/output channel pins for the eTPU_A module. The alternate functions are the output channel pins for the eTPU_A.

3.3.10.8 eTPU_A[12]_DSPI_B_PCS[1]_GPIO[126] — eTPU_A Channel / DSPI_B Chip Select /GPIO

eTPU_A[12]_DSPI_B_PCS[1]_GPIO[126] is an input/output channel pin for the eTPU_A module. The alternate function is a peripheral chip select for the DSPI B module.

3.3.10.9 eTPU_A[13]_DSPI_B_PCS[3]_GPIO[127] — eTPU_A Channel / DSPI_B Chip Select /GPIO

eTPU_A[13]_DSPI_B_PCS[3]_GPIO[127] is an input/output channel pin for the eTPU_A module. The alternate function is a peripheral chip select for the DSPI B module.

3.3.10.10 eTPU_A[14]_DSPI_B_PCS[4]_eTPU_A[9]_GPIO[128] — eTPU_A Channel / DSPI_B Chip Select / eTPU_A Channel (Output Only) / GPIO

eTPU_A[14]_DSPI_B_PCS[4]_eTPU_A[9]_GPIO[128] is an input/output channel pin for the eTPU_A module. The alternate function is a peripheral chip select for the DSPI B module, output channel pin for the eTPU_A.

3.3.10.11 eTPU_A[15]_DSPI_B_PCS[5]_GPIO[129] — eTPU_A Channel / DSPI_B Chip Select /GPIO

eTPU_A[15]_DSPI_B_PCS[5]_GPIO[129] is an input/output channel pin for the eTPU_A module. The alternate function is a peripheral chip select for the DSPI B module.

3.3.10.12 eTPU_A[16]_GPIO[130] — eTPU_A Channel / GPIO

eTPU_A[16]_GPIO[130] is an input/output channel pin for the eTPU_A module.

3.3.10.13 eTPU_A[17]_GPIO[131] — eTPU_A Channel / GPIO

eTPU_A[17]_GPIO[131] is an input/output channel pin for the eTPU_A module.

3.3.10.14 eTPU_A[18]_GPIO[132] — eTPU_A Channel / GPIO

eTPU_A[18]_GPIO[132] is an input/output channel pin for the eTPU_A module.

3.3.10.15 eTPU_A[19]_GPIO[133] — eTPU_A Channel / GPIO

eTPU_A[19]_GPIO[133] is an input/output channel pin for the eTPU_A module.

3.3.10.16 eTPU_A[20:21]_IRQ[8:9]_GPIO[134:135] — eTPU_A Channel / External Interrupt / GPIO

eTPU_A[20:21]_IRQ[8:9]_GPIO[134:135] are input/output channel pins for the eTPU_A module. The alternate functions are external interrupt request inputs for the SIU module.

3.3.10.17 eTPU_A[22]_IRQ[10]_eTPU_A[17]_GPIO[136] — eTPU_A Channel / External Interrupt / eTPU_A Channel (Output Only) / GPIO

eTPU_A[22]_IRQ[10]_eTPU_A[17]_GPIO[136] is input/output channel pin for the eTPU_A module. The alternate function is external interrupt request inputs for the SIU module, output channel pin for the eTPU_A[22].

3.3.10.18 eTPU_A[23]_IRQ[11]_eTPU_A[21]_GPIO[137] — eTPU_A Channel / External Interrupt / eTPU_A Channel (Output Only) / GPIO

eTPU_A[23]_IRQ[11]_eTPU_A[21]_GPIO[137] is input/output channel pin for the eTPU_A module. The alternate function is external interrupt request inputs for the SIU module, output channel pin for the eTPU_A[21].

3.3.10.19 eTPU_A[24]_IRQ[12]_DSPI_C_SCK_LVDS-_GPIO[138] — eTPU_A Channel (Output Only) / External Interrupt / eTPU_A Channel / DSPI_C_SCK_LVDS- / GPIO

eTPU_A[24]_IRQ[12]_DSPI_C_SCK_LVDS-_GPIO[138] is input/output channel pin for the eTPU_A module. The alternate function is external interrupt request inputs for the SIU module, LVDS- output for DSPI C clock.

3.3.10.20 eTPU_A[25]_IRQ[13]_DSPI_C_SCK_LVDS+_GPIO[139] — eTPU_A Channel (Output Only) / External Interrupt / eTPU_A Channel / DSPI_C_SCK_LVDS+ / GPIO

eTPU_A[25]_IRQ[13]_DSPI_C_SCK_LVDS+_GPIO[139] is input/output channel pin for the eTPU_A module. The alternate function is external interrupt request inputs for the SIU module, LVDS+ output for DSPI C clock.

3.3.10.21 eTPU_A[26]_IRQ[14]_DSPI_C_SOUT_LVDS-_GPIO[140] — eTPU_A Channel (Output Only) / External Interrupt / eTPU_A Channel / DSPI_C_SOUT_LVDS- / GPIO

eTPU_A[26]_IRQ[14]_DSPI_C_SOUT_LVDS-_GPIO[139] is input/output channel pin for the eTPU_A module. The alternate function is external interrupt request input for the SIU module, LVDS- output for DSPI C chip select.

3.3.10.22 eTPU_A[27]_IRQ[15]_DSPI_C_SOUT_LVDS_DSPI_B_SOUT_GPIO[141] — eTPU_A Channel (Output Only) / External Interrupt / eTPU_A Channel / DSPI_C_SOUT_LVDS+ / GPIO

eTPU_A[27]_IRQ[15]_DSPI_C_SOUT_LVDS_DSPI_B_SOUT_GPIO[141] is input/output channel pin for the eTPU_A module. The alternate function is the external interrupt request input for the SIU module, LVDS+ output for DSPI C chip select.

3.3.10.23 eTPU_A[28]_DSPI_C_PCS[1]_GPIO[142] — eTPU_A Channel / DSPI_C Chip Select / GPIO

eTPU_A[28]_DSPI_C_PCS[1]_GPIO[142] is an input/output channel pin for the eTPU_A module. The alternate function is a peripheral chip select for the DSPI C module.

3.3.10.24 eTPU_A[29]_DSPI_C_PCS[2]_GPIO[143] — eTPU_A Channel / DSPI_C Chip Select / GPIO

eTPU_A[29]_DSPI_C_PCS[2]_GPIO[143] is an input/output channel pin for the eTPU_A module. The alternate function is a peripheral chip select for the DSPI C module.

3.3.10.25 eTPU_A[30]_DSPI_C_PCS[3]_eTPU_A[11]_GPIO[144] — eTPU_A Channel / DSPI_C Chip Select / eTPU_A Channel (Output Only) / GPIO

eTPU_A[30]_DSPI_C_PCS[3]_eTPU_A[11]_GPIO[144] is an input/output channel pin for the eTPU_A module. The alternate function is a peripheral chip select for the DSPI C module, output channel pin for the eTPU_A[11].

3.3.10.26 eTPU_A[31]_DSPI_C_PCS[4]_eTPU_A[13]_GPIO[145] — eTPU_A Channel / DSPI_C Chip Select / eTPU_A Channel (Output Only) / GPIO

eTPU_A[31]_DSPI_C_PCS[4]_GPIO[145] is an input/output channel pin for the eTPU_A module. The alternate function is a peripheral chip select for the DSPI C module, output channel pin for the eTPU_A[13].

3.3.11 eMIOS

3.3.11.1 eMIOS[0]_eTPU_A[0]_eTPU_A[25]_GPIO[179] — eMIOS Channel / eTPU Channel (Output Only) /eTPU Channel (Output Only)/ GPIO

eMIOS[0]_eTPU_A[0]_eTPU_A[25]_GPIO[179] is an eMIOS channel input and output pin. Alternate functions are eTPU[0,25] channels output pins.

3.3.11.2 eMIOS[1]_eTPU_A[1]_GPIO[180] — eMIOS Channel / eTPU_A Channel (Output Only) / GPIO

eMIOS[1]_eTPU_A[1]_GPIO[180] is an eMIOS channel input and output pin. Alternate function is an eTPU[1] channel output pin.

3.3.11.3 eMIOS[2]_eTPU_A[2]_GPIO[181] — eMIOS Channel / eTPU_A Channel (Output Only) / GPIO

eMIOS[2]_eTPU_A[2]_GPIO[181] is an eMIOS channel input and output pin. Alternate function is an eTPU[2] channel output pin.

3.3.11.4 eMIOS[4]_eTPU_A[4]_GPIO[183] — eMIOS Channel / eTPU_A Channel (Output Only) / GPIO

eMIOS[4]_eTPU_A[4]_GPIO[183] is an eMIOS channel input and output pin. Alternate function is an eTPU[4] channel output pin.

3.3.11.5 eMIOS[8]_eTPU_A[8]_SCI_B_TX_GPIO[187] — eMIOS Channel / eTPU_A Channel (Output Only) / eSCI Transmit / GPIO

eMIOS[8]_eTPU_A[8]_SCI_B_TX_GPIO[187] is an eMIOS channel input and output pin. Primary alternate function is the eTPU[8] channels output pins, second alternate is the eSCI transmit channel.

3.3.11.6 eMIOS[9]_eTPU_A[9]_SCI_B_RX_GPIO[188] — eMIOS Channel / eTPU_A Channel (Output Only) / eSCI Receive / GPIO

eMIOS[8]_eTPU_A[9]_SCI_B_RX_GPIO[188] is an eMIOS channel input and output pin. Primary alternate function is the eTPU[9] channels output pin, second alternate is the eSCI receive channel.

3.3.11.7 eMIOS[10]_GPIO[189] — eMIOS Channel / GPIO

eMIOS[10]_GPIO[189] are eMIOS[10] channel input and output pin.

3.3.11.8 eMIOS[11]_GPIO[190] — eMIOS Channel / GPIO

eMIOS[11]_GPIO[190] are eMIOS[11] channel input and output pins.

3.3.11.9 eMIOS[12]_DSPI_C_SOUT_eTPU_A[27]_GPIO[191] — eMIOS Channel / DSPI C Data Output / eTPU_A Channel (Output Only) / GPIO

eMIOS[12]_DSPI_C_SOUT_eTPU_A[[27]_GPIO[191] is eMIOS[12] channel output pin. The alternate functions are the data output for the DSPI C module, eTPU[27] channel output pin and GPIO[191].

3.3.11.10 eMIOS[13]_GPIO[191] — eMIOS Channel (Input/Output) / GPIO

eMIOS[13]_GPIO[191] is the eMIOS[13] channel input/output pin. The alternate function is not defined.

3.3.11.11 eMIOS[14]_IRQ[0]_eTPU_A[29]_GPIO[193] — eMIOS Channel (Input/Output) / External Interrupt / eTPU_A Channel (Output Only) / GPIO

eMIOS[14]_IRQ[0]_eTPU_A[29]_GPIO[193] is an eMIOS[14] channel input/output pin. The alternate function is external interrupt request input for the SIU module, output channel pin for the eTPU_A[29] and GPIO[193].

3.3.11.12 eMIOS[15]_IRQ[1]_GPIO[194] — eMIOS Channel (Input/Output) / External Interrupt / eTPU_A Channel (Output Only) / GPIO

eMIOS[15]_IRQ[1]_GPIO[194] is the eMIOS[15] channel input/output pin. The alternate function is external interrupt request input for the SIU module.

3.3.11.13 eMIOS[23]_GPIO[202] — eMIOS Channel

eMIOS[23]_GPIO[202] is an eMIOS[23] channel input and output pin.

3.3.12 Clock Synthesizer

3.3.12.1 XTAL — Crystal Oscillator Output

XTAL is the output pin for an external crystal oscillator.

3.3.12.2 EXTAL_EXTCLK — Crystal Oscillator/External Clock Input

EXTAL is the input pin for an external crystal oscillator or an external clock source. The alternate function is the external clock input. The function of this pin is determined by the state of the PLLREF pin during reset.

3.3.12.3 CLKOUT — System Clock Output

CLKOUT is the SPC563Mxx clock output for the calibration external bus interface.

3.3.13 Power / Ground

3.3.13.1 VDDREG — Voltage Regulator Supply

VDDREG is the 5 V voltage regulator supply.

3.3.13.2 VDDPLL - PLL Supply Voltage Input

VDDPLL is the 1.2 V power supply input pin for the FMPLL.

3.3.13.3 VSSPLL - PLL GROUND

VSSPLL is the Ground reference for the FMPLL.

3.3.13.4 VSTBY — Standby RAM Power Supply Input

VSTBY is the supply input pin for standby RAM. It has two ranges:

- Unregulated mode: 0.95–1.2 V
- Regulated mode: 2.0–5.5 V

3.3.13.5 VRC33 — Voltage Regulator Control Bypass Capacitor

VRC33 is the input pin for the bypass capacitor of the 3.3 V voltage regulator.

3.3.13.6 VRCCTL — Voltage Regulator Control Output

VRCCTL is the output pin for the on-chip 1.2 V regulator control circuit.

3.3.13.7 VDDA0/1 — Voltage Reference High

VDDA0/1 are the analog supply input pins for the eQADC.

3.3.13.8 VSSA0/1 — Ground Reference

VSSA0/1 are the analog ground reference input pins for the eQADC.

3.3.13.9 VDDREG — Voltage Regulator Supply

VDDREG is the 5 V voltage regulator supply.

3.3.13.10 VDD — Internal Logic Supply Input

VDD is the 1.2 V logic supply input.

3.3.13.11 VDDEH1a/b — I/O Supply Input

VDDEH1a/b are the 3.3 V to 5.0 V +/- 5% supply input pins to the I/O segment 1.

3.3.13.12 VDDEH6a/b — I/O Supply Input

VDDEH6a/b are the 3.3 V to 5.0 V +/- 5% supply input pins to the I/O segment 6.

3.3.13.13 VDDEH7 — I/O Supply Input

VDDEH7 is the 3.3 V to 5.0 V +/- 5% supply input pin to the I/O segment 7. It is only used on the 144- and 496-pin packages.

3.3.13.14 VDDE12a/b/c/d/e — I/O Supply Input

VDDE12 is the 1.8V to 3.3 V +/- 5% I/O supply input pin to the I/O segment 12.

In the 176-pin package, VDDE12 is the 3.3 V +/-5% I/O supply input pin to the alternate Nexus pins.

In the calibration package, VDDE12 is the 3.3 V +/-5% I/O supply input pin to the I/O segment 12.

3.3.13.15 VSS — Ground

VSS is the ground reference input pin.

3.4 I/O Power/Ground Segmentation

Table 10 gives the power/ground segmentation of the SPC563Mx MCU. Each segment provides the power and ground for the given set of I/O pins, and can be powered by any of the allowed voltages regardless of the power on the other segments.

Table 10. SPC563Mx Power/Ground Segmentation

Power Segment	144-LQFP Pin Number	176-LQFP Pin Number	Voltage Range(1)	I/O Pins Powered by Segment
VDDA0	6	6	5.0 V	AN[0:7], AN[9], AN[11], AN[16:18], AN[21:25], AN[27:28], AN[30:37], AN38, AN39, VRL, REFBYPC, VRH
VDDEH1 (a,b)	24, 34	31, 41	3.3 V – 5.0 V	PCKCFG[2], eTPU_A[0:31], eMIOS[0:2]
VDDEH6 (a, b)	78, 93	95, 110	3.3 V – 5.0 V	RESET, RSTOUT, WKPCFG, BOOTCFG1, PLLREF, SCK_B, PCKCFG[0], CAN_A_TX, CAN_A_RX, CAN_C_TX, CAN_C_RX, SCI_A_TX, SCI_A_RX, SCI_B_TX, SCI_B_RX, SCK_B, SIN_B, SOUT_B, DSPI_B_PCS_B[0:5], eMIOS[4], eMIOS[8:15], eMIOS[23], XTAL, EXTAL
VDDEH7(2)	102, 113	125, 138	3.3 V – 5.0 V	PCKCFG[1], MDO[0:3], EVTI, EVTO, MCKO, MSEO[0:1], TDO, TDI, TMS, TCK, JCOMP, AN[12:15] (GPIO[98:99], GPIO[206:207])
VDDE12(3)	—	16, 119	1.8 V – 3.3 V	CAL_ADDR[12:30], CAL_DATA[0:15], CAL_CS[0], CAL_CS[2:3], CAL_RD_WR, CAL_WE[0:1], CAL_OE, CAL_TS, ALT_MCKO, ALT_EVTO, NEXUSCFG

1. These are nominal voltages. All VDDE and VDDEH voltages are –5 %, +10 % (VDDE 1.62 V to 3.6 V, VDDEH 3.0 V to 5.5 V). VDDA is +5 %, –10 %.
2. The VDDEH7 segment may be powered from 3.0 V to 5.0 V for mux address or SSI functions, but must meet the VDDA specifications of 4.5 V to 5.25 V for analog input function.
3. In the calibration package this signal is named VDDE12.

3.5 DSPI Connections to eTPU_A, eMIOS and SIU

3.5.1 DSPI_B Connectivity

DSPI_B connections are illustrated in *Figure 4*. The full list of connections is given in *Table 11*.

Figure 4. DSPI_B Connectivity

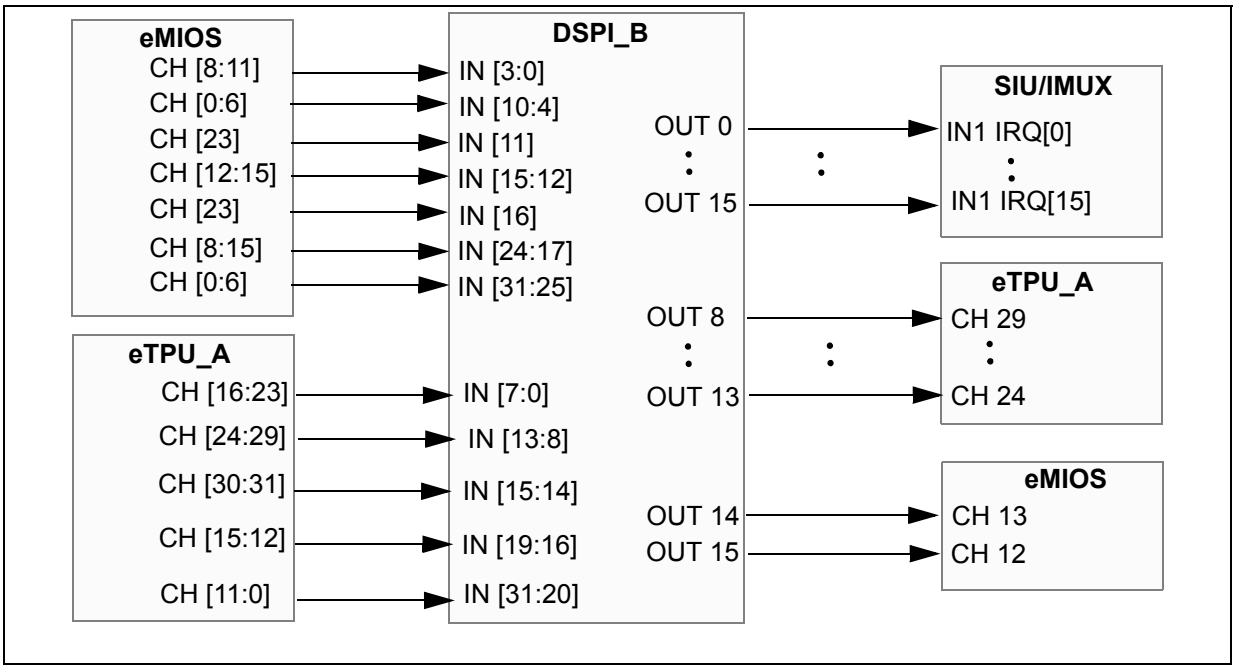


Table 11. DSPI_B I/O Mapping

DSPI_B Serialized Inputs / Outputs	eTPU_A Channel Output (Primary Function	eTPU_A Channel Input	eMIOS Channel Output (Alternate Function)	GPIO Output	Deserialization Destination
31	11	—	0	381	—
30	10	—	1	380	—
29	9	—	2	379	—
28	8	—	3	378	—
27	7	—	4	377	—
26	6	—	5	376	—
25	5	—	6	375	—
24	4	—	8	374	—
23	3	—	9	373	—
22	2	—	10	372	—
21	1	—	11	371	—
20	0	—	12	370	—

Table 11. DSPI_B I/O Mapping (continued)

DSPI_B Serialized Inputs / Outputs	eTPU_A Channel Output (Primary Function	eTPU_A Channel Input	eMIOS Channel Output (Alternate Function)	GPIO Output	Deserialization Destination
19	15	—	13	369	—
18	14	—	14	368	—
17	13	—	15	367	—
16	12	—	23	366	—
15	30	—	12	365	EMIOS_12 IRQ_15 (mux)
14	31	—	13	364	EMIOS_13 IRQ_14 (mux)
13	24	24	14	363	ETPU_A_24 IRQ_13 (mux)
12	25	25	15	362	ETPU_A_25 IRQ_12 (mux)
11	26	26	23	361	ETPU_A_26 IRQ_11 (mux)
10	27	27	0	360	ETPU_A_27 IRQ_10 (mux)
9	28	28	1	359	ETPU_A_28 IRQ_9 (mux)
8	29	29	2	358	ETPU_A_29 IRQ_8 (mux)
7	16	—	3	357	IRQ7 (mux)
6	17	—	4	356	IRQ6 (mux)
5	18	—	5	355	IRQ5 (mux)
4	19	—	6	354	IRQ4 (mux)
3	20	—	8	353	IRQ3 (mux)
2	21	—	9	352	IRQ2 (mux)
1	22	—	10	351	IRQ1 (mux)
0	23	—	11	350	IRQ0 (mux)

3.5.2 DSPI_C Connectivity

DSPI_B connections are illustrated in *Figure 5*. The full list of connections is given in *Table 12*.

Figure 5. DSPI_C Connectivity

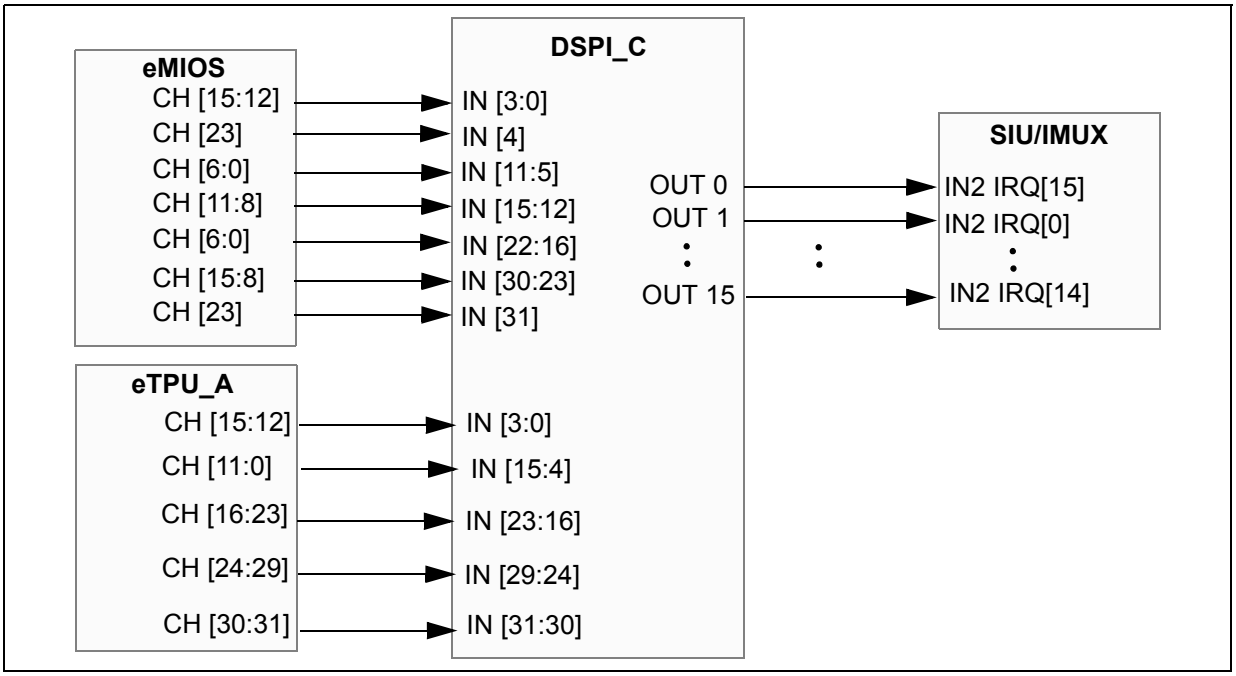


Table 12. DSPI_C I/O Mapping

DSPI_C Serialized Input	eTPU_A Channel Output (Primary Function)	eMIOS Channel Output (Alternate Function)	GPIO Output	Deserialization Destination
31	30	23	413	—
30	31	15	412	—
29	24	14	411	—
28	25	13	410	—
27	26	12	409	—
26	27	11	408	—
25	28	10	407	—
24	29	9	406	—
23	16	8	405	—
22	17	6	404	—
21	18	5	403	—
20	19	4	402	—
19	20	3	401	—
18	21	2	400	—
17	22	1	399	—
16	23	0	398	—
15	11	11	397	IRQ14 (mux)

Table 12. DSPI_C I/O Mapping (continued)

DSPI_C Serialized Input	eTPU_A Channel Output (Primary Function)	eMIOS Channel Output (Alternate Function)	GPIO Output	Deserialization Destination
14	10	10	396	IRQ13 (mux)
13	9	9	395	IRQ12 (mux)
12	8	8	394	IRQ11 (mux)
11	7	6	393	IRQ10 (mux)
10	6	5	392	IRQ9 (mux)
9	5	4	391	IRQ8 (mux)
8	4	3	390	IRQ7 (mux)
7	3	2	389	IRQ6 (mux)
6	2	1	388	IRQ5 (mux)
5	1	0	387	IRQ4 (mux)
4	0	23	386	IRQ3 (mux)
3	15	15	385	IRQ2 (mux)
2	14	14	384	IRQ1 (mux)
1	13	13	383	IRQ0 (mux)
0	12	12	382	IRQ15 (mux)

4 Resets

4.1 Reset sources

This device supports the following reset sources:

- Power-on Reset (POR)
- External Reset (ER)
- Loss of Lock Reset (LLR)
- Loss of Clock Reset (LCR)
- Watchdog Timer/Debug Reset (WTDR)
- JTAG Reset
- Checkstop Reset (CR)
- Software System Reset (SSR)
- Software External Reset (SER)

All reset sources are processed by the reset controller, which monitors the reset input sources, and upon detection of a reset event, resets internal logic and controls the assertion of the RSTOUT pin. The Software External Reset only causes the RSTOUT pin to be asserted for a number of clock cycles determined by the configuration of the PLL (refer to *Section 4.3.2: RSTOUT*), and does not reset the device.

For all reset sources, the device FMPLL is configured according to the value on the PLLREF pin at the negation of the RSTOUT pin. In addition, the FMPLL defaults to bypass mode, with the system clock being supplied directly by the clock reference dictated by the PLLREF pin.

The Reset Status Register (SIU_RSR) gives the source, or sources, of the last reset and indicates whether a glitch has occurred on the RESET pin. The SIU_RSR is updated for all reset sources except JTAG reset.

All reset sources initiate execution of the Boot Assist Module (BAM) program with the exception of the Software External Reset.

The Reset Configuration Half Word (RCHW) determines the MCU configuration after reset. The RCHW is stored in internal flash, or a default configuration is used. During reset, the RCHW is read from internal flash memory. The BOOTCFG pin is defined in *Chapter 15: System Integration Unit (SIU).* The BAM program reads the value of the BOOTCFG pin from the BOOTCFG field of the SIU_RSR, then reads the RCHW from the specified location, and then uses the RCHW value to determine and execute the specified boot procedure.

Note: The reset controller latches the value on the BOOTCFG input to the SIU 4 clock cycles prior to the negation of RSTOUT.

4.2 Reset vector

The reset vector for this device is 0xFFFF_FFFC. This is a fixed location in the BAM. The BAM program executes after every internal reset. The BAM program determines where to branch after its execution completes based on the value on the BOOTCFG pin. See *Section 20.5: Functional description* for details on the BAM program operation and branch location to application software.

4.3 Reset pins

4.3.1 RESET

The $\overline{\text{RESET}}$ pin is an active low input. The $\overline{\text{RESET}}$ pin is asserted by an external device during a power-on or external reset. The internal reset signal asserts only if the $\overline{\text{RESET}}$ pin asserts for 10 clock cycles. Assertion of the $\overline{\text{RESET}}$ pin while the device is in reset causes the reset cycle to start over. The $\overline{\text{RESET}}$ pin has a glitch detector which detects spikes greater than 2 clocks in duration that fall below the switch point of the input buffer logic of the VDDEH input pins. The switch point lies between the maximum VIL and minimum VIH specifications for the VDDEH input pins.

4.3.2 RSTOUT

The $\overline{\text{RSTOUT}}$ pin is an active low output that uses a push/pull configuration. The $\overline{\text{RSTOUT}}$ pin is driven to the low state by the MCU for all internal and external reset sources.

Depending on the PLL configuration, External Reference or Crystal Mode, the $\overline{\text{RSTOUT}}$ pin is asserted after a delay defined in *Table 13*, plus 4 cycles for sampling of the configuration pins.

The $\overline{\text{RSTOUT}}$ pin can also be asserted by a write to the SER bit of the System Reset Control Register (SIU_SRCR). Asserting SER, the $\overline{\text{RSTOUT}}$ duration will follow the value specified in *Table 13*.

Table 13. Timing for reset sources

Reset source	Description	Number of clocks	
		Crystal reference	External reference
POR	Power On Reset	2400	16000
ER	External Reset ($\overline{\text{RESET}}$ pin)	2900	16500
LLR	Loss of Lock Reset	3400	17000
WTR	Watchdog Timer (core) or Debug Reset	3900	17500
CR	Checkstop Reset	4400	18000
SWTR	System Software Watchdog Reset	4900	18500
LCR	Loss of Clock Reset	5400	19000
SSR	SIU Software External Reset	5900	19500
SER	SIU Software System Reset	6400	20000

4.4 Clock quality monitor gating signal

When this device is operating with the crystal oscillator as clock reference, the Clock Quality Monitor module is responsible for keeping reset asserted until the crystal clock is perceived to be of good quality. The time it takes for the crystal oscillator to stabilize is in addition to those indicated in *Table 13*.

4.5 Reset source descriptions

For the following reset source descriptions refer to the reset flow diagrams in *Figure 6* and *Figure 7*. *Figure 6* shows the reset flow for assertion of the RESET pin. *Figure 7* shows the internal processing of reset for all reset sources.

Figure 6. External reset flow diagram

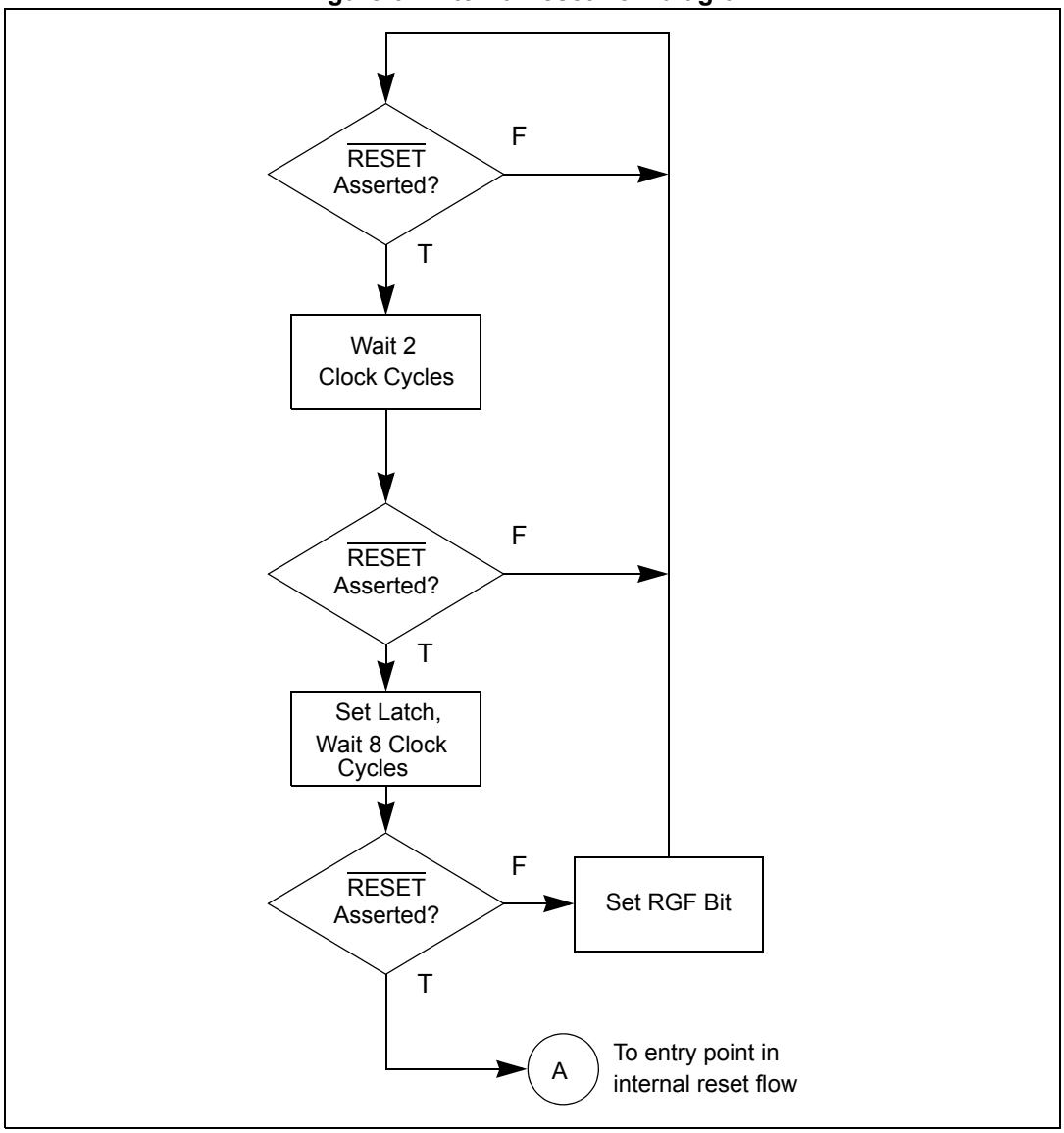
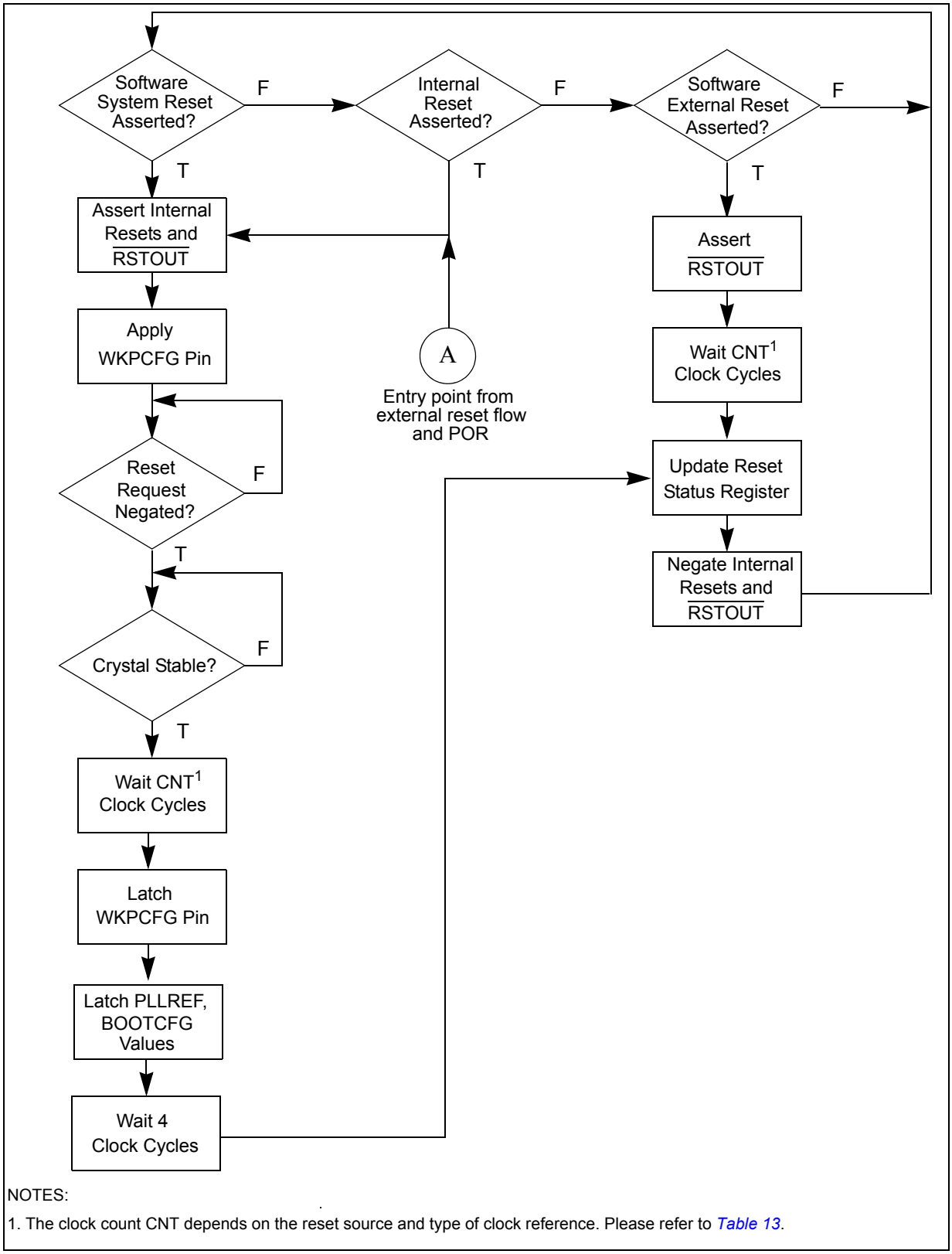


Figure 7. Internal reset flow diagram



NOTES:

1. The clock count CNT depends on the reset source and type of clock reference. Please refer to *Table 13*.

4.5.1 Power-on reset

The internal power-on reset signal is asserted when either the supply voltages, nominally 3.3 V or 1.2 V or the RESET supply (VDDEH6a) fall below defined values. See the device data sheet for the threshold specifications of these voltages. The output signals from the power-on reset circuits are active low signals. All power-on reset output signals are combined into one POR signal at the 1.2 V level and input to the reset controller. Although assertion of the power-on reset signal causes reset, the RESET pin must be asserted during a power-on reset to guarantee proper operation of the MCU.

The PLLREF pin determines the source of reference clock, either crystal or external, at the negation of RSTOUT. During the assertion of RSTOUT, the system clock will switch to the input specified by the PLLREF pin. The value on the PLLREF pin must be kept constant during reset to avoid transients in the system clock. See *Section 16.2.3: Modes of operation* for more details.

The signal on the WKPCFG pin determines whether weak pull up or pull down devices are enabled after reset on the eTPU and eMIOS pins. The WKPCFG pin is applied on the assertion of the internal reset signal (assertion of RSTOUT). See *Section 4.7.3: Reset weak pull up/down configuration* for more information.

Once a power-on-reset is triggered, if the clock reference is the crystal (PLLREF=1), then the clock to the whole chip, including the reset state machine, is kept frozen until the Clock Quality Monitor detects that the crystal oscillator has already stabilized. If the clock reference is external (PLLREF=0) the clock is released to the system immediately. When the clock is stable and released to the chip, the reset controller counts a predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*) before negating the RSTOUT pin. The WKPCFG and BOOTCFG pins are sampled 4 clock cycles before the negation of RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, the PORS and ERS bits are set, and all other reset status bits are cleared in the Reset Status Register.

4.5.2 External reset

When the reset controller detects assertion of the RESET pin, the internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. Once the RESET pin is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*). When the clock count finishes, the WKPCFG and BOOTCFG pins are sampled. The reset controller then waits 4 clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, the ERS bit is set, and all other reset status bits in the SIU_RSR are cleared.

4.5.3 Loss of lock

A Loss of Lock Reset occurs when the FMPLL loses lock and the Loss of Lock Reset Enable (LOLRE) bit in the FMPLL Synthesizer Control Register (SYNCR) is set. The internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. Once the FMPLL Loss of Lock reset request signal is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*). Once the clock count finishes, the WKPCFG and BOOTCFG pins are sampled. The reset controller then waits 4 clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, the LLRS bit is set, and all other reset

status bits in the SIU_RSR are cleared. Refer to *Section 16.5.3: Lock detection* for more information on loss of lock.

4.5.4 Loss of clock

A Loss of Clock Reset occurs when the Clock Quality Monitor Module (CQM) detects a failure in either the reference signal or FMPLL output, and the Loss of Clock Reset Enable (LOCRE) bit in the SYNCR is set. The internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. Once the Loss of Clock reset request signals is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*). Once the clock count finishes, the WKPCFG and BOOTCFG pins are sampled. The reset controller then waits 4 clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, the LCRS bit is set, and all other reset status bits in the SIU_RSR are cleared. Refer to *Section 16.5.3: Lock detection* for more information on loss of clock.

The CQM module when enabled can generates either a system reset or an interrupt signal, refer to *Section 16.5.4: Loss-of-clock detection* for details.

4.5.5 Watchdog timer/debug reset

A Watchdog Timer Reset occurs when the e200z335 core Watchdog Timer is enabled, and a timeout occurs with the Enable Next Watchdog Timer (EWT) and Watchdog Timer Interrupt Status (WIS) bits set in the Timer Status Register, and with the Watchdog Reset Control (WRC) field in the Timer Control Register configured for a reset. The WDRS bit in the SIU_RSR is also set when a debug reset command is issued from a debug tool. To determine whether the WDRS bit was set due to a Watchdog Timer or Debug Reset, see the WRS field in the e200z335 core Timer Status Register. The effect of a Watchdog Timer or Debug Reset request is the same for the reset controller. The internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. After the Watchdog Timer/Debug reset request is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*). Once the clock count finishes the WKPCFG and BOOTCFG pins are sampled. The reset controller then waits 4 clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, the WTRS bit is set, and all other reset status bits in the SIU_RSR are cleared. Refer to the *e200z335 Core Reference Manual* for more information on the Watchdog Timer and debug operation.

Note: In addition to the e200z335 watchdog timer, this device implements a system software watchdog timer (see Chapter 19: Software Watchdog Timer (SWT)).

4.5.6 Software watchdog timer reset

A Software Watchdog Timer Reset occurs when the watchdog timer in the SWT module is enabled and programmed to generate a reset. The effect of a Software Watchdog Timer Reset request is the same for the reset controller. The internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. Once the Software Watchdog Timer reset request is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*). When the clock count finishes the WKPCFG and BOOTCFG pins are sampled. The reset controller then waits 4 clock cycles before negating

RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, the SWTRS bit is set, and all other reset status bits in the SIU_RSR are cleared.

4.5.7 Checkstop reset

When the e200z335 core enters a checkstop state, and the Checkstop Reset is enabled (the CRE bit in the System Reset Control Register (SIU_SRCR) is set), a Checkstop Reset occurs. The internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. After the checkstop state signal is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*). Once the clock count finishes the WKPCFG and BOOTCFG pins are sampled. The reset controller then waits 4 clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, the CRS bit is set, and all other reset status bits in the SIU_RSR are cleared. Refer to the *e200z335 Core Reference Manual* for more information.

4.5.8 JTAG reset

A system reset occurs when JTAG is enabled and either the EXTEST, CLAMP, or HIGHZ instructions are executed by the JTAG controller. The internal reset signal is asserted. The state of the RSTOUT pin is determined by the JTAG instruction. The value on the WKPCFG pin is applied at the assertion of the internal reset signal, as is the PLLREF value. After the JTAG reset request is negated, the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*). Once the clock count finishes the WKPCFG and BOOTCFG pins are sampled, and the associated bits/fields are updated in the SIU_RSR. The reset status bits in the SIU_RSR are unaffected. Refer to *Chapter 30: JTAG Controller (JTAGC)* for more information.

4.5.9 Software system reset

A Software System Reset is caused by a write to the SSR bit in the System Reset Control Register (SIU_SRCR); see *Section 15.8.4: System Reset Control Register (SIU_SRCR)*. A write of one to the SSR bit causes an internal reset of the MCU. The internal reset signal and RSTOUT pin are asserted. The value on the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), as is the PLLREF value. The SSR bit is automatically cleared and the reset controller waits for a predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*). Once the clock count finishes the WKPCFG and BOOTCFG pins are sampled. The reset controller then waits 4 clock cycles before negating RSTOUT, and the associated bits/fields are updated in the SIU_RSR. In addition, the SSRS bit is set, and all other reset status bits in the SIU_RSR are cleared.

4.5.10 Software external reset

A write of one to the SER bit in the SIU_SRCR causes the external RSTOUT pin to be asserted for a predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*). The SER bit automatically clears after the clock counting expires. A Software External Reset does not cause a reset of the MCU, the BAM program is not executed, the PLLREF, BOOTCFG, and WKPCFG pins are not sampled. The SERF bit in the SIU_RSR is set, but no other status bits are affected. The SERF bit in the SIU_RSR is not automatically cleared and remains set until cleared by software or another reset besides the Software External Reset occurs.

For a Software External Reset, the e200z335 core will continue to execute instructions, timers that are enabled will continue to operate, and interrupt requests will continue to be processed. It is the responsibility of the application to ensure devices connected to RSTOUT are not accessed during a Software External Reset, and to determine how to manage MCU resources when using the Software External Reset.

4.6 Reset registers in the SIU

The System Integration Unit (SIU) on this device includes two registers, SIU_RSR and SIU_SRCR, that affect the reset behavior of this device. See *Chapter 15: System Integration Unit (SIU)* for descriptions of these registers.

4.7 Reset configuration

4.7.1 Reset configuration half word (RCHW)

4.7.1.1 RCHW overview

The Reset Configuration Half Word (RCHW) is a collection of control bits from various internal registers that specifies a minimal MCU configuration after reset. The RCHW also contains bits that control the BAM program flow.

The RCHW is read from internal memory. If a valid RCHW is not found, a CAN/eSCI boot is initiated. The BAM program writes the configuration control bits to the appropriate registers and uses the remaining bits for flow control of its program. Refer to the appropriate register given by the RCHW bit descriptions for a detailed description of each control bit.

The RCHW is read and applied each time the BAM program executes, which is for every power-on, external, or internal reset event. The only exception to this is the Software External Reset. The RCHW is read from the following location:

- Address 0x0000_0000 of internal flash memory device

At the negation of the RSTOUT pin, the BOOTCFG field in the SIU_RSR is updated. If BOOTCFG is negated at the negation of the RSTOUT pin, then the BAM program reads the RCHW from the lowest address of the internal flash memory. This address is 0x0000_0000.

If the RCHW stored in the internal flash is not valid (Boot Identifier field of RCHW is not 0x5A), or if BOOTCFG is asserted at the negation of the RSTOUT pin, then the RCHW is not applicable, and the serial boot mode is performed. *Table 14* summarizes the Reset Configuration Half Word location options.

Table 14. Reset configuration half word sources(1)

BOOTCFG	Boot identifier field (RCHW)	Boot mode	Configuration word source
0	Valid	Internal	The lowest address of one of the six low addresses space (LAS) in internal flash memory.
	Invalid	Serial	Not applicable
1	—	Serial	Not applicable

1. Several locations within the Low Address Space memory are used to search for a valid RCHW. Refer to *Section 20.5.3: Reset Configuration Half Word (RCHW)*.

4.7.1.2 RCHW structure

The structure of the RCHW is shown in *Figure 8*.

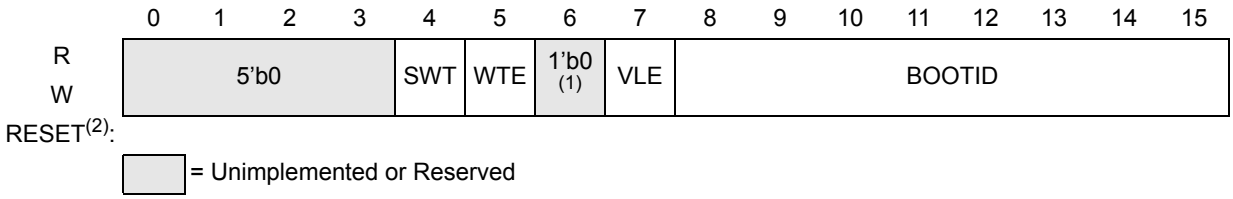


Figure 8. Reset Configuration Half Word (RCHW)

1. PS0 - Port Size. Reserved, should be set to zero for future compatibility.
2. The RCHW is in user programmed non-volatile memory. Therefore, it has no reset value.

Table 15. Reset Configuration Half Word (RCHW) field descriptions

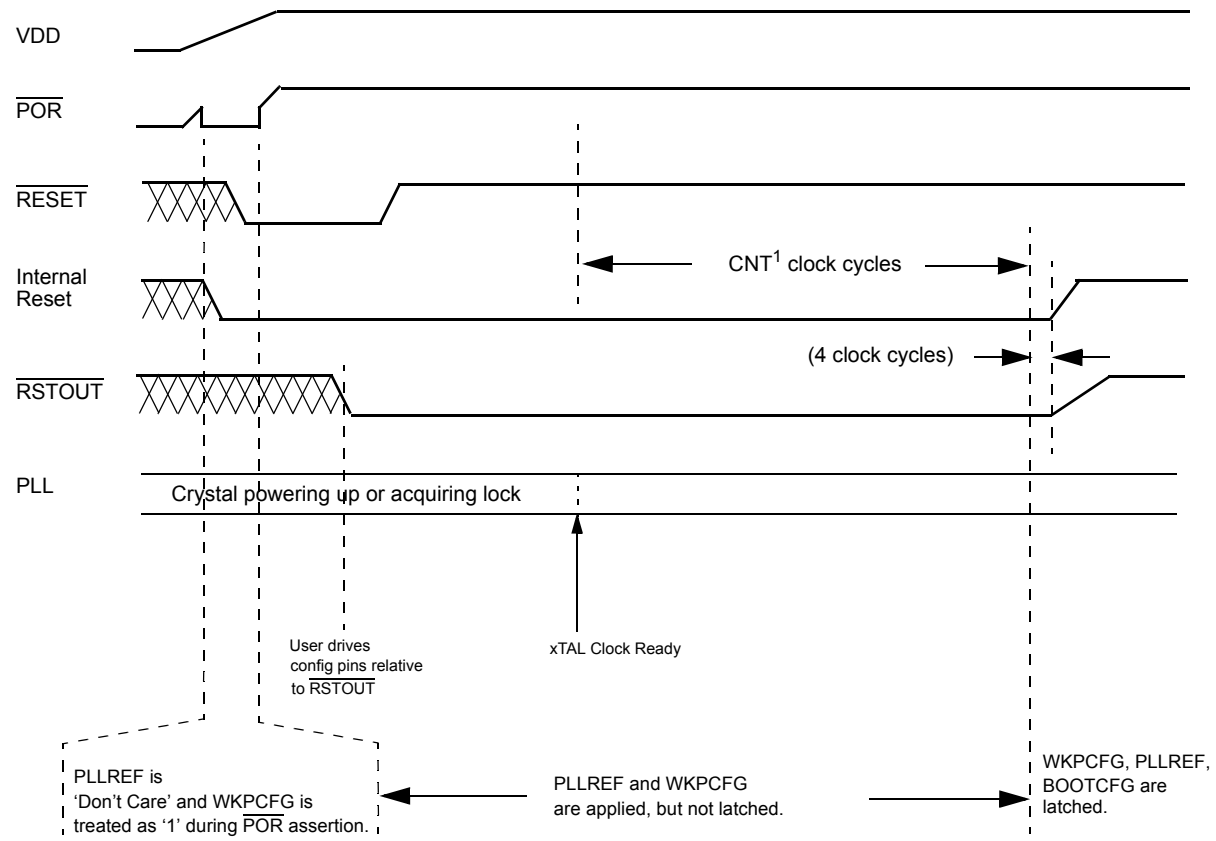
Field	Description
SWT	Software Watchdog Timer Enable This bit defines whether the BAM program enables the Software Watchdog Timer. 1 Software Watchdog Timer enabled 0 Software Watchdog Timer disabled
WTE	Watchdog Timer Enable This bit defines whether the BAM program enables the e200z335 Watchdog Timer. See *Chapter 20: Boot Assist Module (BAM)* for details on the Watchdog Timer configuration. 1 BAM program writes the e200z335 timebase registers (TBU and TBL) to 0x0000_0000_0000_0000 and enables the e200z335 core Watchdog Timer with a timeout period of 3 x 217 system clock cycles. (Example: For 8 MHz crystal -> 12 MHz system clock -> 32.7 ms timeout. For 20 MHz crystal -> 30 MHz system clock -> 13.1 ms timeout) 0 BAM program does not write the e200z335 timebase registers (TBU and TBL) nor enable the e200z335 core Watchdog Timer.
VLE	VLE Indicator This bit is used to configure the MMU for the boot block to execute as either Power Architecture technology code or as VLE code. 1 Boot code executes as VLE code 0 Boot code executes as Power Architecture technology code
BOOTID	Boot Identifier This field serves two functions. First, it is used to indicate which block in flash memory contains the boot program. Second, it identifies whether the flash memory is programmed or invalid. The value of a valid boot identifier is 0x5A. For an internal boot, the BAM program checks the second half word of each flash memory block starting at block 0 until a valid boot identifier is found. If all blocks in the low address space of the internal flash are checked an no valid boot identifier is found, then the internal flash is assumed to be invalid and a CAN/eSCI boot is initiated.

4.7.2 Reset configuration timing

The timing diagram in *Figure 9* shows the sampling of the BOOTCFG, WKPCFG, and PLLREF pin for a power-on reset. The timing diagram is also valid for internal/external resets assuming that VDD and VDD33 are within valid operating ranges. The value of the PLLREF pin is latched at the negation of the RSTOUT pin. The value of the WKPCFG signal is applied at the assertion of the internal reset signal (assertion of RSTOUT). The values of

the WKPCFG and BOOTCFG pins are latched 4 clock cycles before the negation of the RSTOUT pin and stored in the Reset Status Register.

Figure 9. Reset Configuration Timing



NOTE:
1. The clock count CNT depends on the reset source and type of clock reference. Please refer to *Table 13*.

4.7.3 Reset weak pull up/down configuration

The signal on the WKPCFG pin determines whether specified eTPU and eMIOS pins are connected to weak pull up or weak pull down devices at reset (see the Signal Description chapter for the eTPU and eMIOS pins that are affected by WKPCFG). For all reset sources except the Software External Reset, the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT). If the WKPCFG signal is logic high at this time, pull up devices will be enabled on the eTPU and eMIOS pins. If the WKPCFG signal is logic low at the assertion of the internal reset signal, pull down devices will be enabled on those pins. The value on WKPCFG must be held constant during reset to avoid oscillations on the eTPU and eMIOS pins caused by switching pull up/down states. The final value of WKPCFG is latched 4 clock cycles before the negation of RSTOUT. After reset, software may modify the weak pull up/down selection for all I/O pins through the PCR registers in the SIU.

5 Operating modes

5.1 Overview

This section gives a brief overview of the operating modes of this device.

5.2 Modes of operation

5.2.1 Normal Mode

Normal Mode is the functional mode of this device.

5.2.2 Debug Mode

Debug Mode provides access to powerful debugging and development features of this device. The debug and development features are distributed between Nexus blocks in the e200z335 core, eDMA and the eTPU, and some of the peripheral modules. The Nexus debug and development features are described in *Chapter 31: Nexus Port Controller (NPC)*.

The peripheral blocks that implement Debug Mode are:

- DSPI B, DSPI C
- FlexCAN A, FlexCAN C
- eMIOS
- eQADC
- eTPU (referenced as Halt State in *Chapter 22: Enhanced Time Processing Unit (eTPU2)*)

See the “Modes of Operation” section of the individual module for a description of how the Debug Mode affects the behavior of the module.

5.2.3 Low power modes

This device can be configured such that the clock to some or all of the modules can be stopped to reduce the power consumption. A tiered approach towards clock gating is implemented. In the first tier (Module Disable mode) some modules can be configured to stop the clock to the non-memory mapped registers within the module. In the second tier (Module Halt mode) the clock to each of the modules, including the CPU, can be completely stopped.

5.2.3.1 Module Disable Mode

Module Disable Mode is a low-power mode supported by some of the modules on this device, in which the clock to the non-memory mapped registers within the module is gated-off. *Table 16* lists the modules that support Module Disable Mode. The register and bit in each module that must be written to enter or exit this mode are also listed. See the “Modes of Operation” section of the individual module for a description of how the Module Disable Mode affects the behavior of the module.

5.2.3.2 Module Halt Mode

Module Halt mode is a low power mode in which the clock to all registers within each module can be completely halted. The control of the clock gating is centralized in the SIU_HLT register, which has one control bit for each module to be halted. The CPU itself can also be halted.

5.2.3.3 Standby Mode

In this mode, the power is removed from all functions except the standby RAM. Standby mode is entered by removing all power supplies except the one on the VSTBY pin. The device is recovered from the standby mode when powered again; see *Chapter 4: Resets* for more information.

5.3 Modes and clock architecture

5.3.1 Block diagrams

This section contains block diagrams that illustrate the FMPLL, the clock architecture and the various FMPLL and clock configurations that are available.

Figure 10. FMPLL bypass mode with crystal reference

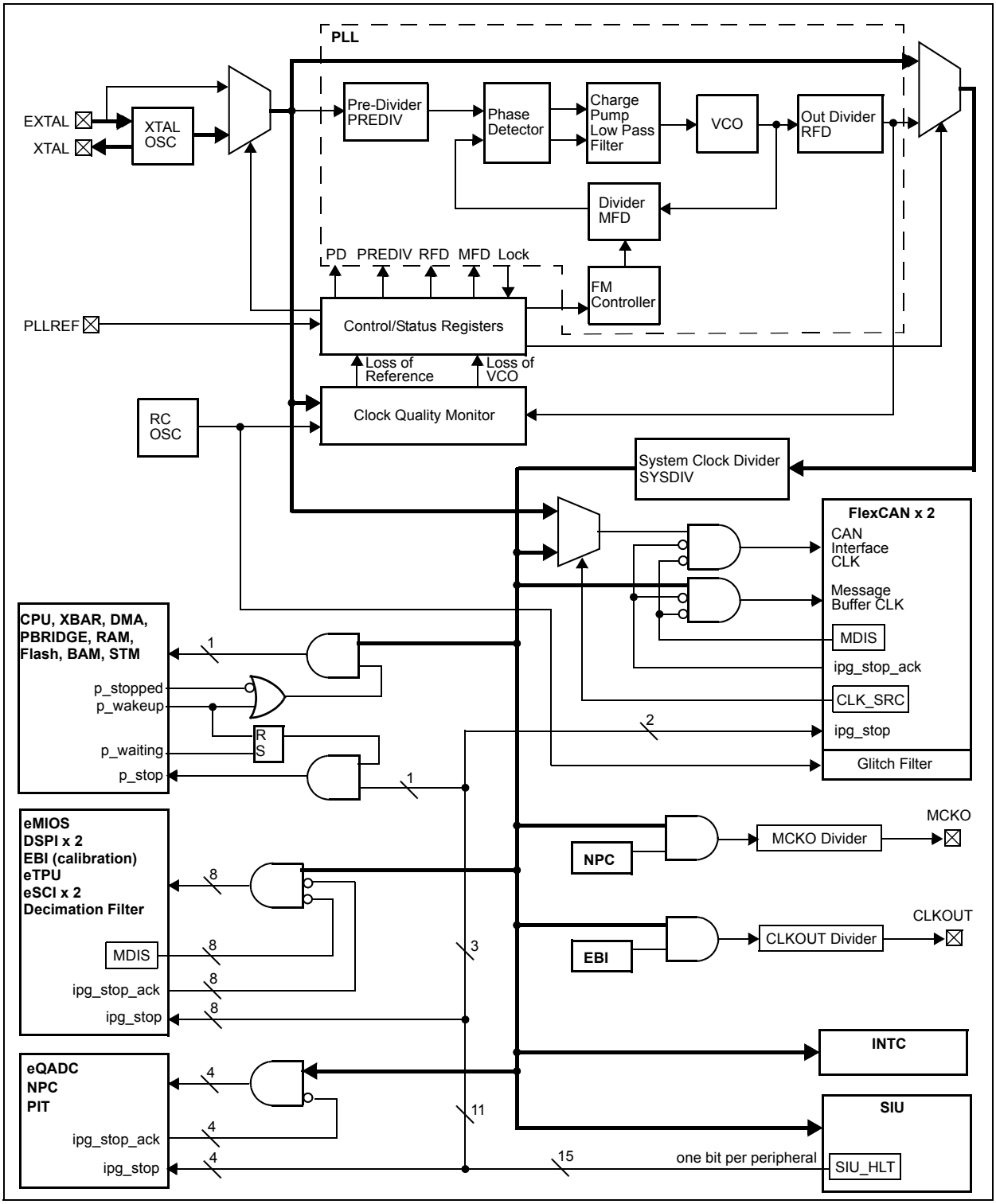


Figure 11. FMPLL bypass mode with external reference

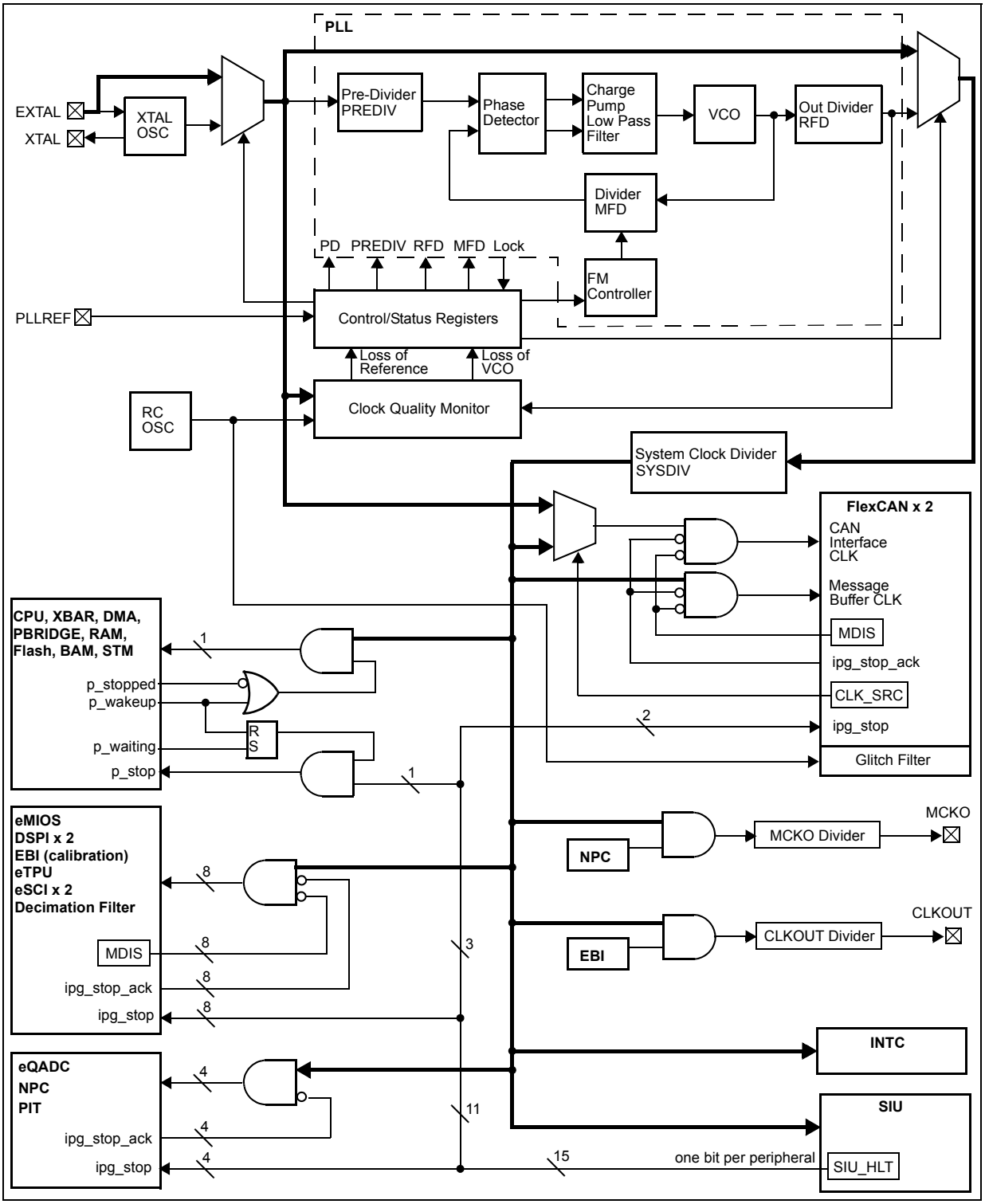


Figure 12. FMPLL normal mode with crystal reference

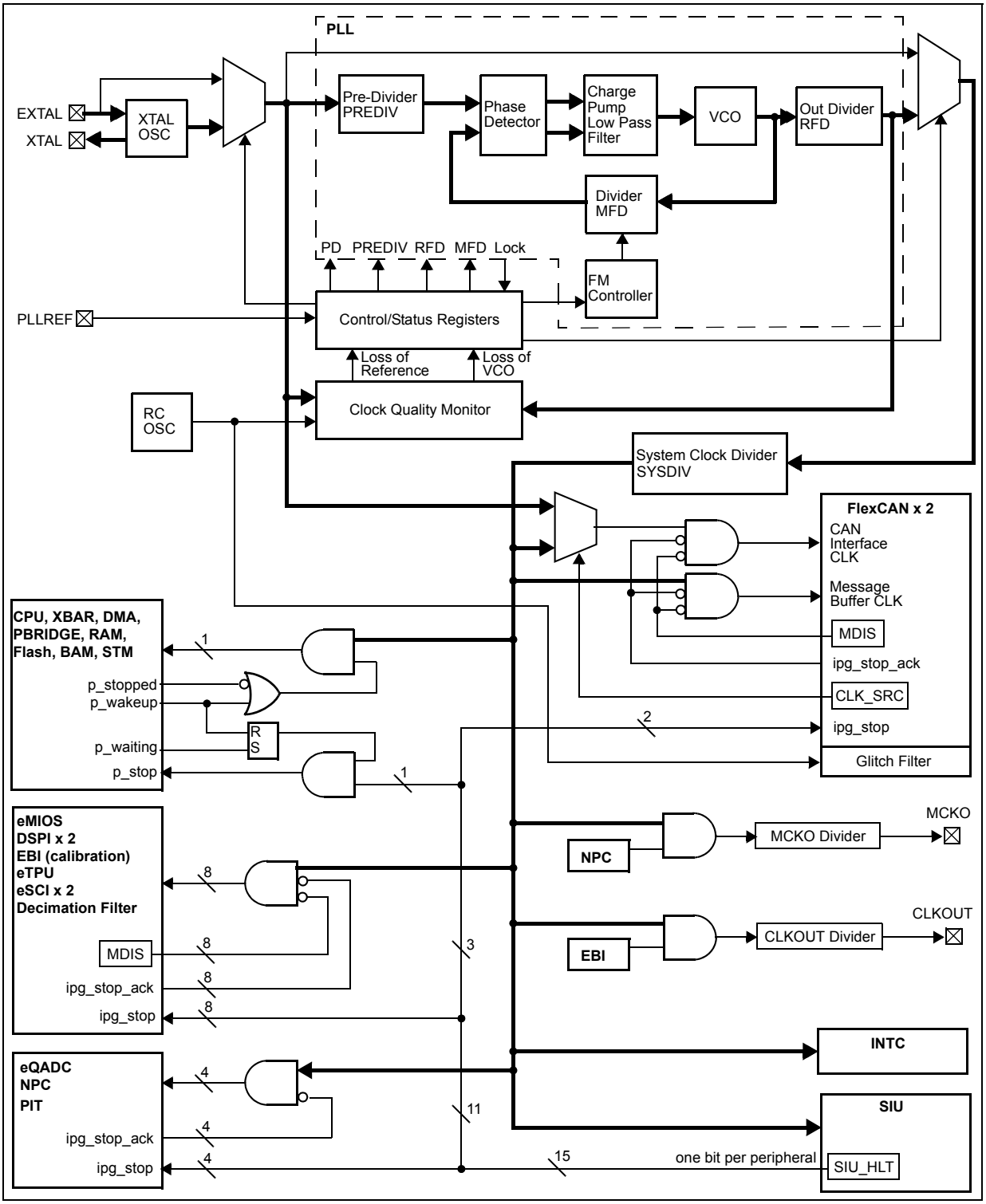
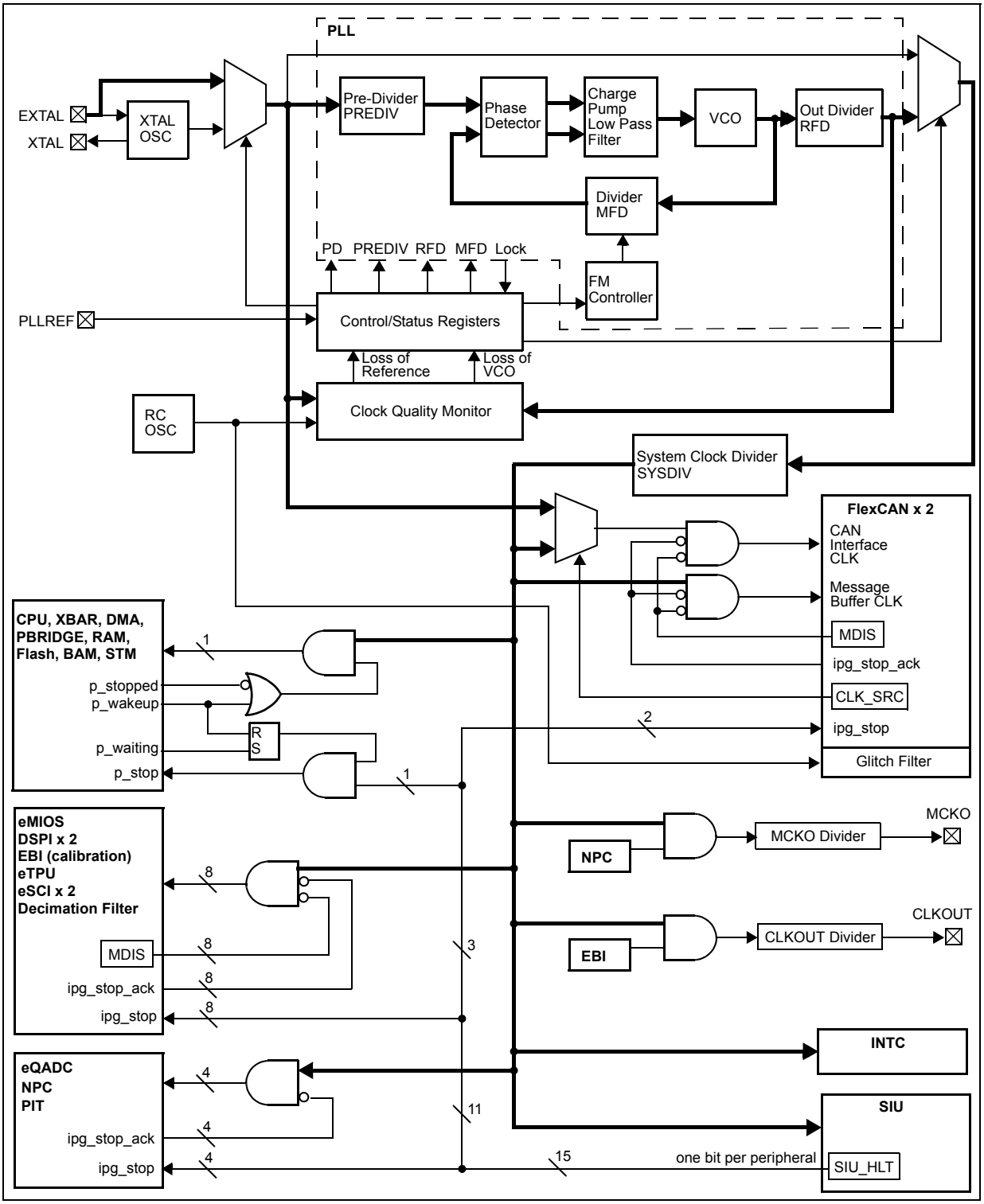


Figure 13. FMPLL normal mode with external reference



5.3.2 Clock architecture

This section describes the clocks and clock architecture on this MCU.

5.3.2.1 Overview

The system clocks are generated from one of the following FMPLL modes:

- Bypass with external reference (FMPLL on or off)
- Bypass with crystal reference (FMPLL on or off)
- Normal with external reference
- Normal mode with crystal reference

See *Chapter 16: Frequency-modulated phase-locked loop (FMPLL)* for information on the different clocking modes available.

Software controlled power management features are available to allow software to selectively gate the clocks to the CPU and each of the peripherals.

The MCU has two clock output pins that are driven by programmable clock dividers. The clock dividers divide the system clock down by even integer values. The two clock output pins are the following:

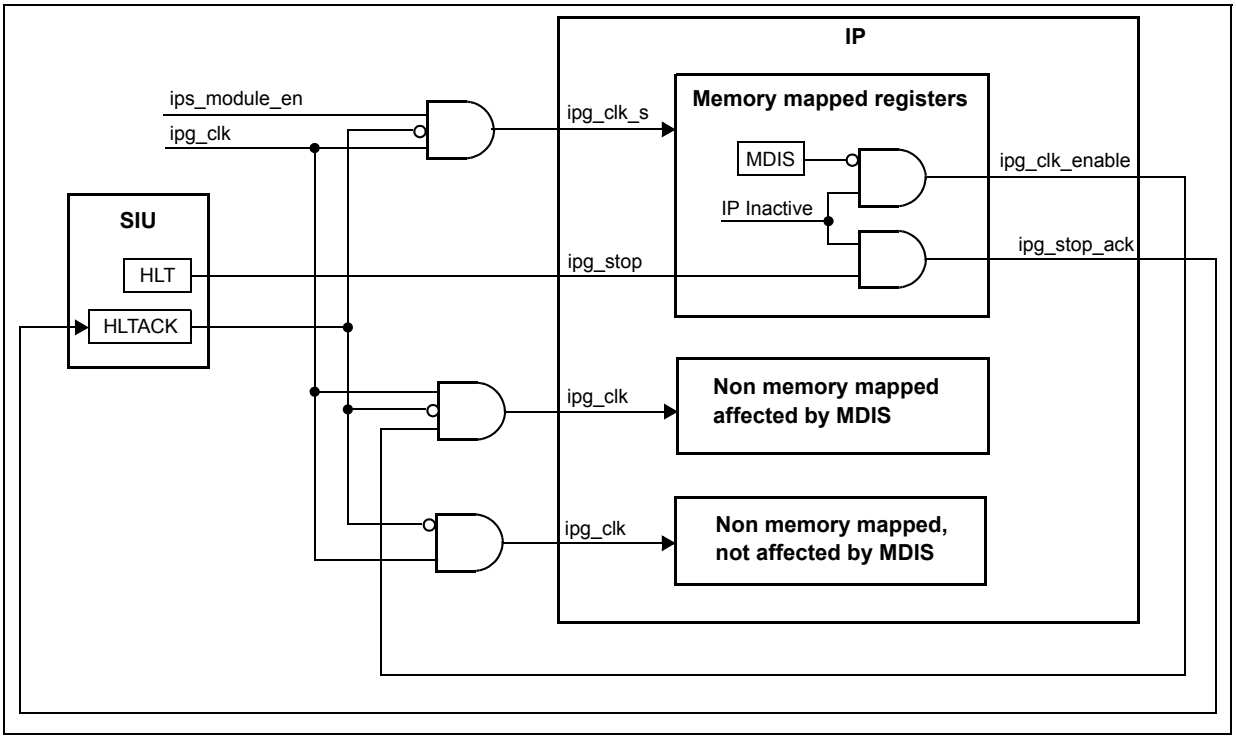
- CLKOUT — External address/data bus clock for the calibration interface; only available in the 496-pin packages.
- MCKO — Nexus auxiliary port clock

The oscillator clock can be selected as the clock source for the CAN interface in the FlexCAN blocks, resulting in very low jitter performance on the CAN bus.

5.3.2.2 Software controlled power management

Software controlled power management and clock gating is supported on a peripheral by peripheral basis, using a three tiered approach. The first tier is a clock gating feature implemented in some of the IP modules (see *Table 16*), which allow software to disable the non-memory-mapped portions of the modules by setting the module disable (MDIS) bits in the registers within the modules. The second tier is provided by the SIU_HLT register, which can be used to halt the clock of both memory-mapped and non-memory-mapped portions of each module. The third tier is provided by the WAIT instruction of the Power Architecture instruction set, which controls the clock gating of the CPU itself. *Figure 14* illustrates how the MDIS and halt bits affect the clocks to the modules.

Figure 14. MDIS and halt clock gating



5.3.2.2.1 MDIS clock gating

The MDIS bit disables the clock to some or all of the non-memory-mapped registers of the module. The memory-mapped portion of the modules are clocked by the system clock when they are accessed.

When the NPC module is disabled by MDIS, the MCKO clock is disabled. Furthermore, the NPC can be configured to disable the MCKO clock when there are no messages pending.

When the EBI module is disabled by MDIS, the CLKOUT clock is disabled. Furthermore, the EBI automatically disables the CLKOUT clock when there are no transactions on the external (calibration) bus.

The flash array can be disabled by writing to a bit in the flash memory map.

The modules that implement the MDIS function are listed in *Table 16*, along with the registers and bits that disable each module. The software controlled clocks are enabled when the CPU comes out of reset.

Table 16. MDIS support(1)

Block name	Register name	Bit name
DSPI_B	DSPI_B_MCR	MDIS
DSPI_C	DSPI_C_MCR	MDIS
EBI	EBI_MCR	MDIS
eTPU	ETPUECR_1	MDIS
FlexCAN A	FLEXCAN_A_MCR	MDIS

Table 16. MDIS support[1] (continued)

Block name	Register name	Bit name
FlexCAN C	FLEXCAN_C_MCR	MDIS
eMIOS	EMIOS_MCR	MDIS
eSCI_A	eSCI_A_SCICR3	MDIS
eSCI_B	eSCI_B_SCICR3	MDIS
Decimation Filter	DECFILTERMCR	MDIS
NPC	NPC_PCR	MCKO_EN, MCKO_GT
Flash Array	FLASH_MCR	STOP

1. By default the MDIS bit reset value is '0'.

5.3.2.2.2 Halt clock gating

Software controlled clock gating can be done via the centralized halt mechanism. The SIU_HLT register bits corresponding to individual modules are configured to determine which modules are clock gated.

The SIU_HLT bits are used to drive a stop request signal to the modules. After the module completes a clean shut down, the module asserts a stop acknowledge handshake signal that is used to gate the clock to the module (see *Figure 14*). The stop acknowledge signals are also captured in the SIU_HLTACK read-only register bits.

The halted modules recover when the SIU_HLT bit is cleared by software. Once the bit is cleared, logic will re-enable the clocks to the modules and then negate the stop request signal after the required timing has been met.

5.3.2.2.3 CPU clock gating

The SIU_HLT register has a bit to halt the clock to the CPU, but in order to prevent accidental CPU halting, a stop request is only activated if the CPU is in wait state due to the execution of the WAIT instruction (indicated by the p_waiting signal).

The CPU recovers from the halted state when one of the following events happens:

- A valid pending interrupt is detected by the core
- A request to enter debug mode is made by setting the DR bit in the OnCE control register (OCR)
- The processor is in a debug session
- A request to enable the CPU clock input has been made by setting the WKUP bit in the OnCE control register

When one of these events is detected, the CPU asserts an asynchronous output signal (p_wakeup) that re-enables the clock to the CPU so that it can exit the stopped state. Typically, the wake-up interrupt request will come from one of three sources: periodic interrupt timer (PIT) interrupt, external pin interrupt or CAN wake-up interrupt.

When the clock to the CPU is gated, the clocks to the platform, the system RAM and the flash memory are also gated. The platform logic includes the crossbar, peripheral bridge, DMA and flash controller. Note that the interrupt controller (INTC) and the SIU are not clock gated to allow interrupts to be used to recover the CPU halt state.

5.3.2.3 Clock dividers

The MCU provides three clock dividers: one for the system clock, one for CLKOUT and one for MCKO.

5.3.2.3.1 System clock divider

The system clock divider is placed right at the output of the FMPLL and before the clock is used by any other circuits, including the other clock dividers. It affects the clock in both normal mode and bypass mode. The system clock divider can only be programmed to divide by 2, 4, 8, and 16 by setting the SYSCLKDIV field in the SIU_SYSDIV register. To achieve an equivalent of a divide-by-1 is to set the bypass bit. The reset value of the bypass bit is logic 1.

5.3.2.3.2 External calibration bus clock (CLKOUT)

The external calibration bus clock (CLKOUT) divider can be programmed to divide the system clock by one, two or four based on the settings of the EBDF bit field in the SIU external clock control register (SIU_ECCR). The reset value of the EBDF selects a CLKOUT frequency of one half of the system clock frequency. The EBI supports gating of the CLKOUT signal when there are no external bus accesses in progress.

The hold time for external bus pins can be changed by writing to the external bus tap select (EBTS) bit in the SIU_ECCR.

Note: The CLKOUT pin is available only in the 496-pin packages.

5.3.2.3.3 Nexus message clock (MCKO)

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

5.3.2.4 FlexCAN clock domains

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

6 e200z335 Core

6.1 Introduction

This device includes the e200z335 processor core. The e200z processor family is a set of CPU cores that implement low-cost versions of the Power Architecture. These processors are designed for deeply embedded control applications which require low-cost solutions rather than maximum performance.

The e200z335 core is a single-issue, 32-bit Power Architecture compliant design with 32 general purpose registers (GPRs). Power Architecture floating-point instructions are not supported by this core in hardware, but are trapped and may be emulated by software. All arithmetic instructions that execute in the core operate on data in the GPRs.

A Signal Processing Extension (SPE) Auxiliary Processing Unit (APU) is provided to support real-time fixed point and single precision, embedded numerics operations using the GPRs. All arithmetic instructions that execute in the core operate on data in the GPRs. The GPRs have been extended to 64-bits in order to support vector instructions defined by the SPE APU. These instructions operate on a vector pair of 16-bit or 32-bit data types, and deliver vector and scalar results.

In addition to the Power Architecture instruction set, the e200z335 core also implements the VLE (Variable Length Encoding) APU, providing improved code density.

6.2 Features

The following is a list of some of the key features of the e200z335 core:

- 32-bit Power Architecture programmer's model
- Single issue, 32-bit Power Architecture compliant CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
  - Dedicated branch address calculation adder
  - Branch acceleration using Branch Lookahead Instruction Buffer
  - Memory Management Unit (MMU) with 16-entry fully-associative translation look-aside buffer
- Load/store unit
  - 1 cycle load latency
  - Fully pipelined
  - Big and Little endian support
  - Misaligned access support
  - Zero load-to-use pipeline bubbles
- Power management
  - Low power design
  - Power saving modes: nap, sleep
  - Dynamic power management of execution units
- Testability
  - Synthesizeable, full Muxed scan design
  - ABIST/MBIST for optional memory arrays

7 Direct Memory Access (DMA)

7.1 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.

7.1.1 Device-specific features

- 32 programmable channels
- The SIU outputs to the eDMA are not connected. If the SIU_DIRSR register selects the eDMA output, the effect will be to disable the interrupts.
- The following registers are not implemented:
  - DMAERQH
  - DMAEEIH
  - DMAINTH
  - DMAERRH
  - DCHPRI32-DCHPRI63
  - TCD32 – TCD63
- Channel groups 2 and 3 are not implemented.

7.1.2 Channel assignments

The assignments between the DMA requests from the blocks to the channels of the eDMA are shown in *Table 17*. The Source column is written in C language syntax. The syntax is block_instance.register[bit].

Table 17. DMA channel assignments

DMA request	Channel	Source	Description
eQADC_FISR0_CFFF0	0	$\overline{\text{eQADC}}$.eQADC_FISR0[CFFF0]	eQADC Command FIFO 0 Fill Flag
eQADC_FISR0_RFDF0	1	$\overline{\text{eQADC}}$.eQADC_FISR0[RFDF0]	eQADC Receive FIFO 0 Drain Flag
eQADC_FISR1_CFFF1	2	$\overline{\text{eQADC}}$.eQADC_FISR1[CFFF1]	eQADC Command FIFO 1 Fill Flag
eQADC_FISR1_RFDF1	3	$\overline{\text{eQADC}}$.eQADC_FISR1[RFDF1]	eQADC Receive FIFO 1 Drain Flag
eQADC_FISR2_CFFF2	4	$\overline{\text{eQADC}}$.eQADC_FISR2[CFFF2]	eQADC Command FIFO 2 Fill Flag
eQADC_FISR2_RFDF2	5	$\overline{\text{eQADC}}$.eQADC_FISR2[RFDF2]	eQADC Receive FIFO 2 Drain Flag
eQADC_FISR3_CFFF3	6	$\overline{\text{eQADC}}$.eQADC_FISR3[CFFF3]	eQADC Command FIFO 3 Fill Flag
eQADC_FISR3_RFDF3	7	$\overline{\text{eQADC}}$.eQADC_FISR3[RFDF3]	eQADC Receive FIFO 3 Drain Flag
eQADC_FISR4_CFFF4	8	$\overline{\text{eQADC}}$.eQADC_FISR4[CFFF4]	eQADC Command FIFO 4 Fill Flag
eQADC_FISR4_RFDF4	9	$\overline{\text{eQADC}}$.eQADC_FISR4[RFDF4]	eQADC Receive FIFO 4 Drain Flag
eQADC_FISR5_CFFF5	10	$\overline{\text{eQADC}}$.eQADC_FISR5[CFFF5]	eQADC Command FIFO 5 Fill Flag
eQADC_FISR5_RFDF5	11	$\overline{\text{eQADC}}$.eQADC_FISR5[RFDF5]	eQADC Receive FIFO 5 Drain Flag
DSPI_B_SR_TFFF	12	$\overline{\text{DSPI_B}}$.DSPI_SR[TFFF]	DSPI_B Transmit FIFO Fill Flag

Table 17. DMA channel assignments (continued)

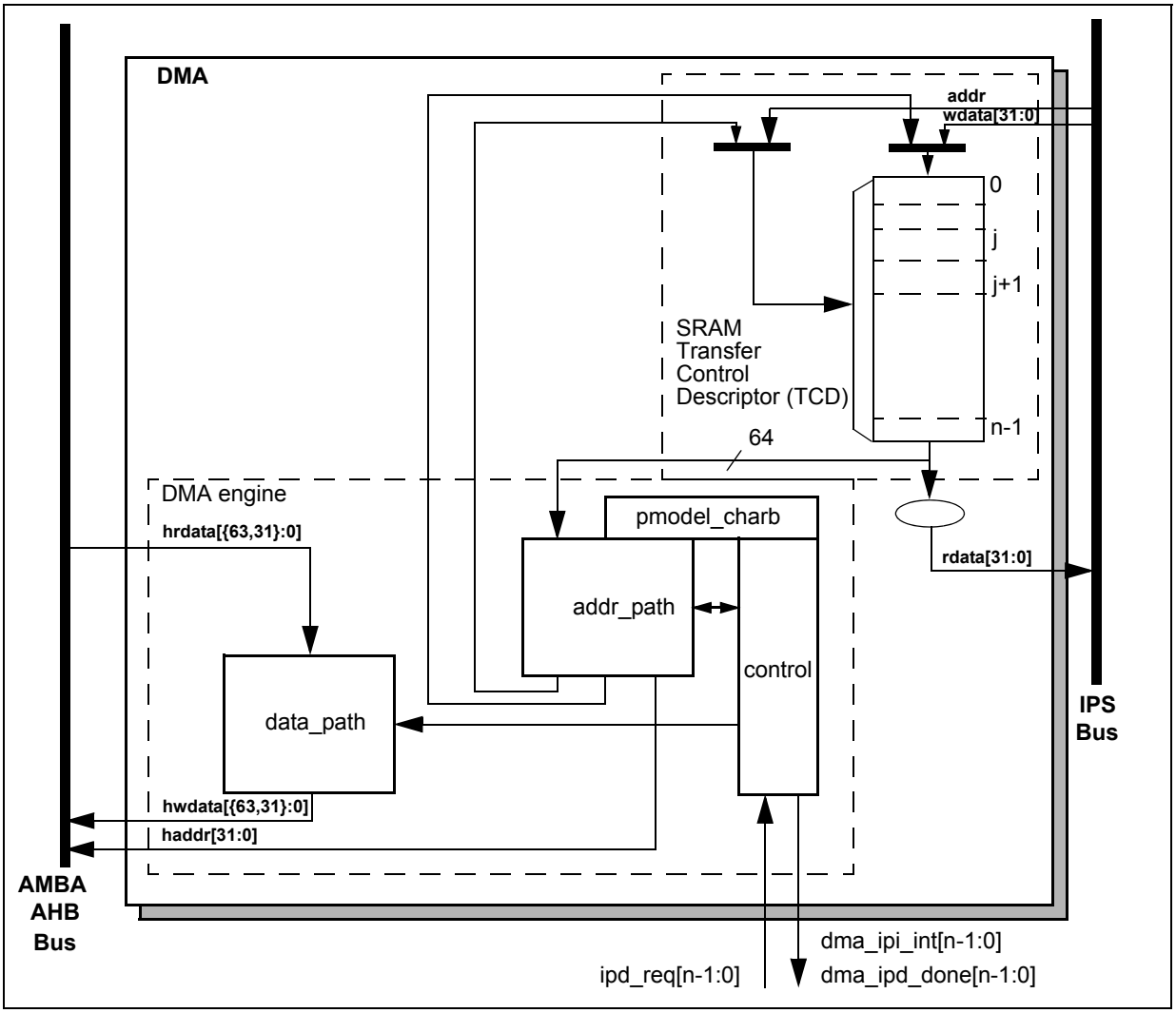
DMA request	Channel	Source	Description
DSPI_B_SR_RFDF	13	DSPI_B.DSPI_SR[RFDF]	DSPI_B Receive FIFO Drain Flag
DSPI_C_SR_TFFF	14	DSPI_C.DSPI_SR[TFFF]	DSPI_C Transmit FIFO Fill Flag
DSPI_C_SR_RFDF	15	DSPI_C.DSPI_SR[RFDF]	DSPI_C Receive FIFO Drain Flag
DECFIL_FILL_BUF	16	DECFIL_FILL_BUF	Decimation Filter Fill Buffer
DECFIL_DRAIN_BUF	17	DECFIL_DRAIN_BUF	Decimation Filter Drain Buffer
SCI_A_TDRE_TC_TXRDY	18	SCI_A.SCISR1[TDRE] || SCI_A.SCISR1[TC] || SCI_A.LINSTAT1[TXRDY]	SCI_A Transmit
SCI_A._RDRF_RXRDY	19	SCI_A.SCISR1[RDRF] || SCI_A.LINSTAT1[RXRDY]	SCI_A Receive Data
EMIOSFLAG_F0	20	EMIOS.EMIOSFLAG[F0]	eMIOS channel 0 Flag
EMIOSFLAG_F1	21	EMIOS.EMIOSFLAG[F1]	eMIOS channel 1 Flag
EMIOSFLAG_F2	22	EMIOS.EMIOSFLAG[F2]	eMIOS channel 2 Flag
EMIOSFLAG_F3	23	EMIOS.EMIOSFLAG[F3]	eMIOS channel 3 Flag
EMIOSFLAG_F4	24	EMIOS.EMIOSFLAG[F4]	eMIOS channel 4 Flag
EMIOSFLAG_F8	25	EMIOS.EMIOSFLAG[F8]	eMIOS channel 8 Flag
EMIOSFLAG_F9	26	EMIOS.EMIOSFLAG[F9]	eMIOS channel 9 Flag
eTPUDTRSR_1_DTRS0	27	eTPU.eTPUDTRSR_1[DTRS0]	eTPU_1 Channel 0 Data Transfer Request Status
eTPUDTRSR_1_DTRS1	28	eTPU.eTPUDTRSR_1[DTRS1]	eTPU_1 Channel 1 Data Transfer Request Status
eTPUDTRSR_1_DTRS2	29	eTPU.eTPUDTRSR_1[DTRS2]	eTPU_1 Channel 2 Data Transfer Request Status
eTPUDTRSR_1_DTRS14	30	eTPU.eTPUDTRSR_1[DTRS14]	eTPU_1 Channel 14 Data Transfer Request Status
eTPUDTRSR_1_DTRS15	31	eTPU.eTPUDTRSR_1[DTRS15]	eTPU_1 Channel 15 Data Transfer Request Status

7.2 Introduction

The DMA (Direct Memory Access) is a second-generation platform module capable of performing complex data transfers with minimal intervention from a host processor via "*n*" programmable channels. Intended for use as part of the Standard Product Platform (SPP), the hardware microarchitecture includes a DMA 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.

Figure 15 is a block diagram of the DMA module.

Figure 15. DMA block diagram



7.2.1 Overview

The DMA 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 DMA hardware supports:

- Single design supporting 16-, 32- and 64-channel implementations, dependent on size of the TCD memory and design parameters
- Connections to the AMBA-AHB crossbar switch for bus mastering the data movement, 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).

7.2.2 Features

The DMA 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 DMA processing

The structure of the transfer control descriptor is fundamental to the operation of the DMA module. It is defined below in a ‘C’ pseudo-code specification, where `int` refers to a 32-bit variable (unless noted otherwise) and `short` is a 16-bit variable.

Note: To compile these structures, change any periods '.' in the variable name to underscores '_'.

```
typedef union {
    struct { /* citer.e_link = 1 */
        unsigned short citer.linkch:6; /* link channel number, */
        unsigned short citer:9; /* current (“major”) iteration count */
    } minor_link_enabled;/* channel link at end of the minor loop */
    struct {/* citer.e_link = 0 */
        unsigned short citer:15; /* current (“major”) iteration count */
    } minor_link_disabled;/* no linking at end of the minor loop */
} t_minor_link_citer;

typedef union {
    struct { /* biter.e_link = 1 */
        unsigned short biter.linkch:6; /* link channel number, */
        unsigned short biter:9; /* beginning (“major”) iteration count */
    } init_minor_link_enabled;/* channel link at end of the minor loop */
    struct {/* biter.e_link = 0 */
        unsigned short biter:15; /* beginning (“major”) iteration count */
```

```
    } init_minor_link_disabled;/* no linking at end of the minor loop */
} t_minor_link_biter;

typedef struct {
   unsigned int saddr;/* source address */
   unsigned int smod:5;/* source address modulo */
   unsigned int ssize:3;/* source transfer size */
   unsigned int dmod:5;/* destination address modulo */
   unsigned int dsize:3;/* destination transfer size */
   short soff;/* signed source address offset */
   unsigned intnbytes;/* inner (“minor”) byte count */
   int slast;/* last source address adjustment */
   unsigned int daddr;/* destination address */
   unsigned short citer.e_link:1;/* enable channel linking on minor loop */
   t_minor_link_citerminor_link_citer;/* conditional current iteration
count */
   short doff;/* signed destination address offset */
   int dlast_sga;/* last destination address adjustment, or
          scatter/gather address (if e_sg = 1) */
   unsigned short biter.e_link:1;/* beginning channel link enable */
   t_minor_link_biterminor_link_biter;/* beginning (“major”) iteration
count */
   unsigned int bwc:2;/* bandwidth control */
   unsigned int major.linkch:6;/* link channel number */
   unsigned int done:1;/* channel done */
   unsigned int active:1;/* channel executing */
   unsigned int major.e_link:1;/* enable channel linking on major loop*/
   unsigned int e_sg:1;/* enable scatter/gather descriptor */
   unsigned int d_req:1;/* disable ipd_req when done */
   unsigned int int_half:1;/* interrupt on citer = (biter >> 1) */
   unsigned int int_maj:1;/* interrupt on major loop completion */
   unsigned int start:1;/* explicit channel start */
} tcd   /* transfer_control_descriptor */
```

The basic operation of a channel is defined as:

1. The channel is initialized by software loading the transfer control descriptor into the DMA’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 DMA engine’s internal register file.
4. The DMA 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 DMA 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. A more detailed description of the channel processing is listed in the pseudo-code below. This simplified example is intended to represent basic data transfers. Detailed processing associated with the error handling is omitted.

```
/* the given DMAchannel is requesting service by the software assertion of
the tcd[channel].start bit, the assertion of an enabled ipd_req from a
device, or the implicit assertion of a channel-to-channel link */

/* begin by reading the transfer control descriptor from the local RAM
   into the local dma_engine registers */
dma_engine        = read_from_local_memory [channel];
dma_engine.active = 1;                /* set active flag */
dma_engine.done   = 0;                /* clear done flag */

/* check the transfer control descriptor for consistency */
if (dma_engine.config_error == 0) {
    / * begin execution of the inner "minor" loop transfers */
    {
        /* convert the source transfer size into a byte count */
        switch (dma_engine.ssize) {
        case 0:                        /* 8-bit transfer */
            src_xfr_size = 1;
            break;
        case 1:                        /* 16-bit transfer */
            src_xfr_size = 2;
            break;
        case 2:                        /* 32-bit transfer */
            src_xfr_size = 4;
            break;
        case 3:                        /* 64-bit transfer */
            src_xfr_size = 8;
            break;
        case 4:                        /* 16-byte burst transfer */
            src_xfr_size = 16;
            break;
        case 5:                        /* 32-byte burst transfer */
            src_xfr_size = 32;
            break;
        }
```

```
        /* convert the destination transfer size into a byte count */
        switch (dma_engine.dsize) {
        case 0:                         /* 8-bit transfer */
            dest_xfr_size = 1;
            break;
        case 1:                         /* 16-bit transfer */
            dest_xfr_size = 2;
            break;
        case 2:                         /* 32-bit transfer */
            dest_xfr_size = 4;
            break;
        case 3:                         /* 64-bit transfer */
            dest_xfr_size = 8;
            break;
        case 4:                         /* 16-byte burst transfer */
            dest_xfr_size = 16;
            break;
        case 5:                         /* 32-byte burst transfer */
            dest_xfr_size = 32;
            break;
        }

        /* determine the larger of the two transfer sizes, this value
           reflects */
        /* the number of bytes transferred per read->write sequence. */
        /* number of iterations of the minor loop = nbytes / xfer_size */
        if (dma_engine.ssize < dma_engine.dsize)
            xfr_size = dest_xfer_size;
         else
            xfr_size = src_xfer_size;

        /* process the source address, READ data into the buffer*/

        /* read "xfr_size" bytes from the source */
        /* if the ssize < dsize, do multiple reads to equal the dsize */
        /* if the ssize => dsize, do a single read of source data */
        number_of_source_reads = xfer_size / src_xfer_size;

        for (number_of_source_reads) {
            dma_engine.data = read_from_amba-ahb (dma_engine.saddr,
src_xfr_size);

            /* generate the next-state source address */
            /* sum the current saddr with the signed source offset */
```

```
                ns_addr = dma_engine.saddr + (int) dma_engine.soff; }

              /* if enabled, apply the power-of-2 modulo to the next-state addr
               */
                if (dma_engine.smod != 0)
                    address_select = (1 << dma_engine.smod) - 1; }
                else
                    address_select = 0xffff_ffff;

                dma_engine.saddr = ns_addr          &  address_select
                            | dma_engine.saddr & ~address_select; }
            }


            /* process the destination address, WRITE data from buffer */

            /* write "xfr_size" bytes to the destination */
            /* if the dsize < ssize, do multiple writes to equal the ssize */
            /* if the dsize => ssize, do a single write of dest data */
            number_of_dest_writes = xfer_size / dest_xfer_size;

            for (number_of_dest_writes) {
                write_to_amba-ahb (dma_engine.daddr, dest_xfr_size) =
dma_engine.data;

                /* generate the next-state destination address */
                /* sum the current daddr with the signed destination offset */
                ns_addr = dma_engine.daddr + (int) dma_engine.doff;

              /* if enabled, apply the power-of-2 modulo to the next-state dest
                 addr */
                if (dma_engine.dmod != 0)
                    address_select = (1 << dma_engine.dmod) - 1;
                else
                    address_select = 0xffff_ffff;

                dma_engine.daddr = ns_addr          &  address_select
                            | dma_engine.daddr & ~address_select;
            }

            /* check for a higher priority channel to service if: */
            /* 1) preemption is enabled */
            /* 2) in fixed arbitration mode */
            /* 3) a higher priority channel is requesting service */
            /* 4) not already servicing a preempting channel */
            if ((CPRn.ecp = 1)  &  fixed_arbitration_mode
```

```
                higher_pri_request & ~current_channel_is_preempt)
                service_preempt_channel;

            /* the bandwidth control field determines when the next read/write
               occurs */
            if (dma_engine.bwc > 1)
                stall_dma_engine (1 << dma_engine.bwc);

            /* decrement the minor loop byte count */
            dma_engine.nbytes = dma_engine.nbytes - xfr_size;

        }while (dma_engine.nbytes > 0) /* end of minor inner loop */

        dma_engine.citer--;           /* decrement major loop iteration count */

        /* if the major loop is not yet exhausted, update certain TCD values in
           the RAM */
        if (dma_engine.citer != 0) {
            write_to_local_memory [channel].saddr = dma_engine.saddr;
            write_to_local_memory [channel].daddr = dma_engine.daddr;
            write_to_local_memory [channel].citer = dma_engine.citer;

                 /* if minor loop linking is enabled, make the channel link */
            if (dma_engine.citer.e_link)
                TCD[citer.linkch].start = 1;     /* specified channel service
                                                    req */

           /* check for interrupt assertion if half of the major iterations are
              done */
           if (dma_engine.int_half && (dma_engine.citer == (dma_engine.biter >>
1)))
                generate_interrupt (channel);

            dma_engine.active = 0;         /* clear the channel busy flag */

         }
         else { /* major loop is complete, dma_engine.citer == 0 */
           /* since the major loop is complete, perform the final address
              adjustments */

           /* sum the current {src,dst} addresses with "last" adjustment */
           write_to_local_memory [channel].saddr = dma_engine.saddr +
dma_engine.slast;
           write_to_local_memory [channel].daddr = dma_engine.daddr +
dma_engine.dlast;
           /* restore the major iteration count to the beginning value */
           write_to_local_memory [channel].citer = dma_engine.biter;
```

```
        /* check for interrupt assertion at completion of the major iteration
         */
         if (dma_engine.int_maj)
           generate_interrupt (channel);

         /* check if the ipd_req is to be disabled at completion of the major
            iteration */
         if (dma_engine.d_req)
             DMAERQ [channel] = 0;

         /* check for a scatter/gather transfer control descriptor */
         if (dma_engine.e_sg) {
          /* load new transfer control descriptor from the address defined by
             dlast_sga */
             write_to_local_memory [channel] =
                                        read_from_amba-
ahb(dma_engine.dlast_sga,32);
         }
         if (dma_engine.major.e_link)
             TCD[major.linkch].start = 1;       /* specified channel service
                                                   req */

         dma_engine.active = 0;     /* clear the channel busy flag */
         dma_engine.done = 1;       /* set the channel done flag */

}
else { /* configuration error detected, abort the channel */
         dma_engine.error_status = error_type; /* record the error */
         dma_engine.active = 0; /* clear the channel busy flag */
         /* check for interrupt assertion on error */
         if (dma_engine.int_err)
             generate_interrupt (channel);
}
```

For more details, consult *Section 7.3.1: Register descriptions,* and *Section 7.4: Functional description*.

7.3 Memory map/register definition

The DMA'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, that is, 32-bits in size.

The DMA module does not include any logic which provides access control. Rather, this function is supported using the standard access control logic provided by the AIPS controller.

Table 18 is a 32-bit view of the DMA's memory map.

Table 18. DMA 32-bit memory map

DMA offset	Register			
0x0000	DMA Control Register (DMA_CR)			
0x0004	DMA Error Status Register (DMA_ESR)			
0x0008	Reserved			
0x000c	DMA Enable Request Register (DMA_ERQRL) (Channels 31–00)			
0x0010	Reserved			
0x0014	DMA Enable Error Interrupt Register (DMA_EEIRL) (Channels 31–00)			
0x0018	DMA Set Enable Request Register (DMA_SERQR)	DMA Clear Enable Request Register (DMA_CERQR)	DMA Set Enable Error Interrupt Register (DMA_SEEIR)	DMA Clear Enable Error Interrupt Register (DMA_CEEIR)
0x001c	DMA Clear Interrupt Request Register (DMA_CIRQR)	DMA Clear Error Register (DMA_CER)	DMA Set START Bit Register (DMA_SSBR)	DMA Clear DONE Status Bit Register (DMA_CDSBR)
0x0020	Reserved			
0x0024	DMA Interrupt Request (DMA_IRQRL) (Channels 31–00)			
0x0028	Reserved			
0x002c	DMA Error (DMA_ERL) Register (Channels 31–00)			
0x0030–0x00fc	Reserved			
0x0100	DMA Channel 0 Priority (DMA_CPR)	DMA Channel 1 Priority (DMA_CPR1)	DMA Channel 2 Priority (DMA_CPR2)	DMA Channel 3 Priority (DMA_CPR3)
0x0104	DMA Channel 4 Priority (DMA_CPR4)	DMA Channel 5 Priority (DMA_CPR5)	DMA Channel 6 Priority (DMA_CPR6)	DMA Channel 7 Priority (DMA_CPR7)
0x0108	DMA Channel 8 Priority (DMA_CPR8)	DMA Channel 9 Priority (DMA_CPR9)	DMA Channel 10 Priority (DMA_CPR10)	DMA Channel 11 Priority (DMA_CPR11)
0x010c	DMA Channel 12 Priority (DMA_CPR12)	DMA Channel 13 Priority (DMA_CPR13)	DMA Channel 14 Priority (DMA_CPR14)	DMA Channel 15 Priority (DMA_CPR15)
0x0110	DMA Channel 16 Priority (DMA_CPR16)	DMA Channel 17 Priority (DMA_CPR17)	DMA Channel 18 Priority (DMA_CPR18)	DMA Channel 19 Priority (DMA_CPR19)
0x0114	DMA Channel 20 Priority (DMA_CPR20)	DMA Channel 21 Priority (DMA_CPR21)	DMA Channel 22 Priority (DMA_CPR22)	DMA Channel 23 Priority (DMA_CPR23)

Table 18. DMA 32-bit memory map (continued)

DMA offset	Register			
0x0118	DMA Channel 24 Priority (DMA_CPR24)	DMA Channel 25 Priority (DMA_CPR25)	DMA Channel 26 Priority (DMA_CPR26)	DMA Channel 27 Priority (DMA_CPR27)
0x011c	DMA Channel 28 Priority (DMA_CPR28)	DMA Channel 29 Priority (DMA_CPR29)	DMA Channel 30 Priority (DMA_CPR30)	DMA Channel 31 Priority (DMA_CPR31)
0x0120	Reserved			
0x0124	Reserved			
0x0128	Reserved			
0x012c	Reserved			
0x0130	Reserved			
0x0134	Reserved			
0x0138	Reserved			
0x013c	Reserved			
0x0140–0x0ffc	Reserved			
0x1000–0x11fc	TCD00–TCD15			
0x1200–0x13fc	TCD16–TCD31			
0x1400–0x15fc	TCD32–TCD47			
0x1600–0x17fc	TCD48–TCD63			

7.3.1 Register descriptions

7.3.1.1 DMA Control Register (DMA_CR)

The 32-bit DMA_CR defines the basic operating configuration of the DMA.

The DMA arbitrates channel service requests in groups of 16 channels. The 32-channel configuration has two groups (1,0). Group 1 contains channels 31–16, and group 0 contains channels 15–0.

Arbitration within a group 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 7.3.1.15: DMA Channel n Priority (DCHPRIn), n = 0,...,31*). In round-robin arbitration mode, the channel priorities are ignored and the channels within each group are cycled through without regard to priority.

The group priorities operate in a similar fashion. In group fixed-priority arbitration mode, channel service requests in the highest priority group are executed first where priority level 3 is the highest and priority level 0 is the lowest. The group priorities are assigned in the GRPnPri registers. All group priorities must have unique values prior to any channel service requests occur, otherwise a configuration error will be reported. Unused group priority registers, per configuration, are unimplemented in the DMA_CR. In group round-robin

mode, the group priorities are ignored and the groups are cycled through without regard to priority.

See *Figure 16* and *Table 19* for the DMA_CR definition.

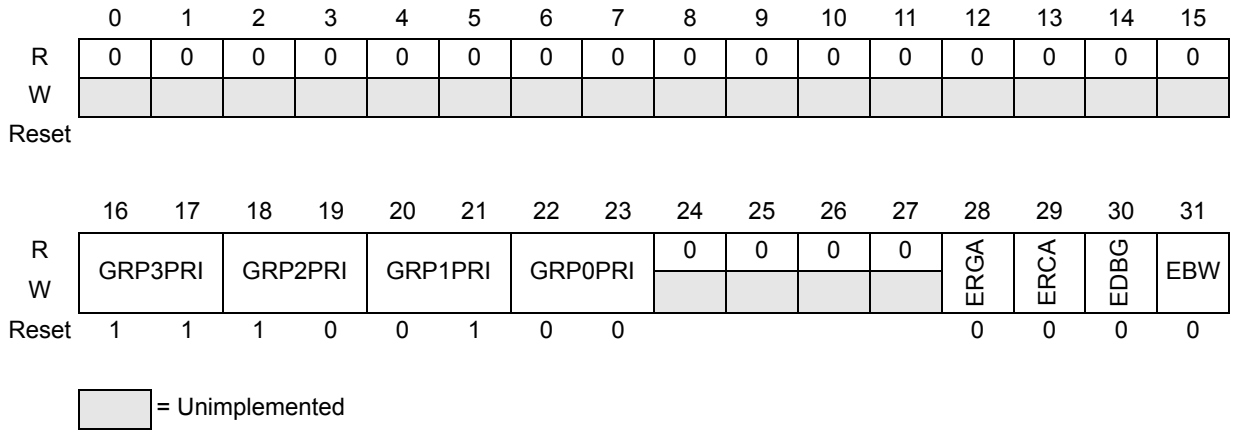


Figure 16. DMA Control Register (DMA_CR)

Table 19. DMA_CR field descriptions

Name	Description	Value
GRP3PRI	Channel Group 3 Priority	Group 3 priority level when fixed priority group arbitration is enabled.
GRP2PRI	Channel Group 2 Priority	Group 2 priority level when fixed priority group arbitration is enabled.
GRP1PRI	Channel Group 1 Priority	Group 1 priority level when fixed priority group arbitration is enabled.
GRP0PRI	Channel Group 0 Priority	Group 0 priority level when fixed priority group arbitration is enabled.
ERGA	Enable Round Robin Group Arbitration	0 Fixed priority arbitration is used for selection among the groups. 1 Round robin arbitration is used for selection among the groups.
ERCA	Enable Round Robin Channel Arbitration	0 Fixed priority arbitration is used for channel selection within each group. 1 Round robin arbitration is used for channel selection within each group.

Table 19. DMA_CR field descriptions (continued)

Name	Description	Value
EDBG	Enable Debug	0 The assertion of the ipg_debug input is ignored. 1 The assertion of the ipg_debug input causes the DMA 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.

7.3.1.2 DMA Error Status Register (DMA_ESR)

The DMA_ESR provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively. In fixed arbitration mode, a configuration error is caused by any two channel priorities being equal within a group, or any group priority levels being equal among the groups. All channel priority levels within a group must be unique and all group priority levels among the groups must be unique when fixed arbitration mode is enabled. If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (dlast_sga) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD.citer.e_link bit does not equal the TCD.biter.e_link bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and assert an error interrupt request, if enabled. 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 DMA 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.

The occurrence of any type of error causes the DMA engine to immediately stop, and the appropriate channel bit in the DMA Error register to be asserted. At the same time, the details of the error condition are loaded into the DMA_ESR. The major loop complete indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are *not* affected when an error is detected. See *Figure 17* and *Table 20* for the DMA_ESR definition.

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPE	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 17. DMA Error Status Register (DMA_ESR)

Table 20. DMA_ESR field descriptions

Name	Description	Value
VLD	Logical OR of all DMA_ERL status bits	0 No DMA_ERL bits are set. 1 At least one DMA_ERL bit is set indicating a valid error exists that has not been cleared.
GPE	Group Priority Error	0 No group priority error. 1 The last recorded error was a configuration error among the group priorities. All group priorities are not unique.
CPE	Channel Priority Error	0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities within a group. All channel priorities within a group are not unique.
ERRCHN[0:5]	Error Channel Number	The channel number of the last recorded error (excluding GPE and CPE errors).
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.

Table 20. DMA_ESR field descriptions (continued)

Name	Description	Value
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.

7.3.1.3 DMA Enable Request Register (DMA_ERQRL)

The DMA_ERQRL register provides a bitmap for the 32 implemented channels (0–31) 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 DMA_SERQR and DMA_CERQR registers. The DMA_SERQR and DMA_CERQR 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 DMA_ERQRL register.

Both the DMA request input signal and the enable request flag must be asserted before a channel's hardware service request is accepted. The state of the DMA enable request flag does *not* affect a channel service request made explicitly through software or a linked channel request. See *Figure 18* and *Table 21* for the DMA_ERQRL definition.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERQ 31	ERQ 30	ERQ 29	ERQ 28	ERQ 27	ERQ 26	ERQ 25	ERQ 24	ERQ 23	ERQ 22	ERQ 21	ERQ 20	ERQ 19	ERQ 18	ERQ 17	ERQ 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	ERQ 15	ERQ 14	ERQ 13	ERQ 12	ERQ 11	ERQ 10	ERQ 09	ERQ 08	ERQ 07	ERQ 06	ERQ 05	ERQ 04	ERQ 03	ERQ 02	ERQ 01	ERQ 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 18. DMA Enable Request (DMA_ERQRL) Registers

Table 21. DMA_ERQRL field descriptions

Name	Description	Value
ERQn, n = 0,... 15 n = 0,... 31	Enable DMA Request n	0 The DMA request signal for channel n is disabled. 1 The DMA request signal for channel n is enabled.

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the DMA_ERQRL bit for that channel. If the TCD.d_req bit is set, then the corresponding DMAERQ bit is cleared, disabling the DMA request; else if the d_req bit is cleared, the state of the DMAERQ bit is unaffected.

7.3.1.4 DMA Enable Error Interrupt Register (DMA_EEIRL)

The DMA_EEIRL register provides a bitmap for the 32 implemented channels to enable the error interrupt signal for each channel (31 – 00). 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 DMA_SEEIR and DMA_CEEIR registers. DMA_SEEIR and DMA_CEEIR 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 DMA_EEIRL register.

Both the DMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted. See *Figure 19* and *Table 22* for the DMAEEI definition.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EEI 31	EEI 30	EEI 29	EEI 28	EEI 27	EEI 26	EEI 25	EEI 24	EEI 23	EEI 22	EEI 21	EEI 20	EEI 19	EEI 18	EEI 17	EEI 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	EEI 15	EEI 14	EEI 13	EEI 12	EEI 11	EEI 10	EEI 09	EEI 08	EEI 07	EEI 06	EEI 05	EEI 04	EEI 03	EEI 02	EEI 01	EEI 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 19. DMA Enable Error Interrupt (DMA_EEIRL) Register

Table 22. DMA_EEIRL field descriptions

Name	Description	Value
EEIn, n = 0,... 15 n = 0,... 31 n = 0,... 63	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.

7.3.1.5 DMA Set Enable Request Register (DMA_SERQR)

The DMA_SERQR provides a simple memory-mapped mechanism to set a given bit in the DMA_ERQRL register to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the DMA_ERQRL 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 DMA_ERQRL to be asserted. Reads of this register return all zeroes. See *Figure 20* and *Table 23* for the DMA_SERQR definition.

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SERQ[0:6]						
Reset		0	0	0	0	0	0	0

= Unimplemented

Figure 20. DMA Set Enable Request Register (DMA_SERQR)

Table 23. DMA_SERQR field descriptions

Name	Description	Value
SERQ[0:6]	Set Enable Request	0–63 Set the corresponding bit in DMA_ERQRL 64–127 Set all bits in DMA_ERQRL

7.3.1.6 DMA Clear Enable Request Register (DMA_CERQR)

The DMA_CERQR provides a simple memory-mapped mechanism to clear a given bit in the DMA_ERQRL register to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the DMA_ERQRL 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 DMA_ERQRL to be zeroed, disabling all DMA request inputs. Reads of this register return all zeroes. See *Figure 21* and *Table 24* for the DMA_CERQR definition.

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CERQ[0:6]						
Reset		0	0	0	0	0	0	0

= Unimplemented

Figure 21. DMA Clear Enable Request Register (DMA_CERQR)

Table 24. DMA_CERQR field descriptions

Name	Description	Value
CERQ[0:6]	Clear Enable Request	0–63 Clear corresponding bit in DMA_ERQRL 64–127 Clear all bits in DMA_ERQRL

7.3.1.7 DMA Set Enable Error Interrupt Register (DMA_SEEIR)

The DMA_SEEIR provides a simple memory-mapped mechanism to set a given bit in the DMA_EEIRL register to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMA_EEIRL 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 DMA_EEIRL to be asserted. Reads of this register return all zeroes. See *Figure 22* and *Table 25* for the DMA_SEEIR definition.

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SEEI[0:6]						
Reset		0	0	0	0	0	0	0

= Unimplemented

Figure 22. DMA Set Enable Error Interrupt Register (DMA_SEEIR)

Table 25. DMA_SEEIR field descriptions

Name	Description	Value
SEEI[0:6]	Set Enable Error Interrupt	0–63 Set the corresponding bit in DMA_EEIRL 64–127 Set all bits in DMA_EEIRL

7.3.1.8 DMA Clear Enable Error Interrupt Register (DMA_CEEIR)

The DMA_CEEIR provides a simple memory-mapped mechanism to clear a given bit in the DMA_EEIRL register to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMA_EEIRL 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 DMA_EEIRL to be zeroed, disabling all DMA request inputs. Reads of this register return all zeroes. See *Figure 23* and *Table 26* for the DMA_CEEIR definition.

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CEEI[0:6]						
Reset		0	0	0	0	0	0	0

= Unimplemented

Figure 23. DMA Clear Enable Error Interrupt Register (DMA_CEEIR)

Table 26. DMA_CEEIR field descriptions

Name	Description	Value
CEEI[0:6]	Clear Enable Error Interrupt	0–63 Clear corresponding bit in DMA_EEIRL 64–127 Clear all bits in DMA_EEIRL

7.3.1.9 DMA Clear Interrupt Request Register (DMA_CIRQR)

The DMA_CIRQR provides a simple memory-mapped mechanism to clear a given bit in the DMA_IRQRL register to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the DMA_IRQRL 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 DMA_IRQRL to be zeroed, disabling all DMA interrupt requests. Reads of this register return all zeroes. See *Figure 24* and *Table 27* for the DMA_CIRQR definition.

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CINT[0:6]						
Reset		0	0	0	0	0	0	0

= Unimplemented

Figure 24. DMA Clear Interrupt Request Register (DMA_CIRQR)

Table 27. DMA_CIRQR field descriptions

Name	Description	Value
CINT[0:6]	Clear Interrupt Request	0–63 Clear the corresponding bit in DMA_IRQRL 64-127 Clear all bits in DMA_IRQRL

7.3.1.10 DMA Clear Error Register (DMA_CER)

The DMA_CER provides a simple memory-mapped mechanism to clear a given bit in the DMA_ERL register to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the DMA_ERL 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 DMA_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes. See *Figure 25* and *Table 28* for the DMA_CER definition.

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CERR[0:6]						
Reset		0	0	0	0	0	0	0

= Unimplemented

Figure 25. DMA Clear Error Register (DMA_CER)

Table 28. DMA_CER field descriptions

Name	Description	Value
CERR[0:6]	Clear Error Indicator	0–63 Clear corresponding bit in DMA_ERL 64-127 Clear all bits in DMA_ERL

7.3.1.11 DMA Set START Bit Register (DMA_SSBR)

The DMA_SSBR provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding Transfer Control Descriptor to be set. 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. Reads of this register return all zeroes. See *Table 42* for the TCD START bit definition.

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SSRT[0:6]						
Reset		0	0	0	0	0	0	0

= Unimplemented

Figure 26. DMA Set START Bit Register (DMA_SSBR)

Table 29. DMA_SSBR field descriptions

Name	Description	Value
SSRT[0:6]	Set START Bit (Channel Service Request)	0–63 Set the corresponding channel's TCD.start 64–127 Set all TCD.start bits

7.3.1.12 DMA Clear DONE Status Bit Register (DMA_CDSBR)

The DMA_CDSBR provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding Transfer Control Descriptor to be cleared. 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. Reads of this register return all zeroes. See *Table 42* for the TCD DONE bit definition.

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CDNE[0:6]						
Reset		0	0	0	0	0	0	0

= Unimplemented

Figure 27. DMA Clear DONE Status Bit Register (DMA_CDSBR)

Table 30. DDMA_CDSBR field descriptions

Name	Description	Value
CDNE[0:6]	Clear DONE Status Bit	0–63 Clear the corresponding channel's DONE bit 64–127 Clear all TCD DONE bits

7.3.1.13 DMA Interrupt Request (DMA_IRQRL)

The DMA_IRQRL register provides a bitmap for the 32 implemented channels signaling the presence of an interrupt request for each channel (31 – 00). The DMA 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 DMA_CIRQR 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 DMA_CIRQR register. On writes to the DMAINT, 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 DMA_CIRQR 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 DMA_IRQRL register. See *Figure 28* and *Table 31* for the DMAINT definition.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INT 31	INT 30	INT 29	INT 28	INT 27	INT 26	INT 25	INT 24	INT 23	INT 22	INT 21	INT 20	INT 19	INT 18	INT 17	INT 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	INT 15	INT 14	INT 13	INT 12	INT 11	INT 10	INT 09	INT 08	INT 07	INT 06	INT 05	INT 04	INT 03	INT 02	INT 01	INT 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 28. DMA Interrupt Request Register (DMA_IRQRL)

Table 31. DMA_IRQRL field descriptions

Name	Description	Value
INTn, n = 0,... 15 n = 0,... 31 n = 0,... 63	DMA Interrupt Request n	0 The interrupt request for channel n is cleared. 1 The interrupt request for channel n is active.

7.3.1.14 DMA Error (DMA_ERL) Register

The DMA_ERL register provides a bitmap for the 32 implemented channels signaling the presence of an error for each channel (31 – 00). The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the DMAEEI register, then logically summed across groups of 16, 32 and 64 channels to form several group error interrupt requests which is then routed to the platform's interrupt controller. During the execution of the interrupt service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the DMA_CER register in the interrupt service routine is used for this purpose. Recall the normal DMA channel completion indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are *not* affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the DMAEEI 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 DMA_CER register. On writes to the DMAERR, 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 DMA_CER register is provided so the error indicator for a *single* channel can easily be cleared. See *Figure 29* and *Table 32* for the DMAERR definition.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERR 31	ERR 30	ERR 29	ERR 28	ERR 27	ERR 26	ERR 25	ERR 24	ERR 23	ERR 22	ERR 21	ERR 20	ERR 19	ERR 18	ERR 17	ERR 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 15	ERR 14	ERR 13	ERR 12	ERR 11	ERR 10	ERR 09	ERR 08	ERR 07	ERR 06	ERR 05	ERR 04	ERR 03	ERR 02	ERR 01	ERR 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 29. DMA Error (DMA_ERL) Registers

Table 32. DMA_ERL field descriptions

Name	Description	Value
ERRn, n = 0,... 15 n = 0,... 31 n = 0,... 63	DMA Error n	0 An error in channel n has not occurred. 1 An error in channel n has occurred.

7.3.1.15 DMA Channel n Priority (DCHPRIn), n = 0,...,31

When the fixed-priority channel arbitration mode is enabled (DMA_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel within a group. The channel priorities are evaluated by numeric value, that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. 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. When read, the GRPPRI bits of the DMA_CPRn register reflect the current priority level of the group of channels in which the corresponding channel resides. GRPPRI bits are not affected by writes to the DMA_CPRn registers. The group priority is assigned in the DMA_CR register. See *Figure 16* and *Table 19* for the DMA_CR definition.

Channel preemption is enabled on a per channel basis by setting the ECP bit in the DMA_CPRn register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all 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 both group and channel arbitration modes. See *Figure 30* and *Table 33* for the DMA_CPRn definition.

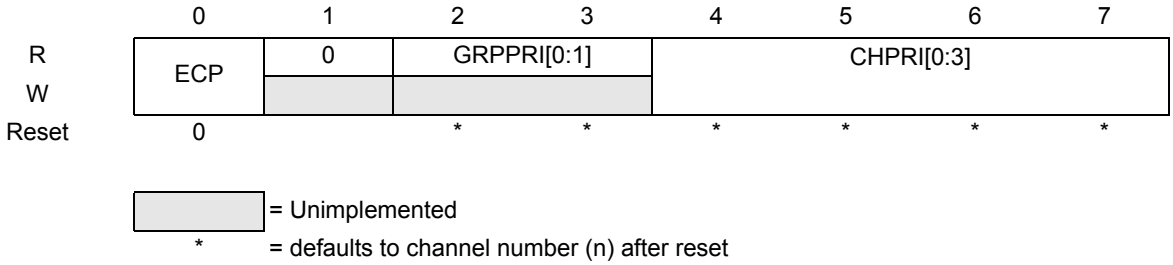


Figure 30. DMA Channel n Priority (DMA_CPRn) Register

Table 33. DMA_CPRn field descriptions

Name	Description	Value
ECP	Enable Channel Preemption	0 Channel n cannot be suspended by a higher priority channel's service request. 1 Channel n can be temporarily suspended by the service request of a higher priority channel.
GRPPRI[0:1]	Channel n Current Group Priority	Group priority assigned to this channel group when fixed-priority arbitration is enabled. These two bits are read only; writes are ignored.
CHPRI[0:3]	Channel n Arbitration Priority	Channel priority when fixed-priority arbitration is enabled.

7.3.1.16 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 7.2.2: 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 34* is a 32-bit view of the basic TCD structure.

There are two configurations for the TCD structure:

1. The standard TCD format is defined by the following conditions:
   - TCD[x].CITER.E_LINK = 0 (see TCD Word 5 details)
     and
   - BITER.E_LINK = 0 (see TCD Word 7 details)
     and
   - DMA_CR[EMLM] = 0 (see DMA_CR register details)
2. The alternate format is selected by the following conditions:
   - TCD[x].CITER.E_LINK = 1 (see TCD Word 5 details)
     and
   - BITER.E_LINK = 1 (see TCD Word 7 details)

Note: The TCD structures are similar. The differences appear in Word5 and Word 7 of the TCD and are noted in the following descriptions.

Table 34. TCDn 32-bit memory structure

DMA 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	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 31 and *Table 35* define word 0 of the TCDn structure, the saddr field.

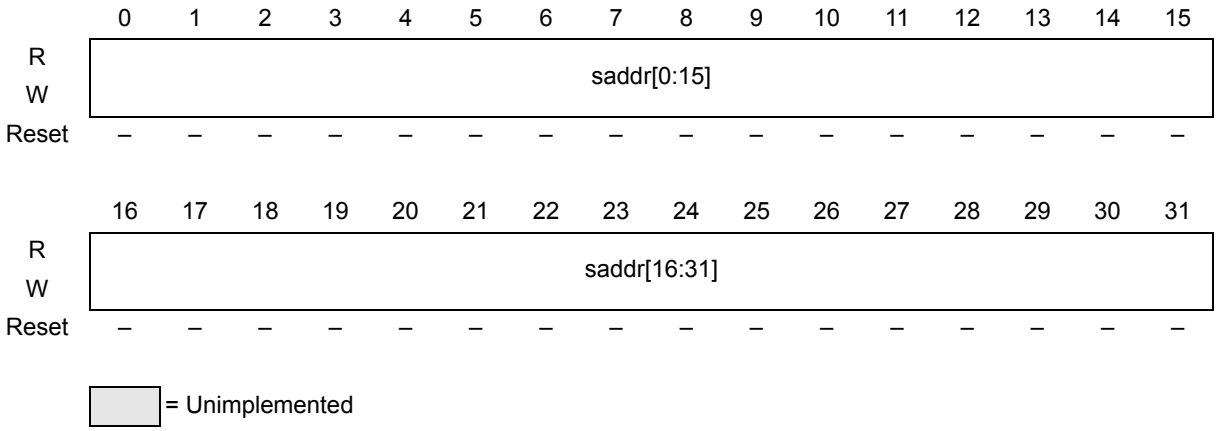


Figure 31. TCDn Word 0 (TCDn.saddr) field

Table 35. TCDn Word 0 (TCDn.saddr) field description

Name	Description	Value
saddr[0:31]	Source address	Memory address pointing to the source data

Figure 32 and *Table 36* define word 1 of the TCDn structure, the soff and transfer attribute fields.

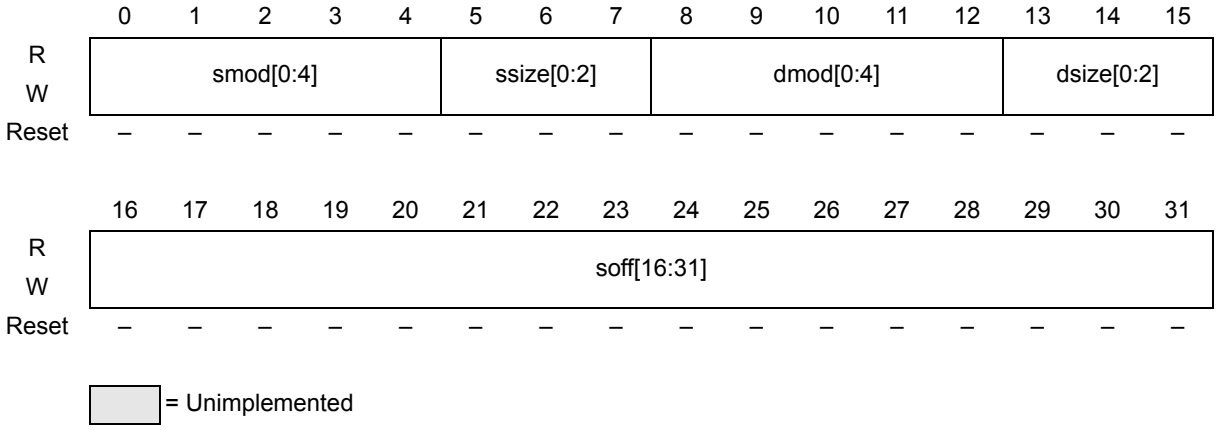


Figure 32. TCDn Word 1 (TCDn.{soff,smod,ssize,dmod,dsize}) fields

Table 36. TCDn Word 1 (TCDn.{smod,ssize,dmod,dsize,soff}) field descriptions

Name	Description	Value
smod[0:4]	Source address modulo	0 Source 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.

Table 36. TCDn Word 1 (TCDn.{smod,ssize,dmod,dsize,soff}) field descriptions (continued)

Name	Description	Value
ssize[0:2]	Source data transfer size	000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 16-byte (32-bit implementations only) 101 32-byte (if supported by the platform) 110 Reserved 111 Reserved The attempted specification of a 64-bit source size in a 32-bit AMBA AHB bus implementation produces a configuration error. Likewise, the attempted specification of a 16-byte source size in a 64-bit AMBA AHB bus implementation generates a configuration error. The attempted specification of a 32-byte burst on platforms that do not support such a transfer type will result in a configuration error.
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 33 and *Table 37* define word 2 of the TCDn structure, the nbytes field

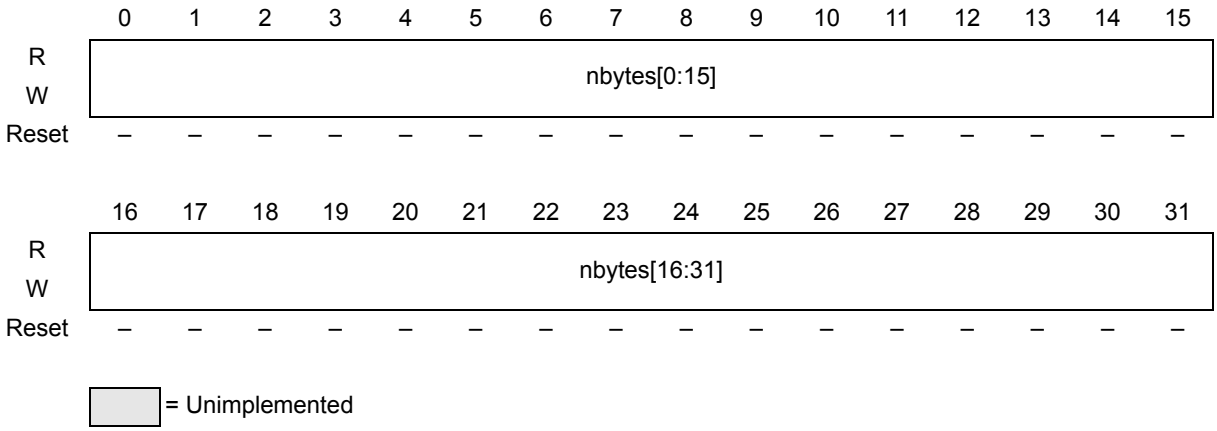


Figure 33. TCDn Word 2 (TCDn.nbytes) field

Table 37. 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 DMA 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.

Figure 34 and *Table 38* define word 3 of the TCDn structure, the slast field.

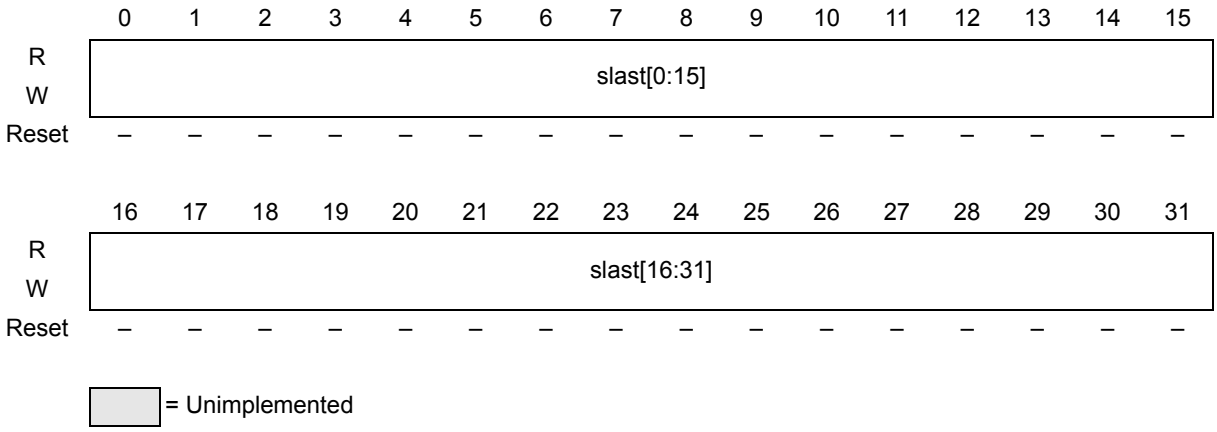


Figure 34. TCDn Word 3 (TCDn.slast) field

Table 38. 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 35 and *Table 39* define word 4 of the TCDn structure, the daddr field.

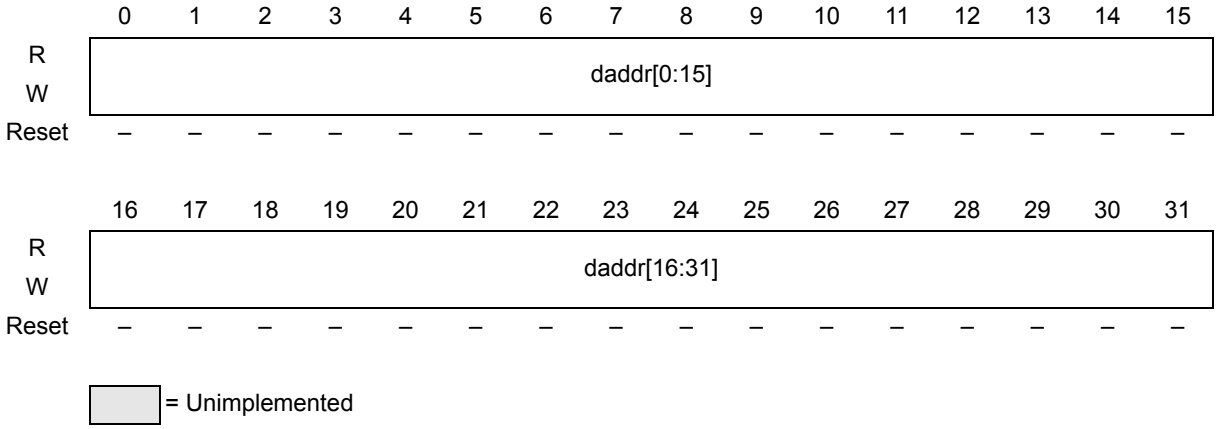


Figure 35. TCDn Word 4 (TCDn.daddr) field

Table 39. TCDn Word 4 (TCDn.daddr) field description

Name	Description	Value
daddr[0:31]	Destination address	Memory address pointing to the destination data.

Figure 36 and *Table 40* define word 5 of the TCDn structure, the citer and doff fields.

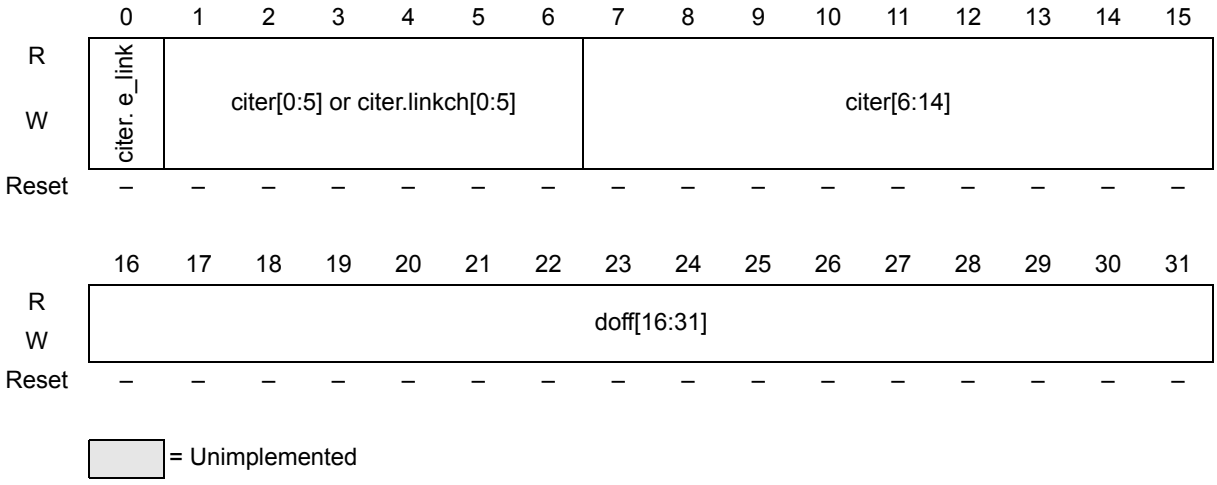


Figure 36. TCDn Word 5 (TCDn.{citer,doff}) fields

Table 40. 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 DMA 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.

Table 40. 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 37 and *Table 41* define word 6 of the TCDn structure, the dlast_sga field.

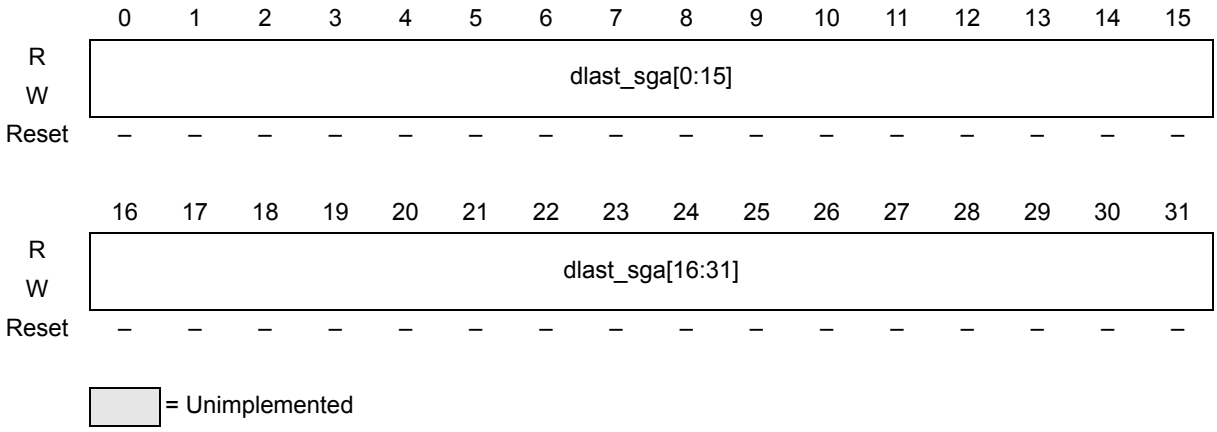


Figure 37. TCDn Word 6 (TCDn.dlast_sga) field

Table 41. TCDn Word 6 (TCDn.dlast_sga) field description

Name	Description	Value

dlast_sga[0:31]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather)	if (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 38 and *Table 42* define word 7 of the TCDn structure, the biter and control/status fields.

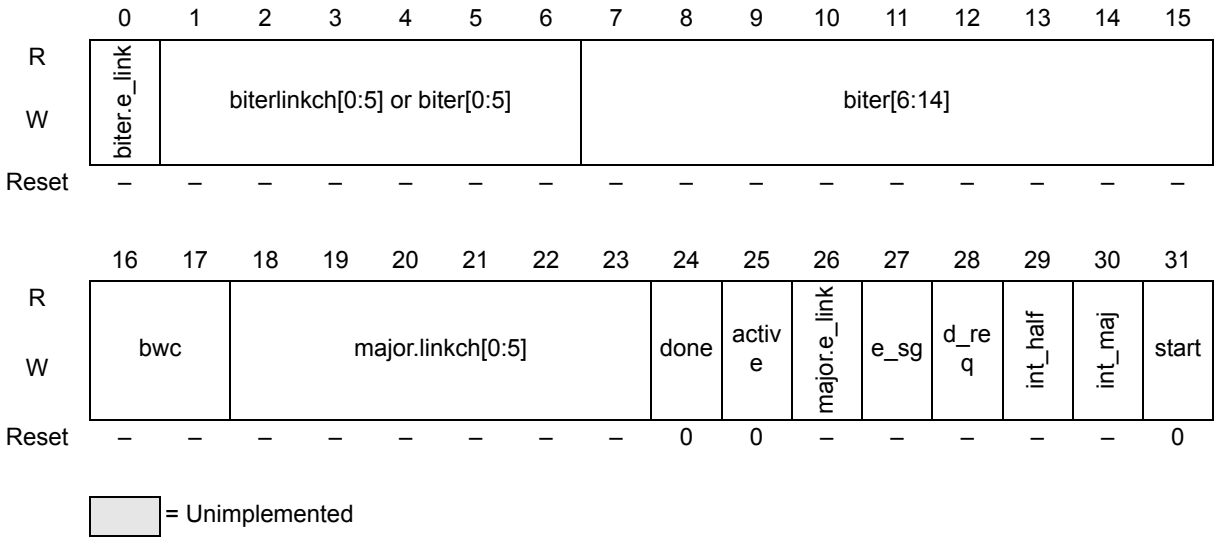


Figure 38. TCDn Word 7 (TCDn.{biter,control/status}) fields

Table 42. 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 DMA 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.

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 42. 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 DMA. In general, as the DMA processes the inner minor loop, it continuously generates read/write, read/write, ... sequences until the minor count is exhausted. This field forces the DMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the platform's crossbar 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 DMA as seen by the crossbar arbitration switch for the executing channel. Dynamic priority elevation is suppressed during the first two AHB bus cycles. 00 No DMA engine stalls 01 Dynamic priority elevation 10 DMA engine stalls for 4 cycles after each r/w 11 DMA 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 DMA 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.

done	Channel done	This flag indicates the DMA has completed the outer major loop. It is set by the DMA engine as the citer count reaches zero; it is cleared by software, or 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 DMA engine as the inner minor loop completes or if any error condition is detected.

Table 42. 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 DMA engine uses dlast_sga as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. *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 DMA 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 DMA 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. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.

Table 42. 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 DMA 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.

7.4 Functional description

This section provides an overview of the microarchitecture and functional operation of the DMA module.

7.4.1 DMA microarchitecture

The DMA module is partitioned into two major modules: the DMA engine and the transfer control descriptor local memory. Additionally, the DMA engine is further partitioned into four submodules, which are detailed below.

- DMA 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 DMA_CPRn[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).

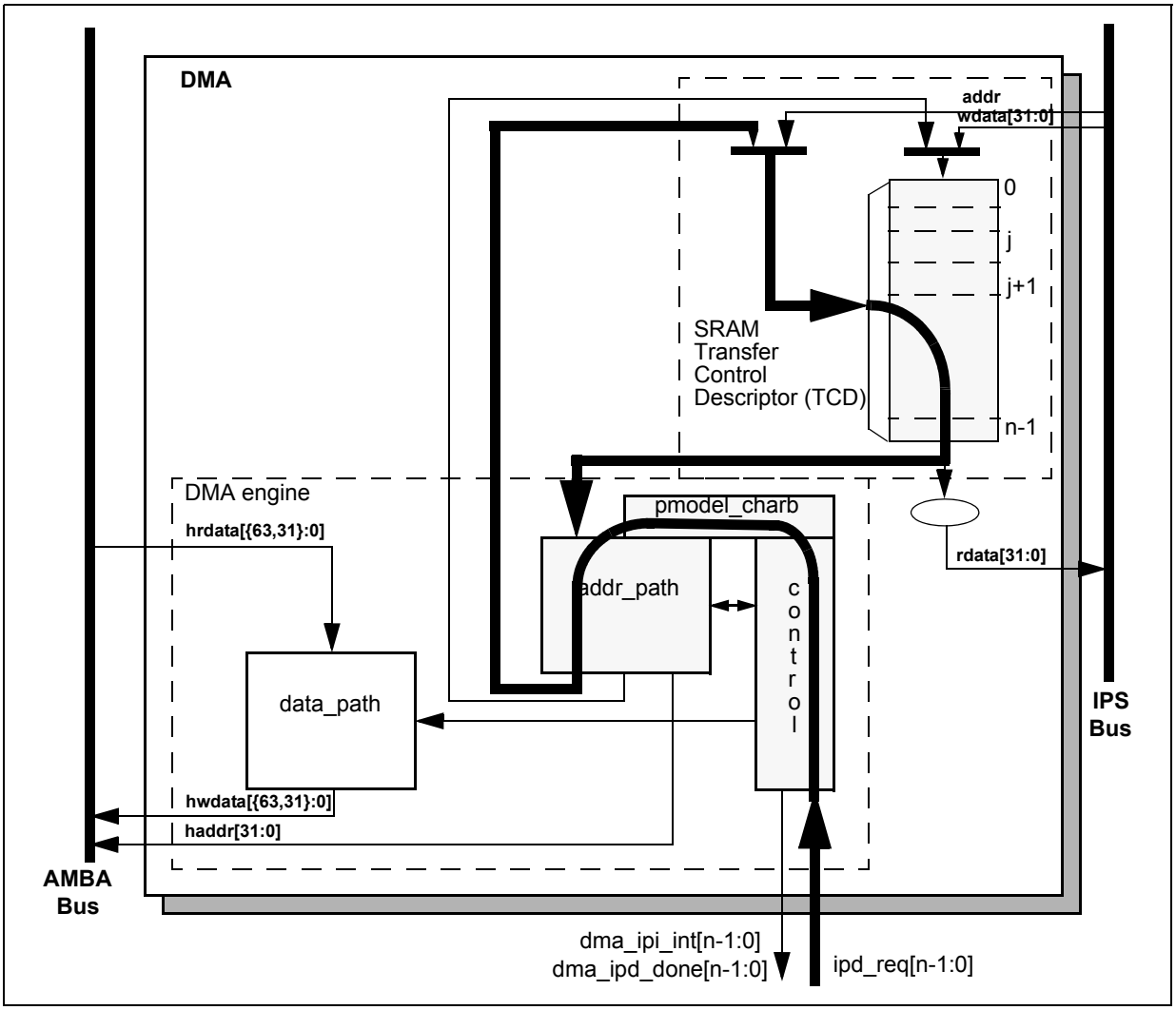
  - *pmodel_charb*: This module implements the first section of DMA's programming model as well as the channel arbitration logic. The programming model registers are connected to the IPS bus (not shown). The ipd_req[n] inputs and dma_ipi_int[n] outputs are also connected to this module (via the control logic).
  - *control*: This module provides all the control functions for the DMA engine. For data transfers where the source and destination sizes are equal, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner 'minor loop' byte count has been moved. 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 DMA engine as well as references from the IPS bus. As noted earlier, in the event of simultaneous accesses, the DMA 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

7.4.2 DMA basic data flow

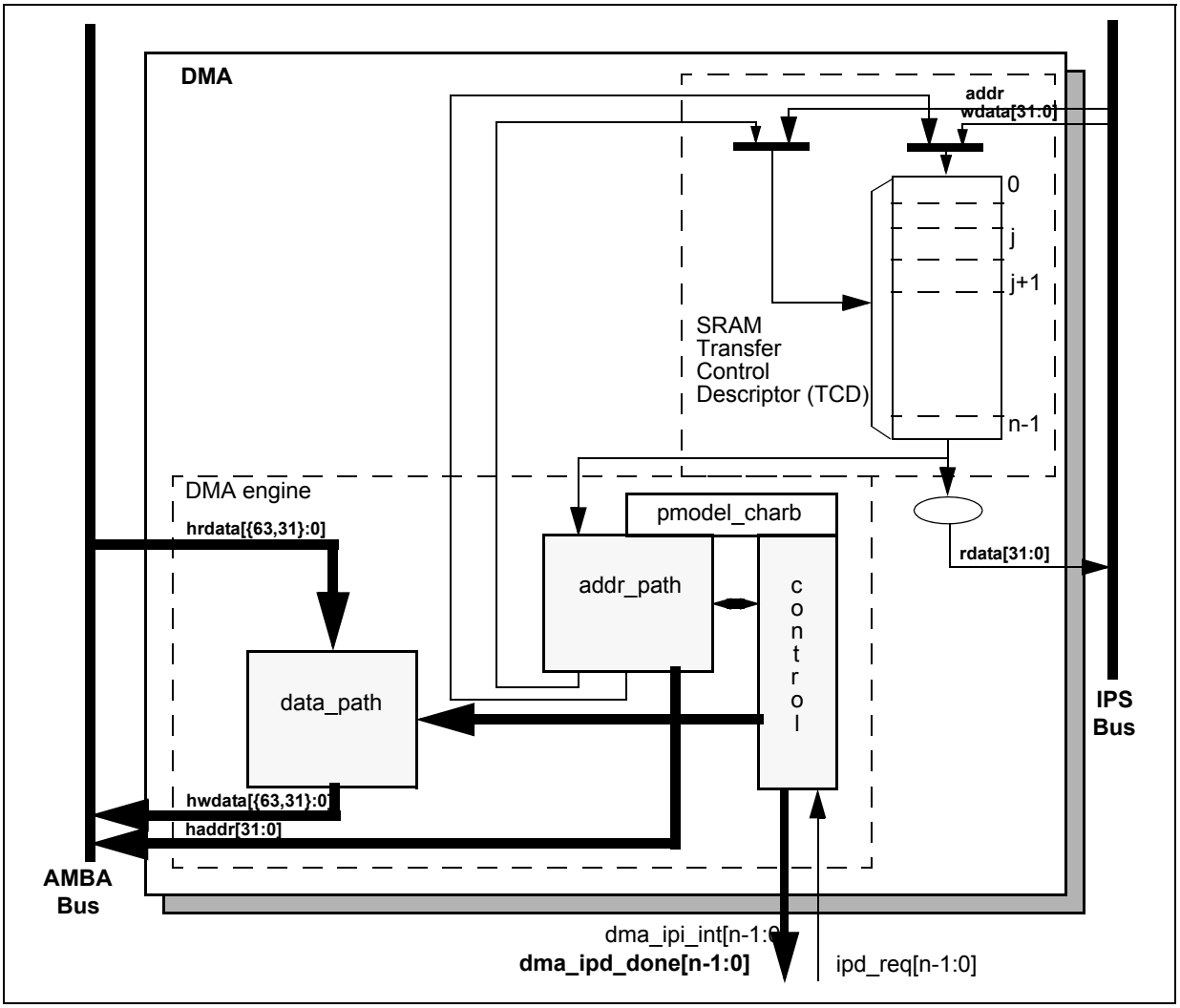
The basic flow of a data transfer can be partitioned into three segments. As shown in *Figure 39*, the first segment involves the channel service request. In the diagram, this example uses the assertion of the ipd_req[n] signal to request service for channel n. Channel service request via software and 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 DMA 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 39. DMA operation, part 1



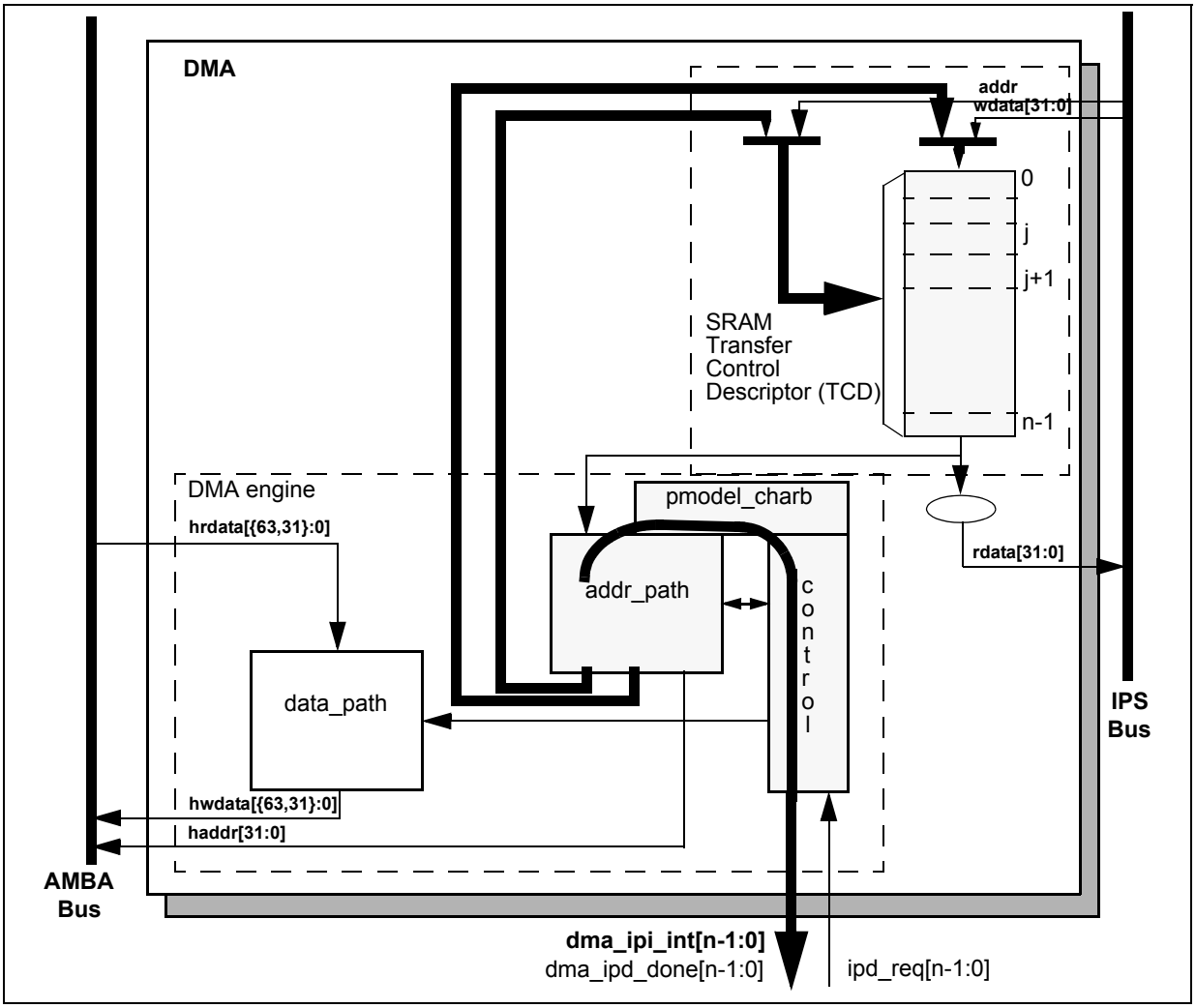
In the second part of the basic data flow as shown in *Figure 40*, the modules associated with the data transfer (addr_path, data_path and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data 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 40. DMA 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 41*.

Figure 41. DMA operation, part 3



7.4.3 DMA performance

This section addresses the performance of the DMA 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 DMA. 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 DMA also factors significantly into the resulting metric.

The peak transfer rates for several different source and destination transfers are shown in *Table 43*. The following assumptions apply to *Table 43* and *Table 44*:

- 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 43 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, that is, 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 43. DMA 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 DMA 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 DMA 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 DMA 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 DMA 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 DMA 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, DMA 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), DMA 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 44*. This metric represents millions of requests per second.

Table 44. DMA peak request rate [MReq/s]

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/s

For an IPS to SRAM transfer:

PEAKreq = 150 MHz ÷ [4 + (1 + 2) + (1 + 1) + 3] cycles = 12.5 Mreq/s

Assuming an even distribution of the two transfer types, the average Peak Request Rate would be:

PEAKreq = (11.5 Mreq/s + 12.5 Mreq/s) ÷ 2 = 12.0 Mreq/s

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, DMA 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.

7.5 Initialization/application information

7.5.1 DMA initialization

A typical initialization of the DMA is:

1. Write the DMA_CR register if a configuration other than the default is desired.
2. Write the channel priority levels into the DMA_CPRn 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 DMA 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 DMA engine's local TCD.saddr, TCD.daddr, and TCD.citer are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed, i.e. interrupts, major loop channel linking, and scatter/gather operations, if enabled.

7.5.2 DMA programming errors

The DMA performs various tests on the Transfer Control Descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of two errors; Group Priority Error and Channel Priority Error, GPE and CPE in the DMA_ESR register respectively.

For all error types other than Group or Channel Priority Errors, the channel number causing the error is recorded in the DMA_ESR register. If the error source is not removed before the next activation of the problem channel, the error will be detected and recorded again.

The sequence listed below is correct. For item 2, the dma_ipd_ack{done} lines will assert only if the selected channel is requesting service via the ipd_req signal. I think the typical application will enable error interrupts for all channels. So the user will get an error interrupt, but the channel number for the DMAERR register and the error interrupt request line may be wrong because they reflect the selected channel.

Channel priority errors are identified within a group after that group has been selected as the active group. For example:

1. The DMA is configured for fixed group and fixed channel arbitration modes.
2. Group3 is the highest priority and all channels are unique in that group.
3. Group2 is the next highest priority and has two channels with the same priority level.
4. If Group3 has any service requests, those requests will be executed.
5. Once all of Group3 requests have completed, Group2 will be the next active group.
6. If Group2 has a service request, then an undefined channel in Group2 will be selected and a channel priority error will occur.
7. This will repeat until the all of Group2 requests have been removed or a higher priority Group3 request comes in.

A group priority error is global and any request in any group will cause a group priority error.

In general, if priority levels are not unique, the highest (channel/group) priority that has an active request will be selected, but the lowest numbered (channel/group) with that priority will be selected by arbitration and executed by the DMA engine. The hardware service request handshake signals, error interrupts and error reporting will be associated with the selected channel.

7.5.3 DMA arbitration mode considerations

7.5.3.1 Fixed group arbitration, fixed channel arbitration

In this mode, the channel service request from the highest priority channel in the highest priority group will be selected to execute. If the DMA is programmed so the channels within one group use “fixed” priorities, and that group is assigned the highest “fixed” priority of all groups, it is possible for that group to take all the bandwidth of the DMA controller—that is, no other groups will be serviced if there is always at least one DMA request pending on a channel in the highest priority group when the controller arbitrates the next DMA request.

The advantage of this scenario is that latency can be small for channels that need to be serviced quickly.

Preemption is available in this scenario only.

7.5.3.2 Round-robin group arbitration, fixed channel arbitration

The occurrence of one or more DMA requests from one or more groups, the channel with the highest priority from a specific group will be serviced first. Groups are serviced starting with the highest group number with an service request and rotating through to the lowest group number containing a service request.

Once the channel request is serviced, the group round robin algorithm will select the highest pending request from the next group in the round robin sequence. Servicing continues round robin, always servicing the highest priority channel in the next group in the sequence, or just skipping a group if it has no pending requests.

If a channel requests service at a rate that equals or exceeds the round robin service rate, then that channel will always be serviced before lower priority channels in the same group, and thus the lower priority channels will never be serviced.

The advantage of this scenario is that no one group uses all the DMA bandwidth.

The highest priority channel selection latency is potentially greater than fixed/fixed arbitration.

Excessive request rates on high priority channels can prevent the servicing of lower priority channels in the same group.

7.5.3.3 Round-robin group arbitration, round-robin channel arbitration

Groups will be serviced as described in *Section 7.5.3.2: Round-robin group arbitration, fixed channel arbitration*, but this time channels will be serviced in channel number order. Only one channel is serviced from each requesting group for each round robin pass through the groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to channel priority levels.

Because channels are serviced in round robin manner, any channel that generates DMA requests faster than a combination of the group round robin service rate and the channel service rate for its group will not prevent the servicing of other channels in its group. Any DMA requests that are not serviced are simply lost, but at least one channel will be serviced.

This scenario ensures that all channels will be guaranteed service at some point, regardless of the request rates. However, the potential latency could be quite high.

All channels are treated equally. Priority levels are not used in round robin/round robin mode.

7.5.3.4 Fixed group arbitration, round-robin channel arbitration

The highest priority group with a request will be serviced. Lower priority groups will be serviced if no pending requests exist in the higher priority groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned within the group.

This scenario could cause the same bandwidth consumption problem as indicated in *Section 7.5.3.1: Fixed group arbitration, fixed channel arbitration*, but all the channels in the highest priority group will be serviced.

Service latency will be short on the highest priority group, but can become much longer as the group priority decreases.

7.5.4 DMA transfer

7.5.4.1 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 DMA 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. DMA engine writes: TCD.done = 0, TCD.start = 0, TCD.active = 1
4. DMA 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. DMA engine writes: TCD.saddr = 0x1000, TCD.daddr = 0x2000, TCD.citer = 1 (TCD.biter)
7. DMA engine writes: TCD.active = 0, TCD.done = 1, DMAINT[n] = 1
8. The channel retires

The DMA goes idle or services next channel.

7.5.4.2 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 DMA is 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. DMA engine writes: TCD.done = 0, TCD.start = 0, TCD.active = 1
4. DMA 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. DMA engine writes: TCD.saddr = 0x1010, TCD.daddr = 0x2010, TCD.citer = 1
7. DMA engine writes: TCD.active = 0
8. The channel retires → *one iteration of the major loop*

The DMA goes idle or services next channel.

9. Second hardware (ipd_req) requests channel service
10. The channel is selected by arbitration for servicing
11. DMA engine writes: TCD.done = 0, TCD.start = 0, TCD.active = 1
12. DMA 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. DMA engine writes: TCD.saddr = 0x1000, TCD.daddr = 0x2000, TCD.citer = 2 (TCD.biter)
15. DMA engine writes: TCD.active = 0, TCD.done = 1, DMAINT[n] = 1
16. The channel retires → *major loop complete*

The DMA goes idle or services the next channel.

7.5.5 TCD status

7.5.5.1 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.

7.5.5.2 Active channel TCD reads

The DMA 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 DMA 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.

7.5.5.3 Preemption status

Preemption is only available when *fixed* arbitration is selected for *both* group and channel arbitration modes. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the DMA engine is not operating in fixed group, fixed channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel and/or group priorities are treated as equal (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)

7.5.6 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 DMA 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.

7.5.7 Dynamic programming

This section provides recommended methods to change the programming model during channel execution.

7.5.7.1 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 within a group, then change the channel priorities within that group only, then enable the appropriate channels.

The following two options are available for dynamically changing ***group*** priority levels:

1. Switch to round-robin group arbitration mode, change the group priorities, then switch back to fixed arbitration mode.
2. Disable ALL channels, change the group priorities, then enable the appropriate channels.

7.5.7.2 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 DMA 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.

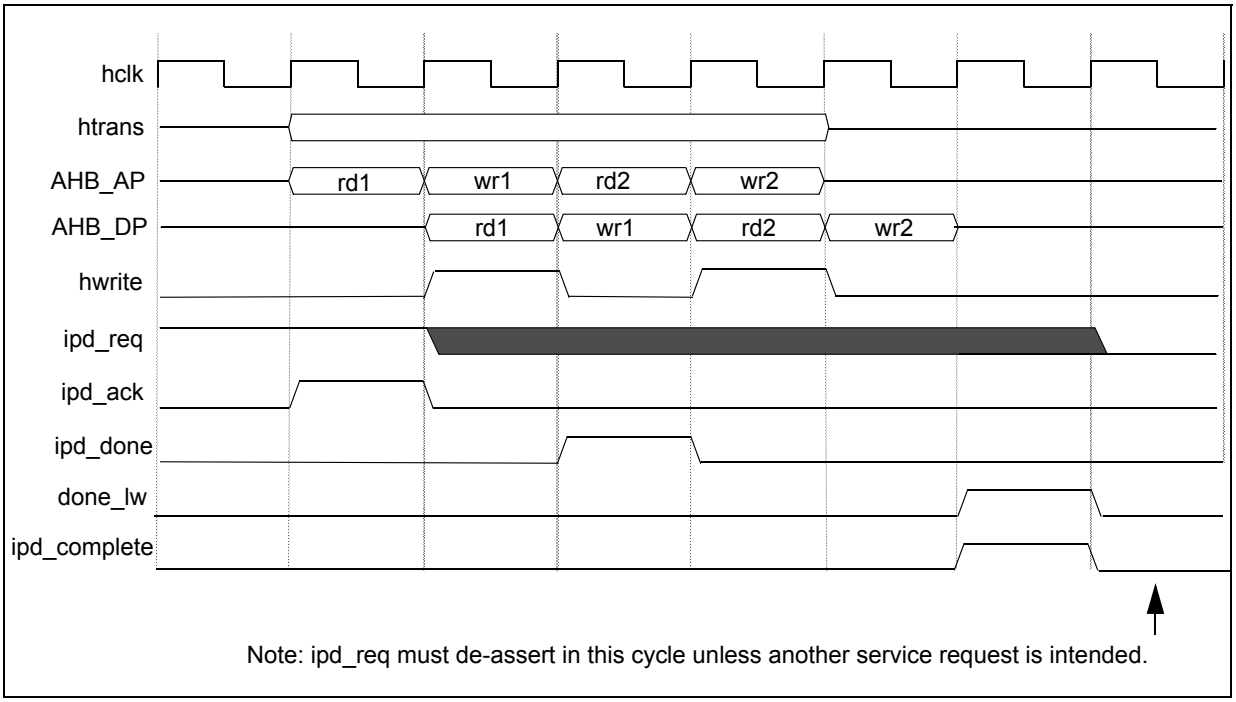
This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the TCD.major.e_link and TCD.e_sg bits to zero on any writes to a channel's TCD.word7 after that channel's TCD.done bit is set indicating the major loop is complete.

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 DMA engine after a channel begins execution.

7.5.8 Hardware request release timing

This section provides a timing diagram for deasserting the ipd_req hardware request signal. *Figure 42* shows two read write sequences with grey indicating the release of the ipd_req hardware request signal.

Figure 42. ipd_req hardware handshake



8 Multi-Layer AHB Crossbar Switch (XBAR)

8.1 Introduction

8.1.1 Overview

This section provides an overview of the multi-layer AHB crossbar switch (XBAR). The purpose of the XBAR is to concurrently support simultaneous connections between master ports and slave ports. The XBAR supports a 32-bit address bus width. Only a single data bus width is supported throughout the design, thus, all master and slave ports have the same data bus width.

The XBAR has four master ports and four slave ports. *Figure 43* shows a block diagram of the XBAR.

Figure 43. XBAR device-specific block diagram

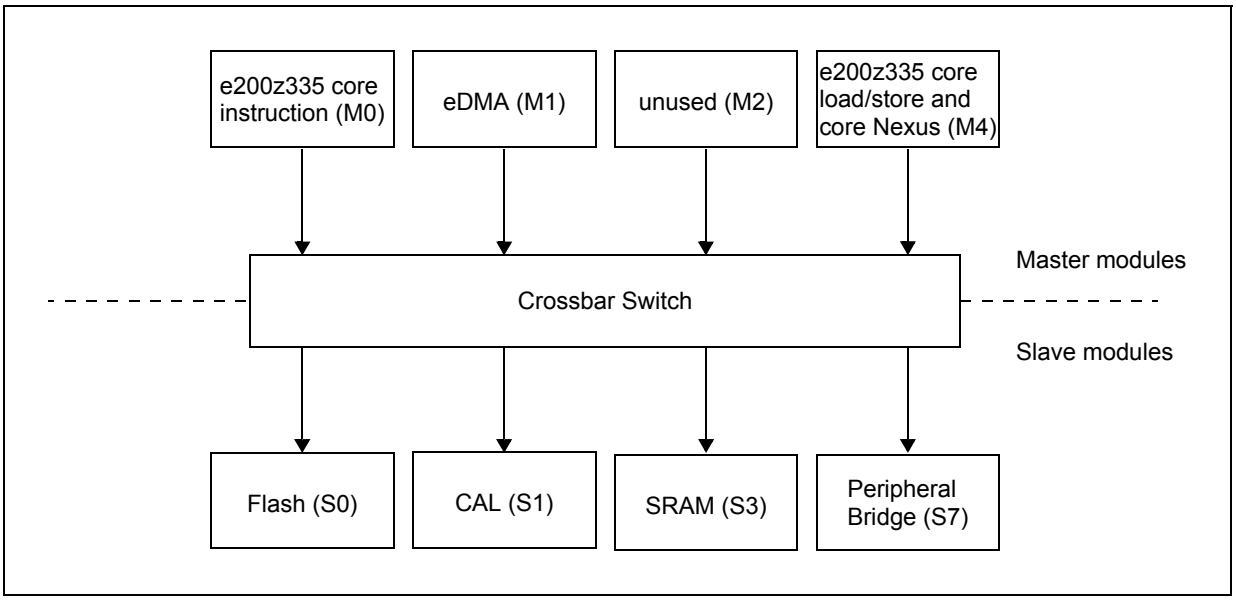


Table 45 shows the port mappings.

Table 45. Master / Slave connections

XBAR port	Module	Master ID
Master 0	e200z335 core instruction	0
Master 1	eDMA	2
Master 2	unused	—
Master 4	e200z335 core Load/Store	0
	e200z335 core Nexus	1
Slave 0	Flash memory	—
Slave 1	Calibration bus	—
Slave 3	SRAM	—

Table 45. Master / Slave connections (continued)

XBAR port	Module	Master ID
Slave 7	Peripheral bridge	—

8.1.2 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 for turning off the clocks to the system and ensuring that no bus activity will be interrupted.

The XBAR can put each slave port into a low power park mode so that the 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—the user can dynamically change master priority levels on a slave port by slave port basis.

The XBAR allows 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.

The XBAR has a 32-bit internal address bus and a 64-bit internal data bus.

8.1.3 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.

8.1.4 General operation

When a master makes an access to the XBAR the access will be immediately taken by the XBAR. If the targeted slave port of the access is available then the access will be immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR. If the targeted slave port of the access is busy or parked on a different master port the requesting master will simply see wait states inserted until the targeted slave port can service the master’s request. The latency in servicing the request depends 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 is 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 remains in control of that slave port until it gives up the slave port by running an IDLE cycle or by leaving that slave port for its next access. The master could also lose control of the slave port if another higher priority master makes a request to the slave port; however, if the master is running a locked or fixed length burst transfer it retains control of the slave port until that transfer is completed.

The XBAR will terminate all master IDLE transfers (as opposed to allowing the termination to come from one of the slave busses). Additionally, when no master is requesting access to a slave port the XBAR will drive IDLE transfers onto the slave bus, even though a default master may be granted access to the slave port.

When a slave bus is being IDLEd by the XBAR it can park the slave port on the master port indicated by the PARK bits in the SGPCR (Slave General Purpose Control Register). This can be done in an attempt to save the initial clock of arbitration delay that would otherwise be seen if the master had to arbitrate to gain control of the slave port. The slave port can also be put into low power park mode in attempt to save power.

8.2 XBAR registers

This section provides information on XBAR registers.

8.2.1 Register summary

There are two registers that reside in each slave 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 46*.

Table 46. XBAR register configuration summary

XBAR base offset	Register	Location	Use
XBAR_Base (0xFFF0_4000)	MPR0	*on page 203*	Master Priority Register for Slave port 0
XBAR_Base + 0x004 – XBAR_Base + 0x00F	—	—	Reserved
XBAR_Base + 0x010	SGPCR0	*on page 205*	General Purpose Control Register for Slave port 0
XBAR_Base + 0x014 – XBAR_Base + 0x0FF	—	—	Reserved
XBAR_Base + 0x100	MPR1	*on page 203*	Master Priority Register for Slave port 1

Table 46. XBAR register configuration summary (continued)

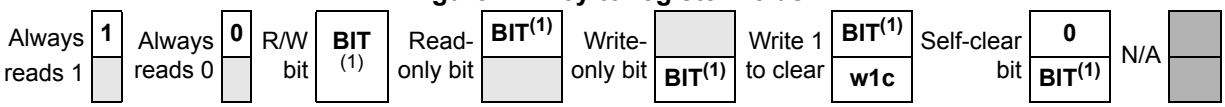
XBAR base offset	Register	Location	Use
XBAR_Base + 0x104 – XBAR_Base + 0x10F	—	—	Reserved
XBAR_Base + 0x110	SGPCR1	*on page 205*	General Purpose Control Register for Slave port 1
XBAR_Base + 0x114 – XBAR_Base + 0X2FF	—	—	Reserved
XBAR_Base + 0x300	MPR3	*on page 203*	Master Priority Register for Slave port 3
XBAR_Base + 0x304 – XBAR_Base + 0x30F	—	—	Reserved
XBAR_Base + 0x310	SGPCR3	*on page 205*	General Purpose Control Register for Slave port 3
XBAR_Base + 0x214 – XBAR_Base + 0X6FF	—	—	Reserved
XBAR_Base + 0x700	MPR7	*on page 203*	Master Priority Register for Slave port 7
XBAR_Base + 0x704 – XBAR_Base + 0X70F	—	—	Reserved
XBAR_Base + 0x710	SGPCR7	*on page 205*	General Purpose Control Register for Slave port 7
XBAR_Base + 0x714 – XBAR_Base + 0XF03	—	—	Reserved

8.2.2 XBAR register descriptions

The following paragraphs provide detailed descriptions of the various XBAR registers.

Refer to *Figure 44* for the various bit configurations that appear in the register maps.

Figure 44. Key to register fields



1. "BIT" refers to a field name in the register. Some fields span multiple bits.

8.2.2.1 Master Priority Register (XBAR_MPRn)

The Master Priority Register (MPR) resides in each slave port and sets the priority of each master port on a per slave port basis, e.g., MPR0 sets priority for each master port for slave port 0.

MPR0: Address: XBAR_Base (0xFFF0_4000) + 0x0000
MPR1: Address: XBAR_Base (0xFFF0_4000) + 0x0100
MPR3: Address: XBAR_Base (0xFFF0_4000) + 0x0300
MPR7: Address: XBAR_Base (0xFFF0_4000) + 0x0700

Access: Supervisor

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	MSTR4		
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	0	MSTR2			0	MSTR1			0	MSTR0		
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0

= not implemented

Figure 45. Master Priority Register (XBAR_MPRn)

Table 47. XBAR Master Priority Register field descriptions

Field	Description
0–11	Reserved
12	Reserved This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
13–15 MSTR4	Master 4 Priority These bits set the arbitration priority for master port 4 (e200z335 core Load/Store and e200z335 core Nexus) on the associated slave port. These bits are initialized by hardware reset. The reset value is 011. 000 This master has the highest priority when accessing the slave port. ... 111 This master has the lowest priority when accessing the slave port.
16–19	Reserved
20	This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
21–23 MSTR2	Master 2 Priority These bits set the arbitration priority for master port 2 (unused) on the associated slave port. These bits are initialized by hardware reset. The reset value is 010. 000 This master has the highest priority when accessing the slave port. ... 111 This master has the lowest priority when accessing the slave port.
24	Reserved This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.

Table 47. XBAR Master Priority Register field descriptions (continued)

Field	Description
25–27 MSTR1	Master 1 Priority These bits set the arbitration priority for master port 1 (eDMA) on the associated slave port. These bits are initialized by hardware reset. The reset value is 001. 000 This master has the highest priority when accessing the slave port. ... 111 This master has the lowest priority when accessing the slave port.
28	Reserved This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
29–31 MSTR0	Master 0 Priority These bits set the arbitration priority for master port 0 (e200z4 core instruction bus) on the associated slave port. These bits are initialized by hardware reset. The reset value is 000. 000 This master has the highest priority when accessing the slave port. ... 111 This 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.

8.2.2.2 Slave General Purpose Control Register (XBAR_SGPCRn)

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 PCTL bits determine how the slave port will park when no master is actively making a request. The available options are to park on the master defined by the PARK bits, park on the last master to use the slave port, or go into a low power park mode which will force all the outputs of the slave port to inactive states when no master is requesting an access. The low power park feature can result in an overall power savings if a the slave port is not saturated; however, it will force an extra clock of latency whenever any master tries to access it when it is not in use because it will not be parked on any master.

The PARK bits determine which master the slave will park on when no master is making an active request. Please use caution to only select master ports that are actually present in the design. If the user programs the PARK bits to a master not present in the current design implementation undefined behavior will result.

Note: The SGPCR can only be accessed in supervisor mode with 32-bit accesses. Once XBAR_SGPCR[RO] has been set, the SGPCR can only be read; attempts to write to it will have no effect on the SGPCR and result in an error response.

SGPCR0: Address: XBAR_Base + 0X0010 + 0x0000 (0xFFF0_4010)
SGPCR1: Address: XBAR_Base + 0X0010 + 0x0100 (0xFFF0_4110)
SGPCR3: Address: XBAR_Base + 0X0010 + 0x0300 (0xFFF0_4310)
SGPCR7: Address: XBAR_Base + 0X0010 + 0x0700 (0xFFF0_4710)

Access: Supervisor

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RO	HLP	0	0	0	0	0	0	0	0	0	HPE4	0	HPE2	HPE1	HPE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

= not implemented

Figure 46. Slave General Purpose Control Register (XBAR_SGPCRn)

Table 48. XBAR Slave General Purpose Control Register field descriptions

Field	Description
0 RO	Read Only This bit is used to force all of a slave port's registers to be read only. Once written to '1' it can only be cleared by hardware reset. This bit is initialized by hardware reset. The reset value is 0. 0 All this slave port's registers can be written. 1 All this slave port's registers are read only and cannot be written (attempted writes have no effect and result in an error response).
1 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. 0 The max_halt_request input has the highest priority for arbitration on this slave port 1 The max_halt_request input has the lowest initial priority for arbitration on this slave port.
2–10	Reserved These bits are reserved for future expansion. They read as zero and should be written with zero for upward compatibility.
11–15 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. 0 The mX_high_priority input is disabled on this slave port 1 The mX_high_priority input is enabled on this slave port.
16–21	Reserved These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.

Table 48. XBAR Slave General Purpose Control Register field descriptions (continued)

Field	Description
22–23 ARB	Arbitration Mode These bits are used to select the arbitration policy for the slave port. These bits are initialized by hardware reset. The reset value is 00. 00 Fixed Priority 01 Round-Robin (rotating) Priority 10 Reserved 11 Reserved
24–25	Reserved These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.
26–27 PCTL	Parking Control These bits determine the parking control used by this slave port. These bits are initialized by hardware reset. The reset value is 00. 00 When no master is making a request the arbiter will park the slave port on the master port defined by the PARK bit field. 01 When no master is making a request the arbiter will park the slave port on the last master to be in control of the slave port. 10 When no master is making a request the arbiter will park the slave port on no master and will drive all outputs to a constant safe state. 11 Reserved
28	Reserved This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
29–31 PARK	PARK These bits are used to determine which master port this slave port parks on when no masters are actively making requests and the PCTL bits are set to 00. These bits are initialized by hardware reset. The reset value is 000. 000 Park on Master Port 0 (e200z335 core instruction) 001 Park on Master Port 1 (eDMA) 010 Park on Master Port 2 (unused) 011 Reserved 100 Park on Master Port 4 (e200z335 core Load/Store and Nexus) 101 Reserved 110 Reserved 111 Reserved

8.2.3 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.

8.3 Function

This section describes in more detail the functionality of the XBAR.

8.3.1 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.

8.3.1.1 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.

8.3.1.2 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).

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 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 occurs 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 is reset to point at master port 0, giving it the highest priority.

8.3.1.3 Parking

If no master is currently requesting the slave port, the slave port is parked. The slave port parks in one of three places, indicated by the value of field XBAR_SGPCR[PCTL].

- If park-on-specific master mode is selected, the slave port parks on the master designated by the PARK field. When the master accesses the slave port again, a one clock arbitration penalty is incurred only for an access request made by another master port to the slave port. No other arbitration penalties are incurred. All other masters pay a one clock penalty.
- If park-on-last (POL) mode is selected, then the slave port parks on the last master to access it, passing that master's signals through to the slave bus. When the master accesses the slave port again, no other arbitration penalties are incurred except that a one clock arbitration penalty is incurred for each access request to the slave port made by another master port. All other masters pay a one clock penalty.
- If the low-power-park (LPP) mode is selected, then the slave port enters low-power park mode. It is not under control by any master and does not transmit any master signals to the slave bus. All slave bus activity halts because all slave bus signals are not toggling. This saves power if the slave port is not used for some time. However, when a master does make a request to a slave port parked in low-power-park, a one clock arbitration delay is incurred to get ownership of the slave port.

8.3.2 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.

9 Peripheral Bridge (PBRIDGE)

9.1 Introduction

The Peripheral Bridge (PBRIDGE) provides an interface between the system crossbar switch bus and the lower-bandwidth peripheral bus.

Note: A single MMU entry can support all of the on-chip peripherals on the SPC563Mxx devices if the "Peripheral Bridge A" peripherals are addressed just below the "Peripheral Bridge B" address space. This can save one MMU entry of the 16 available on the device. There is actually only one peripheral bridge on the device.

9.2 PBRIDGE features

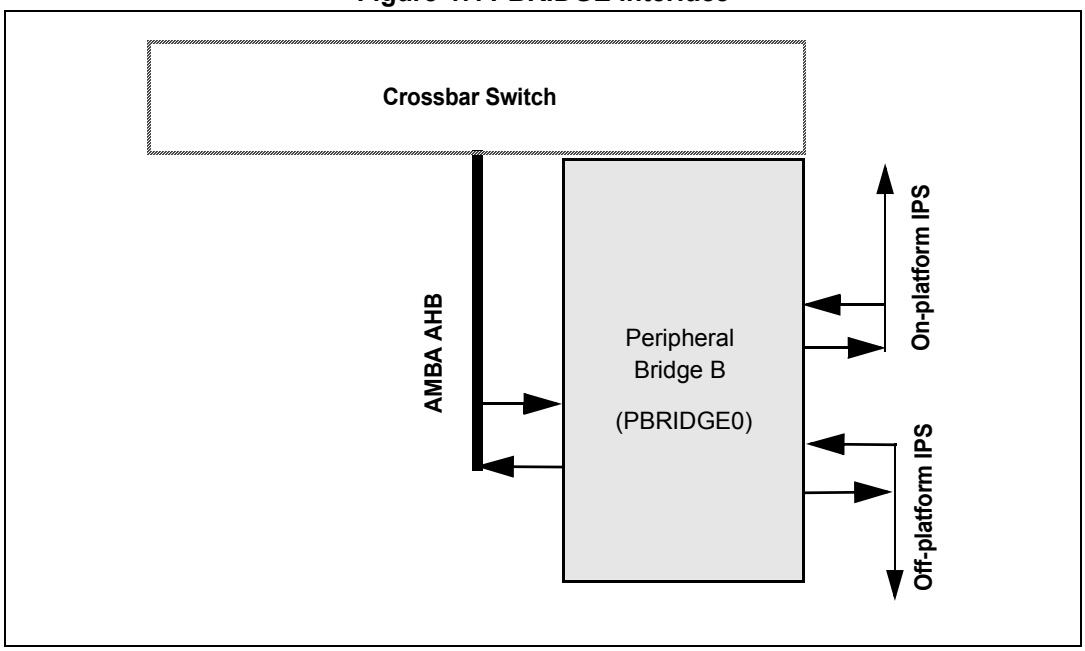
The PBRIDGE:

- Is only meant for slave peripherals
- Supports 32-bit IPS peripherals (byte, halfword, and word reads and write are supported to each)
- Supports a pair of IPS accesses for 64-bit fetches

9.3 PBRIDGE block diagram

The PBRIDGE is the interface between the system bus interface and on-chip peripherals as shown in *Figure 47*.

Figure 47. PBRIDGE interface



9.4 PBRIDGE signal description

The PBRIDGE has no external signals.

9.5 PBRIDGE functional description

The PBRIDGE functions as a protocol translator. Support is provided for generating a pair of 32-bit slave bus instruction accesses (not data accesses) when targeted by a 64-bit system bus access.

Accesses which fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices.

9.5.1 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. If the requested instruction access size is 64 bits, then a minimum of three clocks are required to complete the access. Misaligned read accesses are not supported. 64-bit data reads (not instruction) are not supported.

9.5.2 Write cycles

Three clock write 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. Misaligned writes that do not cross a 32-bit boundary are supported. 64-bit data writes (not instruction) are not supported.

9.6 PBRIDGE registers

The PBRIDGE does not contain any user-programmable registers.

10 C90LC Flash Memory

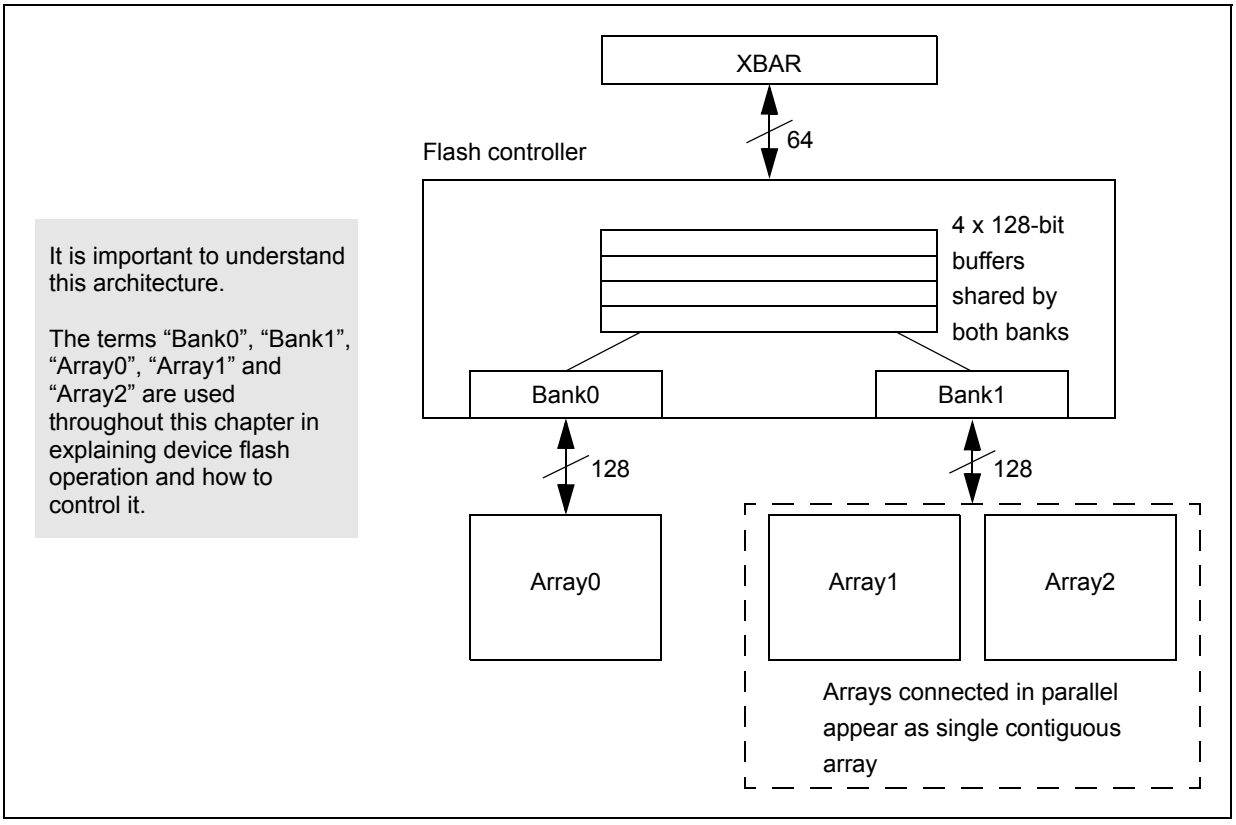
10.1 Overview

Flash memory on SPC563Mxx devices consists of a flash controller and up to three flash memory array modules. The configuration depends on the specific device, but all devices have one controller and at least two array modules arranged in two banks. *Figure 48* shows the architecture.

Recall from the SPC563Mxx block diagram that the Multilayer AHB Crossbar switch (XBAR) controls data flow between three master ports (CPU instruction, CPU data, and DMA) and four slave ports (External Bus Interface (EBI), flash memory, SRAM, and the peripheral bus). The flash controller is the interface between the memory modules and the XBAR.

Note: Throughout this document the Multilayer AHB Crossbar switch is variously referred to as XBAR, AHB Crossbar, AHB, or AMBA-AHB port. The terms are interchangeable.

Figure 48. SPC563Mxx flash block diagram



It is important to note the differences in memory configuration among the devices.

- Devices with 1.5 MB of flash memory are configured as shown *Figure 48*. The memory addresses are contiguous.
- Devices with 1 MB of flash memory have Array0 and Array1 only. There is no Array2. The memory addresses are contiguous.
- Devices with 768 KB of memory have Array0 and Array1 only. There is no Array2. The memory addresses are NOT contiguous. The address range uses the first 256 KB of Array0 and all of Array1.

Accessing read-only flash configuration information and controlling device flash behavior are accomplished using the flash control registers described later in this chapter. The flash controller and each flash array module have control registers.

- Changes to flash controller register values are global in nature—they affect flash behavior at the controller level.
- Changes to flash array module control register values affect only the behavior and configuration of the flash array they are attached to.

Read-While-Write (RWW) is supported between banks.

The remaining sections of this chapter give the functional details of the flash controller and flash arrays, followed by the memory maps and register descriptions.

You should have a clear understanding of at least the overview sections before attempting to program software to configure the flash controller or flash array modules.

10.2 Platform flash memory controller (PFLASH_LCA)

10.2.1 Overview

The flash controller (see *Figure 48*), also called PFLASH_LCA, supports a 64-bit data bus width at the AHB port and connections to 128-bit read data interfaces from two memory banks, where each bank contains one (or more) flash memory array modules.

The flash controller contains a four-entry page buffer, each entry containing 128 bits of data (one flash page) plus an associated prefetch controller that prefetches sequential lines of data from the flash array into the buffer. Page buffer hits support zero-wait AHB data phase responses. AHB read requests which miss the buffers generate the needed flash array access and data is forwarded to the AHB upon completion, typically incurring three wait-states at the maximum operating frequency.

The controller is optimized for applications where a cacheless processor core, e.g., the e200z335, is connected to on-chip memories where the processor and platform operate at the same frequency. For these applications, the 2-stage pipeline AMBA-AHB system bus is effectively mapped directly into stages of the processor's pipeline and zero wait-state responses for most memory accesses is critical for providing an acceptable level of system performance.

10.2.2 Features

The following list summarizes the key features of the Platform Flash Controller:

- 2 flash memory bank interfaces
- Single AHB port interface supports a 64-bit data bus. All AHB aligned and unaligned reads within the 64-bit container are supported. Only aligned word or doubleword writes are supported.(d)
- Array interfaces support a 128-bit read data bus and a 64-bit write data bus for each bank
- 4 page read buffers, each holding 128 bits of flash data, support single-cycle read responses (zero AHB data phase wait-states) on hits. The buffers implement a least-recently-used replacement algorithm to maximize performance.
- Bank interfaces provide configurable read buffering and page prefetch support, operating either as global control or separate per bank control
- Support of address-based read access timing for emulation of other memory types
- Support for reporting of single- and multi-bit flash ECC events
- Hardware interface to the 2 flash banks includes the required address decode and control for all 3 memory arrays.

10.2.3 Detailed description

The flash controller generates read and write enables, flash array address, write size, and write data as inputs to the flash array. It also captures read data from the flash array interface and drives it onto the AHB. Up to four pages of data (128-bit width) are buffered by the controller. Lines may be prefetched in advance of being requested by the AHB interface, allowing single-cycle (zero AHB wait-states) read data responses on buffer hits.

Several prefetch control algorithms are available for controlling page read buffer fills. Prefetch triggering may be restricted to instruction accesses only, data accesses only, or may be unrestricted. Prefetch triggering may also be controlled on a per-AHB-master-port basis.

Buffers may also be selectively enabled or disabled for allocation by instruction and data prefetch.

Access protections may be applied on a per-AHB-master-port basis for both reads and writes to support security and privilege mechanisms.

Note: Throughout this discussion, there are references to fields that control prefetch behavior—the DPFEN, IPFEN, PFLIM and BFEN fields of the BIUCR register. Based on the value of the BIUCR[GCE] field, these fields either control prefetch behavior for both banks (when BIUCR[GCE] = 1) or in conjunction with the BK1_DPFE, BK1_IPFE, BK1_PFLIM and BK1_BFE fields of the PFCR3 register to control prefetch behavior for each bank individually (when BIUCR[GCE] = 0). In the discussion that follows, only the BIUCR fields are mentioned, but the information applies equally when prefetch behavior for each bank is controlled separately.

d. It is important to note that if an application performs a single-word write it should not later do a single-word write to the remaining word. ECC is handled on a 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word in that segment should not be programmed since ECC calculation has already completed for that 64-bit segment.

10.2.3.1 Basic interface protocol

The flash controller interfaces to the flash array by driving addresses and read or write enable signals.

The read or write enable signal is asserted in conjunction with the reference address for a single rising clock when a new access request is made.

Addresses are driven to the flash array in a flow-through fashion to minimize array access time. When no outstanding access is in progress, the flash controller drives addresses and then may change to the next outstanding address in the next cycle.

Accesses are terminated under control of the appropriate read/write wait state control setting. Thus, the access time of the operation is determined by the settings of the wait-state control fields. Access timing can be varied to account for the operating conditions of the device (frequency, voltage, temperature) by appropriately setting the fields in the programming model for either bank.

Prefetching of next sequential page is blocked during wait states. Buffer hits are also blocked as well, regardless of whether the access corresponds to valid data in one of the page read buffers. These steps are taken to ensure that timing emulation is correct and that excessive prefetching is avoided.

10.2.3.2 Access protections

The flash controller provides programmable configurable access protections for both read and write cycles from masters via the Bus Interface Unit Access Protection Register (BIUAPR). It allows restriction of read and write requests on a per-AHB-master basis. This functionality is described in *Section 10.5.1.1.2: Bus Interface Unit Access Protection Register (BIUAPR)*. Detection of a protection violation results in an error response from the flash controller on the AHB transfer.

10.2.3.3 Censorship

This device includes censorship logic which affects the operation of the flash controller as follows:

- Censorship disables access to internal flash based on the censorship control word value and the BOOTCFG[0:1] bits in the SIU_RSR.
- Censorship logic prevents modification of the BIUAPR (Bus Interface Unit Access Protection Register) bitfields associated with all masters except the core based on the censorship control word value, the BOOTCFG[0:1] bits in the SIU_RSR, and the field EBI_MCR[EXTM].

The censorship control word is a 32-bit value located at 0x00FF_FDE0. The flash module latches the value of the control word prior to the negation of system reset. Censorship logic uses the value latched in the flash module to disable access to internal flash, disable the NDI, prevent modification of the BIUAPR bitfields, and/or set the boot default value.

Censorship logic disables read and write access to internal flash according to the logic presented in *Table 49*.

Table 49. Flash access disable logic

BOOTCFG(1)		Censorship control word		Flash access
[0]	[1]	Upper half	Lower half	
0	0	0x55AA	0xXXXX	enabled
0	0	!0x55AA	0xXXXX	enabled
1	0	0x55AA	0xXXXX	enabled
1	0	!0x55AA	0xXXXX	disabled
1	1	0x55AA	0xXXXX	enabled
1	1	!0x55AA	0xXXXX	disabled
0	1	0xXXXX	0x55AA	enabled
0	1	0xXXXX	!0x55AA	disabled

1. BOOTCFG[0:1] bits are located in the SIU_RSR.

The flash controller returns a bus error if an access is attempted while flash access is disabled.

10.2.3.4 Read cycles – Buffer miss

When a request for a read access is received the flash controller compares the address to those stored in page buffers currently marked as valid. If no match is found then the flash controller initiates a read from the appropriate flash array module by driving a valid access address. The flash controller then waits for the programmed number of read wait states before sampling the read. This data is normally stored in the least-recently updated page read buffer in parallel with the requested data being forwarded to the AHB.

If the flash access was the direct result of an AHB transaction, the page buffer is marked as most-recently-used as it is being loaded. If the flash access was the result of a speculative prefetch to the next sequential line, it is first loaded into the least-recently-used buffer. The status of this buffer is not changed to most-recently-used until a subsequent buffer hit occurs.

10.2.3.5 Read cycles – Buffer hit

When a request for a read access is received the flash controller compares the address to those stored in page buffers currently marked as valid. If a match is found then the flash controller returns the data to the AHB data phase with a zero wait-state response.

Single cycle read responses to the AHB are possible with the flash controller when the requested read access was previously loaded into one of the page buffers.

10.2.3.6 Write cycles

In a write cycle, the address, write data, and control signals are launched at the completion of the first AHB data phase cycle. Write cycles to the flash array are initiated by driving a valid access address.

10.2.3.7 Errors

The first case that can cause an error response to the AHB is when an access is attempted by an AHB master whose corresponding Read Access Control or Write Access Control settings do not allow the access, thus causing a protection violation. In this case, the flash controller does not initiate a flash array access.

The second case that can cause an error response to the AHB is when an access is performed to the flash array and is terminated with a flash error response. This may occur for either a read or a write operation.

The third case that can cause an error response to the AHB is when a write access is attempted to the flash array and is disallowed by the state of the control input. This case is similar to case 1.

The flash controller can also terminate the current AHB access if *hready_in* (an internal signal) is asserted before the end of the current bus access. While this circumstance should not occur, this does not result in an error condition being reported, as this behavior is initiated by the AHB. In this circumstance, the flash controller completes any flash array access in progress (without signaling the AHB) before handling a new access request.

10.2.3.8 Access pipelining

The flash controller does not support access pipelining since this capability is not supported by the flash array. As a result, the APC (Address Pipelining Control) field must be the same value as the RWSC (Read Wait State Control) field for best performance, that is, BIUCR[APC] = PFCR[RWSC] (see *Section 10.5.1.1.1: Bus Interface Unit Configuration Register (BIUCR)* and later PFlash Configuration Register sections). It cannot be less than the RWSC.

10.2.3.9 Flash error response operation

The flash array may terminate a requested access with an error. This may occur due to an uncorrectable ECC error, or because of improper sequencing during program/erase operations. When an error response is received, the flash controller does not update or validate a page read buffer. An error response may be signaled on read or write operations.

10.2.3.10 Page read buffers and prefetch operation

The four 128-bit page read buffers are used to hold data read from the flash array. Each buffer operates independently, and is filled using a single array access. The buffers are used for both prefetch and normal demand fetches.

The organization of each page buffer is described below in a pseudo-code representation. The hardware structure includes the buffer address and valid bit, along with 128 bits of page read data and several error flags.

```
struct { // page_buffer
reg addr[23:4]; // page address
reg valid; // valid bit
reg rdata[127:0]; // page read data
reg xfr_error; // transfer error indicator from flash array
reg multi_ecc_error; // multi-bit ECC error indicator from flash array
reg single_ecc_error; // single-bit correctable ECC indicator from
flash array
} page_buffer[4];
```

For the general case, a page buffer is written at the completion of an error-free flash access and the valid bit asserted. Subsequent flash accesses that “hit” the buffer, that is, the current access address matches the address stored in the buffer, can be serviced in 0 AHB wait-states as the stored read data is routed from the given page buffer back to the requesting bus master.

As noted in *Section 10.2.3.9: Flash error response operation*, a page buffer is not marked as valid if the flash array access terminated with any type of transfer error. However, the result is that flash array accesses that are tagged with a single-bit correctable ECC event are loaded into the page buffer and validated. For additional comments on this topic, see *Section 10.2.3.10.4: Buffer invalidation*.

Prefetch triggering is controllable on a per-AHB-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 flash controller may trigger a prefetch to the next sequential page of array data on the first idle cycle following the request. The access address is incremented to the next-higher 16-byte boundary, and a flash array prefetch is initiated if the data is not already resident in a page buffer. Prefetched data is always loaded into the least-recently-used buffer.

Buffers may be in one of six states, listed here in prioritized order:

1. Invalid—The buffer contains no valid data.
2. Used–The buffer contains valid data which has been provided to satisfy an AHB burst type read.
3. Valid—The buffer contains valid data which has been provided to satisfy an AHB single type read.
4. Prefetched— The buffer contains valid data which has been prefetched to satisfy a potential future AHB access.
5. Busy AHB—The buffer is currently being used to satisfy an AHB burst read.
6. Busy Fill—The buffer has been allocated to receive data from the flash array, and the array access is still in progress.

Selection of a buffer to be loaded on a miss is based on the following replacement algorithm:

1. First, the buffers are examined to determine if there are any invalid buffers. If there are multiple invalid buffers, the one to be used is selected using a simple numeric priority, where buffer 0 is selected first, then buffer 1, etc.
2. If there are no invalid buffers, the least-recently-used buffer is selected for replacement. Once the candidate page buffer has been selected, the flash array is accessed and read data loaded into the buffer. If the buffer load was in response to a miss, the just-loaded buffer is immediately marked as most-recently-used. If the buffer load was in response to a speculative fetch to the next-sequential line address after a buffer hit, the recently-used status is not changed. Rather, it is marked as most-recently-used only after a subsequent buffer hit.

This policy maximizes performance based on reference patterns of flash accesses and allows for prefetched data to remain valid when non-prefetch enabled bus masters are granted flash access. Several algorithms are available for prefetch control which trade off performance versus power. They are defined by the PFLM (prefetch limit) register field. More aggressive prefetching increases power slightly due to the number of wasted (discarded) prefetches, but may increase performance by lowering average read latency.

In order for prefetching to occur, a number of control bits must be enabled. Specifically, the global buffer enable (BFEN) must be set, the prefetch limit (PFLM) must be non-zero and either instruction prefetching (IPFEN) or data prefetching (DPFEN) enabled. Refer to *Section 10.5: Register descriptions* for a description of these control fields.

10.2.3.10.1 Instruction/Data prefetch triggering

Prefetch triggering may be enabled for instruction reads via the IPFEN control field, while prefetching for data reads is enabled via the DPFEN control field. Additionally, the PFLIM field must also be set to enable prefetching. Prefetches are never triggered by write cycles.

10.2.3.10.2 Per-AHB-master prefetch triggering

Prefetch triggering may be also controlled for individual bus masters. Refer to *Section 10.5.1.1.2: Bus Interface Unit Access Protection Register (BIUAPR)* for details on these controls.

10.2.3.10.3 Buffer allocation

Allocation of the line read buffers is controlled via page buffer configuration field, BIUCR2[LBCFG]. This field defines the operating organization of the four page buffers. The buffers can be organized as a “pool” of available resources (with all four buffers in the pool) or with a fixed partition between buffers allocated to instruction or data accesses. For the fixed partition, two configurations are supported. In one configuration, buffers 0 and 1 are allocated for instruction fetches and buffers 2 and 3 for data accesses. In the second configuration, buffers 0, 1 and 2 are allocated for instruction fetches and buffer 3 reserved for data accesses.

10.2.3.10.4 Buffer invalidation

The page read buffers may be invalidated under hardware or software control.

Software may invalidate the buffers by clearing field BIUCR[BFEN], which also disables the buffers. Software may then re-assert BIUCR[BFEN] to its previous state, and the buffers will have been invalidated.

One special case needing software invalidation relates to page buffer “hits” on flash data which was tagged with a single-bit ECC event on the original array access. Recall that the page buffer structure includes a status bit signaling the array access detected and corrected a single-bit ECC error. On all subsequent buffer hits to this type of page data, a single-bit ECC event is signaled by the flash controller. Depending on the specific hardware configuration, this reporting of a single-bit ECC event may generate an ECC alert interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

10.2.4 Wait-state emulation

Emulation of other memory array timings are supported by the flash controller on read and write 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 flash controller will insert additional wait states according to the upper address lines ADDR[28:24]. When these address lines are non-zero, additional cycles are added to

system bus transfers. Normal system bus termination will be extended. In addition, no line read buffer prefetches are initiated, and buffer hits are ignored.

See *Section 10.5.1.1.1: Bus Interface Unit Configuration Register (BIUCR)* for details.

10.3 Flash memory block (C90LC)

10.3.1 Flash block overview

The primary function of a 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 flash is addressable by word (32 bits) and page (128 bits).

The C90LC block is arranged as two functional units. The first functional unit is the C90LC 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 C90LC 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 flash controller (PFLASH_LCA).

The flash controller interfaces the SPC563Mxx system bus to the C90LC memory block. The flash controller is described in *Section 10.2: Platform flash memory controller (PFLASH_LCA)*.

The device has three 512 KB arrays implemented for a total of 1.5 MB of flash memory. They are arranged as follows:

Bank0, Array0:

- 512 KB + 16 KB shadow block
  - 8 small blocks organized as 16 KB, 16 KB, 32 KB, 32 KB, 16 KB, 16 KB, 64 KB and 64 KB
  - 2 large blocks organized as 128 KB and 128 KB
  - 1 shadow block, 16 KB

Bank1, Array1:

- 512 KB
  - No small blocks or shadow block

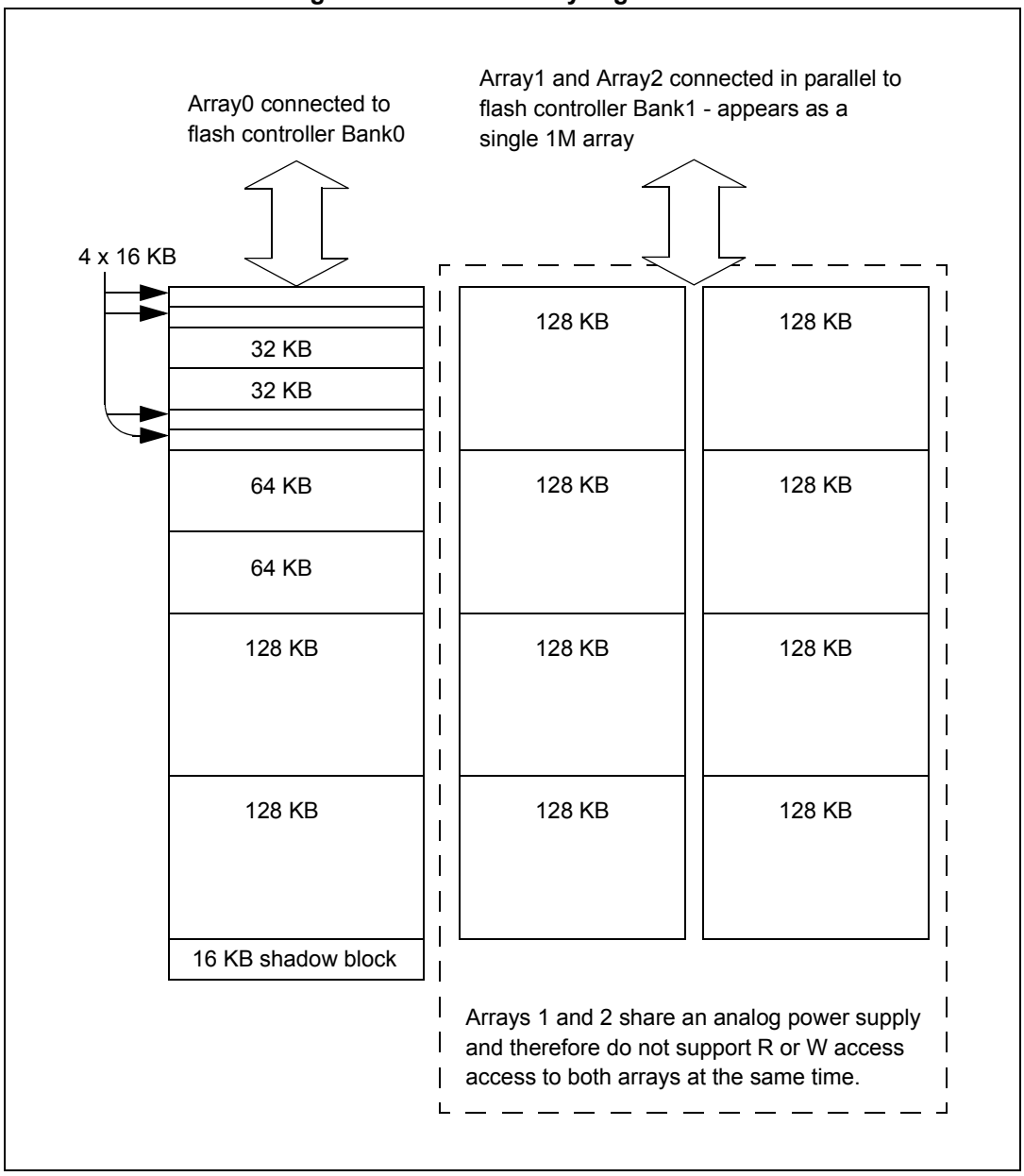
4 large blocks organized as 4 × 128 KBBank1, Array2:

- 512 KB
  - No small blocks or shadow block
  - 4 large blocks organized as 4 × 128 KB

The arrays are organized as two banks. Array0 is connected to Bank0. Array1 and Array2 are connected in parallel to Bank1 and effectively operate as a single 1 MB flash array.

Warning: **Array1 and Array2 share an analog power supply, so accesses (write, erase or fetch) cannot be performed on both Array1 and Array2 simultaneously.**

Figure 49. LC flash array organization



10.3.2 LC flash features

- High Read parallelism (128 bits)
- Error Correction Code (Single Error Correction, Double Error Detection) to enhance Data Retention
- Double Word Program (64 bits)
- Block Erase
- Erase Suspend available (Program Suspend not available)
- Software programmable Program/Erase Protection to avoid unwanted writings
- Censored Mode

10.3.3 Programming considerations

Note: Like all flash memory, before an arbitrary value can be written to a memory location in flash on these devices, the block containing that address must be erased (all values set to '1'). The electrical characteristics of flash memory allow write operations to only transition individual bits from '1' to '0', and to perform erase operations only at the block-level.

Warning: Software executing from flash must not write to registers that control flash behavior, e.g., wait state settings or prefetch enable/disable. Doing so can cause data corruption. On SPC563Mxx devices these registers include BIUCR, BIUAPR, BIUCR2, and PFCR3. These registers must be written while executing from a different memory, such as the internal SRAM, and not from the flash itself. Further, flash configuration registers should be written only with 32-bit write operations to avoid any issues associated with register "incoherence" caused by bit fields spanning smaller size (8- and 16-bit) boundaries.

10.3.3.1 Modify operations

All the modify operations of the flash modules are managed through the flash array control registers. All blocks of each flash array module belong to the same partition (bank), therefore when a modify operation is active on some blocks no read access is possible on any other block within the same array bank.

During a flash modify operation any attempt to read any flash location within the same module outputs invalid data and bit MCR[RWE] is automatically set. This means that the flash module is not fetchable when a modify operation is active within the same array module: the modify operation commands must be executed from another array.

If during a modify operation a reset occurs, the operation is suddenly interrupted and the array is reset to Read Mode. The data integrity of the flash section where the modify operation has been aborted is not guaranteed: the interrupted flash modify operation must be repeated.

In general each modify operation is started through a sequence of three steps:

1. The first instruction is used to select the desired operation by setting its corresponding selection bit in MCR (PGM or ERS) or UT0 (MRE or EIE).
2. The second step is the definition of the operands: the Address and the Data for programming or the blocks for erase or factory margin read.
3. The third instruction is used to start the modify operation, by setting EHV in MCR or AIE in the UT0 register.

Once selected, but not yet started, one operation can be canceled by resetting the operation selection bit.

A summary of the available flash modify operations are shown in *Table 50*.

Table 50. Flash modify operations

Operation	Select bit	Operands	Start bit
Double Word Program	MCR[PGM]	Address and Data by Interlock Writes	MCR[EHV]
Block Erase	MCR[ERS]	LMSR, HSR	MCR[EHV]
Array Integrity Check(1)	None	LMSR, HSR	UT0[AIE]
Factory Margin Read(1)	UT0[MRE]	UT0[MRV] + LMSR, HSR	UT0[AIE]
ECC Logic Check(1)	UT0[EIE]	UT0.DSI, UT1, UT2	UT0[AIE]

1. This operation is executed from User Test Mode. See *Section 10.3.3.1.4: User Test Mode* for details.

In general each modify operation is completed through a sequence of four steps:

1. Wait for operation completion: wait for bit MCR[DONE] (or UT0[AID]) to go high.
2. Check operation result: check bit MCR[PEG] (or compare UMISR0–4 with expected value).
3. Switch off flash controller by resetting MCR[EHV] (or UT0[AIE]).
4. Deselect current operation by clearing MCR[PGM]/[ERS] (or UT0[MRE]/[EIE]).

In the following sections all modify operations are described and some examples of the sequences needed to activate them are presented.

10.3.3.1.1 Double Word Program

A flash program sequence operates on any double word within the flash. Up to two words within the double word may be altered in a single program operation. During a program operation, ECC bits are programmed. ECC is handled on a 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed since ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word will result in an operation failure. It is recommended that all programming operations be of 64 bits. The programming operation should completely fill selected ECC segments within the double word.

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1.

Addresses in locked/disabled blocks cannot be programmed.

You can program the values in any or all of two words, of a double word, with a single program sequence.

Double word-bound words have addresses which differ only in address bit 2.

The Program operation consists of the following sequence of events:

1. Change the value in the MCR[PGM] bit from 0 to 1.
2. Ensure the block that contains the address to be programmed is unlocked.
   - Write the first address to be programmed with the program data.
   - The flash module latches address bits (22:3) 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 (depending on the CPU bus).
3. If more than one word is to be programmed, write the additional address in the double word with data to be programmed. This is referred to as a program data write.

   The flash modules ignore address bits (22:3) for program data writes.

   The eventual unwritten data word default to 0xFFFFFFFF.
4. Write a logic 1 to the MCR[EHV] bit to start the internal program sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the MCR[PGM] bit to terminate the program operation.

A program may be initiated with the 0 to 1 transition of field MCR[PGM] or by clearing the field MCR[EHV] at the end of a previous program.

The first write after a program is initiated determines the page address to be programmed. This first write is referred to as an interlock write. The interlock write determines if the shadow or normal array space will be programmed by causing MCR[PEAS] to be set/cleared.

An interlock write must be performed before setting MCR[EHV]. An application may terminate a program sequence by clearing MCR[PGM] prior to setting MCR[EHV].

While MCR[DONE] is low and MCR[EHV] is high, an application may clear EHV, resulting in a program abort.

A program abort forces the Module to step 8 of the program sequence.

An aborted program will result in MCR[PEG] being set low, indicating a failed operation. MCR[DONE] must be checked to know when the aborting command has completed.

The data space being operated on before the abort will contain indeterminate data. This may be recovered by repeating the same program instruction with the same data or executing an erase of the affected blocks.

Example 1 Double Word Program of data 0x55AA55AA at address 0x00AAA8 and data 0xAA55AA55 at address 0x00AAAC

```
MCR = 0x00000010; /* Set PGM in MCR: Select PGM Operation */
(0x00AAA8) = 0x55AA55AA; /* Latch Address and 32 LSB data */
(0x00AAAC) = 0xAA55AA55; /* Latch 32 MSB data */
MCR = 0x00000011; /* Set EHV in MCR: Operation Start */
do /* Loop to wait for DONE=1 */
{ tmp = MCR; /* Read MCR */
} while ( !(tmp & 0x00000400) );
status = MCR & 0x00000200; /* Check PEG flag */
MCR = 0x00000010; /* Reset EHV in MCR: Operation End */
MCR = 0x00000000; /* Reset PGM in MCR: Deselect Operation */
```

10.3.3.1.2 Block 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 (if available).

- The erase sequence is fully automated within the flash. an application only needs to select the blocks to be erased and initiate the erase sequence.
- Locked/disabled blocks cannot be erased.
- If multiple blocks are selected for erase during an erase sequence, no specific operation order must be assumed.

The Erase operation consists of the following sequence of events:

1. Change the value in the MCR[ERS] bit from 0 to 1.
2. Select the block(s) to be erased by writing 1's to the appropriate register(s) in LMSR or HSR registers.

   If the shadow block is to be erased, this step may be skipped, and LMSR and HSR are ignored.

   Note that Lock and Select are independent. If a block is selected and locked, no erase will occur.
3. Write to any address in the respective flash array. This is referred to as an erase interlock write.
4. Write a logic 1 to the MCR[EHV] bit to start the internal erase sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more blocks are to be erased, return to step 2.
9. Write a logic 0 to the MCR[ERS] bit to terminate the erase operation.

Additional considerations:

- After setting MCR[ERS], one write, referred to as an interlock write, must be performed before MCR[EHV] can be set to 1.
- Data words written during erase sequence interlock writes are ignored.
- An application may terminate the erase sequence by clearing ERS before setting EHV.
- An erase operation may be aborted by clearing MCR[EHV] assuming MCR[DONE] is low,
- MCR[EHV] is high and MCR[ESUS] is low.
- An erase abort forces the Module to step 8 of the erase sequence.
- An aborted erase will result in MCR[PEG] being set low, indicating a failed operation.
- MCR[DONE] must be checked to know when the aborting command has completed.
- The block(s) being operated on before the abort contain indeterminate data. This may be recovered by executing an erase on the affected blocks.
- An application may not abort an erase sequence while in erase suspend.

The following example selects two blocks using the LSEL[2:1] bits of the LMSR register to select blocks 2a and 1b (see *Table 52 on page 233* for the flash space memory map and *Section 10.5.1.2.5: Low/Mid Address Space Block Select Register (LMSR) on page 256*) and performs an erase.

Example 2 Erase of Blocks 2a and 1b

```
MCR = 0x00000004; /* Set ERS in MCR: Select ERS Operation */
LMSR = 0x00000006; /* Set LSEL2-1 in LMSR: Select blocks to erase */
(0x000000) = 0xFFFFFFFF; /* Latch a Flash Address with any data */
MCR = 0x00000005; /* Set EHV in MCR: Operation Start */
do /* Loop to wait for DONE=1 */
{ tmp = MCR; /* Read MCR */
} while ( !(tmp & 0x00000400) );
status = MCR & 0x00000200; /* Check PEG flag */
MCR = 0x00000004; /* Reset EHV in MCR: Operation End */
MCR = 0x00000000; /* Reset ERS in MCR: Deselect Operation */
```

10.3.3.1.3 Erase Suspend/Resume

The erase sequence may be suspended to allow read access to the flash array. It is not possible to program or to erase during an erase suspend. During erase suspend, all reads to blocks targeted for erase return indeterminate data.

An erase suspend is initiated by changing the value of the MCR[ESUS] bit from 0 to 1. MCR[ESUS] can be set to 1 at any time when MCR[ERS] and MCR[EHV] are high and MCR[PGM] is low. A 0 to 1 transition of MCR[ESUS] causes the array module to start the sequence which places it in erase suspend.

An application must wait until MCR[DONE] = 1 before the erase operation is suspended and further actions are attempted. MCR[DONE] will go high after MCR[ESUS] is set to 1.

Once suspended, the array may be read. Reads while MCR[ESUS] = 1 from the block(s) being erased return indeterminate data.

Example 3 Block Erase Suspend

```
MCR = 0x00000007; /* Set ESUS in MCR: Erase Suspend */
do /* Loop to wait for DONE=1 */
{ tmp = MCR; /* Read MCR */
} while ( !(tmp & 0x00000400) );
```

Note that there is no need to clear MCR[EHV] and MCR[ERS] in order to perform reads during erase suspend. The erase sequence is resumed by writing a logic 0 to MCR[ESUS].

MCR[EHV] must be set to 1 before MCR[ESUS] can be cleared to resume the operation.

The array module continues the erase sequence from one of a set of predefined points. This may extend the time required for the erase operation.

Example 4 Block Erase Resume

MCR = 0x00000005; /* Reset ESUS in MCR: Erase Resume */

10.3.3.1.4 User Test Mode

User Test Mode is a mode that customers can put the flash array module in to do specific tests to check integrity.

Three kinds of test can be performed:

- Array Integrity Self Check
- Factory Margin Mode Read
- ECC Logic Check

The User Test Mode is equivalent to a modify operation: read accesses attempted during User Test Mode generate a Read-While-Write Error (MCR[RWE] set).

User Test operations are not allowed on the Test and Shadow blocks.

10.3.3.1.4.1 Array Integrity Self Check

Array Integrity is checked using a predefined address sequence (proprietary), and is executed on selected and unlocked blocks. Once the operation is completed, the results of the reads can be checked by reading the MISR value (stored in UMISR0–4), to determine if an incorrect read, or ECC detection was noted.

The internal MISR calculator is a 32-bit register.

The 128-bit data, the 16 ECC data and the single-bit and double-bit ECC errors of the two double words are therefore captured by the MISR through five different read accesses at the same location.

The entire check is done with five complete scans of the memory address space:

1. The first pass will scan only bits 31:0 of each page.
2. The second pass will scan only bits 63:32 of each page.
3. The third pass will scan only bits 95:64 of each page.
4. The fourth pass will scan only bits 127:96 of each page.
5. The fifth pass will scan only the ECC bits (8 + 8) and the single-bit and double-bit ECC errors (2 + 2) of both double words of each page.

The 128-bit data and the 16 ECC data are sampled before the eventual ECC correction, while the single and double error flags are sampled after the ECC evaluation.

Only data from existing and unlocked locations are captured by the MISR.

The MISR can be seeded to any value by writing the UMISR0–4 registers.

The Array Integrity Self Check consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1's to the appropriate register(s) in LMSR or HSR registers.

   Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.
3. Set UT0[AIS] bit for a sequential addressing only.
4. Write a logic 1 to the UT0[AIE] bit to start the Array Integrity Check.
5. Wait until the UT0[AID] bit goes high.
6. Compare UMISR0–4 content with the expected result.
7. Write a logic 0 to the UT0[AIE] bit.
8. If more blocks are to be checked, return to step 2.

It is recommended to leave UT0[AIS] at 0 and use the proprietary address sequence that checks the read path more fully, although this sequence takes more time. While UT0[AID] is low and UT0[AIE] is high, an application may clear AIE, resulting in a Array Integrity Check abort. UT0[AID] must be checked to know when the aborting command has completed.

The following example selects two blocks using the LSEL[2:1] bits of the LMSR register to select blocks 2a and 1b (see *Table 52 on page 233* for the flash space memory map and *Section 10.5.1.2.5: Low/Mid Address Space Block Select Register (LMSR) on page 256*) and performs an array integrity check of those blocks.

Example 5 Array Integrity Check of Blocks 2a and 1b

```
UT0 = 0xF9F99999; /* Set UTE in UT0: Enable User Test */
LMSR = 0x00000006; /* Set LSEL2-1 in LMSR: Select blocks */
UT0 = 0x80000002; /* Set AIE in UT0: Operation Start */
do /* Loop to wait for AID=1 */
{ tmp = UT0; /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0 = UMISR0; /* Read UMISR0 content*/
data1 = UMISR1; /* Read UMISR1 content*/
data2 = UMISR2; /* Read UMISR2 content*/
data3 = UMISR3; /* Read UMISR3 content*/
data4 = UMISR4; /* Read UMISR4 content*/
UT0 = 0x00000000; /* Reset UTE and AIE in UT0: Operation End */
```

10.3.3.1.4.2 Factory Margin Read

Note: Factory margin read is a diagnostic procedure to check proper programming, for example by 3rd party programming service providers. It is not supported in customer applications because the voltages used for margin reads can reduce the life expectancy of the flash array.

The factory margin read procedure (either Margin 0 or Margin 1) can be run on unlocked blocks to unbalance the sense amplifiers with respect to standard read conditions so that all read accesses reduce the margin vs '0' (UT0[MRV] = '0') or vs '1' (UT0[MRV] = '1'). Locked sectors are ignored by MISR calculation and ECC flagging.

The results of the factory margin reads can be checked by comparing the checksum value in the UMISR0–4 registers.

Since factory margin reads are done at voltages that are higher than the normal read voltages, lifetime expectancy of the flash may be impacted. Doing factory margin reads repeatedly results in degradation of the flash array and shortens the lifetime expected with normal read levels. For these reasons this capability is reserved for factory use only and is not supported in user applications. Charge losses detected via margin reads are not considered failures of the device and no Failure Analysis will be opened on them.

The Margin Read Setup operation consists of the following sequence of events:

1. Set UTE in UT0 by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1's to the appropriate register(s) in LMS or HBS registers.

   Note that Lock and Select are independent. If a block is selected and locked, no Margin Read will occur.
3. Set eventually UT0[AIS] bit for a sequential addressing only.
4. Change the value in the UT0[MRE] bit from 0 to 1.
5. Select the Margin level: UT0[MRV] = 0 for 0's margin, UT0[MRV] = 1 for 1's margin.
6. Write a logic 1 to the UT0[AIE] bit to start the Margin Read Setup or skip to step 6 to terminate.
7. Wait until the UT0[AID] bit goes high.
8. Compare UMISR0–4 content with the expected result.
9. Write a logic 0 to the UT0[AIE], UT0[MRE] and UT0[MRV] bits.
10. If more blocks are to be checked, return to step 2.

It is recommended to leave UT0[AIS] at 1 and use the linear address sequence, which takes less time.

During the execution of the Margin Read operation it is forbidden to modify the content of Block Select (LMS, HBS) and Lock (LML, SLL, HBL) registers, otherwise the MISR value can vary in an unpredictable way.

The read accesses will be done with the addition of a proper number of Wait States to guarantee the correctness of the result.

While UT0[AID] is low and UT0[AIE] is high, the user may clear AIE, resulting in a Array Integrity Check abort.

UT0[AID] must be checked to know when the aborting command has completed.

Example 6 Margin Read Check versus 1's

```
UMISR0 = 0x00000000; /* Reset UMISR0 content */
UMISR1 = 0x00000000; /* Reset UMISR1 content */
UMISR2 = 0x00000000; /* Reset UMISR2 content */
UMISR3 = 0x00000000; /* Reset UMISR3 content */
UMISR4 = 0x00000000; /* Reset UMISR4 content */
UT0 = 0xF9F99999; /* Set UTE in UT0: Enable User Test */
LMS = 0x00000006; /* Set LSL2-1 in LMS: Select Sectors */
UT0 = 0x80000004; /* Set AIS in UT0: Select Operation */
UT0 = 0x80000024; /* Set MRE in UT0: Select Operation */
UT0 = 0x80000034; /* Set MRV in UT0: Select Margin versus 1's */
UT0 = 0x80000036; /* Set AIE in UT0: Operation Start */
do /* Loop to wait for AID=1 */
{ tmp = UT0; /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0 = UMISR0; /* Read UMISR0 content*/
data1 = UMISR1; /* Read UMISR1 content*/
data2 = UMISR2; /* Read UMISR2 content*/
data3 = UMISR3; /* Read UMISR3 content*/
data4 = UMISR4; /* Read UMISR4 content*/
UT0 = 0x80000034; /* Reset AIE in UT0: Operation End */
UT0 = 0x00000000; /* Reset UTE, MRE, MRV, AIS in UT0: Deselect Op. */
```

10.3.3.1.4.3 ECC Logic Check

ECC Logic Check verifies the integrity of the ECC correction and detection logic. The operation provides user control over the 64 data bit + 8 parity bit inputs. Results of the ECC logic can be checked by reading the MISR value.

The ECC Logic Check operation consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Write in UT1.DAI31–0 and UT2.DAI63–32 the double word input value.
3. Write in UT0.DSI7–0 the Syndrome Input value.
4. Select the ECC Logic Check: write a logic 1 to the UT0[EIE] bit.
5. Write a logic 1 to the UT0[AIE] bit to start the ECC Logic Check.
6. Wait until the UT0[AID] bit goes high.
7. Compare UMISR0–4 content with the expected result.
8. Write a logic 0 to the UT0[AIE] bit.

Notice that when UT0[AID] is low UMISR0–4, UT1–2 and bits MRE, MRV, EIE, AIS and DSI7–0 of UT0 are not accessible: reading returns indeterminate data and writing has no effect.

Example 7 ECC Logic Check

```
UT0 = 0xF9F99999; /* Set UTE in UT0: Enable User Test */
UT1 = 0x55555555; /* Set DAI31-0 in UT1: Even Word Input Data */
UT2 = 0xAAAAAAAA; /* Set DAI63-32 in UT2: Odd Word Input Data */
UT0 = 0x80FF0000; /* Set DSI7-0 in UT0: Syndrome Input Data */
UT0 = 0x80FF0008; /* Set EIE in UT0: Select ECC Logic Check */
UT0 = 0x80FF000A; /* Set AIE in UT0: Operation Start */
do /* Loop to wait for AID=1 */
{ tmp = UT0; /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0 = UMISR0; /* Read UMISR0 content (expected 0x55555555) */
data1 = UMISR1; /* Read UMISR1 content (expected 0xAAAAAAAA) */
data2 = UMISR2; /* Read UMISR2 content (expected 0x55555555) */
data3 = UMISR3; /* Read UMISR3 content (expected 0xAAAAAAAA) */
data4 = UMISR4; /* Read UMISR4 content (expected 0x00FF00FF) */
UT0 = 0x00000000; /* Reset UTE, AIE and EIE in UT0: Operation End */
```

10.3.3.2 Error correction code

The flash module provides a method to improve the reliability of the stored data: the usage of an Error Correction Code.

At each double word of 64 bits there are associated 8 ECC bits that are programmed in such a way to guarantee a Single Error Correction and a Double Error Detection (SEC-DED).

10.3.3.2.1 ECC algorithms

The flash arrays support the “All ‘1’s No Error” ECC algorithm, which detects as valid any double word read on a just erased block (all the 72 bits are 1’s).

This option enables performance of a Blank Check after a Block Erase operation.

10.3.3.3 Protection strategy

Two kind of protections are available: Modify Protection to avoid unwanted program/erase in flash blocks and Censored Mode.

10.3.3.3.1 Modify protection

The Flash Modify Protection information is stored in a protected area of the flash. This information is read once during the flash initialization phase following exit from Reset and is stored in program-accessible registers.

The reset state of all the Modify Protection Registers is the protected state.

A software mechanism is provided to independently lock/unlock each Low, Mid and High Address Space Block against program and erase. Software locking is done through the LMLR (Low/Mid Address Space Block Lock Register) or HLR (High Address Space Block Lock Register) registers.

An alternate means to enable software locking for blocks of Low Address Space only is through the SLMLR (Secondary Low/Mid Address Space Block Lock Register).

All of these registers have an image stored in a protected area of flash so the locking information is kept on reset. The registers can be written at any time, therefore an application can lock and unlock blocks when desired.

10.3.3.3.2 Censored mode

The Censored Mode information is stored in the shadow block. This information is read once during the flash initialization phase following exit from Reset and is stored in the appropriate registers. The Censored Mode Registers are not accessible by applications.

10.4 Memory maps

The SPC563Mxx device flash memory has two memory maps: one for the flash memory space and another for the program-visible control and configuration registers. The flash memory space is accessed via the AMBA-AHB port and the program-visible registers are accessed via the slave peripheral bus.

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 system bus side, and no flash array write is initiated.

The following sections contain memory maps for the device flash array and flash control registers.

10.4.1 Overview memory map

Table 51 shows the flash array memory map, and the configuration registers for the flash controller.

Table 51. Flash overview map

Address	Use	Size [KB]
FLASH_BASE (0x0000_0000)	Flash Memory	1536
FLASH_BASE + 0x0018_0000	Reserved	6656
FLASH_BASE + 0x0080_0000	Reserved	3520
FLASH_BASE + 0x00B7_0000	Reserved	64512 bytes
FLASH_BASE + 0x00B7_FB20	Bank1, Array2 Test Block	992 bytes
FLASH_BASE + 0x00B8_0000	Reserved	192
FLASH_BASE+ 0x00BB_0000	Reserved	64512 bytes
FLASH_BASE+ 0x00BB_FB20	Bank1, Array1 Test Block	992 bytes
FLASH_BASE + 0x00BC_0000	Reserved	192
FLASH_BASE + 0x00BF_0000	Reserved	64512 bytes
FLASH_BASE + 0x00BF_FB20	Bank0, Array0 Test Block	992 bytes
FLASH_BASE + 0x00C0_0000	Reserved	3072
FLASH_BASE + 0x00FF_C000	Bank0, Array0 shadow region	16
FLASH_BASE + 0x0100_0000	Flash emulation mapping	507904
FLASH_REGS_BASE_BK0 (0xC3F8_8000)	Bank0, Array0 control registers	

Table 51. Flash overview map (continued)

Address	Use	Size [KB]
FLASH_REGS_BASE_BK0 + 0x01C	Bus Interface Unit Configuration Register (BIUCR)	
FLASH_REGS_BASE_BK0 + 0x020	Bus Interface Unit Access Protection Register (BIUAPR)	
FLASH_REGS_BASE_BK0 + 0x024	Bus Interface Unit Configuration Register 2 (BIUCR2)	
FLASH_REGS_BASE_BK0 + 0x028	Platform Flash Configuration Register 3 (PFCR3)	
FLASH_REGS_BASE_BK1 (0xC3FB_0000)	Bank1, Array1 control registers	
FLASH_REGS_BASE_BK2 (0xC3FB_4000)	Bank1, Array2 control registers	

10.4.2 Flash memory space map

Table 52 shows the mapping for the flash memory storage addresses for all devices.

- Neither the 768 KB devices nor the 1 MB devices contain Array2.
- 768 KB devices use only the first 256 KB of Array0 plus the shadow block.

Table 52. Flash memory storage address map

Address	Use	Block(1), (2)	Size	SPC563M54 768 KB	SPC563M60 1 MB	SPC563M64 1.5 MB	Bank
0x0000_0000	Low Address Space 256 KB	0	16K	Available	Available	Available	Bank0 Array0
0x0000_4000		1a	16K	Available	Available	Available	
0x0000_8000		1b	32K	Available	Available	Available	
0x0001_0000		2a	32K	Available	Available	Available	
0x0001_8000		2b	16K	Available	Available	Available	
0x0001_C000		3	16K	Available	Available	Available	
0x0002_0000		4	64K	Available	Available	Available	
0x0003_0000		5	64K	Available	Available	Available	
0x0004_0000	Mid Address Space 256 KB	6	128K	Not available	Available	Available	
0x0006_0000		7	128K	Not available	Available	Available	

Table 52. Flash memory storage address map (continued)

Address	Use	Block(1), (2)	Size	SPC563M54 768 KB	SPC563M60 1 MB	SPC563M64 1.5 MB	Bank
0x0008_0000	High Address Space 1 MB	8	128K	Available	Available	Available	Bank1 Array1
0x000A_0000		9	128K	Available	Available	Available	
0x000C_0000		10	128K	Available	Available	Available	
0x000E_0000		11	128K	Available	Available	Available	
0x0010_0000		12	128K	Not available	Not available	Available	Bank1 Array2
0x0012_0000		13	128K	Not available	Not available	Available	
0x0014_0000		14	128K	Not available	Not available	Available	
0x0016_0000		15	128K	Not available	Not available	Available	
0x00B7_FB20	OTP(3)	OTP2	992 bytes	Available	Available	Available	Bank1 Array2 Test
0x00BB_FB20	OTP(3)	OTP1	992 bytes	Available	Available	Available	Bank1 Array1 Test
0x00BF_FB20	OTP(3)	OTP0	992 bytes	Available	Available	Available	Bank0 Array0 Test
0x00FF_C000	Shadow Block 16 KB	S0	16K	Available	Available	Available	Bank0 Array0

1. Boot can be performed from any block in the Low Address Space except blocks 1b and 2b.
2. Each block can be “Locked” against modification or “Selected” for erase.
3. One-time programmable test block.

10.4.3 Test block

Devices in the SPC563Mxx family contain a small area of flash memory that can be used as a One-Time Programmable (OTP) storage space. This area can be used to store the default values of some flash registers and also contains information about the device. Each flash array module has this small OTP area.

The addresses are shown in *Table 53*.

Table 53. Test block locations

Address range	Description	Size
0x00B7_FB20 – 0x00B7_FEFF	Bank1, Array 2 Test Block	992 bytes
0x00BB_FB20 – 0x00BB_FEFF	Bank1, Array1 Test Block	992 bytes
0x00BF_FB20 – 0x00BF_FEFF	Bank0, array0 Test Block	992 bytes

Note: The OTP area cannot be erased after it is programmed.

10.4.3.1 Test block memory map

Table 54 contains the test block allocations.

Table 54. Test block allocation

	Address[1]	Use	Default value[2]	Number of words used (32-bit)
Test Flash	0x00Bn_FB20[3]	Flash Controller Micro-code Version	*[4]	2 words
	0x00Bn_FB28	Flash Controller Micro-code Parameter Version	*[4]	1 word
	0x00Bn_FB2C	Reserved	Erased (0xFFFF_FFFF)	1 word
	0x00Bn_FB30 – 0x00Bn_FC0F	Reserved for internal use	This area may contain undocumented data.	56 words
	0x00Bn_FC10	Device Unique Serial Number[5]	*[6]	4 words (128 bits)
	0x00Bn_FC20 – 0x00Bn_FCFF	Reserved for internal use	This area may contain undocumented data.	56 words
	0x00Bn_FD00 – 0x00Bn_FDE7	User reserved	—	58 words
	0x00Bn_FDE8	Flash LMLR default Value	Erased (0xFFFF_FFFF)	1 word
	0x00Bn_FDEC	Unused (except for ECC with the LMLR default value)	Erased (0xFFFF_FFFF)	1 word
	0x00Bn_FDF0	Flash HLR default Value	Erased (0xFFFF_FFFF)	1 word
	0x00Bn_FDF4	Unused (except for ECC with the HLR default value)	Erased (0xFFFF_FFFF)	1 word
	0x00Bn_FDF8	Flash SLMLR default value	Erased (0xFFFF_FFFF)	1 word
	0x00Bn_FDFC	Unused (except for ECC with the SLMLR default value)	Erased (0xFFFF_FFFF)	1 word
	0x00Bn_FE00 – 0x00Bn_FEFF	Reserved	Erased (0xFFFF_FFFF)	64 words

1. n = 0x7, 0xB, or 0xF from the different Flash Modules.
2. Factory programmed values.
3. The 32-bit word beginning at 0x00Bn_FB20 and the 32-bit word beginning at 0x00Bn_FB24 are identical.
4. This value may change in the future.
5. This value only appears in the Test Block 0.
6. This value is different on every device.

10.4.3.2 Unique serial number

For tracking purposes by the customer, a Unique Serial Number is included in the Test Block to allow identification of a particular device. The Unique Serial Number is defined to be 128

bits and is based on the manufacturing lot identifier and additional information specific to each device in a manufacturing lot.

10.4.4 Flash control and configuration registers map

Table 55 shows the mapping for the flash memory register addresses for all devices.

Table 55. Flash memory register addresses

Register	Address
Flash Controller Registers (all devices)	
Bus Interface Unit Configuration Register (BIUCR)	0xC3F8_801C
Bus Interface Unit Access Protection Register (BIUAPR)	0xC3F8_8020
Bus Interface Unit Configuration Register 2 (BIUCR2)	0xC3F8_8024
PFlash Configuration Register 3 (PFCR3)	0xC3F8_8028
Flash Bank0, Array0 (all devices)	
Module Configuration Register (MCR)	0xC3F8_8000
Low/Mid-Address Space Block Locking Register (LMLR)	0xC3F8_8004
High-Address Space Block Locking Register (HLR) - not used	0xC3F8_8008
Secondary Low/Mid-Address Space Block Locking Register (SLMLR)	0xC3F8_800C
Low/Mid-Address Space Block Select Register (LMSR)	0xC3F8_8010
High-Address Space Block Select Register (HSR) - not used	0xC3F8_8014
Address Register (AR)	0xC3F8_8018
User Test 0 (UT0) register	0xC3F8_803C
User Test 1 (UT1) register	0xC3F8_8040
User Test 2 (UT2) register	0xC3F8_8044
User Multiple Input Signature Register 0 (UMISR0)	0xC3F8_8048
User Multiple Input Signature Register 1 (UMISR1)	0xC3F8_804C
User Multiple Input Signature Register 2 (UMISR2)	0xC3F8_8050
User Multiple Input Signature Register 3 (UMISR3)	0xC3F8_8054
User Multiple Input Signature Register 4 (UMISR4)	0xC3F8_8058
Flash Bank1, Array1 (all devices)	
MCR register	0xC3FB_0000
Low/Mid-Address Space Block Locking Register (LMLR) - not used	0xC3FB_0004
High-Address Space Block Locking Register (HLR)	0xC3FB_0008
Secondary Low/Mid-Address Space Block Locking Register (SLMLR) - not used	0xC3FB_000C

Table 55. Flash memory register addresses (continued)

Register	Address
Low/Mid-Address Space Block Select Register (LMSR) - not used	0xC3FB_0010
High-Address Space Block Select Register (HSR) register	0xC3FB_0014
Address Register (AR)	0xC3FB_0018
User Test 0 (UT0) register	0xC3FB_003C
User Test 1 (UT1) register	0xC3FB_0040
User Test 2 (UT2) register	0xC3FB_0044
User Multiple Input Signature Register 0 (UMISR0)	0xC3FB_0048
User Multiple Input Signature Register 1 (UMISR1)	0xC3FB_004C
User Multiple Input Signature Register 2 (UMISR2)	0xC3FB_0050
User Multiple Input Signature Register 3 (UMISR3)	0xC3FB_0054
User Multiple Input Signature Register 4 (UMISR4)	0xC3FB_0058
Flash Bank1, Array2 (1.5 MB devices only)	
Module Configuration Register (MCR)	0xC3FB_4000
Low/Mid-Address Space Block Locking Register (LMLR) - not used	0xC3FB_4004
High-Address Space Block Locking Register (HLR)	0xC3FB_4008
Secondary Low/Mid-Address Space Block Locking Register (SLMLR) - not used	0xC3FB_400C
Low/Mid-Address Space Block Select Register (LMSR) - not used	0xC3FB_4010
High-Address Space Block Select Register (HSR) register	0xC3FB_4014
Address Register (AR)	0xC3FB_4018
User Test 0 (UT0) register	0xC3FB_403C
User Test 1 (UT1) register	0xC3FB_4040
User Test 2 (UT2) register	0xC3FB_4044
User Multiple Input Signature Register 0 (UMISR0)	0xC3FB_4048
User Multiple Input Signature Register 1 (UMISR1)	0xC3FB_404C
User Multiple Input Signature Register 2 (UMISR2)	0xC3FB_4050
User Multiple Input Signature Register 3 (UMISR3)	0xC3FB_4054
User Multiple Input Signature Register 4 (UMISR4)	0xC3FB_0058

10.5 Register descriptions

The following sections detail the flash controller registers and the flash array module registers.

10.5.1 Flash control and configuration registers

SPC563Mxx devices have control registers that affect flash behavior at the flash controller level and at the individual array level. The flash controller registers are global in nature. The flash array registers affect only the flash array module they are associated with.

10.5.1.1 Flash Controller (PFLASH_LCA) Registers

As noted previously, flash controller registers affect flash behavior from the flash controller level (see *Figure 48 on page 212*) and are, therefore, global in nature. The registers are:

- Bus Interface Unit (BIU) Configuration Register 1 (BIUCR): globally controls prefetch behavior and read and write wait states.
- Bus Interface Unit Access Protection Register (BIUAPR) controls the read or write access to the flash memory array by specifying access for each master port on the XBAR.
- Bus Interface Unit Configuration Register 2 (BIUCR2) globally controls operating behavior of the four page buffers.
- Platform Flash Configuration Register 3 (PFCR3) contains certain bank1 configuration fields.

These registers are included on all devices in the SPC563Mxx family.

10.5.1.1.1 Bus Interface Unit Configuration Register (BIUCR)

The BIUCR register contains global configuration fields associated with all attached flash arrays.

Address: 0xC3F8_801C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R								GCE					M3PFE	M2PFE	M1PFE	M0PFE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	APC			WWSC		RWSC				DPFEN		IPFEN		PFLIM		BFEN
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 50. Bus Interface Unit Configuration Register (BIUCR)

Table 56. BIUCR field descriptions

Field	Description
GCE	Global Configuration Enable (Read/Write) This bit determines if specific fields contained in BIUCR apply globally to the PFLASH configuration or apply only to the Bank0 configuration. In particular, this applies to BIUCR[DPFEN, IPFEN, PFLIM, BFEN]. This bit is cleared by reset. 0: Use the contents of BIUCR to configure Bank0 operation, PFCR3 to configure bank1 operation 1: Use the contents of BIUCR to configure both Bank0 and Bank1 operation
M3PFE	Master 3 Prefetch Enable (Read/Write) This field is not implemented; it has no effect.
M2PFE	Master 2 Prefetch Enable (Read/Write) These bits are used to control whether prefetching may be triggered by the DMA module. If prefetch is enabled, prefetch behavior is controlled by a combination of the BIUCR[GCE] value and other settings as follows. If BIUCR[GCE] = 1, the BIUCR[DPFEN, IPFEN, BFEN] settings apply to Bank0 and Bank1. If BIUCR[GCE] = 0, the BIUCR[DPFEN, IPFEN, BFEN] settings apply only to Bank0, and the PFCR3[DPFE, IPFE, BFE] settings control prefetch behavior for Bank1. This field is cleared by reset, disabling all prefetching for the DMA module. 0: No prefetching may be triggered by the DMA module 1: Prefetching may be triggered by the DMA module
M1PFE	Master 1 Prefetch Enable (Read/Write) These bits are used to control whether prefetching may be triggered by the Nexus Development Interface (NDI). If prefetch is enabled, prefetch behavior is controlled by a combination of the BIUCR[GCE] value and other settings as follows. If BIUCR[GCE] = 1, the BIUCR[DPFEN, IPFEN, BFEN] settings apply to Bank0 and Bank1. If BIUCR[GCE] = 0, the BIUCR[DPFEN, IPFEN, BFEN] settings apply only to Bank0, and the PFCR3[DPFE, IPFE, BFE] settings control prefetch behavior for Bank1. This field is cleared by reset, disabling all prefetching for the NDI. 0: No prefetching may be triggered by the NDI 1: Prefetching may be triggered by the NDI
M0PFE	Master 0 Prefetch Enable (Read/Write) These bits are used to control whether prefetching may be triggered by the e200z335 processor core. If prefetch is enabled, prefetch behavior is controlled by a combination of the BIUCR[GCE] value and other settings as follows. If BIUCR[GCE] = 1, the BIUCR[DPFEN, IPFEN, BFEN] settings apply to Bank0 and Bank1. If BIUCR[GCE] = 0, the BIUCR[DPFEN, IPFEN, BFEN] settings apply only to Bank0, and the PFCR3[DPFE, IPFE, BFE] settings control prefetch behavior for Bank1. This field is cleared by reset, disabling all prefetching for the processor core. 0: No prefetching may be triggered by the processor core 1: Prefetching may be triggered by the processor core

Table 56. BIUCR field descriptions (continued)

Field	Description
APC	Address Pipelining Control (Read/Write) This field is used to control the number of cycles between pipelined access requests. This field must be set to a value corresponding to the operating frequency and it must be the same value as RWSC. This field is set to '111' by hardware reset. 000: Accesses may be initiated on consecutive (back-to-back) cycles 001: Access requests require 1 additional hold cycle 010: Access requests require 2 additional hold cycles 011: Access requests require 3 additional hold cycles 100: Access requests require 4 additional hold cycles 101: Access requests require 5 additional hold cycles 110: Access requests require 6 additional hold cycles 111: No address pipelining
WWSC	Write Wait State Control (Read/Write) This field is used to control the number of wait-states to be added to the best-case flash array access time for writes. The best-case flash array access time for writes is two cycles. This field must be set to a value corresponding to the operating frequency. This field is set to '11' 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
RWSC	Read Wait State Control (Read/Write) This field is used to control the number of wait states to be added to the best-case flash array access time for reads. The best-case flash array access time for reads is one cycle. This field must be set to a value corresponding to the operating frequency and it must be the same value as APC. This field is set to '111' by hardware reset. 000: No additional wait-states are added 001: 1 additional wait-state is added 010: 2 additional wait-states are added 011: 3 additional wait-states are added 100: 4 additional wait-states are added 101: 5 additional wait-states are added 110: 6 additional wait-states are added 111: 7 additional wait-states are added
DPFEN	Data Prefetch Enable (Read/Write) This field enables or disables prefetching initiated by a data read access. This field is a global control affecting the operation of both flash banks if BIUCR[GCE] = 1, else it controls only Bank0. It is cleared by hardware reset. 0: No prefetching is triggered by a data read access 1: If page buffers are enabled (BFEN = 1), prefetching is triggered by any data read access

Table 56. BIUCR field descriptions (continued)

Field	Description
IPFEN	Instruction Prefetch Enable (Read/Write) This field enables or disables prefetching initiated by an instruction fetch read access. This field is a global control affecting the operation of both flash banks if BIUCR[GCE] = 1, else it controls only Bank0. It is cleared by hardware reset. 0: No prefetching is triggered by an instruction fetch read access 1: If page buffers are enabled (BFEN = 1), prefetching is triggered by any instruction fetch read access
PFLIM	Prefetch Limit (Read/Write) 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 a global control affecting the operation of both flash banks if BIUCR[GCE] = 1, else it controls only Bank0. It is cleared by hardware reset. 00: No prefetching is performed. 01: The referenced line is prefetched on a buffer miss, that is, prefetch on miss. 1x: 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.
BFEN	Buffer Enable (Read/Write) This bit enables or disables page buffer read hits and is a global control affecting the operation of both flash banks if BIUCR[GCE] = 1, else it controls only Bank0. It is also used to invalidate the buffers. This bit is cleared 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.

10.5.1.1.2 Bus Interface Unit Access Protection Register (BIUAPR)

The BIUAPR controls the read or write access to the flash memory array given to the master ports of the XBAR.

Address: 0xC3F8_8020

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
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									M3AP		M2AP		M1AP		M0AP	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

= Unimplemented or Reserved

Figure 51. Bus Interface Unit Access Protection Register (BIUAPR)

Table 57. BIUAPR field descriptions

Field	Description
M3AP	Master 3 Access Protection (Read/Write) This field is not implemented; it has no effect.
M2AP	Master 2 Access Protection (Read/Write) Controls whether the DMA module is allowed to make read and write accesses to the flash. 00: No accesses may be performed by the DMA module 01: Only read accesses may be performed by the DMA module 10: Only write accesses may be performed by the DMA module 11: Both read and write accesses may be performed by the DMA module
M1AP	Master 1 Access Protection (Read/Write) Controls whether the Nexus Development Interface (NDI) is allowed to make read and write accesses to the flash. 00: No accesses may be performed by the NDI 01: Only read accesses may be performed by the NDI 10: Only write accesses may be performed by the NDI 11: Both read and write accesses may be performed by the NDI
M0AP	Master 0 Access Protection (Read/Write) Controls whether the e200z335 processor core is allowed to make read and write accesses to the flash. 00: No accesses may be performed by the processor core 01: Only read accesses may be performed by the processor core 10: Only write accesses may be performed by the processor core 11: Both read and write accesses may be performed by the processor core

See *Section 10.2.3.3: Censorship* for information on how flash censorship can affect accesses to the BIUAPR. See *Chapter 8: Multi-Layer AHB Crossbar Switch (XBAR)* for more information on the Crossbar.

10.5.1.1.3 Bus Interface Unit Configuration Register 2 (BIUCR2)

The BIUCR2 globally defines the logical partitioning of the four page buffers.

To temporarily change the values of any of the fields in the BIUCR2, a write to the register is performed. To change the values loaded into the BIUCR2 at reset, the word location at address 0x00FF_FE00 of the shadow block in Bank0, Array0 must be programmed using the normal sequence of operations

Address: 0xC3F8_8024

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LBCFG															
W																
Reset	See Note(1)															

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

= Unimplemented or Reserved

Figure 52. Bus Interface Unit Configuration Register 2 (BIUCR2)

1. Default reset value comes from the Shadow Block - 0xFF_FE00.

Table 58. BIUCR2 field descriptions

Field	Description
LBCFG	Line Buffer Configuration (Read/Write) This field globally controls the configuration of the four line 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. In all cases, 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. This field is initialized by hardware reset to the value contained in address 0x00FF_FE00 (in the Shadow Block of Bank0, Array0). An erased or unprogrammed flash sets this field to 0b11. 00: All four buffers are available for any flash access, that is, there is no partitioning 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.

10.5.1.1.4 Platform Flash Configuration Register 3 (PFCR3)

The PFLASH Configuration Register 3 (PFCR3) defines certain bank1 configuration fields.

To temporarily change the values of any of the fields in the PFCR3, a write to the register is performed. To change the values loaded into the PFCR3 at reset, the word location at address 0x00FF_FE08 of the shadow block in Bank0, Array0 must be programmed using the normal sequence of operations.

Address: 0xC3F8_8028

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R										B1_DPFE		B1_IPFE		B1_PFLIM		B1_BFE
W																
Reset																

= Unimplemented or Reserved

Figure 53. PFLASH Configuration Register 3 (PFCR3)

1. Reset value is read from 0xFF_FE08 in the shadow block.

Table 59. PFCR3 field descriptions

Field	Description
B1_DPFE	Bank1 Data Prefetch Enable (Read/Write) This field enables or disables prefetching initiated by a data read access to Bank1. This field is operational only if BIUCR[GCE] = 0. An erased or unprogrammed flash sets this field to 0b1. 0: No prefetching is triggered by a data read access 1: If page buffers are enabled (BFE = 1), prefetching is triggered by any data read access
B1_IPFE	Bank1 Instruction Prefetch Enable (Read/Write) This field enables or disables prefetching initiated by an instruction fetch read access to Bank1. This field is operational only if BIUCR[GCE] = 0. An erased or unprogrammed flash sets this field to 0b1. 0: No prefetching is triggered by an instruction fetch read access 1: If page buffers are enabled (BFE = 1), prefetching is triggered by any instruction fetch read access
B1_PFLIM	Bank1 Prefetch Limit (Read/Write) This field controls the prefetch algorithm used by the PFLASH controller for Bank1. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is operational only if BIUCR[GCE] = 0. An erased or unprogrammed flash sets this field to 0b11. 00: No prefetching is performed. 01: The referenced line is prefetched on a buffer miss, that is, prefetch on miss. 1x: 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.
B1_BFE	Bank1 Buffer Enable (Read/Write) This bit enables or disables page buffer read hits and it is also used to invalidate the buffers. This field only configures bank1 if BIUCR[GCE] = 0. 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.

10.5.1.2 Flash array module registers

As detailed previously (see *Figure 48*), devices in the SPC563Mxx family have up to three flash memory array modules in two banks.

- Bank0 has one 528 KB module that includes a 16 KB shadow block. On devices with 768 KB of flash memory, only the first 256 KB of addressable space plus the shadow block is used.
- Bank1 contains one or two flash array modules, one (Array1) in 768 KB and 1 MB devices, and two (Array1 and Array2) in 1.5 MB devices.

Each flash array module register controls configuration and behavior for a specific array. Each flash array has the following registers but not all registers are used for all arrays:

- Module configuration register (MCR)
- Low/mid address space block locking (LMLR) register
- High address space block locking (HLR) register (not used on Bank0, Array0)
- Secondary low/mid address space block lock (SLMLR) register (used only on Bank0, Array0)
- Low/Mid address space block select (LMSR) register (used only on Bank0, Array0)
- High address space block select (HSR) register (not used on Bank0, Array0)
- Address (AR) register
- User test 0 (UT0) register
- User test 1 (UT1) register
- User test 2 (UT2) register
- User multiple input signature register 0 (UMISR0) register
- User multiple input signature register 1 (UMISR1) register
- User multiple input signature register 2 (UMISR2) register
- User multiple input signature register 3 (UMISR3) register
- User multiple input signature register 4 (UMISR4) register

Note: When a register is described as not being used, e.g., the HLR register is not used in Bank0, Array0, that means it physically exists but is not used to control the behavior of its associated array. The register may be used for some internal functions so programs should not use those registers for any reason.

10.5.1.2.1 Module Configuration Register (MCR)

The MCR contains ECC status data, read-only configuration information, read-while-write status information and read/write control fields.

All flash array modules have an associated MCR. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The MCR is defined in *Figure 54* and *Table 61*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_8000
Bank1, Array1: 0xC3FB_0000
Bank1, Array2: 0xC3FB_4000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EDC					SIZE				LAS						MAS
W																
Reset	See Note(1)															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE			PEAS	DONE	PEG					PGM	PSUS	ERS	ESUS	EHV
W																
Reset																

= Unimplemented or Reserved

Figure 54. Module Configuration Register (MCR)

1. The reset value is: 0x02700600 for the 544 KB array module (Array0); 0x02000600 for the 528 KB array modules (Array1 and Array2).

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 *Table 61*, but those locks do not consider the effects of trying to write two or more bits simultaneously.

The flash does not allow an application to write bits simultaneously that would put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in *Table 60*.

Table 60. MCR bits set/clear priority levels

Priority level	MCR bits
1	ERS
2	PGM
3	EHV
4	ESUS

Table 61. MCR field descriptions

Field	Description
EDC	ECC Data Correction (Read/Clear) EDC provides information on previous reads. If an ECC Single Error detection and correction occurred, the EDC bit is set to 1. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by an application. In the event of an ECC Double Error detection, this bit is not set. If EDC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EDC) were not corrected through ECC. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the bit. A write of 0 has no effect. The function of this bit is device dependent and it can be configured to be disabled (through UT0[SBCE]). 0: Reads are occurring normally. 1: An ECC Single Error occurred and was corrected during a previous read.
SIZE	Array Size (Read Only) 00: 128 KB 001: 256 KB 010: 512 KB 011: 1056 KB 100: 1536 KB 101: Reserved 110: 64 KB 111: Reserved
LAS	Low Address Space (Read Only) Indicates configuration of the Low Address Space. Value is array-dependent. Bank0, Array0: 111: 2 × 16 KB + 2 × 32 KB + 2 × 16 KB + 2 × 64 KB blocks in low address space. Bank1, Array1 and Bank1, Array2: 000: No small blocks
MAS	Mid Address Space (Read Only) Indicates configuration of the Mid Address Space. Bank0, Array0: 0: 2 × 128 KB blocks in mid address space Bank1, Array1 and Bank1, Array2: No mid address space
EER	ECC Event Error (Read/Clear) EER provides information on previous reads. If a double bit error detection occurs, 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 an application. In the event of a single bit detection and correction, this bit is not set. If EER is not set, or remains 0, this indicates all previous reads (from the last reset, or clearing of EER) were correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the bit. A write of 0 has no effect. 0: Reads are occurring normally 1: An ECC Double Error occurred during a previous read

Table 61. MCR field descriptions (continued)

Field	Description
RWE	Read-While-Write Event Error (Read/Clear) RWE provides information on previous reads when a Modify operation is ongoing. If a RWW Error occurs, the RWE bit is set to 1. Read-While-Write Error means that a read access to the flash array has occurred while a Program or Erase operation or an Array Integrity Check was being performed. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by an application. If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) were correct. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the bit. A write of 0 will have no effect. 0: Reads are occurring normally. 1: A RWW Error occurred during a previous read.
PEAS	Program/Erase Access Space (Read Only) 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 flash controller 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. 0: Shadow address space is disabled for program/erase and main address space enabled 1: Shadow address space is enabled for program/erase and main address space disabled
DONE	Status (Read Only) DONE indicates whether the flash module is performing a high voltage operation. – DONE is set to a 1 on termination of the flash module reset. – DONE is cleared by a 0 to 1 transition of EHV which initiates a high voltage operation. – DONE is cleared by 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 by a 1 to 0 transition of EHV which aborts a high voltage operation. 0: Flash is executing a high voltage operation 1: Flash is not executing a high voltage operation

Table 61. MCR field descriptions (continued)

Field	Description
PEG	Program/Erase Good (Read Only) 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 will cause PEG to be cleared to 0, indicating the sequence failed. PEG is set to 1 when the flash module is reset, unless a flash initialization error has been detected. The value of PEG is valid only when PGM = 1 and/or ERS = 1 and after DONE transitions from 0 to 1 due to an abort or the completion of a Program/Erase operation. PEG is valid until PGM/ERS makes a 1 to 0 transition or EHV makes a 0 to 1 transition. The value in PEG is not valid after a 0 to 1 transition of DONE caused by ESUS being set to logic 1. If Program or Erase operations are attempted on blocks that are locked, the response will be PEG = 1, indicating that the operation was successful, and the content of the block were properly protected from the Program or Erase operation. If a Program operation tries to program to ‘1’ bits that are at ‘0’, the program operation is correctly executed on the new bits to be programmed at ‘0’, but PEG is cleared, indicating that the requested operation has failed. In Array Integrity Check or Margin Mode PEG is set to 1 when the operation is completed, regardless of the occurrence of any error. The presence of errors can be detected only comparing checksum value stored in UMIRS0–1. Aborting an Array Integrity Check or a Margin Mode operation will cause PEG to be cleared to 0, indicating the sequence failed. 0: Program or Erase operation failed. 1: Program or Erase operation successful.
PGM	Program (Read/Write) PGM is used to setup the flash array module for a Program operation. A 0 to 1 transition of PGM initiates a Program sequence. A 1 to 0 transition of PGM ends the Program sequence. PGM can be set only under User Mode Read (ERS is low and UT0[AIE] is low). PGM can be cleared by an application only when EHV is low and DONE is high. PGM is cleared on reset. 0: Flash is not executing a Program sequence. 1: Flash is executing a Program sequence.
PSUS	Program Suspend (Read/Write) A write to this bit has no effect, but the written data can be read back.
ERS	Erase (Read/Write) Used to setup the flash array module for an Erase operation. A 0 to 1 transition of ERS initiates an Erase sequence. A 1 to 0 transition of ERS ends the Erase sequence. ERS can be set only under User Mode Read (PGM is low and UT0[AIE] is low). ERS can be cleared by an application only when ESUS and EHV are low and DONE is high. ERS is cleared on reset. 0: Flash is not executing an Erase sequence. 1: Flash is executing an Erase sequence.

Table 61. MCR field descriptions (continued)

Field	Description
ESUS	Erase Suspend (Read/Write) Used to indicate the flash array module is in Erase Suspend or in the process of entering a Suspend state. The flash array 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. 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. 0: Erase sequence is not suspended. 1: Erase sequence is suspended.
EHV	Enable High Voltage (Read/Write) The EHV bit enables the flash module for a high voltage Program/Erase operation. EHV is cleared on reset. EHV must be set after an interlock write to start a Program/Erase sequence. EHV may be set under one of the following conditions: – Erase (ERS = 1, ESUS = 0, UT0[AIE] = 0) – Program (ERS = 0, ESUS = 0, PGM = 1, UT0[AIE] = 0) In normal operation, a 1 to 0 transition of EHV with DONE high and ESUS low terminates the current Program/Erase high voltage operation. When an operation is aborted, there is a 1 to 0 transition of EHV with DONE low and the eventual Suspend bit low. An abort causes the value of PEG to be cleared, indicating a failing Program/Erase. Address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted. Aborting a high voltage operation will leave the flash module addresses in an indeterminate data state. This may be recovered by executing an Erase on the affected blocks. EHV may be written during Suspend. EHV must be high to exit Suspend. EHV may not be written after ESUS is set and before DONE transitions high. EHV may not be cleared after ESUS is cleared and before DONE transitions low. 0: Flash is not enabled to perform an high voltage operation. 1: Flash is enabled to perform an high voltage operation.

10.5.1.2.2 Low/Mid Address Space Block Locking Register (LMLR)

The LMLR provides a means to protect blocks from being modified. These bits, along with bits in the SLMLR register, determine if the block is locked from program or erase. A logical “OR” of the LMLR and SLMLR register values determine the final lock status.

Only Bank0 Array0 has uses an LMLR register. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The LMLR register is defined in *Figure 55* and *Table 62*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_8004
Bank1, Array1: 0xC3F8_0004
Bank1, Array2: 0xC3F8_0004

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME											SLOCK			MLOCK	
W																
Reset(1)	0	0	0	0	0	0	0	0	0	0	0	X	0	0	X	X

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

= Unimplemented or Reserved

Figure 55. Low/Mid Address Space Block Locking Register (LMLR)

1. Default reset value comes from the Test Flash - Address is 0xBF_FDE8.

Table 62. LMLR field descriptions

Field	Description
LME	Low/Mid address space block enable (Read Only) This bit indicates whether the lock fields (SLOCK, MLOCK and LLOCK) can be set or cleared by register writes. This bit is a status bit only. This bit is set by writing a password, and if the password matches, the LME bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the LMLR register. 0: Low address locks are disabled: SLOCK, MLOCK1–0 and LLOCK7–0 cannot be written. 1: Low address locks are enabled: SLOCK, MLOCK1–0 and LLOCK7–0 can be written.
SLOCK	Shadow address space block lock (Read/Write) This bit is used to lock the block of Shadow Address Space from Program and Erase. It is also used to unlock the flash test sector. A value of 1 in the SLOCK field signifies that the Shadow block is locked for Program and Erase. A value of 0 signifies that the Shadow block is available to receive Program and Erase pulses. The SLOCK field is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SLOCK field is not writable if a high voltage operation is suspended. Upon reset, information from a protected area of flash is loaded into the SLOCK field. Reset will cause the bit to go back to its stored. The default value of the SLOCK bit (assuming erased fuses) is locked. SLOCK is not writable unless LME is high. 0: Shadow address space block is unlocked and can be modified (if also SLMLR[SSLOCK] = 0). 1: Shadow address space block is locked and cannot be modified.

Table 62. LMLR field descriptions (continued)

Field	Description
MLOCK	Mid address space block lock (Read/Write) These bits are used to individually lock or unlock blocks of flash memory residing in the mid address space of Array0 from being programmed or erased. Referring to *Table 52 on page 233*, MLOCK[1–0] apply to blocks 7 and 6, respectively. A value of 1 signifies that the block is locked from Program and Erase operations. A value of 0 signifies that the block is available to receive Program and Erase pulses. The MLOCK field is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the MLOCK field is not writable if a high voltage operation is suspended. Upon reset, information from a protected area of flash is loaded into the MLOCK field. Reset will cause the bits to go back to their stored values. The default value of each MLOCK bit (assuming erased fuses) is locked. In the event that blocks are not present (due to configuration or total memory size), the MLOCK bits default to locked, and are not writable. The reset value will always be 1, and writes will have no effect. MLOCK is not writable unless LME is high. 0: Mid address space block is unlocked and can be modified (if also SLMLR[SSLOCK] = 0). 1: Mid address space block is locked and cannot be modified.
LLOCK	Low address space block lock (Read/Write) These bits are used to individually lock or unlock blocks of flash memory residing in the low address space of Array0 from being programmed or erased. Referring to *Table 52 on page 233*, LLOCK[7–0] apply to blocks 5, 4, 3, 2b, 2a, 1b, 1a and 0, respectively. A value of 1 in a bit of the LLOCK field signifies that the corresponding block is locked for Program and Erase. A value of 0 signifies that the corresponding block is available to receive Program and Erase pulses. The LLOCK field is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the LLOCK field is not writable if a high voltage operation is suspended. Upon reset, information from a protected area of flash is loaded into the LLOCK field. Reset will cause the bits to go back to their stored values. The default value of each LLOCK bit (assuming erased fuses) is locked. In the event that blocks are not present (due to configuration or total memory size), the LLOCK bits default to locked, and are not writable. The reset value will always be 1, and writes will have no effect. LLOCK is not writable unless LME is high. 0: Low address space block is unlocked and can be modified (if also SLMLR[SLLOCK] = 0). 1: Low address space block is locked and cannot be modified.

10.5.1.2.3 High Address Space Block Locking Register (HLR)

The HLR provides a means to protect blocks from being modified.

All flash array modules except Bank0, Array0 use an HLR. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.1

The HLR is defined in *Figure 56* and *Table 63*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: N/A
Bank1, Array1: 0xC3FB_0008
Bank1, Array2: 0xC3FB_4008

Access: User read/write

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

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

= Unimplemented or Reserved

Figure 56. High Address Space Block Locking Register (HLR)

1. Default reset value comes from the Test Flash - Address is 0xBF_FDF0.

Table 63. HLR field descriptions

Field	Description
HBE	High address space Block Enable (Read Only) This bit is used to enable the lock bits (HBLOCK[3–0]) to be set or cleared by register writes. This bit is a status bit only. This bit is set by writing 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 0xB2B22222 must be written to the HLR. 0: High address space locks are disabled: HBLOCK[3–0] cannot be written. 1: High address space locks are enabled: HBLOCK[3–0] can be written.

Table 63. HLR field descriptions (continued)

Field	Description
HBLOCK	High address space block lock (Read/Write) These bits are used to lock or unlock blocks of High Address Space from Program and Erase operations. A value of 1 in a bit of the HBLOCK field signifies that the corresponding block is locked for Program and Erase. A value of 0 signifies that the corresponding block is available to receive Program and Erase pulses. Referring to *Table 52 on page 233*, HBLOCK[3–0] correspond to the blocks in the high address space of the flash array associated with this register. In Array1, HBLOCK[3-0] apply to blocks 11, 10, 9 and 8, respectively. In Array2, HBLOCK[3-0] apply to blocks 15, 14, 13 and 12, respectively. The HBLOCK field is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the HBLOCK field is not writable if a high voltage operation is suspended. Upon reset, information from a protected flash block is loaded into the HBLOCK field. Reset will cause the bits to go back to their stored value. The default value of the HBLOCK bits (assuming erased fuses) is locked. In the event that blocks are not present (due to configuration or total memory size), the HBLOCK bits will default to locked, and will not be writable. The reset value will always be 1 and writes will have no effect. HBLOCK is not writable unless HBE is high. 0: High address space block is unlocked and can be modified. 1: High address space block is locked and cannot be modified.

10.5.1.2.4 Secondary Low/Mid Address Space Block Lock Register (SLMLR)

The SLMLR provides an alternative means to protect blocks from being modified. These bits, along with bits in the LMLR, determine if the block is locked from Program or Erase. An “OR” of LMLR and SLMLR determine the final lock status.

Only Bank0 Array0 uses an SLMLR. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The SLMLR is defined in *Figure 57* and *Table 64*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_800C
Bank1, Array1: N/A
Bank1, Array2: N/A

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE											SSLOCK			SMLOCK	
W																
Reset(1)	0	0	0	0	0	0	0	0	0	0	0	X	0	0	X	X

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

= Unimplemented or Reserved

Figure 57. Secondary Low/Mid Address Space Block Lock Register (SLMLR)

1. The SLMLR register default reset value comes from the Test Flash - Address is 0xBF_FDF8.

Table 64. SLMLR field descriptions

Field	Description
SLE	Secondary low/mid address space block enable (Read Only) This bit is used to enable the lock bit (STSLK, SMK1-0, and SLK15-0) to be set or cleared by register writes. This bit is a status bit only. The bit is set by writing a password, and if the password matches, the SLE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE, the password 0xC3C3_3333 must be written to the SLL register. 0: Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0, and SLK15-0 cannot be written. 1: Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0, and SLK15-0 can be written.
SSLOCK	Secondary shadow address space block lock (Read/Write) This bit is used as an alternate means to lock the block of Shadow address space from Program and Erase. It also locks/unlocks the flash test block. A value of 1 signifies that the Shadow block is locked for Program and Erase. A value of 0 signifies that the Shadow block is available to receive Program and Erase pulses. The SSLOCK field is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SSLOCK field is not writable if a high voltage operation is suspended. Upon reset, information from a protected area of flash is loaded into the SSLOCK field. Reset will cause the bit to go back to its stored block value. The default value of the SSLOCK bit (assuming erased fuses) would be locked. 0: Shadow address space block is unlocked and can be modified (if also LMLR[SLOCK] = 0). 1: Shadow address space block is locked and cannot be modified.

Table 64. SLMLR field descriptions (continued)

Field	Description
SMLOCK	Secondary mid address space block lock (Read/Write) These bits provide an alternative means to individually lock or unlock blocks of flash memory residing in the mid address space of Array0 from being programmed or erased. Referring to *Table 52 on page 233*, MLOCK[1–0] apply to blocks 7 and 6, respectively. A value of 1 signifies that the block is locked from Program and Erase operations. A value of 0 signifies that the block is available to receive Program and Erase pulses. The SMLOCK field is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SMLOCK field is not writable if a high voltage operation is suspended. Upon reset, information from a protected area of flash is loaded into the SMLOCK field. Reset will cause the bits to go back to their stored values. The default value of each SMLOCK bit (assuming erased fuses) is locked. In the event that blocks are not present (due to configuration or total memory size), the SMLOCK bits default to locked, and are not writable. The reset value will always be 1, and writes will have no effect. 0: Mid address space block is unlocked and can be modified (if also LMLR[MLOCK] = 0). 1: Mid address space block is locked and cannot be modified.
SLLOCK	Secondary low address space block lock (Read/Write) These bits are used to individually lock or unlock blocks of flash memory residing in the low address space of Array0 from being programmed or erased. Referring to *Table 52 on page 233*, SLLOCK[7–0] apply to blocks 5, 4, 3, 2b, 2a, 1b, 1a and 0, respectively. A value of 1 in a bit of the SLLOCK field signifies that the corresponding block is locked for Program and Erase. A value of 0 signifies that the corresponding block is available to receive Program and Erase pulses. The SLLOCK field is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SLLOCK field is not writable if a high voltage operation is suspended. Upon reset, information from a protected area of flash is loaded into the SLLOCK field. Reset will cause the bits to go back to their stored values. The default value of each SLLOCK bit (assuming erased fuses) is locked. In the event that blocks are not present (due to configuration or total memory size), the SLLOCK bits default to locked, and are not writable. The reset value will always be 1, and writes will have no effect. 0: Low address space block is unlocked and can be modified (if also LMLR[LLOCK] = 0). 1: Low address space block is locked and cannot be modified.

10.5.1.2.5 Low/Mid Address Space Block Select Register (LMSR)

The LMSR provides a means to select blocks to be operated on during erase.

Only Bank0 Array0 uses an LMSR. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The LMSR is defined in *Figure 58* and *Table 65*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_8010
Bank1, Array1: N/A
Bank1, Array2: N/A

Access: User read/write

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

= Unimplemented or Reserved

Figure 58. Low/Mid Address Space Block Select Register (LMSR)

Table 65. LMSR field descriptions

Field	Description
MSEL	Mid address space block select (Read/Write) A value of 1 in an MSEL bit signifies that the associated block is selected for erase. A value of 0 signifies that the block is not selected for erase. The reset value for the field is 0, or unselected. Referring to *Table 52 on page 233*, MSEL[1–0] affect the blocks in the mid address space of Array0 and apply to blocks 7 and 6, respectively. The blocks must be selected (or unselected) before doing an erase interlock write as part of the Erase sequence. The select field is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding MSEL bits will default to unselected, and will not be writable. The reset value will always be 0, and writes will have no effect. 0: Mid address space block is unselected for Erase. 1: Mid address space block is selected for Erase.
LSEL	Low address space block select (Read/Write) A value of 1 in an LSEL bit signifies that the associated block is selected for erase. A value of 0 signifies that the block is not selected for erase. The reset value is 0, or unselected. Referring to *Table 52 on page 233*, LSEL[7–0] affect the blocks in the low address space of Array0. Bits LSEL[7–0] apply to blocks 5, 4, 3, 2b, 2a, 1b, 1a and 0, respectively. The blocks must be selected (or unselected) before doing an erase interlock write as part of the Erase sequence. The select field is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding LSEL bits will default to unselected, and will not be writable. The reset value will always be 0, and writes will have no effect. 0: Low address space block is unselected for Erase. 1: Low address space block is selected for Erase.

10.5.1.2.6 High Address Space Block Select Register (HSR)

The HSR register provides a means to select blocks to be operated on during erase.

All flash array modules except Bank0, Array0 use an HSR register. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The HSR register is defined in *Figure 59* and *Table 66*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: N/A
Bank1, Array1: 0xC3FB_0014
Bank1, Array2: 0xC3FB_4014

Access: User read/write

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

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

= Unimplemented or Reserved

Figure 59. High Address Space Block Select Register (HSR)

Table 66. HSR field descriptions

Field	Description
HBSEL	High address space block select (Read/Write) A value of 1 in an HBSEL bit signifies that the associated block is selected for erase. A value of 0 signifies that the block is not selected for erase. The reset value is 0, or unselected. Referring to *Table 52 on page 233*, HBSEL[3–0] correspond to the blocks in the high address space of the flash array associated with this field. In Array1, HBLOCK[3–0] apply to blocks 11, 10, 9 and 8, respectively. In Array2, HBLOCK[3–0] apply to blocks 15, 14, 13 and 12, respectively. The blocks must be selected (or unselected) before doing an erase interlock write as part of the Erase sequence. The select field is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding HBSEL bits will default to unselected, and will not be writable. The reset value will always be 0, and writes will have no effect. 0: High Address Space Block is unselected for Erase. 1: High Address Space Block is selected for Erase.

10.5.1.2.7 Address Register (AR)

The Address Register provides the first failing address in the event of module-level failures (ECC, RWW or flash controller) or the first address at which an ECC single error correction occurs.

All flash array modules have an associated AR. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The AR is defined in *Figure 60* and *Table 67*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_8018
Bank1, Array1: 0xC3FB_0018
Bank1, Array2: 0xC3FB_4018

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SAD											ADDR[20–16]				
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	ADDR[15–3]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 60. Address Register (AR)

Table 67. AR field descriptions

Field	Description
SAD	Shadow address (Read Only) When this bit is high, the address indicated by ADDR[20–3] belongs to the Shadow block.
ADDR	Address 20–3 (Read Only) The Address Register provides the first failing address in the event of an ECC error (MCR[EER] set) or the first failing address in the event of an RWW error (MCR[RWE] set), or the address of a failure that may have occurred in a flash controller operation (MCR[PEG] cleared). The Address Register also provides the first address at which an ECC single error correction occurs (MCR[EDC] set), if the device is configured to show this feature (through UT0[SBCE]). The ECC double error detection takes the highest priority, followed by the RWW error, the flash controller error and the ECC single error correction. When accessed AR will provide the address related to the first event occurred with the highest priority. The priorities between these four possible events is summarized in *Table 68*. This address is always a double word address that selects 64 bits. In case of a simultaneous ECC Double Error Detection on both double words of the same page, bit ADDR3 will output 0. The same is valid for a simultaneous ECC Single Error Correction on both double words of the same page.

Table 68. AR content priority list

Priority level	Error flag	AR content
1	MCR[EER] = 1	Address of first ECC Double Error
2	MCR[RWE] = 1	Address of first RWW Error

Table 68. AR content priority list (continued)

Priority level	Error flag	AR content
3	MCR[PEG] = 0	Address of first flash controller Error
4	MCR[EDC] = 1	Address of first ECC Single Error Correction

10.5.1.2.8 User Test 0 (UT0) Register

The UT0 register controls ability to perform test features on the flash.

All flash array modules have a UT0 register. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The UT0 register is defined in *Figure 61* and *Table 69*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_803C
Bank1, Array1: 0xC3FB_003C
Bank1, Array2: 0xC3FB_403C

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R										X	MRE	MRV	EIE	AIS	AIE	AID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

= Unimplemented or Reserved

Figure 61. User Test 0 (UT0) Register

Table 69. UT0 field descriptions

Field	Description
UTE	User test enable (Read/Clear) This status bit gives indicates User Test is enabled. All bits in the UT0–2 and UMISR0–4 registers are locked when this bit is 0. This bit is not writable 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 write. For UTE the password 0xF9F99999 must be written to the UT0 register.
SBCE	Single bit correction enable (Read/Clear) This bit, when high, enables the device to flag the information about any eventual ECC single bit correction in the flash array (MCR[EDC]).

Table 69. UT0 field descriptions (continued)

Field	Description
DSI	Data syndrome input (Read/Write) These bits represent the input of syndrome bits of ECC logic used in the ECC Logic Check. The DSI7–0 bits correspond to the 8 syndrome bits on a double word. These bits are not accessible whenever MCR[DONE] or UT0[AID] are low. Reading returns indeterminate data; writing has no effect. 0: The syndrome bit is forced at 0. 1: The syndrome bit is forced at 1.
Reserved (bit 25)	Reserved (Read/Write) This bit can be written and its value can be read back, but there is no function associated. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading returns indeterminate data; writing has no effect.
MRE	Factory Margin Read Enable (Read/Write) MRE enables margin reads to be done. This bit, combined with MRV, enables regular user mode reads to be replaced by margin reads. Margin reads are only active during Array Integrity Checks; Normal User reads are not affected by MRE. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading returns indeterminate data; writing has no effect. 0: Margin reads are not enabled. 1: Margin reads are enabled.
MRV	Factory Margin Read Value (Read/Write) If MRE is high, MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0). This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading returns indeterminate data; writing has no effect. 0: Zero's (programmed) margin reads are requested (if MRE = 1). 1: One's (erased) margin reads are requested (if MRE = 1).
EIE	ECC data Input Enable (Read/Write) EIE enables the ECC Logic Check operation to be done. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading returns indeterminate data; writing has no effect. 0: ECC Logic Check is not enabled. 1: ECC Logic Check is enabled.
AIS	Array Integrity Sequence (Read/Write) AIS determines the address sequence to be used during array integrity checks. The default sequence (AIS = 0) is meant to replicate sequences normal user code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS = 1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading returns indeterminate data; writing has no effect. 0: Array Integrity or Margin Mode sequence is proprietary sequence. 1: Array Integrity or Margin Mode sequence is sequential.

Table 69. UT0 field descriptions (continued)

Field	Description
AIE	Array Integrity Enable (Read/Write) AIE set to 1 starts the Array Integrity Check done on all selected and unlocked blocks. The pattern is selected by AIS, and the MISR (UMISR0–4) can be checked after the operation is complete, to determine if a correct signature is obtained. AIE can be set only if MCR[ERS], MCR[PGM] and MCR[EHV] are all low. 0: Array Integrity Checks, Factory Margin Reads and ECC Logic Checks are not enabled. 1: Array Integrity Checks, Factory Margin Reads and ECC Logic Checks are enabled.
AID	Array Integrity Done (Read Only) 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 (UMISR0–4) can be checked. 0: Array Integrity Check is ongoing. 1: Array Integrity Check is done.

10.5.1.2.9 User Test 1 (UT1) Register

The UT1 register allows to enable the checks on the ECC logic related to the 32 LSB of the double word. The register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading this register returns indeterminate data; writing has no effect.

All flash array modules have a UT1 register. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The UT1 register is defined in *Figure 62* and *Table 70*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_8040
Bank1, Array1: 0xC3FB_0040
Bank1, Array2: 0xC3FB_4040

Access: User read/write

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

= Unimplemented or Reserved

Figure 62. User Test 1 (UT1) Register

Table 70. UT1 field descriptions

Field	Description
DAI [31-0]	Data Array Input (Read/Write) These bits represent the input of even word of ECC logic used in the ECC Logic Check. The DAI bits correspond to the 32 array bits representing Word 0 within the double word. 0: The array bit is forced at 0. 1: The array bit is forced at 1.

10.5.1.2.10 User Test 2 (UT2) Register

The UT2 register enables checks on the ECC logic related to the 32 MSB of the double word.

This register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading this register returns indeterminate data; writing has no effect.

All flash array modules have a UT2 register. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The UT2 register is defined in *Figure 63* and *Table 71*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_8044
Bank1, Array1: 0xC3FB_0044
Bank1, Array2: 0xC3FB_4044

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DAI[63–48]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DAI[47–32]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 63. User Test 2 (UT2) Register

Table 71. UT2 field descriptions

Field	Description
DAI [63-32]	Data Array Input (Read/Write) These bits represent the input of odd word of ECC logic used in the ECC Logic Check. The DAI63–32 bits correspond to the 32 array bits representing Word 1 within the double word. 0: The array bit is forced at 0. 1: The array bit is forced at 1.

10.5.1.2.11 User Multiple Input Signature Register 0 (UMISR0) Register

The UMISR0 register provides a means to evaluate array integrity. Its value represents bits 31:0 of the whole 144-bit word (2 double words including ECC).

This is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading returns indeterminate data; writing has no effect.

All flash array modules have a UMISR0 register. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The UMISR0 register is defined in *Figure 64* and *Table 72*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_8048
Bank1, Array1: 0xC3FB_0048
Bank1, Array2: 0xC3FB_4048

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MS[031–016]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MS[015–000]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 64. User Multiple Input Signature Register 0 (UMISR0) Register

Table 72. UMISR0 field descriptions

Field	Description
MS [031-000]	Multiple input Signature (Read/Write) These bits represent the MISR value obtained accumulating the bits 31–0 of all the pages read from the flash memory. The MS can be seeded to any value by writing the UMISR0 register.

10.5.1.2.12 User Multiple Input Signature Register 1 (UMISR1) Register

The UMISR1 register provides a means to evaluate array integrity. Its value represents bits 63:32 of the whole 144-bit word (2 double words including ECC).

This register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading this register returns indeterminate data; writing has no effect.

All flash array modules have a UMISR1 register. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The UMISR1 register is defined in *Figure 65* and *Table 73*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_804C
Bank1, Array1: C3FB_004C
Bank1, Array2: 0xC3FB_404C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MS[063-048]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MS[047-032]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 65. User Multiple Input Signature Register 1 (UMISR1) Register

Table 73. UMISR1 field descriptions

Field	Description
MS [063-032]	Multiple input Signature 063–032 (Read/Write) These bits represent the MISR value obtained accumulating the bits 63–32 of all the pages read from the flash memory. The MS can be seeded to any value by writing the UMISR1 register.

10.5.1.2.13 User Multiple Input Signature Register 2 (UMISR2) Register

The UMISR2 register provides a means to evaluate array integrity. Its value represents bits 95:64 of the whole 144-bit word (2 double words including ECC).

This register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading this register returns indeterminate data; writing has no effect.

All flash array modules have a UMISR2 register. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The UMISR2 register is defined in *Figure 66* and *Table 74*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_8050
Bank1, Array1: 0xC3FB_0050
Bank1, Array2: 0xC3FB_4050

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MS[095–080]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MS[079–064]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 66. User Multiple Input Signature Register 2 (UMISR2) Register

Table 74. UMISR2 field descriptions

Field	Description
MS [095-064]	Multiple input Signature 095–064 (Read/Write) These bits represent the MISR value obtained accumulating the bits 95–64 of all the pages read from the flash memory. The MS can be seeded to any value by writing the UMISR2 register.

10.5.1.2.14 User Multiple Input Signature Register 3 (UMISR3) Register

The UMISR3 register provides a means to evaluate array integrity. Its value represents bits 127:96 of the whole 144-bit word (2 double words including ECC).

This register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading this register returns indeterminate data; writing has no effect.

All flash array modules have a UMISR3 register. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The UMISR3 register is defined in *Figure 67* and *Table 75*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_8054
Bank1, Array1: 0xC3FB_0054
Bank1, Array2: 0xC3FB_4054

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MS[127–112]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MS[111–096]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 67. User Multiple Input Signature Register 3 (UMISR3) Register

Table 75. UMISR3 field descriptions

Field	Description
MS [127–096]	Multiple input Signature 127–096 (Read/Write) These bits represent the MISR value obtained accumulating the bits 127–96 of all the pages read from the flash memory. The MS can be seeded to any value by writing the UMISR3 register.

10.5.1.2.15 User Multiple Input Signature Register 4 (UMISR4) Register

The UMISR4 register provides a means to evaluate array integrity. Its value represents the ECC bits of the whole 144-bit word (2 double words including ECC):

- Bits 23:16 are ECC bits for the odd double word
- Bits 7:0 are the ECC bits for the even double word
- Bits 27:26 and 11:10 of MISR are respectively the double and single ECC error detection for odd and even double word.

This register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reading this register returns indeterminate data; writing has no effect.

All flash array modules have a UMISR4 register. See *Section 10.4.4: Flash control and configuration registers map* for the flash memory registers address map.

The UMISR4 register is defined in *Figure 68* and *Table 76*.

Address is array-dependent—see *Table 55*.
Bank0, Array0: 0xC3F8_8058
Bank1, Array1: 0xC3FB_0058
Bank1, Array2: 0xC3FB_0058

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MS[159–144]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MS[143–128]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 68. User Multiple Input Signature Register 4 (UMISR4) Register

Table 76. UMISR4 field descriptions

Field	Description
MS [159–128]	Multiple input Signature 159–128 (Read/Write) These bits represent the MISR value obtained accumulating: – The 8 ECC bits for the even double word (on MS135–128); – The single ECC error detection for even double word (on MS138); – The double ECC error detection for even double word (on MS139); – The 8 ECC bits for the odd double word (on MS151–144); – The single ECC error detection for odd double word (on MS154); – The double ECC error detection for odd double word (on MS155). The MS can be seeded to any value by writing the UMISR4 register.

11 General-Purpose Static RAM (SRAM)

11.1 Overview

This device includes up to 94 KB of general-purpose SRAM. The first 32 KB block of the SRAM is powered by its own power supply pin only during standby operation.

11.2 Features

The SRAM controller includes these features:

- Supports read/write accesses mapped to the SRAM memory from any master
- 32 KB block powered by separate supply for standby operation
- Byte, halfword, word and doubleword addressable
- ECC performs single bit correction, double bit detection

11.3 Modes of operation

11.3.1 Normal (Functional) Mode

Normal Mode allows for reads and writes of the SRAM memory arrays.

11.3.2 Standby Mode

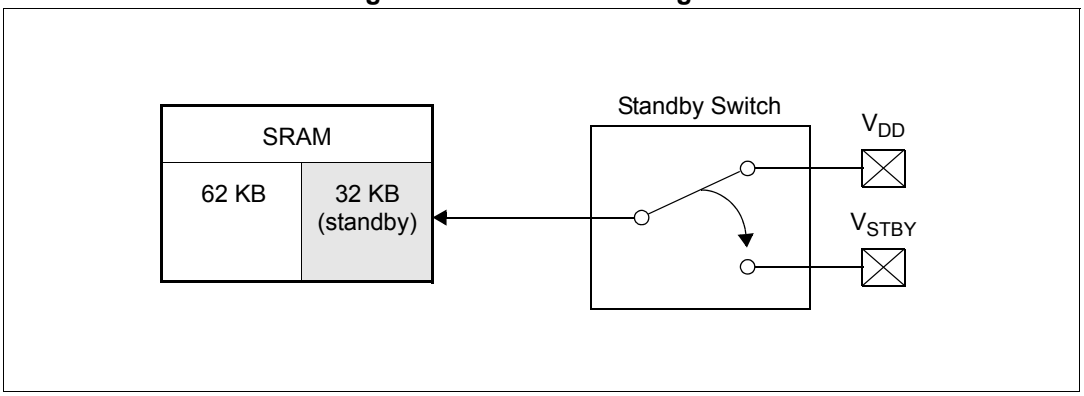
Standby Mode preserves contents of the standby portion of the memory when the 1.2 V (V_{DD}) power drops below the level of the standby power supply voltage. There are two possible supplies for standby: 1.0 V directly from the VSTBY pin and 2–5 volts (also on the VSTBY pin), which enables a standby regulator.

Updates to the standby portion of the SRAM are inhibited during system reset or during Standby Mode.

11.4 Block diagram

The SRAM block diagram is shown in *Figure 69*.

Figure 69. SRAM block diagram



11.5 External signal description

$V_{standby}$ RAM power supply pin.

11.6 Functional description

The crossbar bus is a two-stage pipelined bus which would require the SRAM controller to insert a wait state when a read follows a write. The SRAM controller implements a late-write-buffer to enable zero wait state write-read combinations.

ECC handling in this device is done on a 32-bit boundary. Because the e200z3 core in this device is a cacheless processor, the platform RAM is organized in on a 32-bit boundary versus the 64-bit organization used on MCUs based on the e200z6 core. This organization was driven by performance reasons: for the cacheless e200z3 processor, most RAM writes will be 32 bits or smaller in size. Each 32-bit word requires a 7-bit ECC code, so the RAM array is organized in two banks of 39 bits each.

The ECC code performs single bit corrections and indicates a multiple bit error on all double-bit read errors. Multiple bit errors will assert an error indication in the bus cycle, as well as setting field ECSM_ESR[RNCE].

During a write operation for 8-bit and 16-bit data, a read of 32-bit data will be checked for ECC, prior to merging in the write data. If a correction is required, it is corrected prior to merging in the write data. Then a new ECC code word is generated and written to the RAM. If a multiple bit error occurs during the read portion of the write operation, then the write is not performed.

It is essential for the ECC check bits to be initialized after Power On Reset. The write transfer must be 32 or 64 bits in size because a less than 32-bit write transfer will generate a read/modify/write operation which will check the ECC value upon the read.

11.6.1 Access timing

The crossbar bus is a two stage pipelined bus, which makes the timing of any access dependent on the access during the previous clock. *Table 77* shows the wait states for accesses. **Current** is the access being measured; **Previous** is the RAM access during the previous clock.

Table 77. Wait states during RAM access

Current	Previous	Wait states
Read	Idle	0
	Read	0
	32/64-bit Write	0
	8/16-bit Write	1
32/64-bit Write	Idle	0
	Read	0
	32/64-bit Write	0
	8/16-bit Write	1
8/16-bit Write	Idle	0
	Read	0
	32/64-bit Write	0
	8/16-bit Write	1

11.7 Module memory map

The SRAM occupies up to 94 KB of address space. See *Table 4* and *Table 6*.

Table 78 shows the SRAM memory map.

Table 78. SRAM memory map

Address	Use
L2SRAM_BASE	24 or 32 KB RAM, powered by V_{STBY}
L2SRAM_BASE + 0x8000	62 KB RAM

11.8 Register descriptions

The SRAM has no registers. Registers associated with the ECC are located in the ECSM. See *Section 17.3.1: ECC registers*.

12 External Bus Interface for Calibration Bus

12.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

12.1.1 Device-specific features

- Available only on the calibration system package
- Only calibration bus implemented
- 22-bit address bus (ADDR[10:31])
- 16-bit data bus (DATA[0:15])
- 3 chip-select: Cal_CS[0] and Cal_CS[2:3] (multiplexed with 2 most significant address signals)
- Two Write/Byte Enable (Cal_WE[0:1]_BE[0:1]) signals
- Possible division factors for CLKOUT: 1, 2, and 4

12.1.2 Unsupported features

- External master mode
- Burst access
- Transfer acknowledge
- Bus arbitration
- Following register fields are not supported:
  - EBI_BR0–3[LWRN], EBI_CAL_BR0–3[LWRN]
  - EBI_BR0–3[EOE], EBI_CAL_BR0–3[EOE]
  - EBI_BR0–3[SETA], EBI_CAL_BR0–3[SETA]
  - EBI_BR0–3[TBDIP], EBI_CAL_BR0–3[TBDIP]
  - EBI_BR0–3[GCSN], EBI_CAL_BR0–3[GCSN]
  - EBI_BR0–3[SBL], EBI_CAL_BR0–3[SBL]
- Following pins are not supported: CAL_CS1, TA and TSIZ[0:1]
- The two most significant bytes of the Base Registers can not be set since the external address space is restricted to 0x2000_0000 and 0x3FF_FFFF.

12.1.3 Device-specific register information

*Note: The convention is to **not** use the module name prefix (“EBI_”) for register names in software.*

Table 79 lists several register bit values for this device.

Table 79. Register bit values

Register	Bits	Value
EBI_BR0–EBI_BR3 EBI_CAL_BR0–3	BA[0:2]	0b001
EBI_OR0–EBI_OR3 EBI_CAL_OR0–3	AM[0:2]	0b111

12.2 Introduction

12.2.1 Overview

The External Bus Interface (EBI) handles the transfer of information between the internal buses and the memories or peripherals in the external address space. The EBI includes a memory controller that generates interface signals to support a variety of external memories. This includes Single Data Rate (SDR) burst mode flash, SRAM, and asynchronous memories. It supports up to four regions (via chip selects), each with its own programmed attributes (plus four calibration chip-select regions).

12.2.2 Features

Note: This list is a superset list of all possible features the EBI supports. Refer to Section 12.1: Information specific to this device for details on specifics for a particular device due to package limitations.

- 32-bit Address bus with transfer size indication (only 24–29 available on pins)
- 32-bit Data bus (16-bit Data Bus Mode also supported)
- Multiplexed Address on Data pins (single master)
- Memory controller with support for various memory types:
  - Synchronous burst SDR flash and SRAM
  - Asynchronous/legacy flash and SRAM
- Burst support (wrapped only)
- Bus monitor
- Port size configuration per chip select (16 or 32 bits)
- Configurable wait states
- Configurable internal or external transfer acknowledge ($\overline{TA}$) per chip select
- Support for Dynamic Calibration with up to 4 chip-selects
- 4 Write/Byte Enable ($\overline{WE}$[0:3]/$\overline{BE}$[0:3]) signals
- Slower-speed clock modes
- Stop and Module Disable Modes for power savings
- Optional automatic CLKOUT gating to save power and reduce EMI
- Misaligned access support (for chip-select accesses only)

12.2.3 Modes of operation

The mode of the EBI is determined by the MDIS, EXTM, and AD_MUX bits in the EBI_MCR. See *Section 12.4.1.1: EBI Module Configuration Register (EBI_MCR)* for details. Slower-

speed modes, Debug Mode, Stop Mode, and Factory Test Mode are modes that the MCU may enter, in parallel to the EBI being configured in one of its block-specific modes.

12.2.3.1 Single Master Mode

In Single Master Mode, the EBI responds to internal requests matching one of its regions, but ignores all externally-initiated bus requests. The MCU is the only master allowed to initiate transactions on the external bus in this mode; therefore, it acts as a parked master and does not have to arbitrate for the bus before starting each cycle. The $\overline{BR}$, $\overline{BG}$ and $\overline{BB}$ signals are not used by the EBI in this mode, and are available for use in an alternate function by another block of the MCU. Single Master Mode is entered when EXTM = 0 and MDIS = 0 in the EBI_MCR.

12.2.3.2 Module Disable Mode

The Module Disable Mode is used for MCU power management. The clock to the non-memory mapped logic in the EBI can be stopped while in Module Disable Mode. Logic on the MCU external to the EBI is needed to fully implement the Module Disable Mode (to shut off the clock). Internal master requests made to the external bus in Module Disable Mode are terminated with transfer error (internally, no external $\overline{TEA}$ assertion). Module Disable Mode is entered when MDIS = 1 in the EBI_MCR.

12.2.3.3 Stop Mode

The EBI supports the IPI Green-Line Interface Stop Mode mechanism used for MCU power management. When a request is made to enter Stop Mode (controlled in device logic outside EBI), the EBI block completes any pending bus transactions and acknowledges the stop request. After the acknowledgement, the system clock input may be shut off by the clock driver on the MCU. While the clocks are shut off, the EBI is not accessible. While in stop mode, accesses to the EBI from the internal master will terminate with transfer error (internally, no external $\overline{TEA}$ assertion).

12.2.3.4 Slower-Speed Modes

In slower-speed modes, the external CLKOUT frequency is divided (by 2, 3, etc.) compared with that of the internal system bus. The EBI behavior remains dictated by the mode of the EBI, except that it drives and samples signals at the CLKOUT frequency rather than the internal system frequency. This mode is selected by writing a clock control register in a block outside of the EBI.

12.2.3.5 16-bit Data Bus Mode

For MCUs that have only 16 data bus signals pinned out, or for systems where the use of a different multiplexed function (e.g. GPIO) is desired on 16 of the 32 data pins, the EBI supports a 16-bit Data Bus Mode. In this mode, only 16 data signals are used by the EBI. The user can select which 16 data signals are used (DATA[0:15] or DATA[16:31]) by writing the D16_31 bit in the EBI_MCR.

For EBI-mastered accesses, the operation in 16-bit Data Bus Mode (DBM = 1, PS=x) is similar to a chip-select access to a 16-bit port in 32-bit Data Bus Mode (DBM = 0, PS = 1), except for the case of a non-chip-select access of exactly 32-bit size.

EBI-mastered non-chip-select accesses of exactly 32-bit size are supported via a two (16-bit) beat burst for both reads and writes. See *Section 12.5.2.9: Non-chip-select burst in 16-bit Data Bus Mode*. Non-chip-select transfers of non-32-bit size are supported in standard

non-burst fashion.

16-bit Data Bus Mode is entered when DBM = 1 in the EBI_MCR. Some MCUs may have DBM = 1 by default out of reset.

12.2.3.6 Multiplexed Address on Data Bus Mode

This mode covers several cases aimed at reducing pin count on MCU and external components. In this mode, the DATA pins will drive (for internal master cycles) the address value on the first clock of the cycle (while $\overline{TS}$ is asserted).

The memory controller supports per-chip-select selection of multiplexing address/data through the EBI_BRx[AD_MUX] bit.

Address on Data bus multiplexing also supports the 16-bit data bus mode (EBI_MCR[DBM] = 1) and 16-bit memories (EBI_ORx[PS] = 1). The user can select which 16 data signals are used (DATA[0:15] or DATA[16:31]) by writing the D16_31 bit in the EBI_MCR. For either setting of D16_31, the 16 LSBs of external address (ADDR[16:31]) are driven onto the selected 16 DATA pins. If additional address lines are required to interface to the memory, then non-muxed address pins are sometimes (see note below) required to complete the address space (e.g. ADDR[8:15] are commonly present as non-muxed address pins).

Note: The EBI also drives the unused 16 DATA signals with the MSBs of the external address, zero-padded in front (e.g. when D16_31 bit is set for a device with 24 ADDR pins, the EBI drives (0b00000000,ADDR[8:15]) on DATA[0:15]. This allows the device to optionally use DATA[8:15] for the upper 8 external address lines instead of requiring separate non-muxed ADDR[8:15] pins. This is relevant primarily for devices that support both 32-bit and 16-bit A/D muxed operation, so therefore have DATA[0:31] pins present on the device, and in that case are not required to have separate ADDR pins.

For more details (e.g. timing diagrams), see *Section 12.5.2.12: Address data multiplexing*.

12.2.3.7 Debug Mode

When the MCU is in Debug Mode, the EBI behavior is unaffected and remains dictated by the mode of the EBI.

12.3 External signal description

12.3.1 Overview

Table 80 lists the external pins used by the EBI. Not all signals listed here are available external to the chip.

Table 80. Signal properties

Name	I/O type	Function	Pull(1)
ADDR[3:31]	I/O	Address bus	—
BDIP	Output	Burst Data in Progress	Up
CLKOUT(2)	Output	Clockout	—
CAL_CS[0:3]	Output	Calibration Chip Selects	Up
DATA[0:31]	I/O	Data bus(3)	—

Table 80. Signal properties (continued)

Name	I/O type	Function	Pull(1)
OE	Output	Output Enable	Up
RD_$\overline{WR}$	I/O	Read_Write	Up
TA	I/O	Transfer Acknowledge	Up
TEA	I/O	Transfer Error Acknowledge	Up
TS	I/O	Transfer Start	Up
TSIZ[0:1]	I/O	Transfer Size	—
$\overline{WE}$[0:3]/$\overline{BE}$[0:3]	Output	Write/Byte Enables	Up

1. This column shows which signals require a weak pullup or pulldown. The EBI block does not contain these pullup/pulldown devices within the block. They are assumed to be in another module of the MCU (e.g. pads module).
2. The CLKOUT signal is driven by the System Clock Block outside the EBI.
3. In Address/Data multiplexing modes, Data will also show the address during the address phase.

12.3.2 Detailed signal descriptions

Note: This section lists the superset of signals for the EBI. Refer to Section 12.1: Information specific to this device for device-specific package limitations and possible signal renaming.

12.3.2.1 ADDR[3:31] — Address lines 3–31

The ADDR[3:31] signals specify the physical address of the bus transaction.

The 29 address lines correspond to bits 3–31 of the EBI's 32-bit internal address bus.

12.3.2.2 $\overline{BDIP}$ — Burst Data in Progress

$\overline{BDIP}$ is asserted to indicate that the master is requesting another data beat following the current one.

This signal is driven by the EBI on all EBI-mastered external burst cycles, but is only sampled by burst mode memories that have a corresponding pin. See *Section 12.5.2.5: Burst transfer*.

12.3.2.3 CLKOUT — Clockout

CLKOUT is a general-purpose clock output signal to connect to the clock input of SDR external memories and in some cases to the input clock of another MCU in multi-master configurations.

12.3.2.4 $\overline{CS}$[0:3] — Chip Selects 0–3

$\overline{CS}$x is asserted by the master to indicate that this transaction is targeted for a particular memory bank on the Primary external bus.

$\overline{CS}$ is driven in the same clock as the assertion of $\overline{TS}$ and valid address, and is kept valid until the cycle is terminated. See *Section 12.5.1.4: Memory controller with support for various memory types* for details on chip-select operation.

12.3.2.5 CAL_CS[0:3] — Calibration Chip Selects 0–3

CAL_CSx is asserted by the master to indicate that this transaction is targeted for a particular memory bank on the Calibration external bus.

The calibration chip selects are driven only by the EBI. External master accesses on the Calibration bus are not supported. In all other aspects, the calibration chip-selects behave exactly as the primary chip-selects. See *Section 12.5.1.4: Memory controller with support for various memory types* for details on chip-select operation.

12.3.2.6 DATA[0:31] — Data Lines 0–31

The DATA[0:31] signals contain the data to be transferred for the current transaction.

DATA[0:31] is driven by the EBI when it owns the external bus and it initiates a write transaction to an external device.

DATA[0:31] is driven by an external device during a read transaction from the EBI.

For 8-bit and 16-bit transactions, the byte lanes not selected for the transfer do not supply valid data.

DATA[0:31] is driven by the EBI in the address phase with the ADDR value if the Address on Data multiplexing mode is enabled. See *Section 12.2.3.6: Multiplexed Address on Data Bus Mode* for details.

In 16-bit Data Bus Mode, (or for chip-select accesses to a 16-bit port), only DATA[0:15] or DATA[16:31] are used by the EBI, depending on the setting of the D16_31 bit in the EBI_MCR. See *Section 12.2.3.5: 16-bit Data Bus Mode*.

12.3.2.7 OE — Output Enable

OE is used to indicate when an external memory is permitted to drive back read data. External memories must have their data output buffers off when OE is negated. OE is only asserted for chip-select accesses.

For read cycles, OE is asserted one clock after TS assertion (by default with EBI_BR[EOE] = 0b00) and held until the termination of the transfer. For write cycles, OE is negated throughout the cycle.

12.3.2.8 RD_WR — Read / Write

RD_WR indicates whether the current transaction is a read access or a write access.

RD_WR is driven in the same clock as the assertion of TS and valid address, and is kept valid until the cycle is terminated.

12.3.2.9 TA — Transfer Acknowledge

TA is asserted to indicate that the slave has received the data (and completed the access) for a write cycle, or returned data for a read cycle. If the transaction is a burst read, TA is asserted for each one of the transaction beats. For write transactions, TA is only asserted once at access completion, even if more than one write data beat is transferred.

TA is driven by the EBI when the access is controlled by the chip selects (and SETA = 0). Otherwise, TA is driven by the slave device to which the current transaction was addressed.

See *Section 12.5.2.8: Termination signals protocol* for more details.

12.3.2.10 $\overline{\text{TEA}}$ — Transfer Error Acknowledge

$\overline{\text{TEA}}$ is asserted by either the EBI or an external device to indicate that an error condition has occurred during the bus cycle.

$\overline{\text{TEA}}$ is asserted by the EBI when the internal bus monitor detected a timeout error.

See *Section 12.5.2.8: Termination signals protocol* for more details.

12.3.2.11 $\overline{\text{TS}}$ — Transfer Start

$\overline{\text{TS}}$ is asserted by the current bus owner to indicate the start of a transaction on the external bus.

$\overline{\text{TS}}$ is only asserted for the first clock cycle of the transaction, and is negated in the successive clock cycles until the end of the transaction.

12.3.2.12 TSIZ[0:1] — Transfer Size 0–1

TSIZ[0:1] indicates the size of the requested data transfer, for aligned internal master transfers only. For the special case of misaligned chip-select transfers, TSIZ[0:1] values are not specified, and must not be used by external devices for those transfers.

The TSIZ[0:1] signals may be used with ADDR[30:31] to determine which byte lanes of the data bus are involved in the transfer. For non-burst transfers, the TSIZ[0:1] signals specify the number of bytes starting from the byte location addressed by ADDR[30:31]. In burst transfers, the value of TSIZ[0:1] is always 00.

Table 81. TSIZ[0:1] encoding

Burst cycle	TSIZ[0:1]	Transfer size
N	0b01	Byte
N	0b10	16-bit
N	0b11	Reserved
N	0b00	32-bit
Y	0b00	Burst

The SIZEEN bit has no effect on the EBI when it is mastering a transaction on the external bus. TSIZ[0:1] is still driven appropriately by the EBI. See *Section 12.4.1.1: EBI Module Configuration Register (EBI_MCR)*.

12.3.2.13 $\overline{\text{WE}}$[0:3] / $\overline{\text{BE}}$[0:3] — Write/Byte Enables 0–3

Write enables are used to enable program operations to a particular memory. These signals can also be used as byte enables for read and write operation by setting the WEBS bit in the appropriate Base Register. $\overline{\text{WE}}$[0:3]/BE[0:3] are only asserted for chip-select accesses.

For chip-select accesses to a 16-bit port, only $\overline{\text{WE}}$[0:1]/$\overline{\text{BE}}$[0:1] are used by the EBI, regardless of which half of the DATA bus is selected via the D16_31 bit in the EBI_MCR.

See *Section 12.5.1.11: Four write/byte enable (WE/BE) signals* for more details on $\overline{\text{WE}}$[0:3]/$\overline{\text{BE}}$[0:3] functionality.

12.3.3 Signal output buffer enable logic by mode

Table 82 describes how the EBI drives its output buffer enable (OBE) signals. These are internal signals from the EBI to device logic outside the EBI, that determine when the EBI strongly drives values on pins. When the OBE for an EBI signal is asserted (1), the EBI strongly drives the value on that pin. When the OBE is negated (0), the EBI does not drive the signal, and the value is determined by internal or external pullups/pulldowns, and/or device logic outside EBI block.

Table 82. Signal output buffer enable logic by mode(1)

Signal	OBE value by mode (1 = strongly driven, 0 = not driven by EBI)		
	Module disable mode(2) (EXTM = X, MDIS = 1)	Single master mode (EXTM = 0, MDIS = 0)	External master mode (EXTM = 1, MDIS = 0)
ADDR[3:31]	0	1	1 during int. master access
BDIP	0	1	1 during int. master access
$\overline{\text{CAL_CS}}$[0:3]	0	1	1 during int. master access
DATA[0:31]	0	Only 1 during write access or on Address phase when Addr/Data muxing is enabled.	
OE	0	1	1 during int. master access
RD_$\overline{\text{WR}}$	0	1	1 during int. master access
TA	0	Only 1 during chip-select (or cal-chip-select) SETA = 0 access	
TEA	0	Only 1 for 2 cycles when timeout occurs	
TS	0	1	1 during int. master access
TSIZ[0:1]	0	1	1 during int. master access
$\overline{\text{WE}}$[0:3]/$\overline{\text{BE}}$[0:3]	0	1	1 during int. master access

1. The values in this table only indicate when signals are strongly driven, not the logic value on the pin itself.
2. This assumes that the clock to the EBI is shut off when MDIS = 1. This is an optional device feature. If the clocks are left running to EBI even when MDIS = 1, then the EBI OBE behavior is as if in Single Master Mode (though EBI accesses are not supported in this scenario).

12.4 Memory map/register definition

Table 83 shows the EBI registers.

Note: The convention is to not use the module name prefix ("EBI_") for register names in software.

Table 83. EBI address map

Address	Use	Location
EBI_BASE	EBI Module Configuration Register (EBI_MCR)	*on page 280*
EBI_BASE+0x4	Reserved	—
EBI_BASE+0x8	EBI Transfer Error Status Register (EBI_TESR)	*on page 283*
EBI_BASE+0xC	EBI Bus Monitor Control Register (EBI_BMCR)	*on page 284*
EBI_BASE+0x10	EBI Base Register Bank 0 (EBI_BR0)	*on page 285*

Table 83. EBI address map (continued)

Address	Use	Location
EBI_BASE+0x14	EBI Option Register Bank 0 (EBI_OR0)	on page 288
EBI_BASE+0x18	EBI Base Register Bank 1 (EBI_BR1)	on page 285
EBI_BASE+0x1C	EBI Option Register Bank 1 (EBI_OR1)	on page 288
EBI_BASE+0x20	EBI Base Register Bank 2 (EBI_BR2)	on page 285
EBI_BASE+0x24	EBI Option Register Bank 2 (EBI_OR2)	on page 288
EBI_BASE+0x28	EBI Base Register Bank 3 (EBI_BR3)	on page 285
EBI_BASE+0x2C	EBI Option Register Bank 3 (EBI_OR3)	on page 288
EBI_BASE+0x30 – EBI_BASE+0x3C	Reserved	—
EBI_BASE+0x40	EBI Calibration Base Register Bank 0 (EBI_CAL_BR0)	on page 285
EBI_BASE+0x44	EBI Calibration Option Register Bank 0 (EBI_CAL_OR0)	on page 288
EBI_BASE+0x48	EBI Calibration Base Register Bank 1 (EBI_CAL_BR1)	on page 285
EBI_BASE+0x4C	EBI Calibration Option Register Bank 1 (EBI_CAL_OR1)	on page 288
EBI_BASE+0x50	EBI Calibration Base Register Bank 2 (EBI_CAL_BR2)	on page 285
EBI_BASE+0x54	EBI Calibration Option Register Bank 2 (EBI_CAL_OR2)	on page 288
EBI_BASE+0x58	EBI Calibration Base Register Bank 3 (EBI_CAL_BR3)	on page 285
EBI_BASE+0x5C	EBI Calibration Option Register Bank 3 (EBI_CAL_OR3)	on page 288

12.4.1 Register descriptions

Note: Other than the exceptions noted below, EBI registers must not be written while a transaction to the EBI (from internal master) is in progress (or within two CLKOUT cycles after a transaction has just completed, to allow internal state machines to go IDLE). In those cases, the behavior is undefined.

Exceptions that can be written while an EBI transaction is in progress:

– All bits in EBI_TESR

– SIZE, SIZEEN fields in EBI_MCR

See Section 12.6.1: Booting from external memory for related application information.

12.4.1.1 EBI Module Configuration Register (EBI_MCR)

The EBI Module Configuration Register contains bits which configure various attributes associated with EBI operation.

EBI_BASE+0x8

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ACGE	EXTM	EARB	EARP		0	0	0	0	MDIS	0	0	0	D16_31	AD_MUX	DBM
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1

= Unimplemented or Reserved

Figure 70. EBI Module Configuration Register (EBI_MCR)

Table 84. EBI_MCR field descriptions

Field	Description
1–4	Reserved
5 SIZEEN	SIZE enable The SIZEEN bit enables the control of transfer size by the SIZE field (as opposed to external TSIZ pins) for external master transactions to internal address space. 1 Transfer size controlled by SIZE field 0 Transfer size controlled by TSIZ[0:1] pins
6–7	SIZE - Transfer size The SIZE field determines the transfer size of external master transactions to internal address space when SIZEEN = 1. This field is ignored when SIZEEN = 0. This field must be written before doing an external master transfer of a different size than the current setting. The DBM bit affects how the SIZE field should be set for the special case of 64-bit external master transfers. *Table 85* shows how to set SIZE appropriately for all the cases.
8–15	Reserved
16 ACGE	Automatic CLKOUT Gating Enable The ACGE bit enables the EBI feature of turning off CLKOUT (holding it high) during idle periods in-between external bus accesses. 1 Automatic CLKOUT Gating is enabled 0 Automatic CLKOUT Gating is disabled
17 EXTM	External Master Mode The EXTM bit enables the External Master Mode of operation when MDIS = 0. When MDIS = 1, the EXTM bit is a ‘don’t care’, and is treated as 0. In External Master Mode, an external master on the external bus can access any internal memory-mapped space while the internal e200z core is fully operational. When EXTM = 0, only internal masters can access the internal memory space. This bit also determines the functionality of the $\overline{BR}$, $\overline{BG}$ and $\overline{BB}$ signals. 1 External Master Mode is active 0 External Master Mode is inactive (Single Master Mode)

Table 84. EBI_MCR field descriptions (continued)

Field	Description
18 EARB	External Arbitration 1 When EXTM = 0, the EARB bit is a ‘don’t care’, and is treated as 0. External arbitration is used 0 Internal arbitration is used
19–20 EARP	External Arbitration Request Priority This field defines the priority of an external master’s arbitration request (0–2), with 2 being the highest priority level (EARP = 3 is reserved). This field is valid only when EARB = 0 (internal arbitration). The internal masters of the MCU have a fixed priority of 1. By default, internal and external masters have equal priority.0b00 MCU has priority 0b01 Equal priority, round robin used 0b10 External master has priority 0b11 Reserved **Note:**
21–24	Reserved
25 MDIS	Module Disable Mode The MDIS bit controls an internal EBI “enable clk” signal which can be used (if MCU logic supports) to control the clocks to the EBI. The MDIS bit allows the clock to be stopped to the non-memory mapped logic in the EBI, effectively putting the EBI in a software controlled power-saving state. See *Section 12.2.3.2: Module Disable Mode* for more information. No external bus accesses can be performed when the EBI is in Module Disable Mode (MDIS = 1). 1 Module Disable Mode is active (negate “enable clk” signal) 0 Module Disable Mode is inactive (assert “enable clk” signal)
26–28	Reserved
29 D16_31	Data Bus 16_31 Select The D16_31 bit controls whether the EBI uses the DATA[0:15] or DATA[16:31] signals, when in 16-bit Data Bus Mode (DBM = 1) or for chip-select accesses to a 16-bit port (PS = 1). 1 DATA[16:31] signals are used for 16-bit port accesses 0 DATA[0:15] signals are used for 16-bit port accesses **Note:** There is one usage case where D16_31 = 0 **cannot** be used. For systems that are using both a 32-bit non-chip-select device AND a 16-bit device (chip-select or non-chip-select), the user **must** set D16_31 = 1. Otherwise (if D16_31 = 0 is used), when an access to the 16-bit device is performed, the 32-bit non-chip-select device may decode the wrong upper address bits and may respond to that access, causing contention on the shared address/data lines going to both devices. This is due to the 32-bit device expecting address 8:15 on DATA[8:15] pins, whereas D16_31 = 0 has address 24:31 on those pins. This contention case is avoided when D16_31 = 1, because the shared address/data pins have the same functions for both the 16-bit and 32-bit memory accesses (address 16:31 on DATA[16:31] pins). This contention case is also avoided when only chip-select devices are used, since the devices ignore accesses when their corresponding chip-select is not asserted.
30 AD_MUX	Address on Data Bus Multiplexing Mode The AD_MUX bit controls whether non-chip-select accesses have the address driven on the data bus in the address phase of a cycle. 1 Address on Data Multiplexing Mode is used for non-CS accesses. 0 Only Data on data pins for non-CS accesses.
31 DBM	Data Bus Mode The DBM bit controls whether the EBI is in 32-bit or 16-bit Data Bus Mode. 1 16-bit Data Bus Mode is used 0 32-bit Data Bus Mode is used

Table 85. SIZE encoding(1)

DBM	Size of external master transfer(2)	SIZE
X(3)	Byte	0b01
X	16-bit	0b10
X	32-bit(4)	0b00
0	64-bit	0b00
1	64-bit	0b10
X	Reserved	0b11

1. This table is not affected by width of internal AMBA bus (32 or 64 bits), only by the size of the transfer.
2. This refers to size of external master transfer on external master MCU as determined by the source code. For example, if the external master code does a 64-bit load (or store) to an EBI with a 32-bit bus (DBM = 0), this is broken into two 32-bit transfers, so SIZE field must be set to 32 bits (0b00). For a 16-bit bus, the 64-bit access is broken into four 16-bit transfers, so SIZE must be set to 16 bits (0b01).
3. 'Don't care' value for this case.
4. 32-bit transfers use SIZE 0b00 regardless of DBM because there is special logic in the EBI to handle 32-bit coherent non-chip-select accesses even when DBM=1. Similar logic is not present for 64-bit transfers, so these are broken into separate bus-size accesses.

12.4.1.2 EBI Transfer Error Status Register (EBI_TESR)

The EBI Transfer Error Status Register contains a bit for each type of transfer error on the external bus. A bit set to logic 1 indicates what type of transfer error occurred since the last time the bits were cleared. Each bit can be cleared by reset or by writing a 1 to it. Writing a 0 has no effect. For some MCUs with this EBI, this register may not be writable in Module Disable Mode due to the use of power saving clock modes.

EBI_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	0	0	TEAF	BMTF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 71. EBI Transfer Error Status Register (EBI_TESR)

Table 86. EBI_TESR field descriptions

Field	Description
0–29	Reserved
30 TEAF	Transfer Error Acknowledge Flag This bit is set if the cycle was terminated by an externally generated $\overline{\text{TEA}}$ signal. 1 External $\overline{\text{TEA}}$ occurred 0 No error
31 BMTF	Bus Monitor Timeout Flag This bit is set if the cycle was terminated by a bus monitor timeout. 1 Bus monitor timeout occurred 0 No error

12.4.1.3 EBI Bus Monitor Control Register (EBI_BMCR)

The EBI Bus Monitor Control Register controls the timeout period of the bus monitor and whether it is enabled or disabled.

EBI_BASE+0xC

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

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

= Unimplemented or Reserved

Figure 72. EBI Bus Monitor Control Register (EBI_BMCR)

Table 87. EBI_BMCR field descriptions

Field	Description
0–15	Reserved
16–23 BMT	Bus Monitor Timing This field defines the timeout period, in 8 external bus clock resolution, for the Bus Monitor. See *Section 12.5.1.6: Bus monitor* for more details on bus monitor operation. Timeout Period = (2 + (8 * BMT)) / external bus clock frequency.
24 BME	Bus Monitor Enable This bit controls whether the bus monitor is enabled for internal to external bus cycles. The BME bit is ignored (treated as 0) for chip-select accesses with internal $\overline{\text{TA}}$ (SETA = 0). 1 Enable bus monitor (for external $\overline{\text{TA}}$ accesses only) 0 Disable bus monitor

12.4.1.4 EBI Base Registers (EBI_BR0–EBI_BR3, EBI_CAL_BR0–3)

The EBI Base Registers are used to define the base address and other attributes for the corresponding chip select.

EBI_BASE+0x10, EBI_BASE+0x18, EBI_BASE+0x20, EBI_BASE+0x28,
EBI_BASE+0x40, EBI_BASE+0x48, EBI_BASE+0x50, EBI_BASE+0x58

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R		0	0													
W	BA			LWRN	PS	EOE		SBL	AD_MUX	BL	WEBS	TBDIP	GCSN	SETA	BI	V
Reset	0	0	0	0	0	00		0	0	0	0	0	0	0	1	0

= Unimplemented or Reserved

1. Some upper bits of the BA field may be tied to a fixed value, in which case the reset value is this fixed value and not zero. Refer to *Section 12.1: Information specific to this device* to see which bits this applies to, if any.

Figure 73. EBI Base Registers (EBI_BR0–EBI_BR3, EBI_CAL_BR0–3)

Table 88. EBI_BR0–EBI_BR3, EBI_CAL_BR0–3 field descriptions

Field	Description
0–16 BA	Base Address These bits are compared to the corresponding unmasked address signals among ADDR[0:16] of the internal address bus to determine if a memory bank controlled by the memory controller is being accessed by an internal bus master. **Note:** An MCU may have some of the upper bits of the BA field tied to a fixed value internally in order to restrict the address range of the EBI for that MCU. Tied-off bits can be read but not written. These bits are ignored by the EBI during the chip-select address comparison. However, the internal bridge of the MCU most likely requires that the chip-select banks be located in memory regions corresponding to the fixed values chosen.
17–18	Reserved
19 LWRN(1)	Late RD_$\overline{WR}$ Negation The LWRN bit determines the timing of RD_$\overline{WR}$ signal negation for a write transfer. 1 Negate RD_$\overline{WR}$ the same cycle as $\overline{CS}$ negation. See *Figure 86*. 0 Negate RD_$\overline{WR}$ one cycle earlier than $\overline{CS}$ negation. See *Figure 83*.
20 PS	Port Size The PS bit determines the data bus width of transactions to this chip-select bank. **Note:** In the case where the DBM bit in EBI_MCR is set for 16-bit Data Bus Mode, the PS bit value is ignored and is always treated as a ‘1’ (16-bit port). 1 16-bit port 0 32-bit port

Table 88. EBI_BR0–EBI_BR3, EBI_CAL_BR0–3 field descriptions (continued)

Field	Description
21–22 EOE	Early $\overline{OE}$ The EOE field determines the timing of $\overline{OE}$ signal assertion for a read transfer. When EBI_BR[ADMUX] = 1, the EOE value is ignored and treated as 0b00 (in order to avoid contention on shared address/data bus for muxed transfers). See *Table 89* for EOE values.
23 SBL	Short Burst Length The SBL bit provides support for a 2-word external burst (see *Figure 102* in *Section Example 12: Small access example #5: 32-byte read to 32-bit port with SBL = 1*). 1 The number of beats in a burst is automatically determined by the EBI to be 2 or 4 according to the Port Size (PS bit), regardless of BL bit value. 0 The number of beats in a burst is determined by the BL bit. See *Table 90* for SBL and BL encodings.
24 AD_MUX	Address on Data Bus Multiplexing The AD_MUX bit controls whether accesses for this chip select have the address driven on the data bus in the address phase of a cycle 1 Address on Data Multiplexing Mode is enabled for this chip select. 0 Address on Data Multiplexing Mode is disabled for this chip select.
25 BL	Burst Length The BL bit (along with SBL bit) determines the amount of data transferred in a burst for this chip select, measured in 32-bit words. When SBL = 0, the number of beats in a burst is automatically determined by the EBI to be 4, 8, or 16 according to the Port Size (PS bit) so that the burst fetches the number of words chosen by BL. For internal AMBA data bus width of 32-bits, the BL bit is ignored (treated as 1). When SBL = 1, the BL bit value is a ‘don’t care’. See *Table 90* for SBL and BL encodings.
26 WEBS	Write Enable / Byte Select This bit controls the functionality of the $\overline{WE}$[0:3]/$\overline{BE}$[0:3] signals. 1 The $\overline{WE}$[0:3]/$\overline{BE}$[0:3] signals function as $\overline{BE}$[0:3] 0 The $\overline{WE}$[0:3]/$\overline{BE}$[0:3] signals function as $\overline{WE}$[0:3]
27 TBDIP(2)	Toggle Burst Data in Progress This bit determines how long the $\overline{BDIP}$ signal is asserted for each data beat in a burst cycle. See *Section 12.5.2.5.1: TBDIP effect on burst transfer* for details. 1 Only assert $\overline{BDIP}$ (BSCY+1) external cycles before expecting subsequent burst data beats 0 Assert $\overline{BDIP}$ throughout the burst cycle, regardless of wait state configuration
28 GCSN(3)	Guarantee $\overline{CS}$ Negation The GCSN bit allows support for guaranteeing that the EBI will negate $\overline{CS}$ between all back-to-back transfer cases (even those that are part of a set of Small Accesses). See *Figure 92*. 1 Negate $\overline{CS}$ between all external transfers. This adds an extra dead cycle for some back-to-back cases. 0 Default operation ($\overline{CS}$ may or may not negate between transfers, depending on the particular back-to-back case).

Table 88. EBI_BR0–EBI_BR3, EBI_CAL_BR0–3 field descriptions (continued)

Field	Description
29 SETA(4)	Select External Transfer Acknowledge The SETA bit controls whether accesses for this chip select will terminate (end transfer without error) based on externally asserted $\overline{TA}$ or internally asserted $\overline{TA}$. SETA should only be set when the BI bit is 1 as well, since burst accesses with SETA = 1 are not supported. Setting SETA = 1 causes the BI bit to be ignored (treated as 1, burst inhibited). 1 Transfer Acknowledge ($\overline{TA}$) is an input to the EBI, data phase will be terminated by an external device 0 Transfer Acknowledge ($\overline{TA}$) is an output from the EBI, data phase will be terminated by the EBI
30 BI(5)	Burst Inhibit This bit determines whether or not burst read accesses are allowed for this chip-select bank. The BI bit is ignored (treated as 1) for chip-select accesses with external $\overline{TA}$ (SETA = 1). 1 Disable burst accesses for this bank. This is the default value out of reset (or when SETA = 1). 0 Enable burst accesses for this bank
31 V	Valid bit The user writes this bit to indicate that the contents of this Base Register and Option Register pair are valid. The appropriate $\overline{CS}$ signal does not assert unless the corresponding V-bit is set. 1 This bank is valid 0 This bank is not valid

1. LWRN is not supported on SPC563Mxx devices.
2. TBDIP is writable but is not supported on SPC563Mxx devices since burst is not supported.
3. GCSN is not supported on SPC563Mxx devices.
4. SETA is writable but is not supported on SPC563Mxx devices since burst is not supported.
5. BI is writable but is not supported on SPC563Mxx devices since burst is not supported. It must always be set to 1.

Table 89. EOE values

Value	Timing of $\overline{OE}$ assertion
0b00(1)	Assert $\overline{OE}$ 1 CLKOUT cycle after $\overline{TS}$ assertion (see *Figure 78*)
0b01	Assert $\overline{OE}$ the same CLKOUT cycle as $\overline{TS}$ assertion (see *Figure 81*)
0b10	Assert $\overline{OE}$ one internal clock cycle after $\overline{TS}$ assertion (see *Figure 81*)(2)
0b11	Reserved

1. The 0b00 timing case also applies when EBI_BR[ADMUX] = 1, regardless of EOE value in Base Register.
2. In divided bus speed modes (where the EBI runs at a slower frequency than the internal system bus), the EBI uses the internal higher-frequency clock to assert $\overline{OE}$ partway through the CLKOUT cycle where $\overline{TS}$ is asserted. In 1:1 bus speed mode (where CLKOUT and internal system bus run at same frequency), the behavior for EOE value 0b10 is identical to that of value 0b00.

Table 90. SBL, BL values

Value		Burst length(1)	PS	Number of beats in burst(2)
SBL	BL			
0	0(3)	8-word(4)	0 (32-bit)	8
			1 (16-bit)	16

Table 90. SBL, BL values (continued)

Value		Burst length(1)	PS	Number of beats in burst(2)
SBL	BL			
0	1	4-word	0 (32-bit)	4
			1 (16-bit)	8
1	X	2-word(4)	0 (32-bit)	2
			1 (16-bit)	4

1. Total amount of data fetched in a burst transfer, measured in 32-bit words.
2. Number of external data beats used in external burst transfer. The size of each beat is determined by PS value.
3. An 8-word burst length is only supported for devices using 64-bit AMBA data bus width to EBI.
4. A word always refers to 32-bits of data, regardless of PS.

12.4.1.5 EBI Option Registers (EBI_OR0–EBI_OR3, EBI_CAL_OR0–3)

The EBI Option Registers are used to define the address mask and other attributes for the corresponding chip select.

EBI_BASE+0x14, EBI_BASE+0x1C, EBI_BASE+0x24, EBI_BASE+0x2C,
EBI_BASE+0x44, EBI_BASE+0x4C, EBI_BASE+0x54, EBI_BASE+0x5C

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

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

= Unimplemented or Reserved

1. Some upper bits of the AM field may be tied to a fixed value, in which case the reset value is this fixed value and not zero. Refer to *Section 12.1: Information specific to this device* to see which bits this applies to, if any.

Figure 74. EBI Option Registers (EBI_OR0–EBI_OR3, EBI_CAL_OR0–3)

Table 91. EBI_OR0–EBI_OR3, EBI_CAL_OR0–3 field descriptions

Field	Description
0–16 AM	Address Mask This field allows masking of any corresponding bits in the associated Base Register. Masking the address independently allows external devices of different size address ranges to be used. Any clear bit masks the corresponding address bit. Any set bit causes the corresponding address bit to be used in comparison with the address pins. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. This field can be read or written at any time. **Note:** An MCU may have some of the upper bits of the AM field tied to a fixed value internally in order to restrict the address range of the EBI for that MCU. See the corresponding Note for the Base Register BA field for more details. Tied-off bits can be read but not written.
17–23	Reserved
24–27 SCY	Cycle length in clocks This field represents the number of wait states (external cycles) inserted after the address phase in the single transfer case, or in the first beat of a burst, when the memory controller handles the external memory access. Values range from 0 to 15. This is the main parameter for determining the length of the cycle. These bits are ignored when SETA = 1. The total cycle length for the first beat (including the $\overline{TS}$ cycle) = (2 + SCY) external clock cycles. See *Section 12.6.3.1: Example wait state calculation* for related application information.
28	Reserved
29–30 BSCY	Burst beats length in clocks This field determines the number of wait states (external cycles) inserted in all burst beats except the first, when the memory controller starts handling the external memory access and thus is using SCY[0:3] to determine the length of the first beat. These bits are ignored when SETA = 1. The total memory access length for each beat is (1 + BSCY) external clock cycles. The total cycle length (including the $\overline{TS}$ cycle) = (2 + SCY) + (#beats(1) – 1) * (BSCY + 1). 00: 0-clock cycle wait states (1 clock per data beat) 01: 1-clock cycle wait states (2 clocks per data beat) 10: 2-clock cycle wait states (3 clocks per data beat) 11: 3-clock cycle wait states (4 clocks per data beat)
31	Reserved

1. #beats is the number of beats (4,8,16) determined by BL and PS bits in Base Register.

12.5 Functional description

12.5.1 External bus interface features

12.5.1.1 32-bit address bus with transfer size indication (up to 29 available on pins)

The transfer size for an external transaction is indicated by the TSIZ[0:1] signals during the clock where address is valid. Valid transaction sizes are 8, 16 and 32 bits. Only 24-29 address lines are pinned out externally (device-specific option), but a full 32-bit decode is done internally to determine the target of the transaction and whether a chip select should be asserted.

12.5.1.2 32-bit data bus (16-bit data bus mode also supported)

The entire 32-bit data bus is available for external memory accesses. There is also a 16-bit Data Bus Mode available via the DBM bit in EBI_MCR. See *Section 12.2.3.5: 16-bit Data Bus Mode*.

12.5.1.3 Multiplexed address on data pins (single master)

When this mode is enabled, the address shows up on the data pins during the address phase of the cycle. This mode can be enabled separately for non-chip-select accesses and per chip-select access. See *Section 12.2.3.6: Multiplexed Address on Data Bus Mode*.

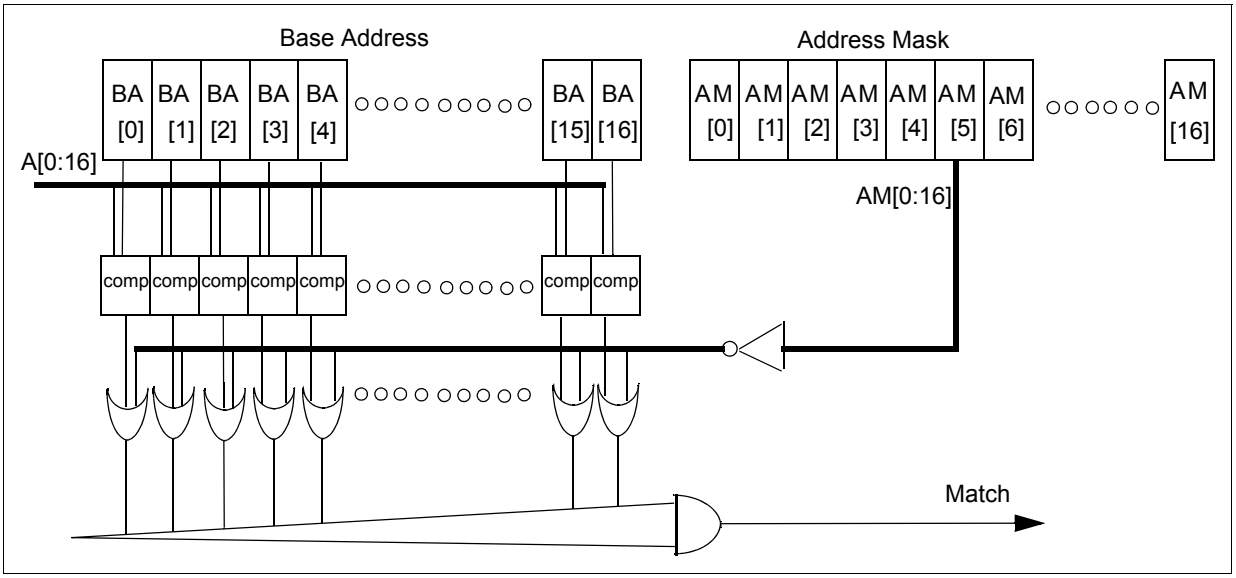
12.5.1.4 Memory controller with support for various memory types

The EBI contains a memory controller that supports a variety of memory types, including synchronous burst mode flash and SRAM, and asynchronous/legacy flash and SRAM with a compatible interface.

Each $\overline{CS}$ bank is configured via its own pair of Base and Option Registers. Each time an internal to external bus cycle access is requested, the internal address is compared with the base address of each valid Base Register (with 17 bits having mask). See *Figure 75*. If a match is found, the attributes defined for this bank in its EBI_BR and EBI_OR are used to control the memory access. If a match is found in more than one bank, the lowest bank matched handles the memory access (e.g., bank 0 is selected over bank 1).

A match on a valid calibration chip-select register overrides a match on any non-calibration chip-select register, with CAL_CS0 having the highest priority. Thus the full priority of the chip-selects is: CAL_CS0,...,CAL_CS3,CS0,...,CS3.

Figure 75. Bank base address & match structure



When a match is found on one of the chip-select banks, all its attributes (from the appropriate Base and Option Registers) are selected for the functional operation of the external memory access, such as:

- Number of wait states for a single memory access, and for any beat in a burst access
- Burst enable
- Port size for the external accessed device

See *Section 12.4.1.4: EBI Base Registers (EBI_BR0–EBI_BR3, EBI_CAL_BR0–3)* and *Section 12.4.1.5: EBI Option Registers (EBI_OR0–EBI_OR3, EBI_CAL_OR0–3)* for a full description of all chip-select attributes.

When no match is found on any of the chip-select banks, the default transfer attributes shown in *Table 92* are used.

Table 92. Default attributes for non-chip-select transfers

CS attribute	Default value	Comment
PS	0	32-bit port size
BL	0	burst length is ‘don’t care’ since burst is disabled
WEBS	0	write enables
TBDIP	0	‘don’t care’ since burst is disabled
BI	1	burst inhibited
SCY	0	‘don’t care’ since external $\overline{TA}$ is used
BSCY	0	‘don’t care’ since external $\overline{TA}$ is used
AD_MUX	0	Address on Data multiplexing
SETA	1	Select external TA to terminate access

12.5.1.5 Burst support (wrapped only)

The EBI supports burst read accesses of external burstable memory. To enable bursts to a particular memory region, clear the BI (Burst Inhibit) bit in the appropriate Base Register. External burst lengths of two, four and eight words are supported. Burst length is configured for each chip select by using the BL and SBL bits in the appropriate Base Register. See *Section 12.5.2.5: Burst transfer* for more details on burst operation.

In 16-bit data bus mode (DBM = 1 in EBI_MCR), a special 2-beat burst case is supported for reads and writes for 32-bit non-chip-select accesses only. This is to allow 32-bit coherent accesses to another MCU. See *Section 12.5.2.9: Non-chip-select burst in 16-bit Data Bus Mode*.

Bursting of accesses that are not controlled by the chip selects is not supported for any other case besides the special case of 32-bit accesses in 16-bit data bus mode.

Burst writes are not supported for any other case besides the special case of 32-bit non-chip-select writes in 16-bit data bus mode. Internal requests to write >32 bits (such as a cache line) externally are broken up into separate 32-bit or 16-bit external transactions according to the port size. See *Section 12.5.2.6: Small accesses (small port size and short burst length)* for more detail on these cases.

12.5.1.6 Bus monitor

When enabled (via the BME bit in the EBI_BMCR), the bus monitor detects when no $\overline{TA}$ assertion is received within a maximum timeout period for external $\overline{TA}$ accesses. The timeout for the bus monitor is specified by the BMT field in the EBI_BMCR. Each time a timeout error occurs, the BMTF bit is set in the EBI_TESR. The timeout period is measured in external bus (CLKOUT) cycles. Thus the effective real-time period is multiplied (by 2, 3, etc.) when a slower-speed mode is used, even though the BMT field itself is unchanged.

12.5.1.7 Port size configuration per chip select (16 or 32 bits)

The EBI supports memories with data widths of 16 or 32 bits. The port size for a particular chip select is configured by writing the PS bit in the corresponding Base Register.

12.5.1.8 Configurable wait states

From 0 to 15 wait states can be programmed for any cycle that the memory controller generates, via the SCY bits in the appropriate Option Register. From 0 to 3 wait states between burst beats can be programmed using the BSCY bits in the appropriate Option Register.

12.5.1.9 Configurable internal or external $\overline{TA}$ per chip select

Each chip select can be configured (via the SETA bit) to have $\overline{TA}$ driven internally (by the EBI), or externally (by an external device). See *Section 12.4.1.4: EBI Base Registers (EBI_BR0–EBI_BR3, EBI_CAL_BR0–3)*" for more details on SETA bit usage.

12.5.1.10 Support for dynamic calibration with up to four chip-selects

The EBI contains 4 calibration chip select signals, controlling 4 independent memory banks on an optional 2nd external bus for calibration. See *Section 12.5.2.10: Calibration bus operation*" for more details on using the calibration bus.

12.5.1.11 Four write/byte enable ($\overline{WE}$/$\overline{BE}$) signals

The functionality of the $\overline{WE}$[0:3]/$\overline{BE}$[0:3] signals depends on the value of the WEBS bit in the corresponding Base Register. Setting WEBS to 1 configures these pins as $\overline{BE}$[0:3], while resetting it to 0 configures them as $\overline{WE}$[0:3]. $\overline{WE}$[0:3] are asserted only during write accesses, while $\overline{BE}$[0:3] is asserted for both read and write accesses. The timing of the $\overline{WE}$[0:3]/$\overline{BE}$[0:3] signals remains the same in either case.

The upper Write/Byte Enable ($\overline{WE0}$/$\overline{BE0}$) indicates that the upper eight bits of the data bus (DATA[0:7]) contain valid data during a write/read cycle. The upper middle Write/Byte Enable ($\overline{WE1}$/$\overline{BE1}$) indicates that the upper middle eight bits of the data bus (DATA[8:15]) contain valid data during a write/read cycle. The lower middle Write/Byte Enable ($\overline{WE2}$/$\overline{BE2}$) indicates that the lower middle eight bits of the data bus (DATA[16:23]) contain valid data during a write/read cycle. The lower Write/Byte Enable ($\overline{WE3}$/$\overline{BE3}$) indicates that the lower eight bits of the data bus (DATA[24:31]) contain valid data during a write/read cycle.

Note: The exception to the preceding $\overline{WE}$/$\overline{BE}$ description is that for 16-bit port transfers (DBM = 1 or PS = 1), only the $\overline{WE}$[0:1]/$\overline{BE}$[0:1] signals are used, regardless of whether DATA[0:15] or DATA[16:31] are selected (via the D16_31 bit in the EBI_MCR). This means for the case where DATA[16:31] are selected, that $\overline{WE0}$ indicates that DATA[16:23] contains valid data, and $\overline{WE1}$ indicates that DATA[24:31] contains valid data.

The Write/Byte Enable lines affected in a transaction for a 32-bit port (PS = 0) and a 16-bit port (PS = 1) are shown in *Table 93*. Only Big Endian byte ordering is supported by the EBI.

Table 93. Write/byte enable signals function(1)

Transfer size	TSIZ[0:1]	Address		32-bit port size				16-bit port size(2)			
		A30	A31	$\overline{WE0}$/ BE0	$\overline{WE1}$/ BE1	$\overline{WE2}$/ BE2	$\overline{WE3}$/ BE3	$\overline{WE0}$/ BE0	$\overline{WE1}$/ BE1	$\overline{WE2}$/ BE2	$\overline{WE3}$/ BE3
Byte	01	0	0	X				X			
	01	0	1		X				X		
	01	1	0			X		X			
	01	1	1				X		X		
16-bit	10	0	0	X	X			X	X		
	10	1	0			X	X	X	X		
32-bit	00	0	0	X	X	X	X	X(3)	X(3)		
Burst	00	0	0	X	X	X	X	X	X		

1. This table applies to aligned internal master transfers only. In the case of a misaligned internal master transfer that is split into multiple aligned external transfers, not all of the write enables X'd in the table will necessarily assert. See *Section 12.5.2.11: Misaligned access support*.
2. Also applies when DBM = 1 for 16-bit data bus mode.
3. This case consists of two 16-bit external transactions, but for both transactions the $\overline{WE}$[0:1]/$\overline{BE}$[0:1] signals are the only WE/BE signals affected.

12.5.1.12 Slower-speed clock modes

For memories that cannot run with a full-speed external bus, the EBI supports slower-speed clock modes. Refer to *Section 12.2.3.4: Slower-Speed Modes* for more details on this feature. The timing diagrams for slower-speed modes are identical to those for full-speed mode, except that the frequency of CLKOUT is reduced.

12.5.1.13 Stop and module disable modes for power savings

See *Section 12.2.3: Modes of operation* for a description of the power saving modes.

12.5.1.14 Optional automatic CLKOUT gating

The EBI has the ability to hold the external CLKOUT pin high when the EBI's internal master state machine is idle and no requests are pending. The EBI outputs a signal to the pads logic in the MCU to disable CLKOUT. This feature is disabled out of reset, and can be enabled or disabled by the ACGE bit in the EBI_MCR.

Note: This feature must be disabled for multi-master systems. In those cases, one master is getting its clock source from the other master and needs it to stay valid continuously.

12.5.1.15 Misaligned access support

The EBI has limited misaligned access support. Misaligned non-burst chip-select transfers from internal masters are supported. The EBI aligns the accesses when it sends them out to the external bus (splitting them into multiple aligned accesses if necessary), so that external devices are not required to support misaligned accesses. Burst accesses (internal master) must match the internal bus size (64-bit aligned or 32 bit aligned). See *Section 12.5.2.11:*

Misaligned access support or *Section 12.5.2.11.2: Misaligned access support (32 bit AMBA)* for more details.

12.5.2 External bus operations

The following sections provide a functional description of the external bus, the bus cycles provided for data transfer operations, and error conditions.

12.5.2.1 External clocking

The CLKOUT signal sets the frequency of operation for the bus interface directly. Internally, the MCU uses a phase-locked loop (PLL) circuit to generate a master clock for all of the MCU circuitry (including the EBI) which is phase-locked to the CLKOUT signal. In general, all signals for the EBI are specified with respect to the rising-edge of the CLKOUT signal, and they are guaranteed to be sampled as inputs or changed as outputs with respect to that edge.

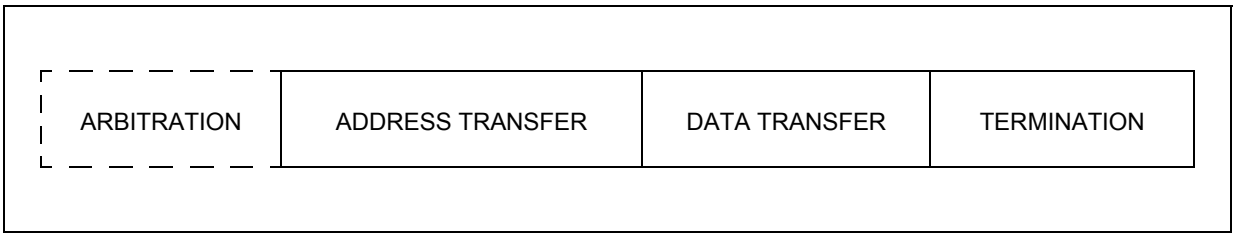
12.5.2.2 Reset

Upon detection of internal reset assertion, the EBI immediately ends all transactions (abruptly, not through normal termination protocol), and ignores any transaction requests that take place while reset is asserted.

12.5.2.3 Basic transfer protocol

The basic transfer protocol defines the sequence of actions that must occur on the external bus to perform a complete bus transaction. A simplified scheme of the basic transfer protocol is shown in *Figure 76*.

Figure 76. Basic transfer protocol



The arbitration phase is where bus ownership is requested and granted. This phase is not needed in Single Master Mode because the EBI is the permanent bus owner in this mode.

The address transfer phase specifies the address for the transaction and the transfer attributes that describe the transaction. The signals related to the address transfer phase are TS, ADDR (or DATA if Address/Data multiplexing is used), CS[0:3], RD_WR, TSIZ[0:1], and BDIP. The address and its related signals (with the exception of TS, BDIP) are driven on the bus with the assertion of the TS signal, and kept valid until the bus master receives TA asserted (the EBI holds them one cycle beyond TA for writes and external TA accesses). Note that for writes with internal TA, RD_WR is not held one cycle past TA.

The data transfer phase performs the transfer of data, from master to slave (in write cycles) or from slave to master (on read cycles), if any is to be transferred. The data phase may transfer a single beat of data (1-4 bytes) for non-burst operations or a 2-beat, 4-beat, 8-beat, or 16-beat burst of data (2 or 4 bytes per beat depending on Port Size) when burst is enabled. On a write cycle, the master must not drive write data until after the address transfer phase is complete. This is to avoid electrical contentions when switching between

drivers. The master must start driving write data one cycle after the address transfer cycle. The master can stop driving the data bus as soon as it samples the TA line asserted on the rising edge of CLKOUT. To facilitate asynchronous write support, the EBI keeps driving valid write data on the data bus until 1 clock after the rising edge where RD_WR and WE are negated (for chip-select accesses only). See *Figure 83* for an example of write timing. On a read cycle, the master accepts the data bus contents as valid on the rising edge of the CLKOUT in which the TA signal is sampled asserted. See *Figure 78* for an example of read timing.

The termination phase is where the cycle is terminated by the assertion of either TA (normal termination) or TEA (termination with error). Termination is discussed in detail in *Section 12.5.2.8: Termination signals protocol*.

Note: In the timing diagrams in this document, asynchronous relationships between signals that switch in the same CLKOUT cycle are not guaranteed. For example, in Figure 83, WE and write DATA change during the same CLKOUT cycle. There is no guarantee that DATA will be stable before WE assertion. External devices should not be latching write DATA on WE assertion, but instead must use a signal edge that takes place in a later CLKOUT cycle, such as WE negation.

12.5.2.4 Single beat transfer

The flow and timing diagrams in this section assume that the EBI is configured in Single Master Mode. Therefore, arbitration is not needed and is not shown in these diagrams.

12.5.2.4.1 Single beat read flow

The handshakes for a single beat read cycle are illustrated in the following flow and timing diagrams.

Figure 77. Basic flow diagram of a single beat read cycle

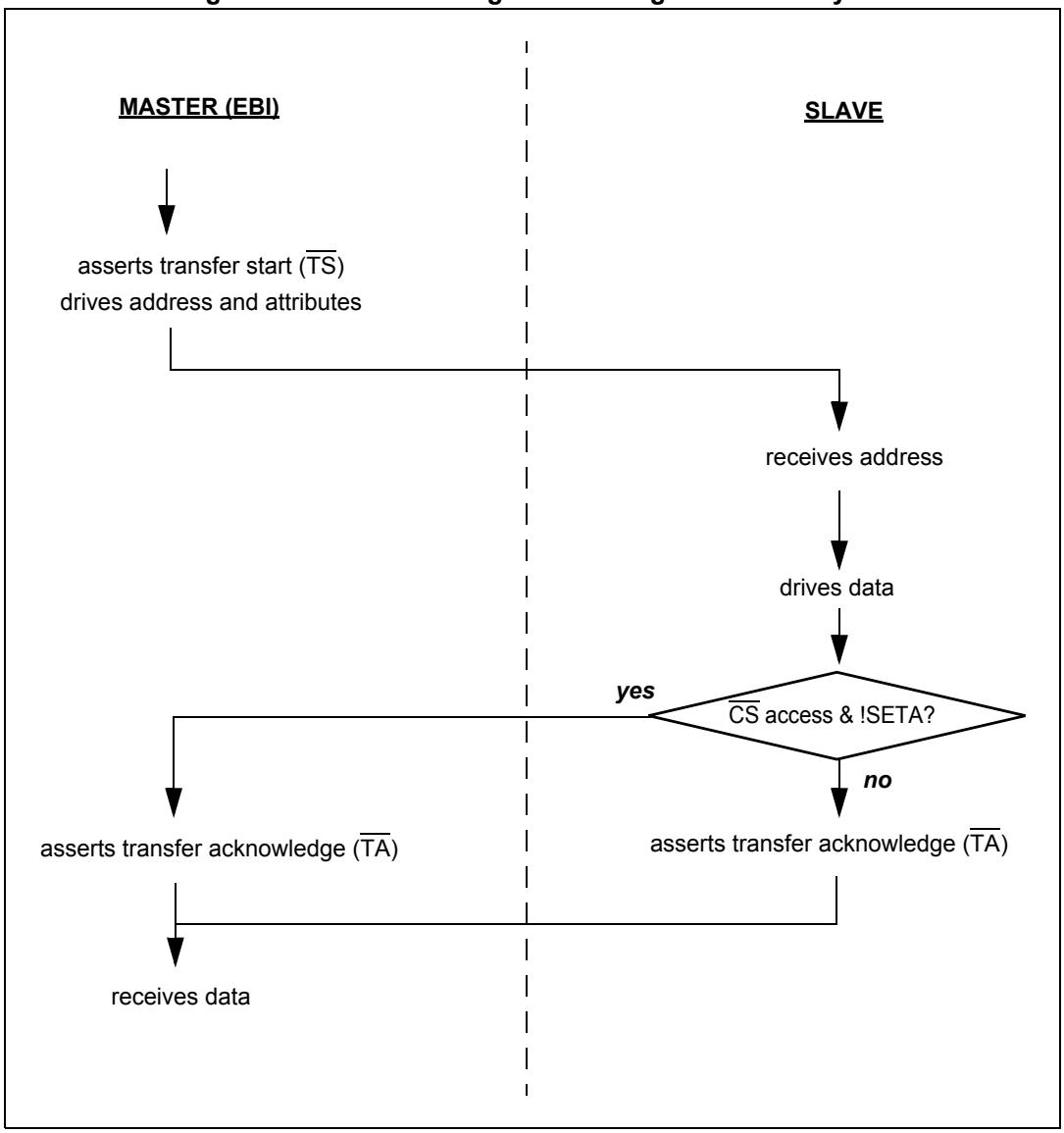


Figure 78. Single beat 32-bit read cycle, $\overline{CS}$ access, zero wait states

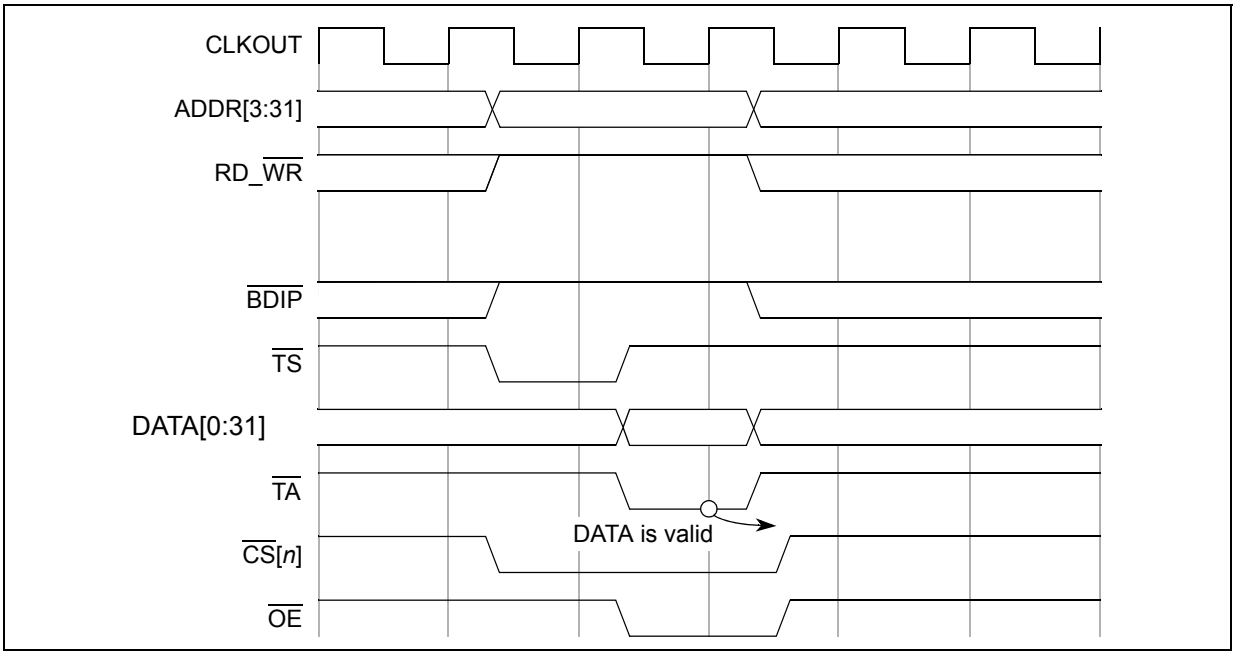


Figure 79. Single beat 32-bit read cycle, $\overline{CS}$ access, one wait state

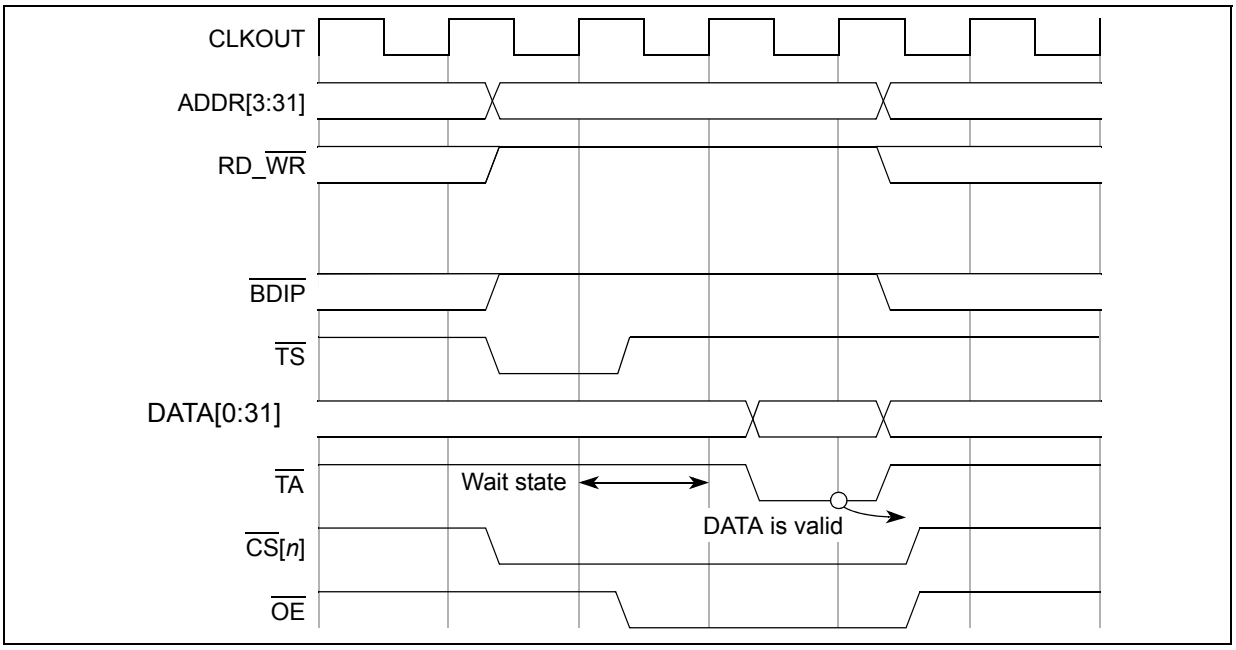
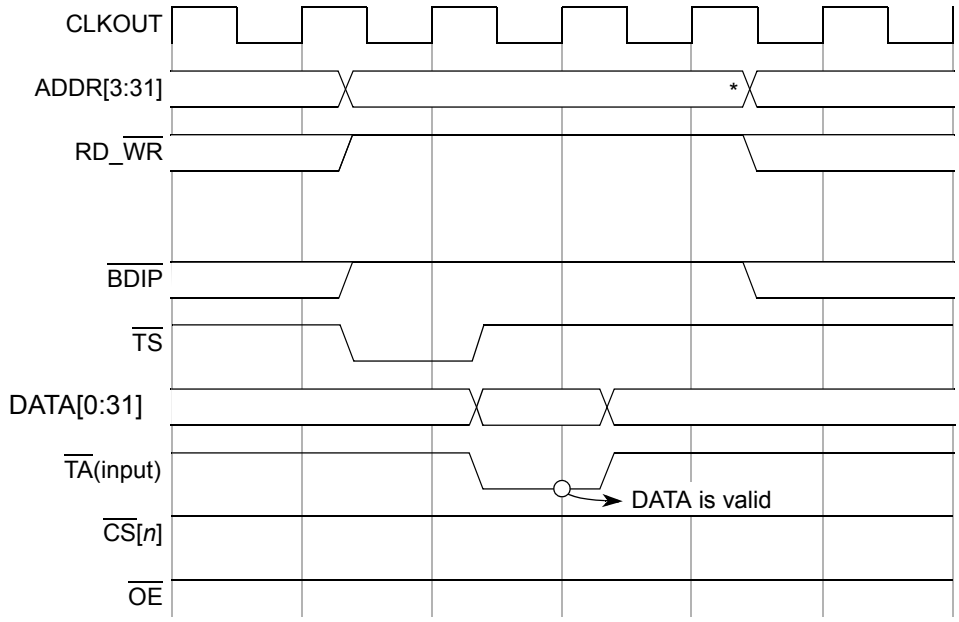
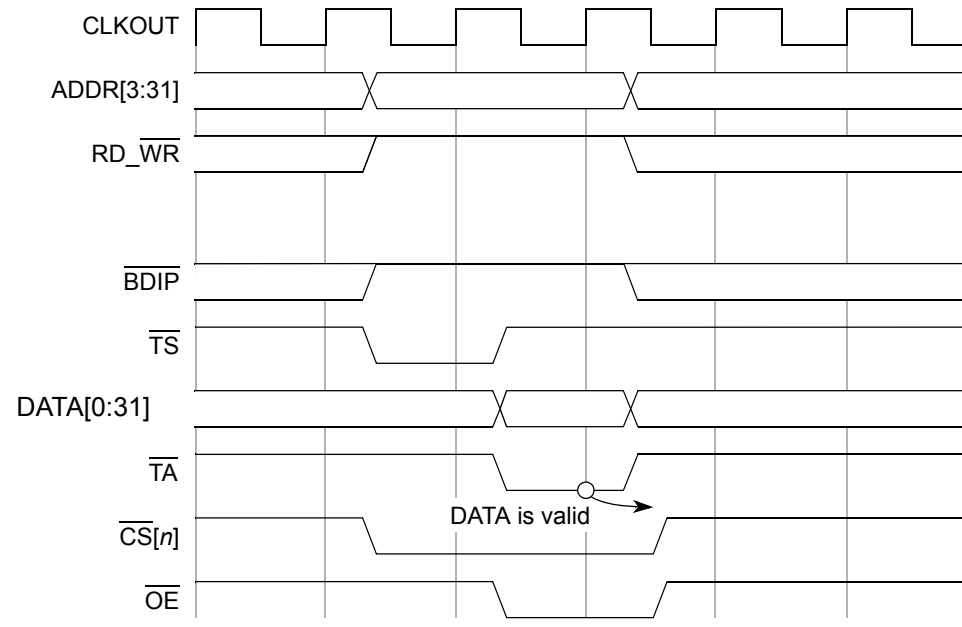


Figure 80. Single beat 32-bit read cycle, non-$\overline{CS}$ access, zero wait states



* The EBI drives address and control signals an extra cycle because it uses a latched version of the external TA (1 cycle delayed) to terminate the cycle.

Figure 81. Single beat 32-bit read cycle, $\overline{CS}$ access, zero wait states (EOE = 0b01, 0b10)



12.5.2.4.2 Single beat write flow

The handshakes for a single beat write cycle are illustrated in the following flow and timing diagrams.

Figure 82. Basic flow diagram of a single beat write cycle

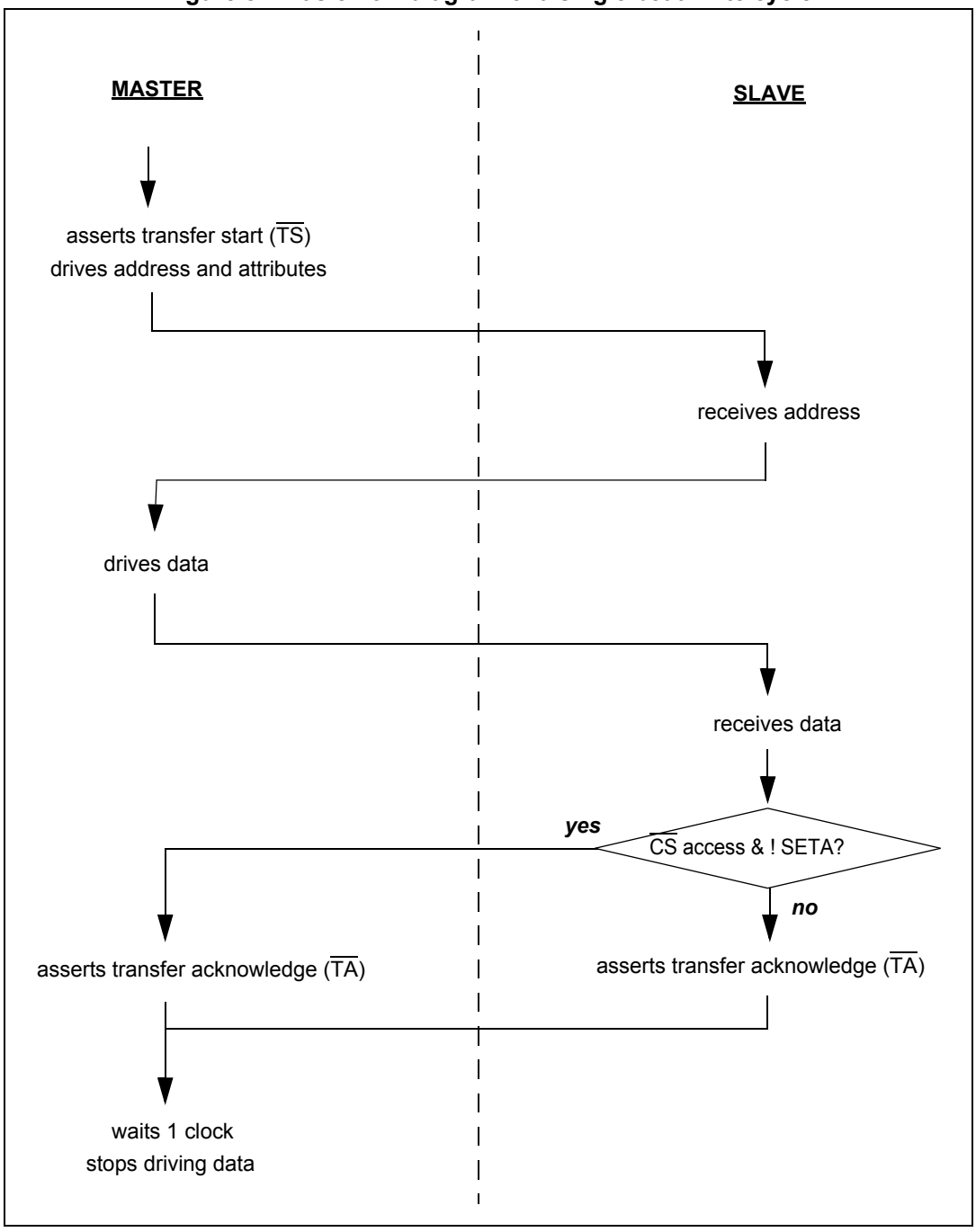


Figure 83. Single beat 32-bit write cycle, $\overline{CS}$ access, zero wait states

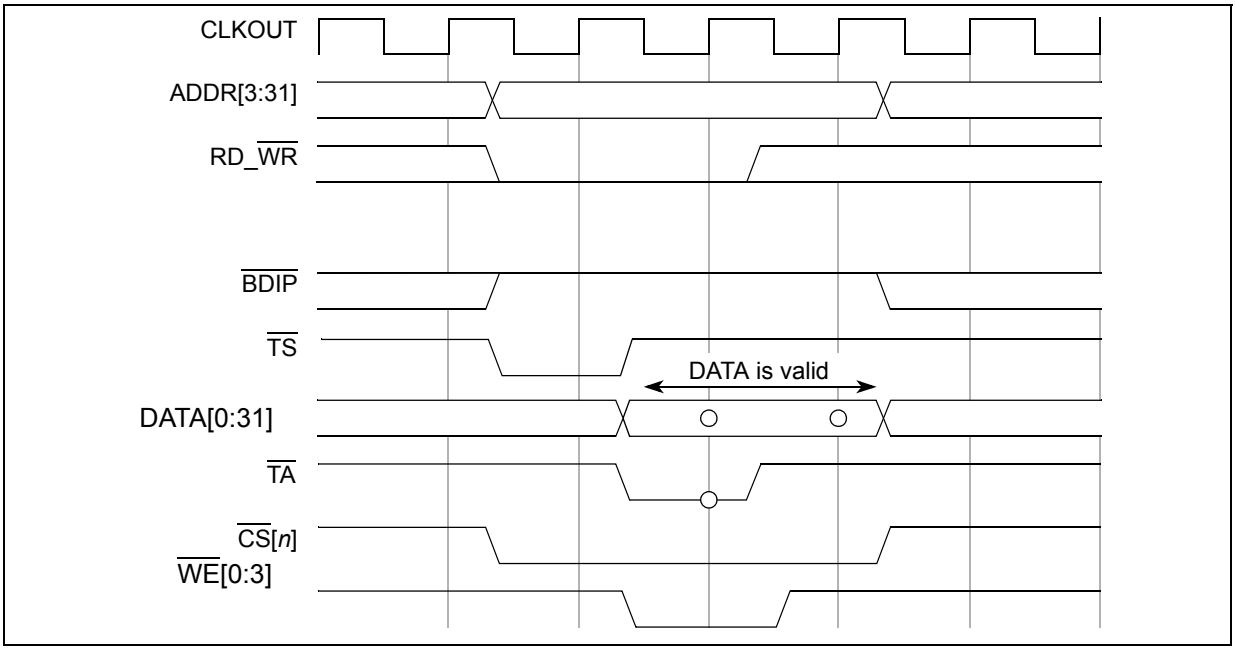


Figure 84. Single beat 32-bit write cycle, $\overline{CS}$ access, one wait state

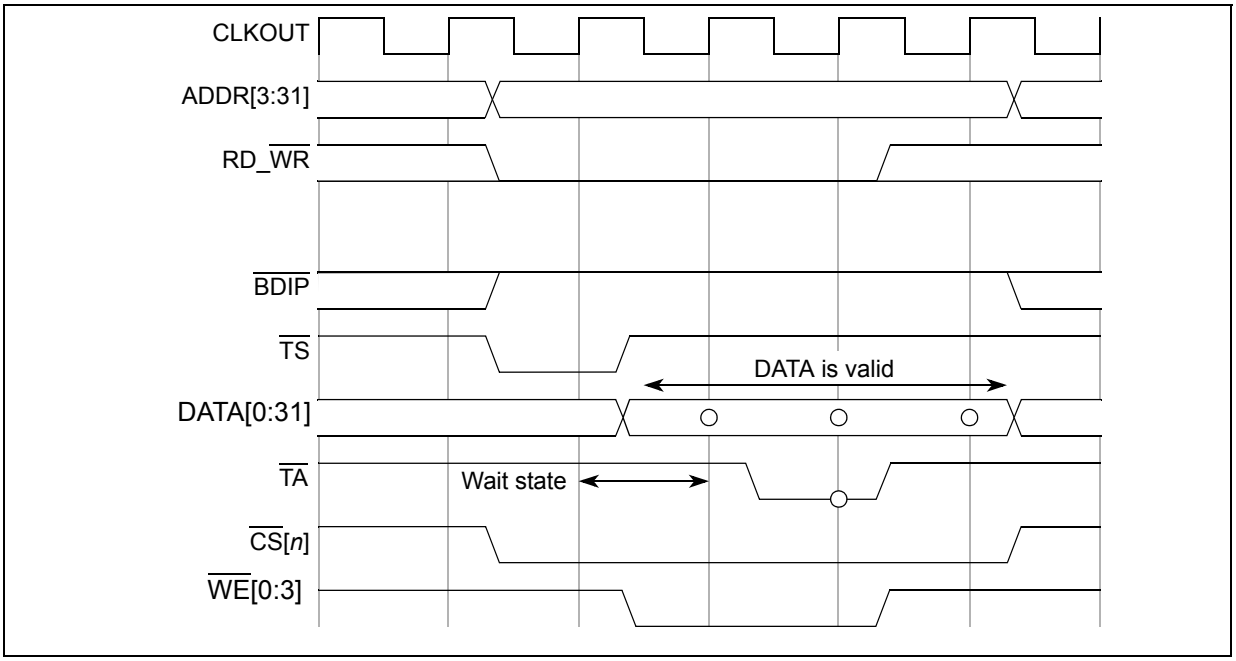
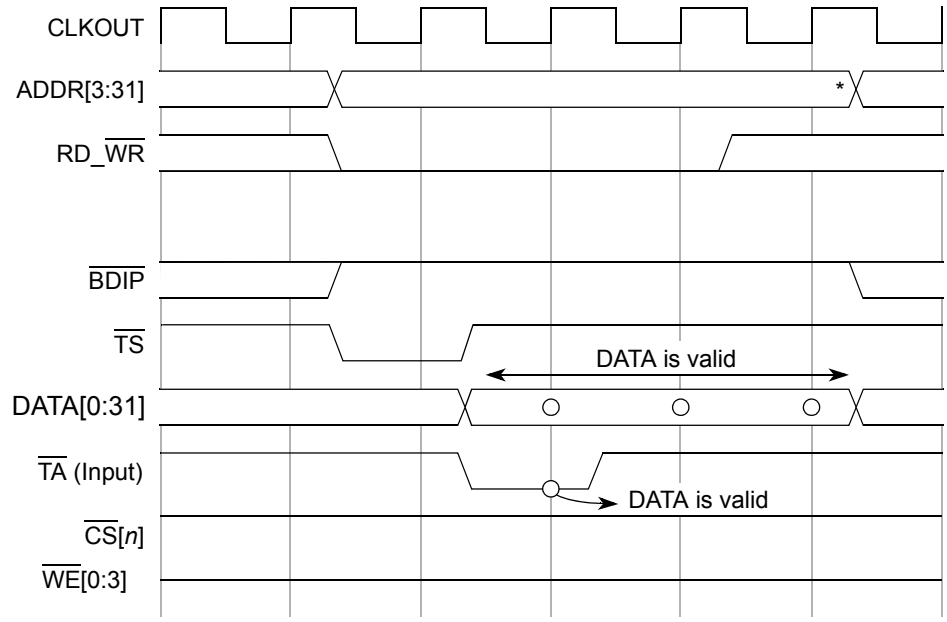
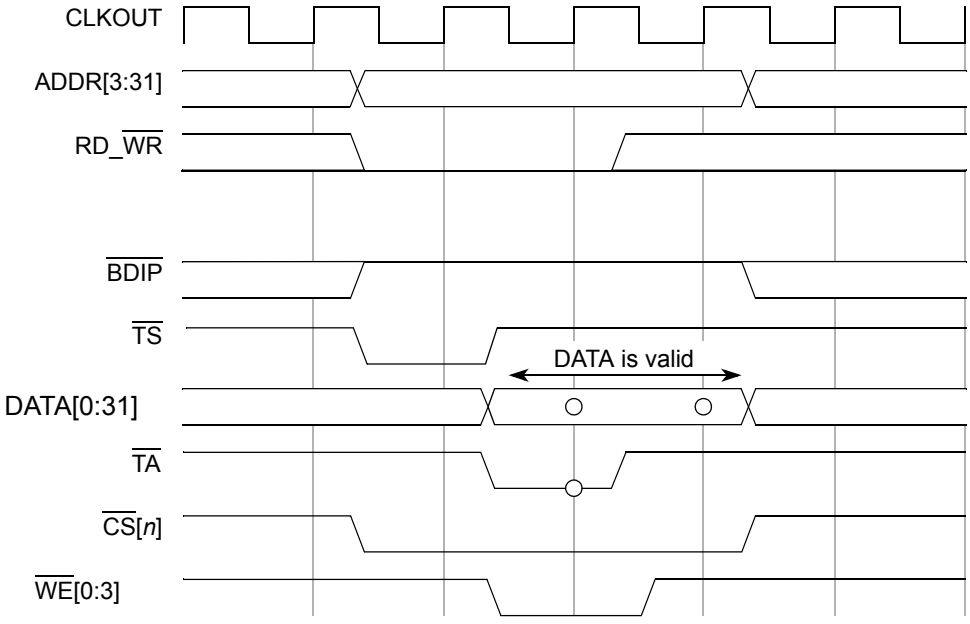


Figure 85. Single beat 32-bit write cycle, non-$\overline{CS}$ access, zero wait states



* The EBI drives address and control signals an extra cycle because it uses a latched version of the external TA (1 cycle delayed) to terminate the cycle.

Figure 86. Single beat 32-bit write cycle, $\overline{CS}$ access, zero wait states, LWRN = 1



12.5.2.4.3 Back-to-back accesses

Due to internal bus protocol, one dead cycle is necessary between back-to-back external bus accesses that are not part of a set of small accesses (see *Section 12.5.2.6: Small accesses (small port size and short burst length)* for small access timing). A dead cycle refers to a cycle between the TA of a previous transfer and the TS of the next transfer.

Note: In some cases, CS remains asserted during this dead cycle, such as the cases of back-to-back writes or read-after-write to the same chip-select (with EBI_BR[GCSN] = 0). See Figure 90 and Figure 91. However, if EBI_BR[GCSN] = 1 (see Figure 92), then the EBI inserts an extra dead cycle between the accesses for these cases in order to guarantee CS negation.

Besides this dead cycle, in most cases, back-to-back accesses on the external bus do not cause any change in the timing from that shown in the previous diagrams, and the two transactions are independent of each other. The only exceptions to this are listed below:

- Back-to-back accesses where the first access ends with an externally-driven TA or TEA. In these cases, an extra cycle is required between the end of the first access and the TS assertion of the second access. See *Section 12.5.2.8: Termination signals protocol* for more details.

The following diagrams show a few examples of back-to-back accesses on the external bus.

Figure 87. Back-to-back 32-bit reads to the same CS bank

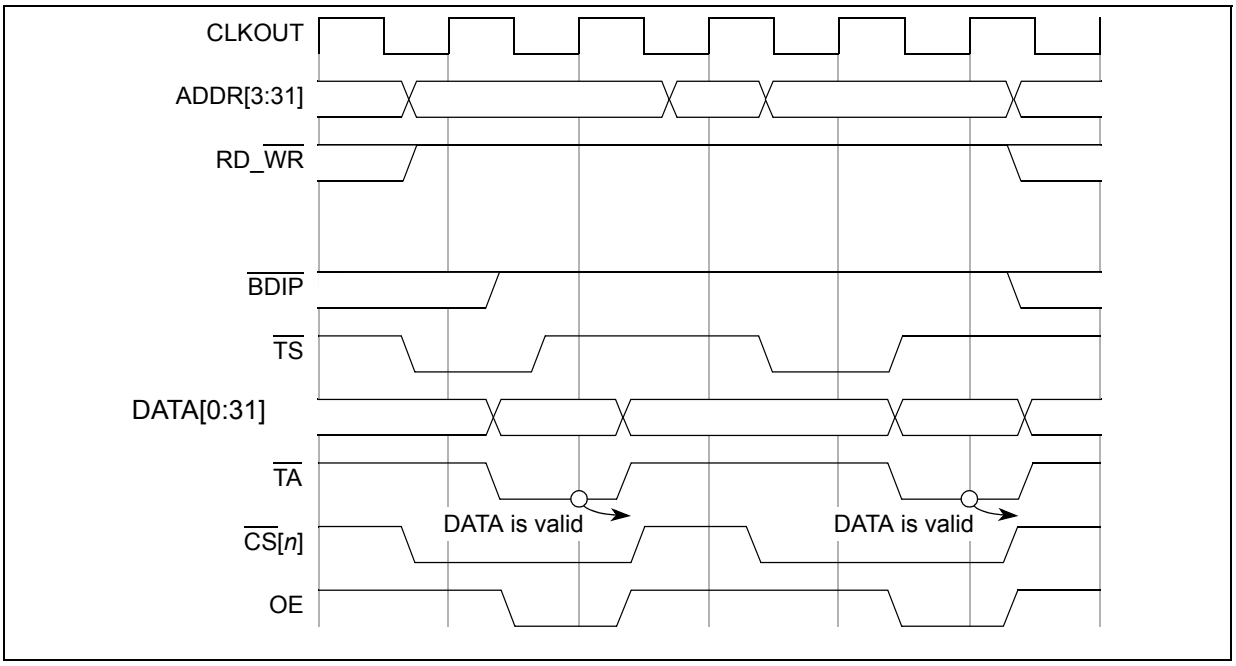


Figure 88. Back-to-back 32-bit reads to different $\overline{CS}$ banks

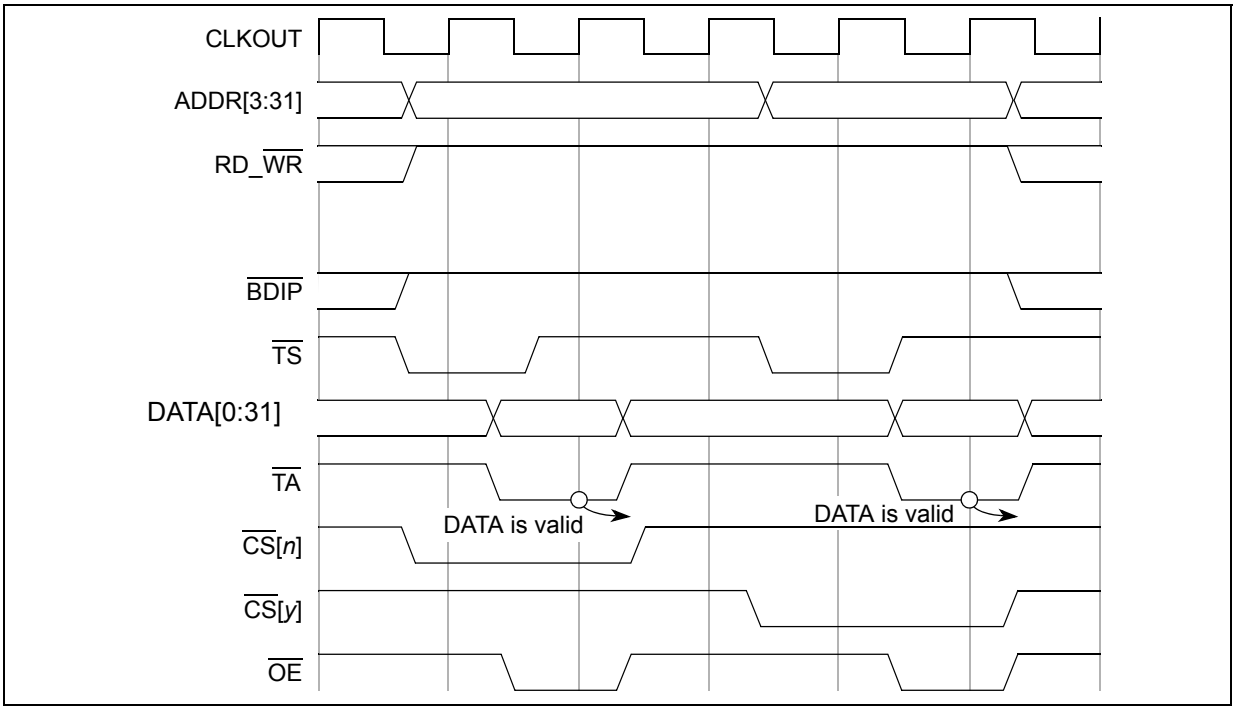


Figure 89. Write after read to the same $\overline{CS}$ bank

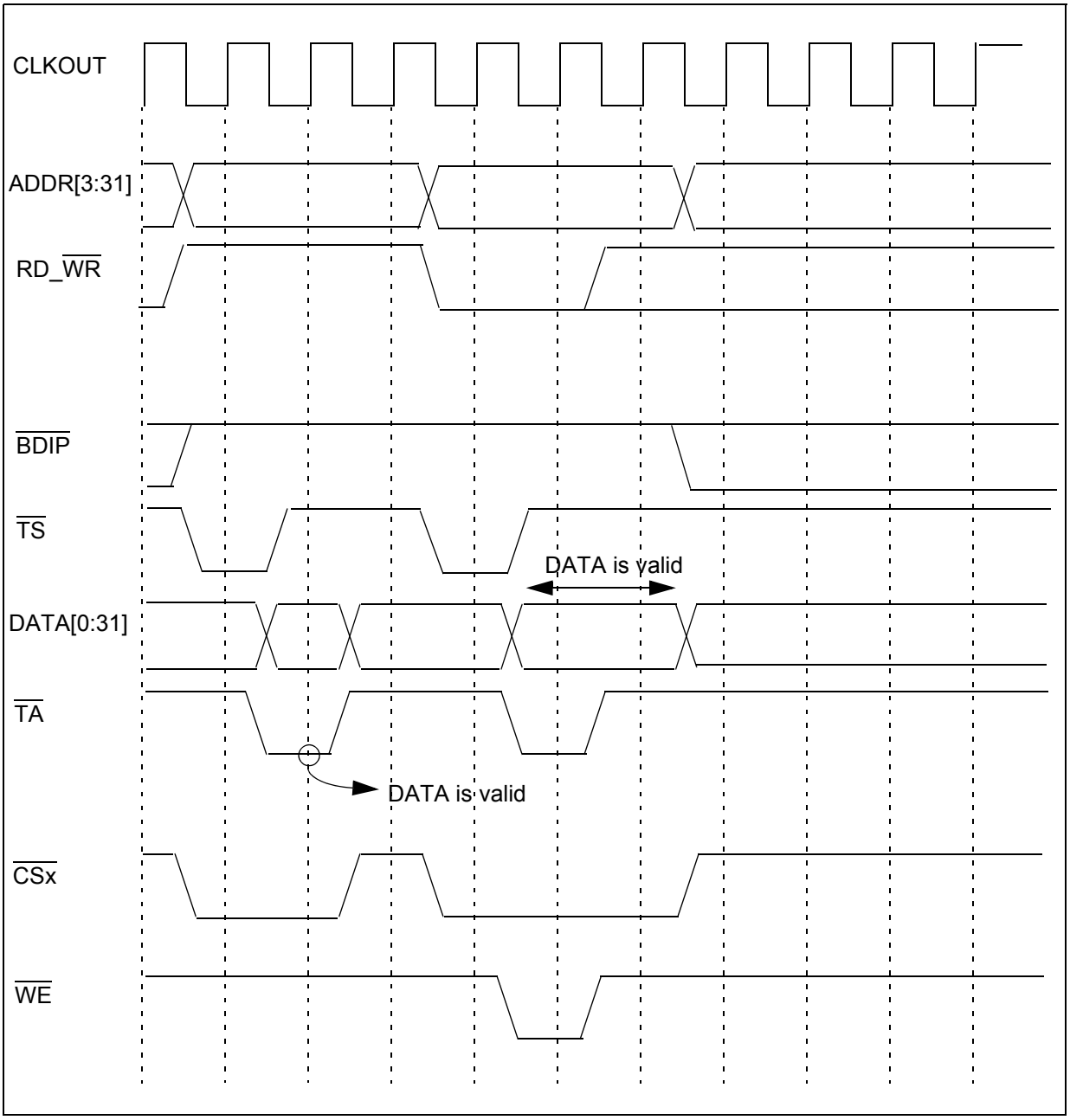


Figure 90. Back-to-back 32-bit writes to the same $\overline{CS}$ bank

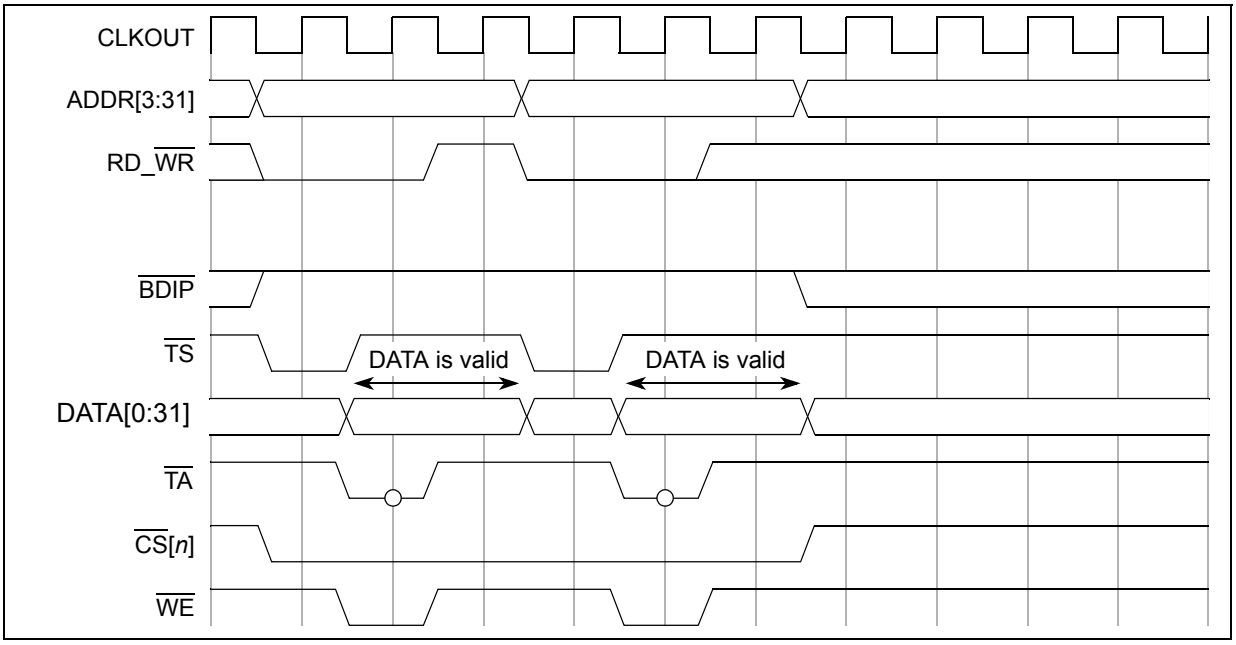


Figure 91. Read after write to the same $\overline{CS}$ bank

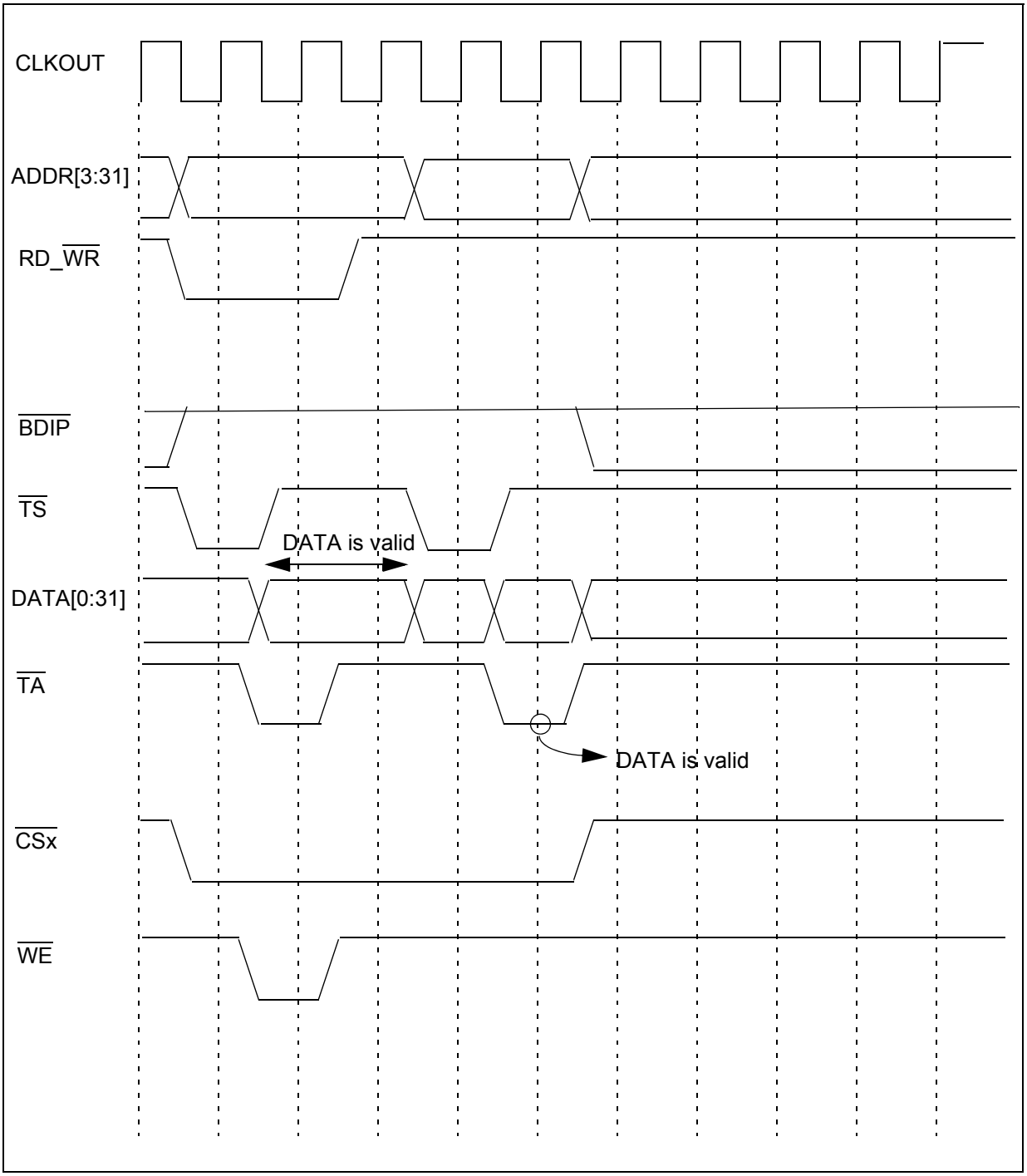
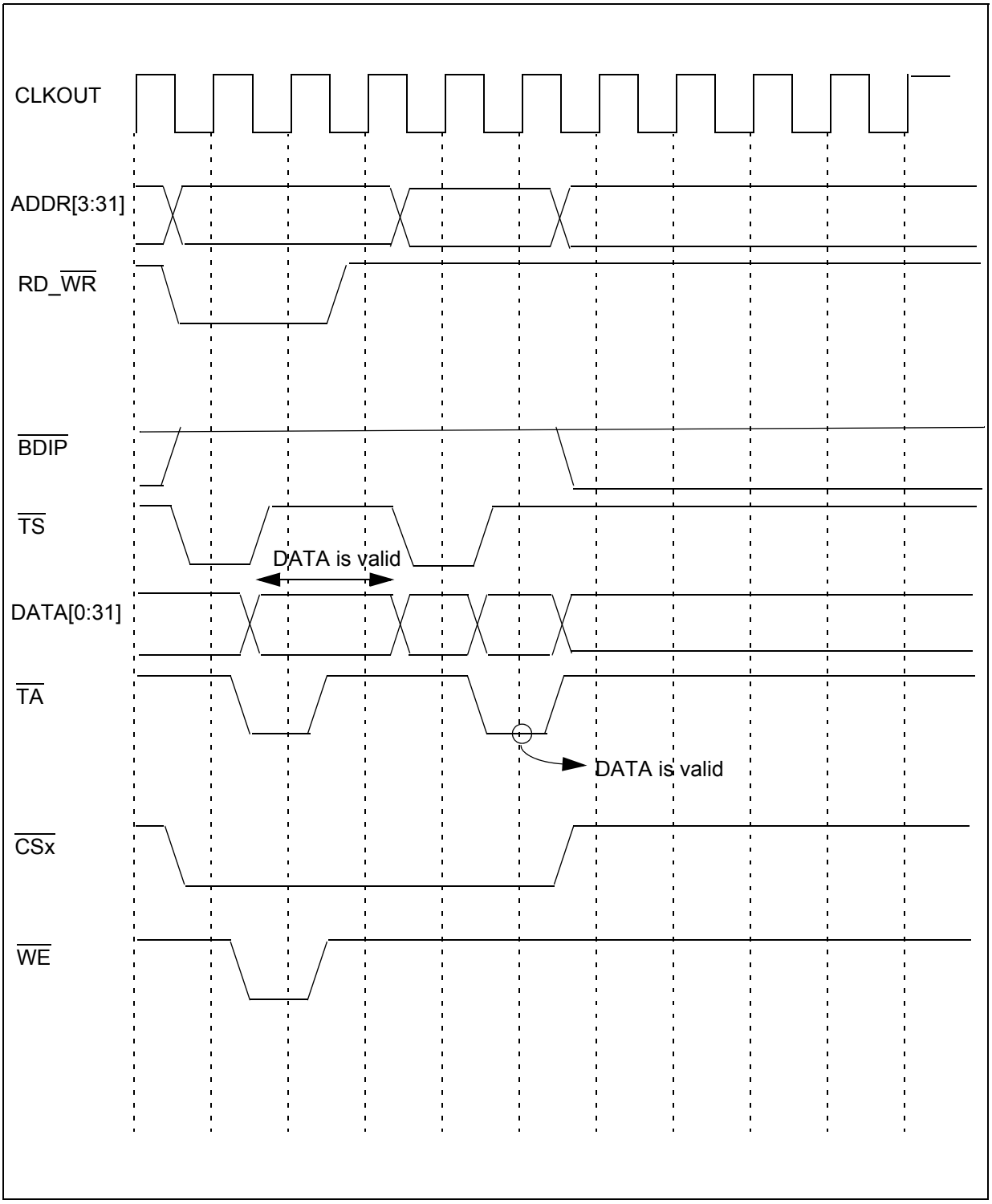


Figure 92. Read after write to the same $\overline{CS}$ bank (EBI_BR[GCSN] = 1)



12.5.2.5 Burst transfer

The EBI supports wrapping 32-byte critical-doubleword-first burst transfers. Bursting is supported only for internally-requested (e.g. core, DMA, etc.) cache-line size (32-byte) read accesses to external devices that use the chip selects[e].

Note: In the case where the internal AMBA data bus width is 32 bits (instead of 64), the EBI supports a 16-byte critical-word-first burst transfer (instead of 32-byte critical-doubleword-first). In general, this document assumes the default case of 64-bit AMBA bus, 32-byte burst transfer.

Accesses to devices operating without a chip select are always single beat. If an internal request to the EBI indicates a size of less than 32 bytes, the request is fulfilled by running one or more single-beat external transfers, not by an external burst transfer.

An 8-word wrapping burst reads eight 32-bit words by supplying a starting address that points to one of the words (doubleword aligned) and requiring the memory device to sequentially drive each word on the data bus. The selected slave device must internally increment ADDR[27:29] (also ADDR30 in the case of a 16-bit port size device) of the supplied address for each transfer, until the address reaches an 8-word boundary, and then wrap the address to the beginning of the 8-word boundary. The address and transfer attributes supplied by the EBI remain stable during the transfers. Termination of each beat transfer occurs by the EBI asserting $\overline{TA}$ (SETA = 1 is not supported for burst transfers). The EBI requires that addresses be aligned to a doubleword boundary on all burst cycles.

Table 94 shows the burst order of beats returned for an 8-word burst to a 32-bit port.

Table 94. Wrap bursts order

Burst starting address ADDR[27:28]	Burst order (assuming 32-bit port size)
00	word0 -> word1 -> word2 -> word3 -> word4 -> word5 -> word6 -> word7
01	word2 -> word3 -> word4 -> word5 -> word6 -> word7 -> word0 -> word1
10	word4 -> word5 -> word6 -> word7 -> word0 -> word1 -> word2 -> word3
11	word6 -> word7 -> word0 -> word1 -> word2 -> word3 -> word4 -> word5

The general case of burst transfers assumes that the external memory has 32-bit port size and 8-word burst length (SBL = 0, BL = 0). The EBI can also burst from 16-bit port size memories, taking twice as many external beats to fetch the data as compared to a 32-bit port with the same burst length. The EBI can also burst from 16-bit or 32-bit memories that have a 4-word burst length (SBL = 0, BL = 1 in the appropriate Base Register). In this case, two external 4-word burst transfers (wrapping on 4-word boundary) are performed to fulfill the internal 8-word request[f]. This operation is considered atomic by the EBI, so the EBI does not allow other unrelated master accesses or bus arbitration to intervene between the transfers. For more details and a timing diagram, see *Example 10: Small access example #3: 32-byte read to 32-bit port with BL = 1*.

e. Except for the special case of a 32-bit non-chip-select access in 16-bit data bus mode. See *Section 12.5.2.9: Non-chip-select burst in 16-bit Data Bus Mode*.

f. This case (of two external 4-word burst transfers being required) applies only to AMBA data bus width of 64 bits.

The EBI can also burst from 16-bit or 32-bit memories that have a 2-word burst length (SBL = 1, BL = X in the appropriate Base Register). In this case, four[g] external 2-word burst transfers (wrapping on 2-word boundary) are performed to fulfill the internal 8-word request. This operation is considered atomic by the EBI, so the EBI does not allow other unrelated master accesses or bus arbitration to intervene between the transfers. For more details and a timing diagram, see *Example 12: Small access example #5: 32-byte read to 32-bit port with SBL = 1*.

During burst cycles, the $\overline{\text{BDIP}}$ (Burst Data In Progress) signal is used to indicate the duration of the burst data. During the data phase of a burst read cycle, the EBI receives data from the addressed slave. If the EBI needs more than one data, it asserts the $\overline{\text{BDIP}}$ signal. Upon receiving the data prior to the last data, the EBI negates $\overline{\text{BDIP}}$. Thus, the slave stops driving new data after it receives the negation of $\overline{\text{BDIP}}$ on the rising edge of the clock. Some slave devices have their burst length and timing configurable internally and thus may not support connecting to a $\overline{\text{BDIP}}$ pin. In this case, $\overline{\text{BDIP}}$ is driven by the EBI normally, but the output is ignored by the memory and the burst data behavior is determined by the internal configuration of the EBI and slave device. When the TBDIP bit is set in the appropriate Base Register, the timing for $\overline{\text{BDIP}}$ is altered. See *Section 12.5.2.5.1: TBDIP effect on burst transfer* for this timing.

Since burst writes are not supported by the EBI[h], the EBI negates $\overline{\text{BDIP}}$ during write cycles.

g. Applies to 64-bit AMBA data bus width. For 32-bit AMBA data bus width, only two transfers are performed.

h. Except for the special case of a 32-bit non-chip-select access in 16-bit data bus mode. See *Section 12.5.2.9: Non-chip-select burst in 16-bit Data Bus Mode*.

Figure 93. Basic flow diagram of a burst read cycle

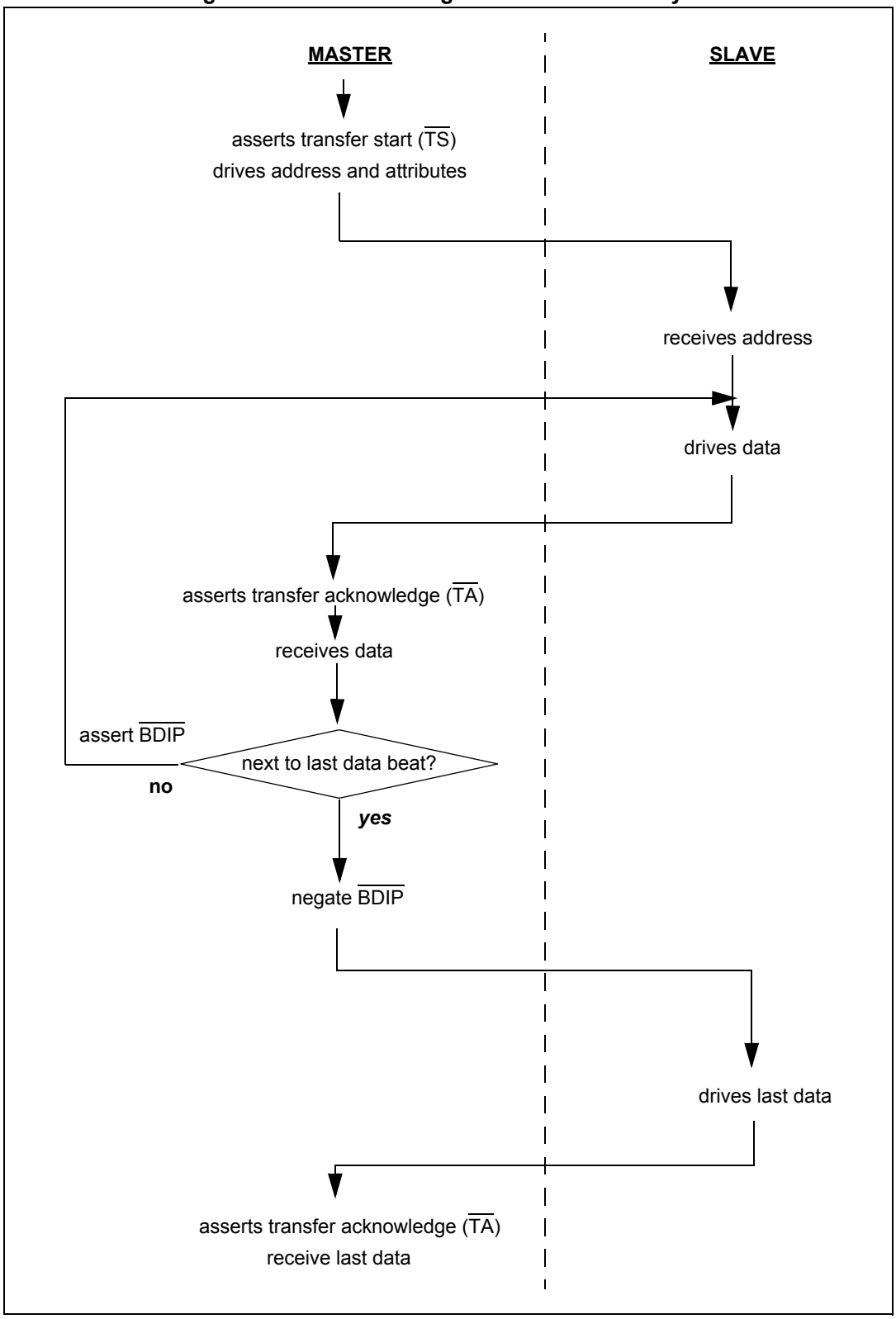


Figure 94. Burst 32-bit read cycle, zero wait states

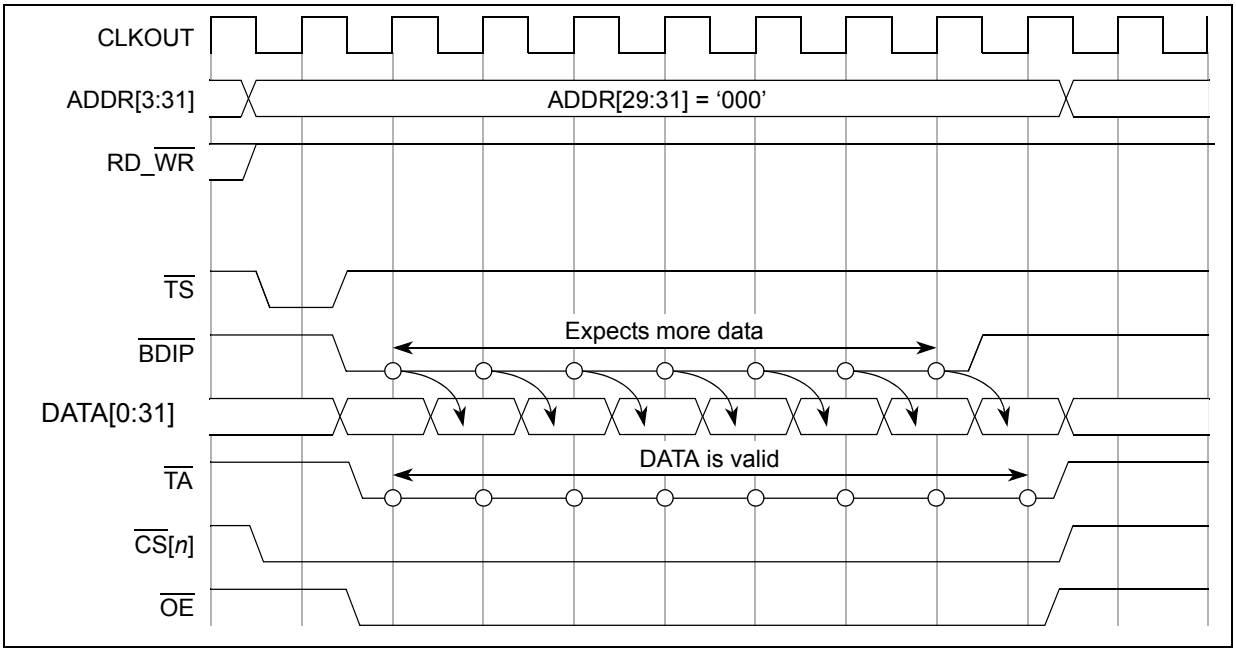
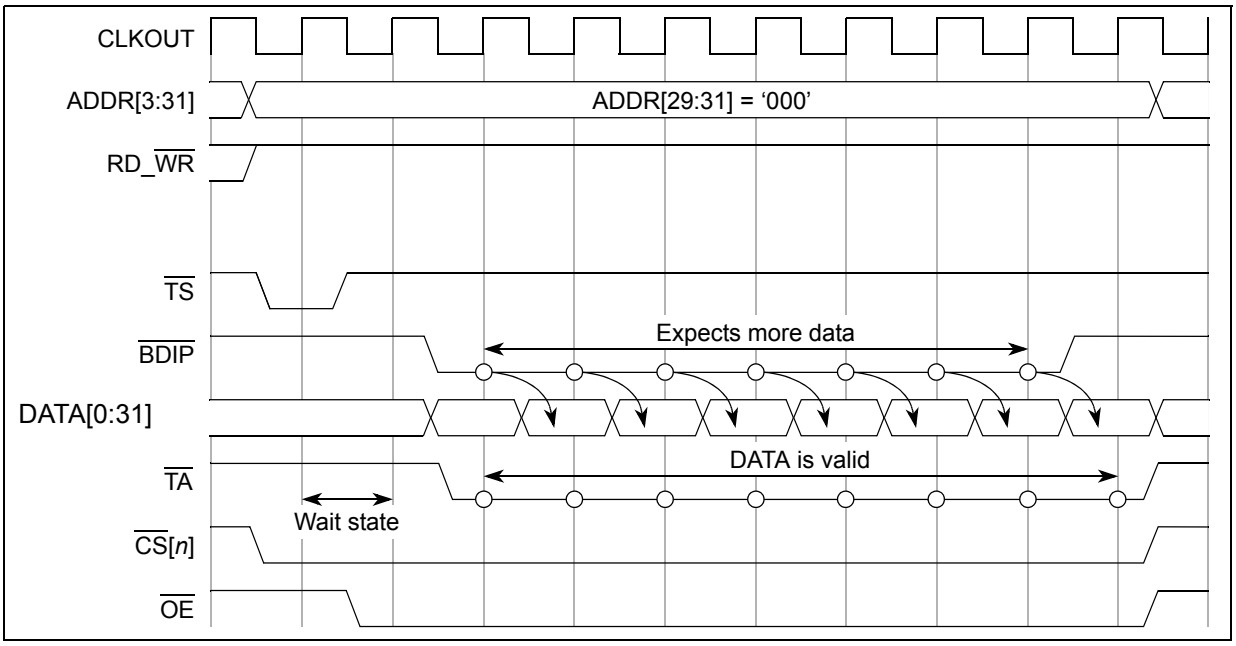


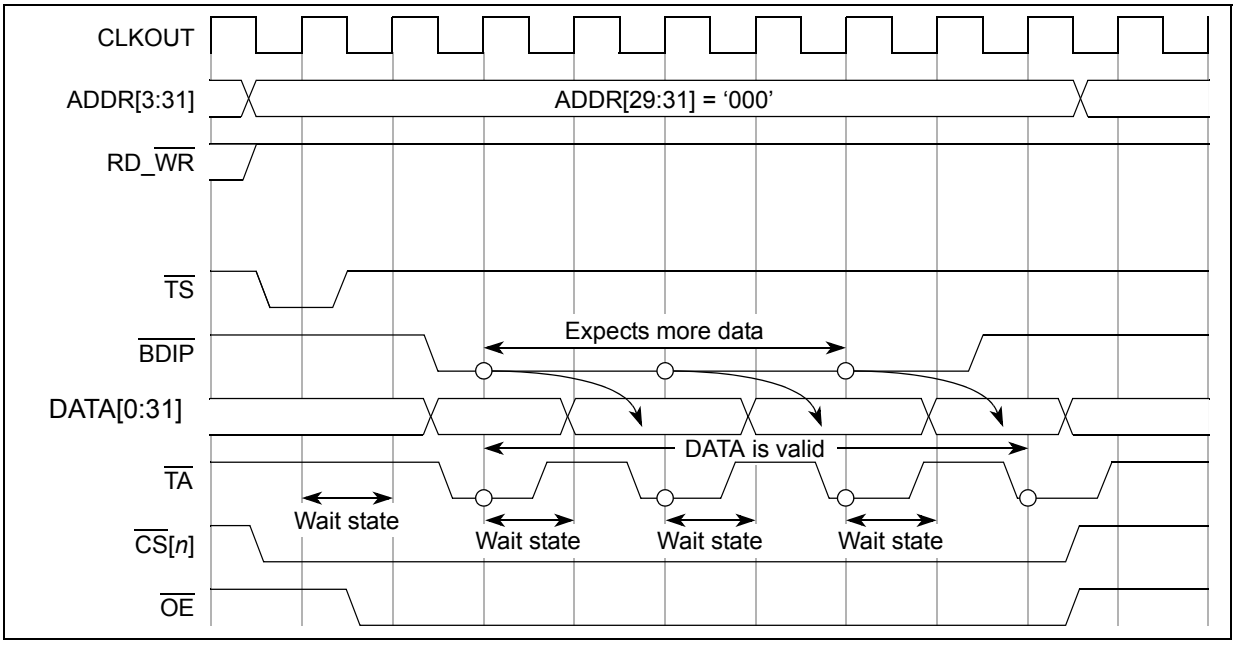
Figure 95. Burst 32-bit read cycle, one initial wait state



12.5.2.5.1 TBDIP effect on burst transfer

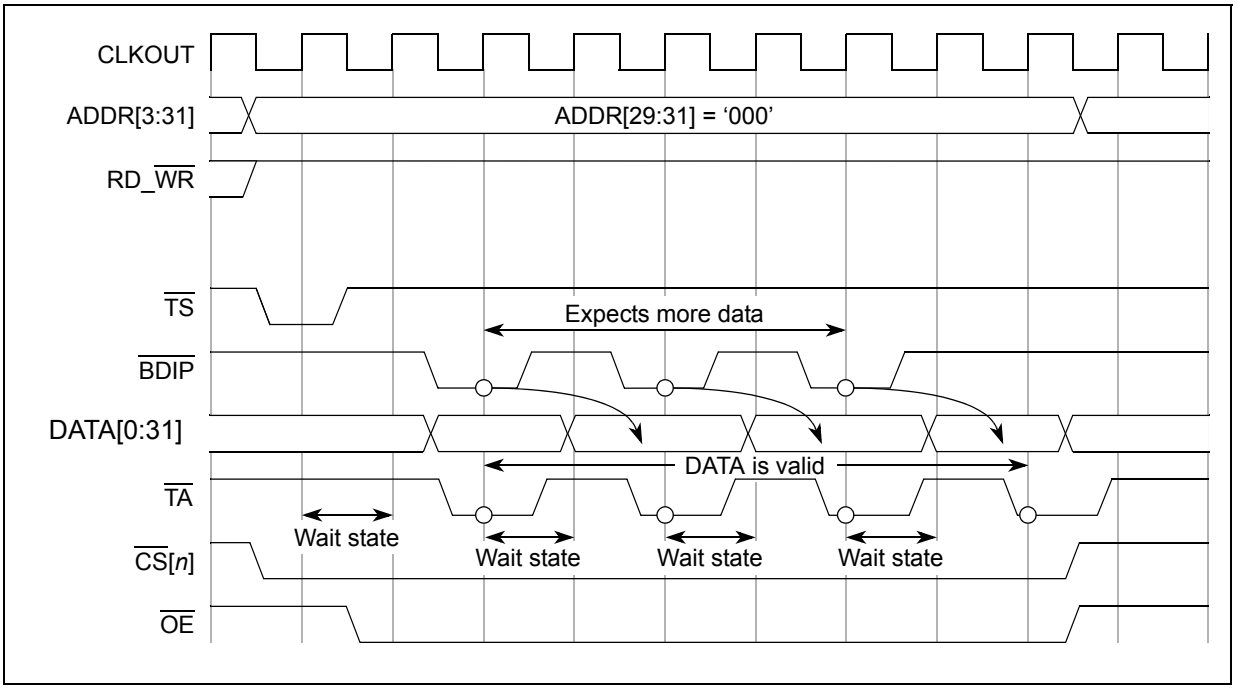
Some memories require different timing on the $\overline{\text{BDIP}}$ signal than the default to run burst cycles. Using the default value of TBDIP = 0 in the appropriate EBI Base Register results in $\overline{\text{BDIP}}$ being asserted (SCY+1) cycles after the address transfer phase, and being held asserted throughout the cycle regardless of the wait states between beats (BSCY). *Figure 96* shows an example of the TBDIP = 0 timing for a 4-beat burst with BSCY = 1.

Figure 96. Burst 32-bit read cycle, one wait state between beats, TBDIP = 0



When using TBDIP = 1, the BDIP behavior changes to toggle between every beat when BSCY is a non-zero value. *Figure 97* shows an example of the TBDIP = 1 timing for the same 4-beat burst shown in *Figure 96*.

Figure 97. Burst 32-bit read cycle, one wait state between beats, TBDIP = 1



12.5.2.6 Small accesses (small port size and short burst length)

In this context, a *small access* refers to an access whose burst length and port size (SBL, BL, PS bits in Base Register for chip-select access or default burst disabled, 32-bit port for non-chip-select access) are such that the number of bytes requested by the internal master cannot all be fetched (or written) in one external transaction. If this is the case, the EBI initiates multiple transactions until all the requested data is transferred. It should be noted that all the transactions initiated to complete the data transfer are considered as an atomic transaction, so the EBI does not allow other unrelated master accesses to intervene between the transfers.

Table 95 shows all the combinations of burst length, port size, and requested byte count that cause the EBI to run multiple external transactions to fulfill the request.

Table 95. Small access cases

Byte count requested by internal master	Burst length	Port size	# External accesses to fulfill request
Non-Burstable Chip-Select Banks (BI = 1) or Non-Chip-Select Access			
4	1 beat	16-bit	2/1(1)
8	1 beat	32-bit	2
8	1 beat	16-bit	4
16(2)	1 beat	32-bit	4
16(2)	1 beat	16-bit	8
32(3)	1 beat	32-bit	8
32(3)	1 beat	16-bit	16
Burstable Chip-Select Banks (BI = 0)			
32(3)	4 words	16-bit (8 beats), 32-bit (4 beats)	2
32(4)	2 words	16-bit (4 beats), 32-bit (2 beats)	4

1. In 32-bit data bus mode (DBM = 0 in EBI_MCR), two accesses are performed. In 16-bit data bus mode (DBM = 1), one 2-beat burst access is performed and this is not considered a “small access” case. See *Section 12.5.2.9: Non-chip-select burst in 16-bit Data Bus Mode* for this special DBM = 1 case.
2. Only supported for case of 32-bit internal AMBA data bus.
3. Only supported for case of 64-bit internal AMBA data bus.
4. SBL = 1 (2-word burst case).

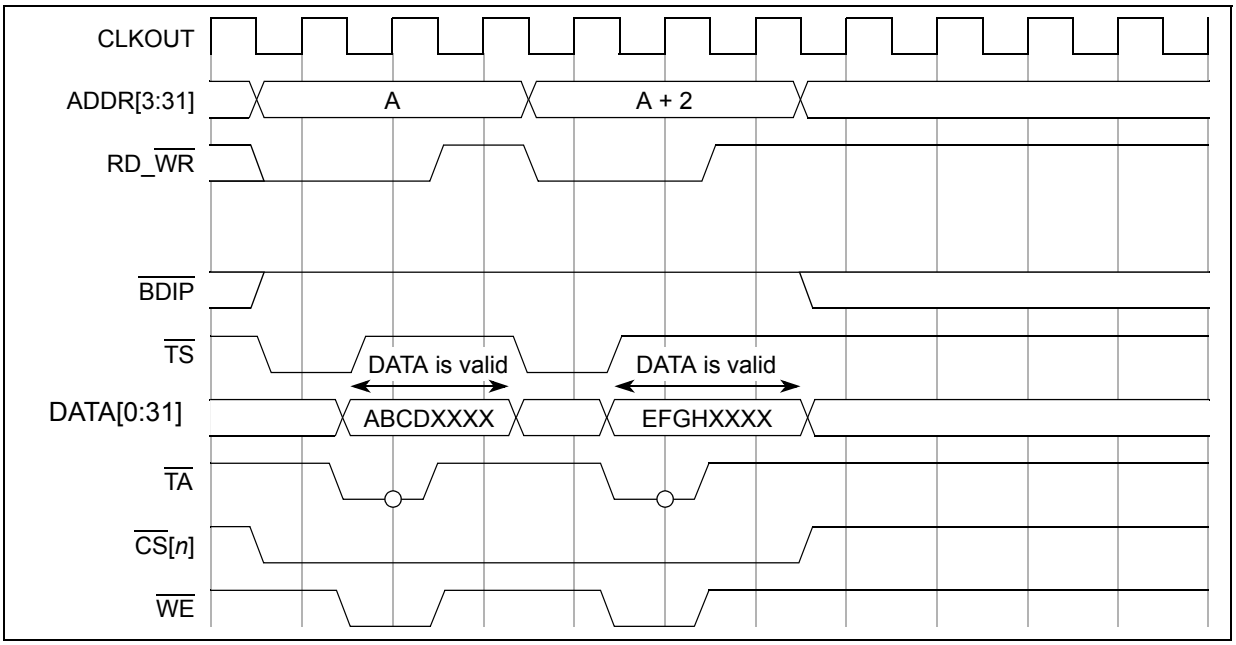
In most cases, the timing for small accesses is the same as for normal single-beat and burst accesses, except that multiple back-to-back external transfers are executed for each internal request. These transfers have no additional dead cycles in-between that are not present for back-to-back stand-alone transfers except for the case of writes with an internal request size of > 64 bits, discussed in *Example 9: Small access example #2: 32-byte write with external TA*.

A few examples of small access are provided on the following pages. The timing for the remaining cases in *Table 95* can be extrapolated from these and the other timing diagrams in this document.

Example 8 Small access example #1: 32-bit write to 16-bit port

Figure 98 shows an example of a 32-bit write to a 16-bit port, requiring two 16-bit external transactions.

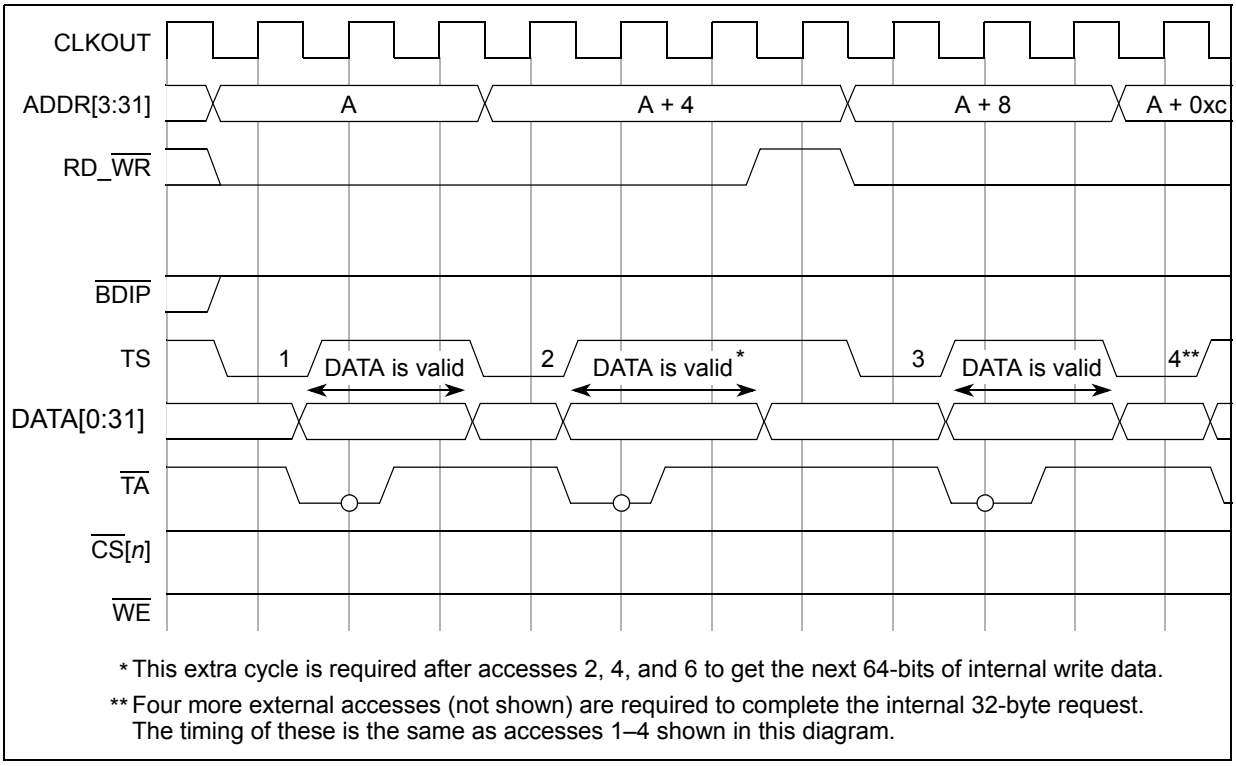
Figure 98. Single beat 32-bit write cycle, 16-bit port size, basic timing



Example 9 Small access example #2: 32-byte write with external TA

Figure 99 shows an example of a 32-byte write to a non-chip-select device using external TA, requiring eight 32-bit external transactions. Note that due to the use of external TA, RD_WR does not toggle between the accesses unless that access is the end of a 64-bit boundary. In this case, an extra cycle is required between TA and the next TS in order to get the next 64-bits of write data internally and RD_WR negates during this extra cycle.

Figure 99. 32-byte write cycle with external $\overline{TA}$, basic timing



* This extra cycle is required after accesses 2, 4, and 6 to get the next 64-bits of internal write data.

** Four more external accesses (not shown) are required to complete the internal 32-byte request. The timing of these is the same as accesses 1–4 shown in this diagram.

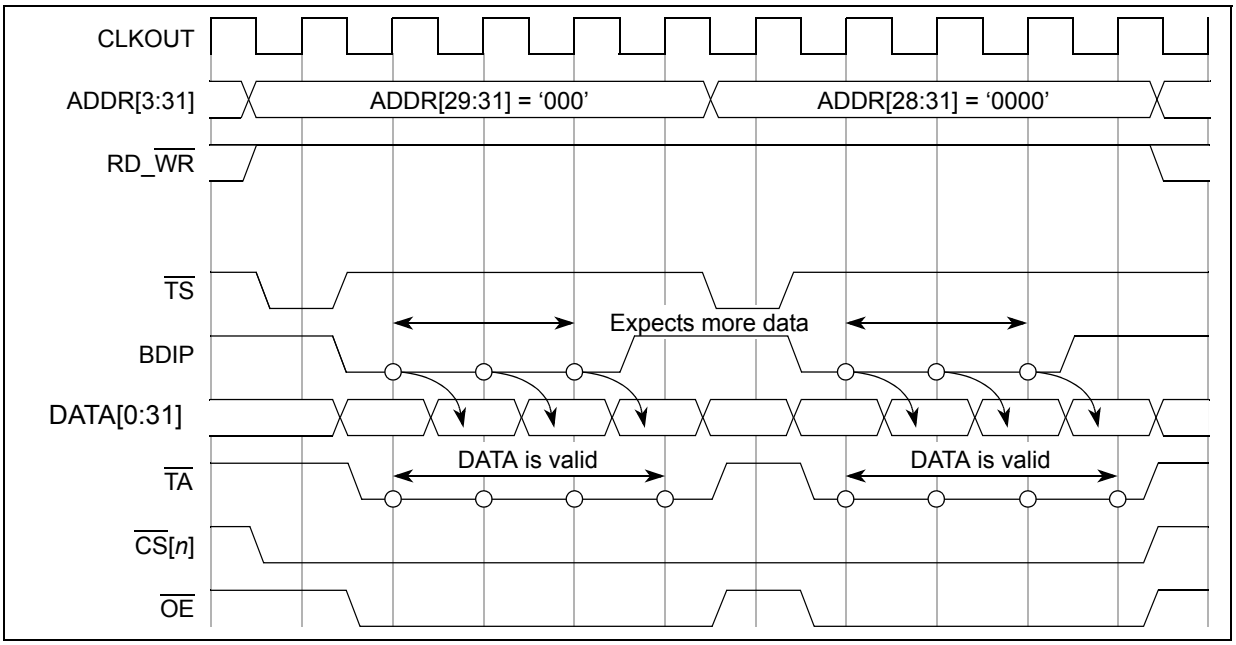
Example 10 Small access example #3: 32-byte read to 32-bit port with BL = 1

Figure 100 shows an example of a 32-byte read to a 32-bit burst enabled port with burst length of 4 words, requiring two 4-word (16-byte) external transactions. For this case, the address for the second 4-word burst access is calculated by adding 0x10 to the lower 5 bits of the first address (no carry), and then masking out the lower 4 bits to fix them at zero.

Table 96. Examples of 4-word burst addresses

1st address	Lower 5 bits of 1st address + 0x10 (no carry)	Final 2nd address (after masking lower 4 bits)
0x000	0x10	0x10
0x008	0x18	0x10
0x010	0x00	0x00
0x018	0x08	0x00
0x020	0x30	0x30
0x028	0x38	0x30
0x030	0x20	0x20
0x038	0x28	0x20

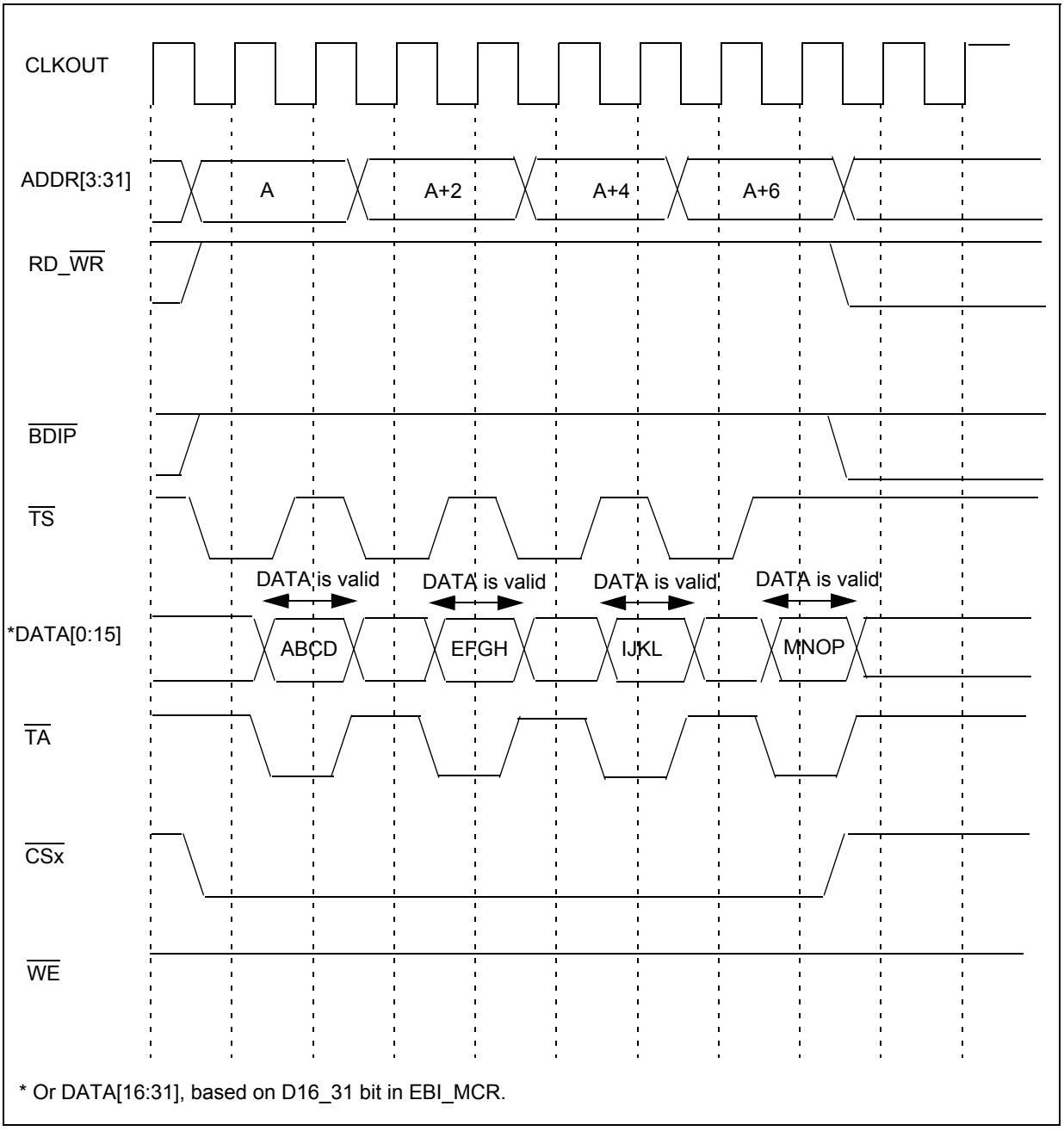
Figure 100. 32-byte read with back-to-back 4-word bursts to 32-bit port, zero wait states



Example 11 Small access example #4: 64-bit read to 16-bit port

Figure 101 shows an example of a 64-bit read to a 16-bit port, requiring four 16-bit external transactions.

Figure 101. Single beat 64-bit read cycle, 16-bit port size, basic timing



* Or DATA[16:31], based on D16_31 bit in EBI_MCR.

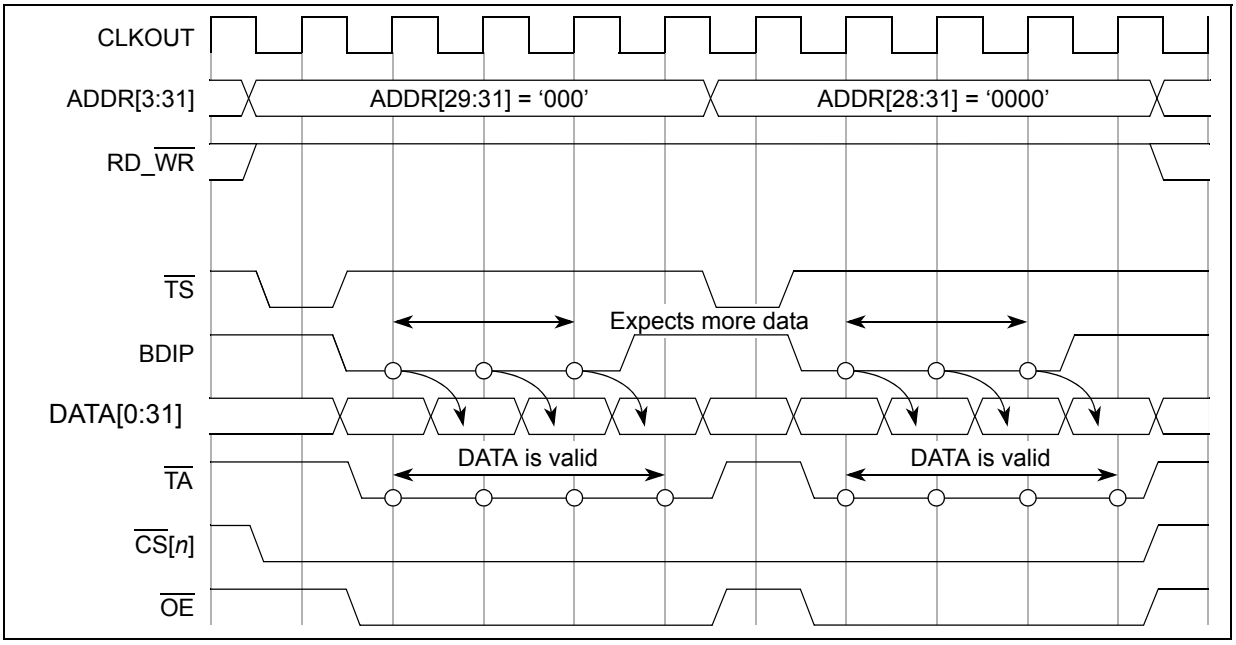
Example 12 Small access example #5: 32-byte read to 32-bit port with SBL = 1

Figure 100 shows an example of a 32-byte read to a 32-bit burst enabled port with burst length of 2 words, requiring four 8-byte external transactions. For this case, the address for the second–fourth 2-word burst accesses is calculated by adding 0x08 to the lower 5 bits of the first address (no carry), and then masking out the lower 3 bits to fix them at zero.

Table 97. Examples of 2-word burst addresses

1st address	Lower 5 bits of 1st address + 0x08 (no carry)	Final 2nd–4th addresses (after masking lower 3 bits)
0x000	0x08	0x08, 0x10, 0x18
0x008	0x10	0x10, 0x18, 0x00
0x010	0x18	0x18, 0x00, 0x08
0x018	0x00	0x00, 0x8, 0x10
0x020	0x28	0x28, 0x30, 0x38
0x028	0x30	0x30, 0x38, 0x20
0x030	0x38	0x38, 0x20, 0x28
0x038	0x20	0x20, 0x28, 0x30

Figure 102. 32-byte read with 4 back-to-back 2-word bursts to 32-bit port, zero wait states (SBL = 1)



12.5.2.7 Size, alignment and packaging on transfers

Table 98 shows the allowed sizes that an internal master can request from the EBI. The behavior of the EBI for request sizes not shown below is undefined. No error signal is asserted for these erroneous cases.

Table 98. Transaction sizes supported by EBI

Number of bytes (internal master)	Number of bytes (external master)
1	1
2	2

Table 98. Transaction sizes supported by EBI (continued)

Number of bytes (internal master)	Number of bytes (external master)
4	4
3(1)	—
8	—
16(2)	—
32(3)	—

1. Some misaligned access cases may result in 3-byte writes. These cases are treated as power-of-2 sized requests by the EBI, using WE_BE[0:3] to make sure only the appropriate 3 bytes get written.
2. Only supported for case of 32-bit internal AMBA data bus.
3. Only supported for case of 64-bit internal AMBA data bus.

Even though misaligned non-burst transfers from internal masters are supported, the EBI naturally aligns the accesses when it sends them out to the external bus, splitting them into multiple aligned accesses if necessary. See *Section 12.5.2.11: Misaligned access support* for these cases.

Natural alignment for the EBI means:

- Byte access can have any address
- 16-bit access, address bit 31 must be 0
- 32-bit access, address bits 30–31 must be 0
- For burst accesses of any size, address bits 29–31 must be 0

The EBI never generates a misaligned external access. In the erroneous case that an externally-initiated misaligned access does occur, the EBI errors the access (by asserting TEA externally) and does not initiate the access on the internal bus.

The EBI requires that the portion of the data bus used for a transfer to/from a particular port size be fixed. A 32-bit port must reside on data bus bits 0–31,and a 16-bit port must reside on bits 0–15.

In the following figures and tables the following convention is adopted:

- The most significant byte of a 32-bit operand is OP0, and OP3 is the least significant byte.
- The two bytes of a 16-bit operand are OP0 (most significant) and OP1, or OP2 (most significant) and OP3, depending on the address of the access.
- The single byte of a byte-length operand is OP0, OP1, OP2, or OP3, depending on the address of the access.

This can be seen in *Figure 103*.

Figure 103. Internal operand representation

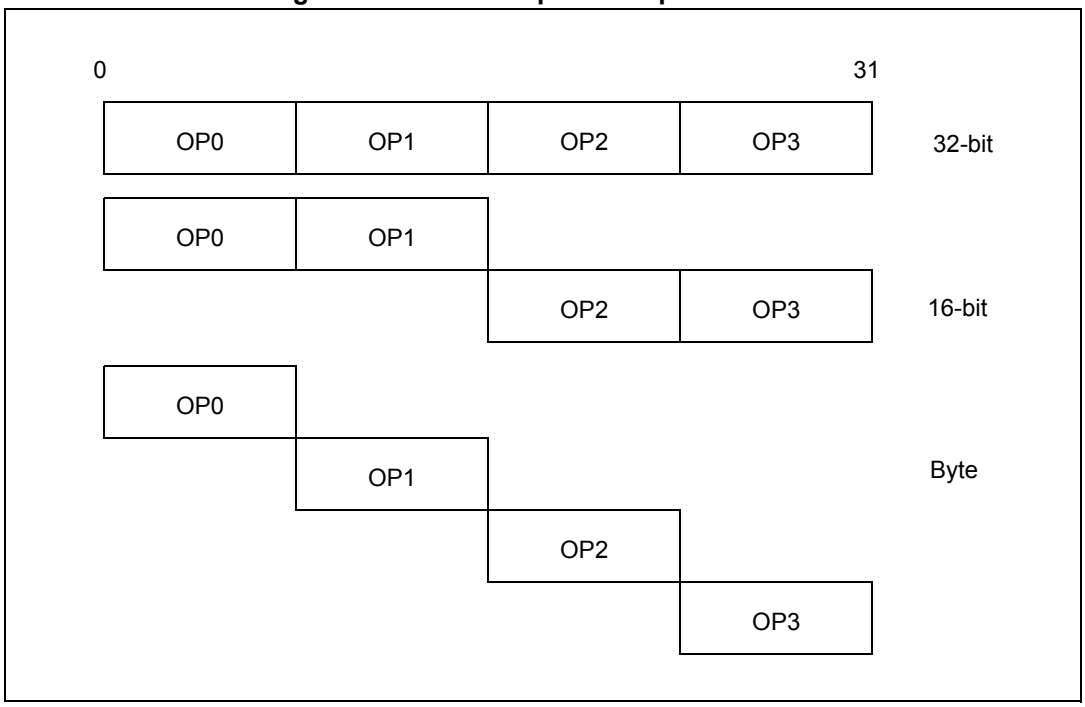


Figure 104 shows the device connections on the DATA[0:31] bus.

Figure 104. Interface to different port size devices

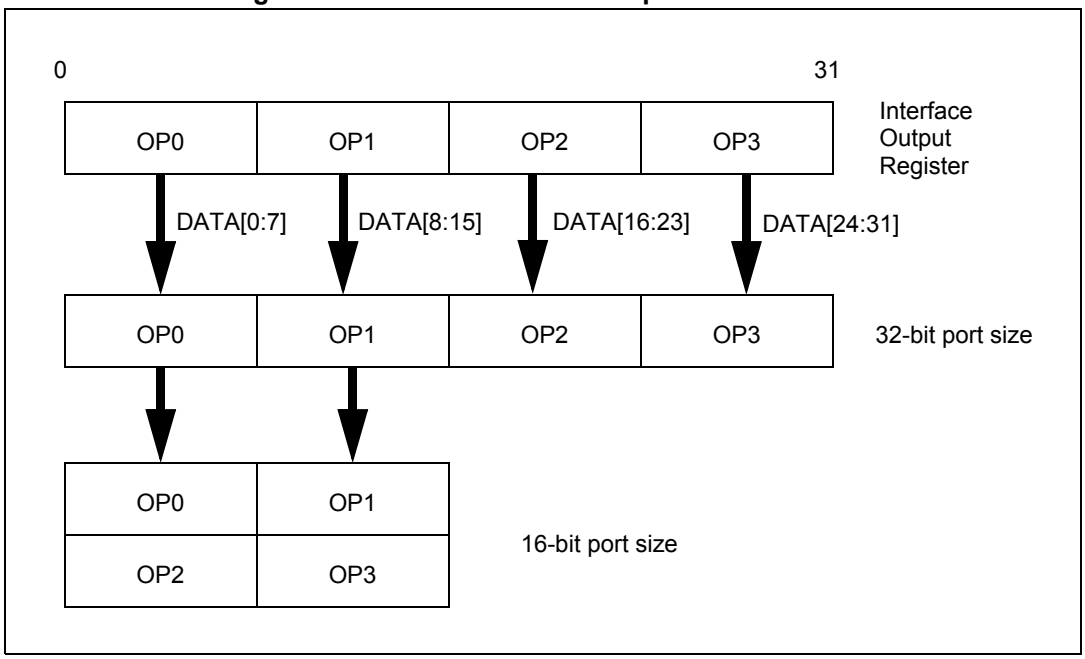


Table 99 lists the bytes required on the data bus for read cycles. The bytes indicated as '—' are not required during that read cycle.

Table 99. Data bus requirements for read cycles

Transfer size	TSIZ[0:1]	Address		32-bit port size				16-bit port size(1)	
		A30	A31	D0:D7	D8:D15	D16:D23	D24:D31	D0:D7(2)	D8:D15(3)
Byte	01	0	0	OP0	—	—	—	OP0	—
	01	0	1	—	OP1	—	—	—	OP1
	01	1	0	—	—	OP2	—	OP2	—
	01	1	1	—	—	—	OP3	—	OP3
16-bit	10	0	0	OP0	OP1	—	—	OP0	OP1
	10	1	0	—	—	OP2	OP3	OP2	OP3
32-bit	00	0	0	OP0	OP1	OP2	OP3	OP0/OP2 (4)	OP1/OP3

1. Also applies when DBM = 1 for 16-bit data bus mode.
2. For address/data muxed transfers, DATA[16:23] are used externally, not DATA[0:7].
3. For address/data muxed transfers, DATA[24:31] are used externally, not DATA[8:15].
4. This case consists of two 16-bit external transactions, the first fetching OP0 and OP1, the second fetching OP2 and OP3.

Table 100 lists the patterns of the data transfer for write cycles when accesses are initiated by the MCU. The bytes indicated as '—' are not driven during that write cycle.

Table 100. Data bus contents for write cycles

Transfer size	TSIZ[0:1]	Address		32-bit port size				16-bit port size(1)	
		A30	A31	D0:D7	D8:D15	D16:D23	D24:D31	D0:D7(2)	D8:D15(3)
Byte	01	0	0	OP0	—	—	—	OP0	—
	01	0	1	—	OP1	—	—	—	OP1
	01	1	0	—	—	OP2	—	OP2	—
	01	1	1	—	—	—	OP3	—	OP3
16-bit	10	0	0	OP0	OP1	—	—	OP0	OP1
	10	1	0	—	—	OP2	OP3	OP2	OP3
32-bit	00	0	0	OP0	OP1	OP2	OP3	OP0/OP2 (4)	OP1/OP3

1. Also applies when DBM = 1 for 16-bit data bus mode.
2. For address/data muxed transfers, DATA[16:23] are used externally, not DATA[0:7].
3. For address/data muxed transfers, DATA[24:31] are used externally, not DATA[8:15].
4. This case consists of two 16-bit external transactions, the first writing OP0 and OP1, the second writing OP2 and OP3.

12.5.2.8 Termination signals protocol

The termination signals protocol was defined in order to avoid electrical contention on lines that can be driven by various sources. In order to do that, a slave must not drive signals associated with the data transfer until the address phase is completed and it recognizes the address as its own. The slave must disconnect from signals immediately after it acknowledges the cycle and not later than the termination of the next address phase cycle.

For EBI-mastered non-chip-select accesses, the EBI requires assertion of $\overline{TA}$ from an external device to signal that the bus cycle is complete. The EBI uses a latched version of $\overline{TA}$ (1 cycle delayed) for these accesses to help make timing at high frequencies. This results in the EBI driving the address and control signals 1 cycle longer than required, as seen in *Figure 105*. However, the DATA does not need to be held 1 cycle longer by the slave, because the EBI latches DATA every cycle during non-chip-select accesses. During these accesses, the EBI does not drive the $\overline{TA}$ signal, leaving it up to an external device (or weak internal pullup) to drive $\overline{TA}$.

For EBI-mastered chip-select accesses, when the SETA bit is 0, the EBI drives $\overline{TA}$ the entire cycle, asserting according to internal wait state counters to terminate the cycle. When the SETA bit is 1, the EBI samples the $\overline{TA}$ for the entire cycle. During idle periods on the external bus, the EBI drives $\overline{TA}$ negated as long as it is granted the bus; when it no longer owns the bus, it lets go of $\overline{TA}$.

If no device responds by asserting $\overline{TA}$ within the programmed timeout period (BMT in EBI_BMCR) after the EBI initiates the bus cycle, the internal Bus Monitor (if enabled) asserts $\overline{TEA}$ to terminate the cycle. An external device may also drive $\overline{TEA}$ when it detects an error on an external transaction. $\overline{TEA}$ assertion causes the cycle to terminate and the processor to enter exception processing for the error condition. To properly control termination of a bus cycle for a bus error with external circuitry, $\overline{TEA}$ must be asserted at the same time or before (external) $\overline{TA}$ is asserted. $\overline{TEA}$ must be negated before the second rising edge after it was sampled asserted in order to avoid the detection of an error for the following bus cycle initiated. $\overline{TEA}$ is only driven by the EBI during the cycle where the EBI is asserting $\overline{TEA}$ and the cycle immediately following this assertion (for fast negation). During all other cycles, the EBI relies on a weak internal pullup to hold $\overline{TEA}$ negated. This allows an external device to assert $\overline{TEA}$ when it needs to indicate an error. External devices must follow the same protocol as the EBI, only driving $\overline{TEA}$ during the assertion cycle and 1 cycle afterwards for negation.

When $\overline{TEA}$ is asserted from an external source, the EBI uses a latched version of $\overline{TEA}$ (1 cycle delayed) to help make timing at high frequencies. This means that for any accesses where the EBI drives $\overline{TA}$ (chip-select accesses with SETA = 0), a $\overline{TEA}$ assertion that occurs 1 cycle before or during the last $\overline{TA}$ of the access could be ignored by the EBI, since it will have completed the access internally before it detects the latched $\overline{TEA}$ assertion. This means that non-burst chip-select accesses with no wait states (SCY = 0) cannot be reliably terminated by external $\overline{TEA}$. If external error termination is required for such a device, the EBI must be configured for SCY>=1.

Note: For the cases discussed above where $\overline{TEA}$ "could be ignored", this is not guaranteed. For some small access cases (which always use chip-select and internally-driven $\overline{TA}$), a $\overline{TEA}$ that occurs one cycle before or during the $\overline{TA}$ cycle or for SCY = 0 may in fact lead to terminating the cycle with error. However, proper error termination is not guaranteed for these cases, so $\overline{TEA}$ must always be asserted at least two cycles before an internally-driven $\overline{TA}$ cycle for proper error termination.

External $\overline{TEA}$ assertion that occurs during the same cycle that $\overline{TS}$ is asserted by the EBI is always treated as an error (terminating the access) regardless of SCY.

Table 101 summarizes how the EBI recognizes the termination signals provided from an external device.

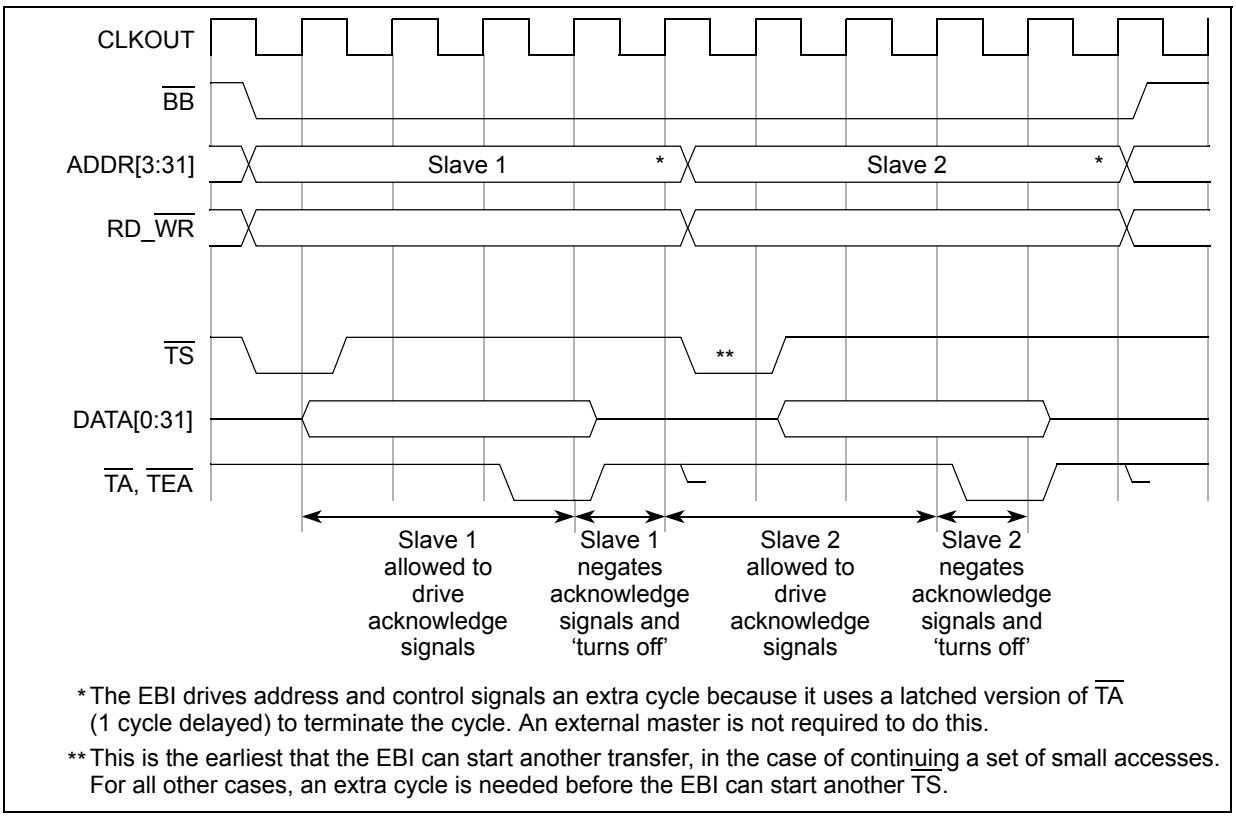
Table 101. Termination Signals Protocol

TEA(1)	TA(1)	Action
Negated	Negated	No Termination
Asserted	X	Transfer Error Termination
Negated	Asserted	Normal Transfer Termination

1. Latched version (1 cycle delayed) used for externally driven $\overline{TEA}$ and $\overline{TA}$.

Figure 105 shows an example of the termination signals protocol for back-to-back reads to two different slave devices who properly “take turns” driving the termination signals. This assumes a system using slave devices that drive termination signals.

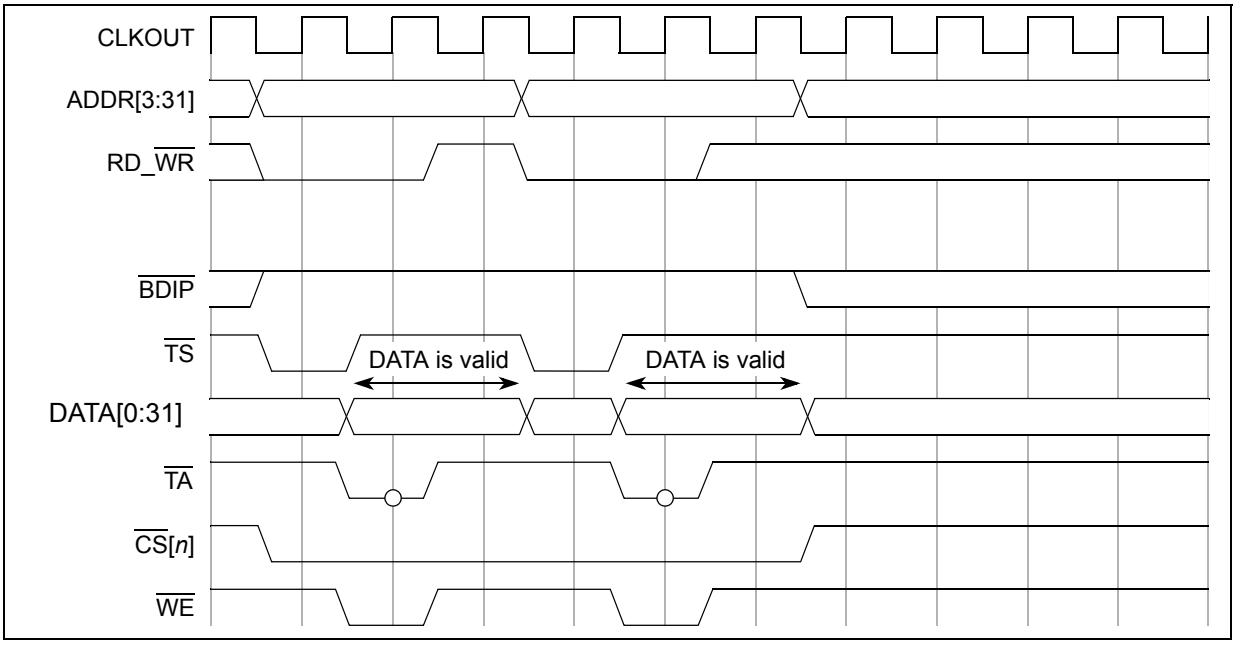
Figure 105. Termination signals protocol timing diagram



* The EBI drives address and control signals an extra cycle because it uses a latched version of $\overline{TA}$ (1 cycle delayed) to terminate the cycle. An external master is not required to do this.

** This is the earliest that the EBI can start another transfer, in the case of continuing a set of small accesses. For all other cases, an extra cycle is needed before the EBI can start another $\overline{TS}$.

Figure 106. External master back-to-back writes to same $\overline{CS}$ bank



12.5.2.9 Non-chip-select burst in 16-bit Data Bus Mode

The timing diagrams in this section apply only to the special case of a non-chip-select 32-bit access in 16-bit data bus mode (DBM = 1 in EBI_MCR).

For this case, a special 2-beat burst protocol is used for reads and writes, so that a slave device (using the same EBI) can internally generate one 32-bit read or write access (thus 32-bit coherent), as opposed to two separate 16-bit accesses. This behavior is independent of the width of the internal AMBA bus. So even with a 32-bit AMBA bus, 32-bit accesses are coherent (just as for default 64-bit AMBA bus).

Note: If the device does not support multi-master systems, the original intent of this protocol does not apply. However, this 2-beat burst protocol can also occur in a single-master system, if a non-chip-select 32-bit access to a 16-bit port is performed.

Figure 107 shows a 32-bit (and non-chip-select in a single-master system) read from an external master in 16-bit data bus mode.

Figure 108 shows a 32-bit (and non-chip-select in a single-master system) write from an external master in 16-bit data bus mode.

Figure 107. 32-bit read from MCU with DBM = 1

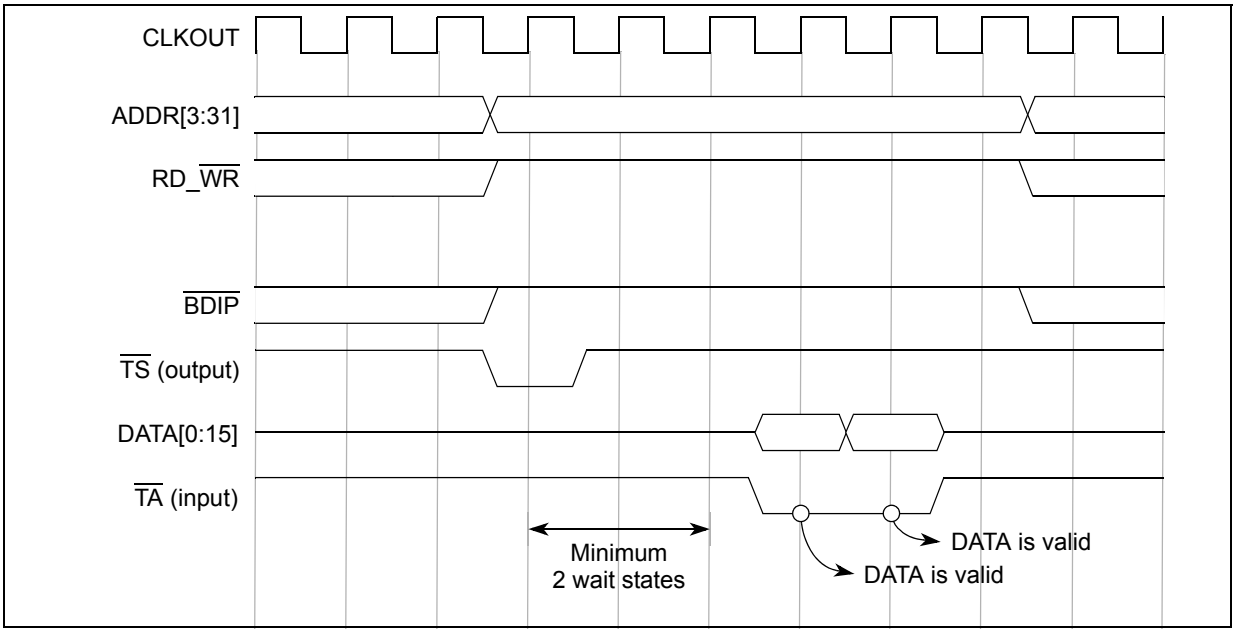
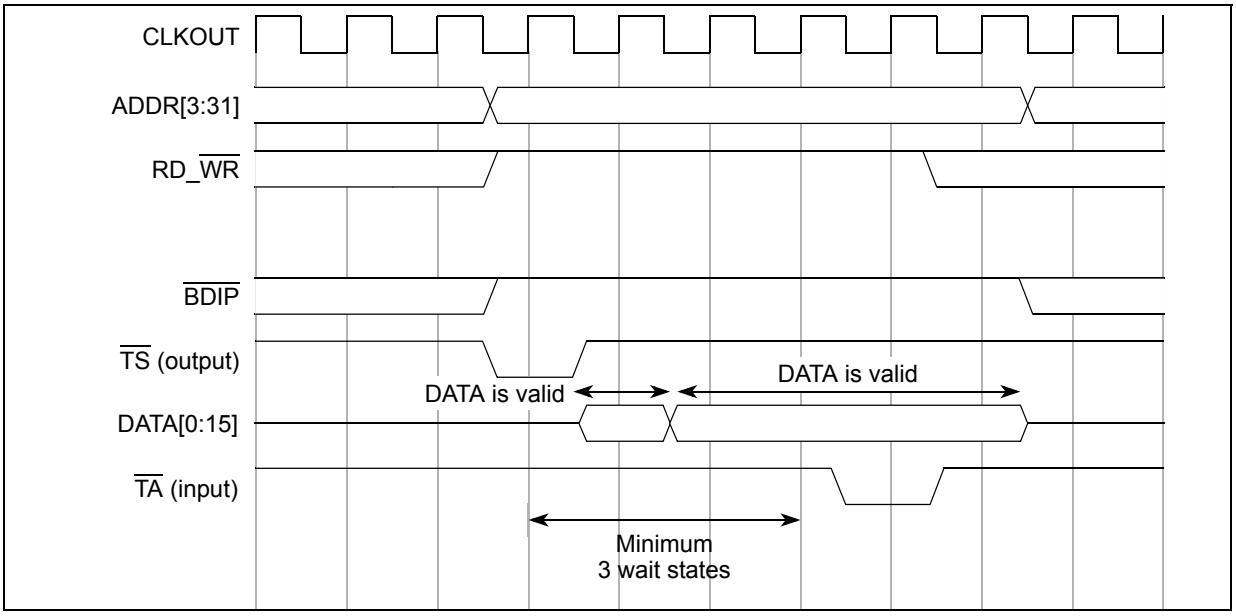


Figure 108. 32-bit write to MCU with DBM = 1



12.5.2.10 Calibration bus operation

Some devices with this EBI have a second external bus, intended for calibration use. This bus consists of a second set of the same signals present on the Primary external bus, except that arbitration, (and optionally other signals also) are excluded. Both buses can be supported with one EBI block, by using the calibration chip-selects to steer accesses to the calibration bus instead of the primary external bus.

Since the calibration bus has no arbitration signals, the arbitration on the primary bus controls accesses on the calibration bus as well, and no external master accesses can be

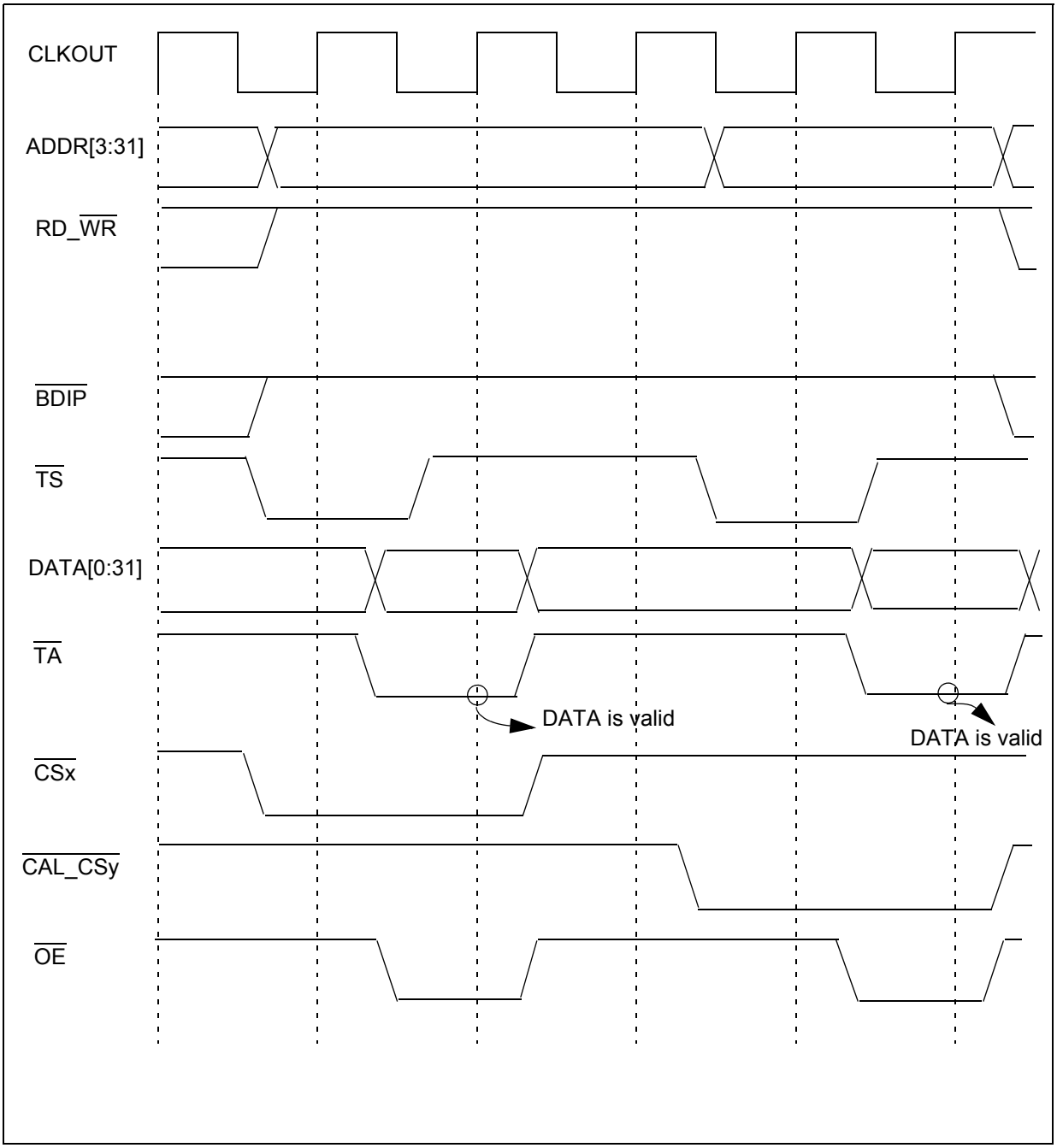
performed on the calibration bus. Accesses cannot be performed in parallel on both external buses. However, back-to-back accesses can switch from one bus to the other, as determined by the type of chip-select each address matches.

The timing diagrams and protocol for the calibration bus is identical to the primary bus, except that some signals are missing on the calibration bus.

There is an inherent dead cycle between a calibration chip-select access and a non-calibration access (chip-select or non-chip-select), just like the one between accesses to two different non-calibration chip-selects (described in *Section 12.5.2.4.3: Back-to-back accesses*).

Figure 109 shows an example of a non-calibration chip-select read access followed by a calibration chip-select read access. Note that this figure is identical to *Figure 88*, except the CSy is replaced by CAL_CSy. Timing for other cases on calibration bus can similarly be derived from other figures in this document (by replacing $\overline{CS}$ with $\overline{CAL_CS}$).

Figure 109. Back-to-back 32-bit reads to $\overline{CS}$, $\overline{CAL_CS}$ banks



12.5.2.11 Misaligned access support

This section describes all the misaligned cases supported by the EBI. These cases are a subset of the full set of cases allowed by the AMBA AHB V6 specification. The EBI works under the assumption that all internal masters on the device do not produce any misaligned access cases (to the EBI) other than the ones below.

12.5.2.11.1 Misaligned access support (64-bit AMBA)

Table 102 shows all the misaligned access cases supported by the EBI (using a 64-bit AMBA implementation), as seen on the internal master AMBA bus. All other misaligned cases are not supported. If an unsupported misaligned access to the EBI is attempted (such as non-chip-select or burst misaligned access), the EBI errors the access on the internal bus and does not start the access (nor assert $\overline{TEA}$) externally.

Table 102. Misalignment cases supported by a 64-bit AMBA EBI (internal bus)

No. (1)	Program size and byte offset	Address [29:31](2)	Data bus byte strobes(3)	HSIZE(4)	HUNALIGN(5)
1	Half @0x1,0x9	001	0110_0000	10	1
2	Half @0x3,0xB	011	0001_1000	11	1
3	Half @0x5,0xD	101	0000_0110	10	1
4 -	Half @0x7, 0xF (2 AHB transfers)	111 000	0000_0001 1000_0000	01(6) 00	1 0
5	Word @0x1,0x9	001	0111_1000	11	1
6	Word @0x2,0xA	010	0011_1100	11	1
7	Word @0x3,0xB	011	0001_1110	11	1
8 -	Word @0x5,0xD (2 AHB transfers)	101 000	0000_0111 1000_0000	10 00	1 0
9 -	Word @0x6, 0xE (2 AHB transfers)	110 000	0000_0011 1100_0000	10(7) 01	1 0
10 11	Word @0x7,0xF (2 AHB transfers)	111 000	0000_0001 1110_0000	10(6) 10	1 1
12 -	Doubleword @0x4,0x8 (2 AHB transfers)	100 000	0000_1111 1111_0000	11(8) 10	1 0
13 -	Doubleword @0x2,0xA (2 AHB transfers)	010 000	0011_1111 1100_0000	11 01	1 0
14 15	Doubleword 0x6,0xE (2 AHB transfers)	110 000	0000_0011 1111_1100	11(7) 11	1 1

1. Misaligned case number. Only transfers where HUNALIGN = 1 are numbered as misaligned cases.
2. Address on internal master AHB bus, not necessarily address on external ADDR pins.
3. Internal byte strobe signals on AHB bus. Shown with Big-Endian byte ordering in this table, even though internal master AHB bus uses Little-Endian byte-ordering (EBI flips order internally).
4. Internal signal on AHB bus; 00 = 8 bits, 01 = 16 bits, 10 = 32 bits, 11 = 64 bits. HSIZE is driven according to the smallest aligned container that contains all the requested bytes. This results in extra EBI external transfers in some cases.
5. Internal signal on AHB bus that indicates that this transfer is misaligned (when 1).
6. For this case, the EBI internally treats HSIZE as 00 (1-byte access).
7. For this case, the EBI internally treats HSIZE as 01 (2-byte access).
8. For this case, the EBI internally treats HSIZE as 10 (4-byte access).

Table 103 shows which external transfers are generated by the EBI for the misaligned access cases in *Table 102*, for each port size.

The number of external transfers for each internal AHB master request is determined by the HSIZE value for that request relative to the port size. For example, a half-word write to @011 (misaligned case #2) with 16-bit port size results in four external 16-bit transfers because the HSIZE is 64-bits. For cases where two or more external transfers are required for one internal transfer request, these external accesses are considered part of a “small access” set, as described in *Section 12.5.2.6: Small accesses (small port size and short burst length)*.

Since all transfers are aligned on the external bus, normal timing diagrams and protocol apply. Note that the TSIZ[0:1] signals are not intended to be used for misaligned accesses, so they are not specified in *Table 103*.

Table 103. Misalignment cases supported by a 64-bit AMBA EBI (external bus)

No.(1)	PS(2)	Program size and byte offset	ADDR[29:31](3)	$\overline{\text{WE_BE}}$[0:3](4)
1	0	Half @0x1,0x9	000	1001
	1		000 010	1011 0111
2	0	Half @0x3,0xB	000 100	1110 0111
	1		010 100	1011 0111
3	0	Half @0x5,0xD	100	1001
	1		100 110	1011 0111
4	0	Half @0x7,0xF (2 AHB transfers)	111(5)	1110
-			000	0111
4	1		110	1011
-			000	0111
5	0	Word @0x1,0x9	000 100	1000 0111
	1		000 010 100	1011 0011 0111
6	0	Word @0x2,0xA	000 100	1100 0011
	1		010 100	0011 0011
7	0	Word @0x3,0xB	000 100	1110 0001
	1		010 100 110	1011 0011 0111

Table 103. Misalignment cases supported by a 64-bit AMBA EBI (external bus) (continued)

No.[1]	PS[2]	Program size and byte offset	ADDR[29:31][3]	$\overline{\text{WE_BE}}$[0:3][4]
8	0	Word @0x5,0xD (2 AHB transfers)	100	1000
-			000	0111
8	1		100 110	1011 0011
-			000	0111
9	0	Word @0x6,0xE (2 AHB transfers)	110[6]	1100
-			000	0011
9	1		110[6]	0011
-			000	0011
10	0	Word @0x7,0xF (2 AHB transfers)	111[5]	1110
11			000	0001
10	1		111[5]	1011
11			000 010	0011 0111
12	0	Doubleword @0x4,0xC (2 AHB transfers)	100[7]	0000
-			000	0000
12	1		100[7] 110	0011 0011
-			000 010	0011 0011
13	0	Doubleword @0x2,0xA (2 AHB transfers)	000 100	1100 0000
-			000	0011
13	1		010 100 110	0011 0011 0011
-			000	0011
14	0	Doubleword @0x6,0xE (2 AHB transfers)	110[6]	1100
15			000 100	0000 0011
14	1		110[6]	0011
15			000 010 100	0011 0011 0011

1. Misaligned case number, from *Table 102*.
2. Port size; 0 = 32 bits, 1 = 16 bits.
3. External ADDR pins, not necessarily the address on internal master AHB bus.
4. External $\overline{\text{WE_BE}}$ pins. Note that these pins have negative polarity, opposite of the internal byte strobes in *Table 102*.

5. Treated as 1-byte access.
6. Treated as 2-byte access.
7. Treated as 4-byte access.

12.5.2.11.2 Misaligned access support (32 bit AMBA)

Table 104 shows all the misaligned access cases supported by the EBI (using a 32-bit AMBA implementation), as seen on the internal master AMBA bus. All other misaligned cases are not supported. If an unsupported misaligned access to the EBI is attempted (such as non-chip-select or burst misaligned access), the EBI errors the access on the internal bus and does not start the access (nor assert TEA) externally.

Table 104. Misalignment cases supported by a 32-bit AMBA EBI (internal bus)

No. (1)	Program size and byte offset	Address [30:31] (2),(3)	Data bus byte strobes (4)	HSIZE (5)	HUNALIGN (6)
1	Half @0x1	01	0110	10	1
4 -	Half @0x3 (2 AHB transfers)	11 z00	0001 1000	01 (7) 00	1 0
8	Word @0x1 (2 AHB transfers)	01	0111 1000	10 00	1 0
9	Word @0x2 (2 AHB transfers)	10	0011 1100	10 01	1 0
10 11	Word @0x3 (2 AHB transfers)	11 z00	0001 1110	10 (7) 10	1 1

1. Misaligned case number. Only transfers where HUNALIGN = 1 are numbered as misaligned cases. The missing case numbers cannot occur on a 32-bit AMBA implementation.
2. Address on internal master AHB bus, not necessarily address on external ADDR pins.
3. Address Z is incremented by one 32 bit word compared to previous access on the AMBA bus.
4. Internal byte strobe signals on AHB bus. Shown with Big-Endian byte ordering in this table, even though internal master AHB bus uses Little-Endian byte-ordering (EBI flips order internally).
5. Internal signal on AHB bus; 00 = 8 bits, 01 = 16 bits, 10 = 32 bits. HSIZE is driven according to the smallest aligned container that contains all the requested bytes. This results in extra EBI external transfers in some cases.
6. Internal signal on AHB bus that indicates that this transfer is misaligned (when 1).
7. For this case, the EBI internally treats HSIZE as 00 (1-byte access).

Table 105 shows which external transfers are generated by the EBI for the misaligned access cases in *Table 104*, for each port size.

The number of external transfers for each internal AHB master request is determined by the HSIZE value for that request relative to the port size. For example, a half-word write to @0x3 (misaligned case #4) with 16-bit port size results in 4 external 16-bit transfers because the transfer granularity of 32 bits. For cases where two or more external transfers are required for one internal transfer request, these external accesses are considered part of a “small access” set, as described in *Section 12.5.2.6: Small accesses (small port size and short burst length)*.

Since all transfers are aligned on the external bus, normal timing diagrams and protocol apply. Note that the TSIZ[0:1] signals are not intended to be used for misaligned accesses, so they are not specified in *Table 105*.

Table 105. Misalignment cases supported by a 32-bit AMBA EBI (external bus)

No. (1)	PS(2)	Program size and byte offset	ADDR[30:31](3),(4)	$\overline{\text{WE_BE}}$[0:3](5)
1	0	Half @0x1	00	1001
	1		00 10	1011 0111
4	0	Half @0x3 (2 AHB transfers)	11(6)	1110
-			z00	0111
4	1		11(6)	1011
-			z00	0111
8	0	Word @0x1 (2 AHB transfers)	00	1000
-			z00	0111
8	1		00 10	1011 0011
-			z00	0111
9	0	Word @0x2 (2 AHB transfers)	00	1100
-			z00	0011
9	1		10	0011
-			z00	0011
10	0	Word @0x3 (2 AHB transfers)	11(6)	1110
11			z00	0001
10	1		11(6)	1011
11			z00 z10	0011 0111

1. Misaligned case number, from *Table 104*.
2. Port size; 0 = 32 bits, 1 = 16 bits.
3. External ADDR pins, not necessarily the address on internal master AHB bus.
4. For address with Z - address bit 29 will increment to next word. For all other cases, address bit 29 will be unchanged.
5. External $\overline{\text{WE_BE}}$ pins. Note that these pins have negative polarity, opposite of the internal byte strobes in *Table 104*.
6. Treated as 1-byte access.

12.5.2.12 Address data multiplexing

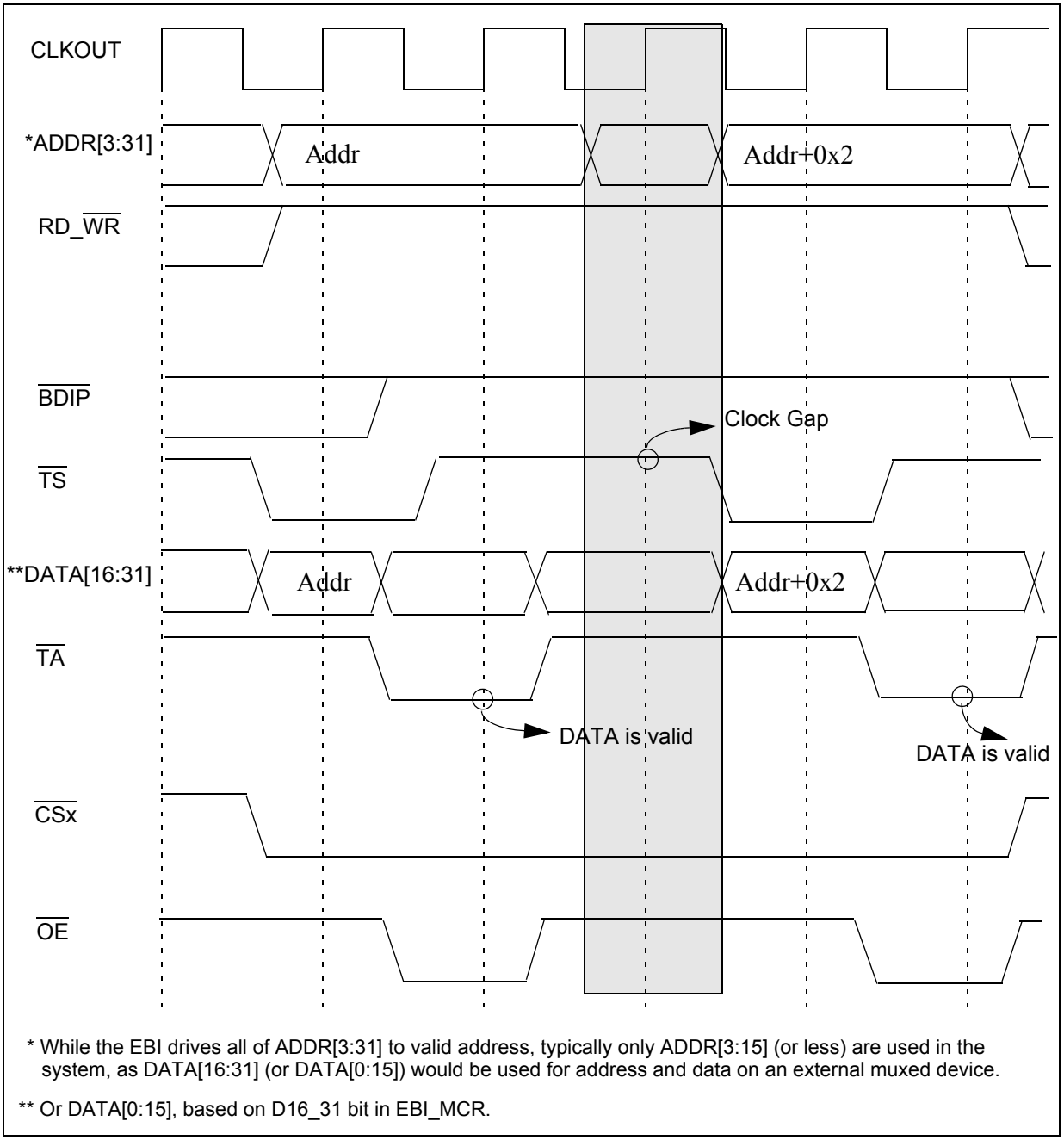
Address/Data multiplexing enables the design of a system with reduced pin count. In such a system, multiplexed address/data functions (on DATA pins) are used, instead of having separate address and data pins. Compared to the normal EBI specification (e.g. 24 address pins + 32 data pins), only 32 data pins are required. Compared to a 16-bit bus implementation, only 24 pins are required (e.g. ADDR[8:15] + ADDR[16:31]/DATA[16:31]).

When performing a small access read, as described in *Section 12.5.2.6: Small accesses (small port size and short burst length)*, with A/D multiplexing enabled for this access, the EBI inserts an idle clock cycle with $\overline{\text{OE}}$ negated and $\overline{\text{CS}}$ asserted, to allow for the memory to three-state the bus prior to the EBI driving the address on the next clock. This clock gap

already exists (for other reasons) for non-small-access transfers, so no additional clock gap is inserted for those cases. See *Figure 110* for an example of a small access read with A/D multiplexing enabled.

In general, timing diagrams in A/D multiplexing mode are very similar to other diagrams in this document (including support for Burst accesses), except for the behavior of the ADDR and DATA buses, which can be seen in *Figure 110*.

Figure 110. Small access (32-bit read to 16-bit port) on address/data multiplexed bus



* While the EBI drives all of ADDR[3:31] to valid address, typically only ADDR[3:15] (or less) are used in the system, as DATA[16:31] (or DATA[0:15]) would be used for address and data on an external muxed device.

** Or DATA[0:15], based on D16_31 bit in EBI_MCR.

12.6 Initialization/Application information

12.6.1 Booting from external memory

The EBI block does not support booting directly to external memory (i.e. fetching the first instruction after reset externally). One common method for an MCU to resemble an external boot with this EBI is to use an internal Boot Assist Module on the MCU, which fetches the first instruction internally and configures EBI registers before branching to an external address to “boot” externally.

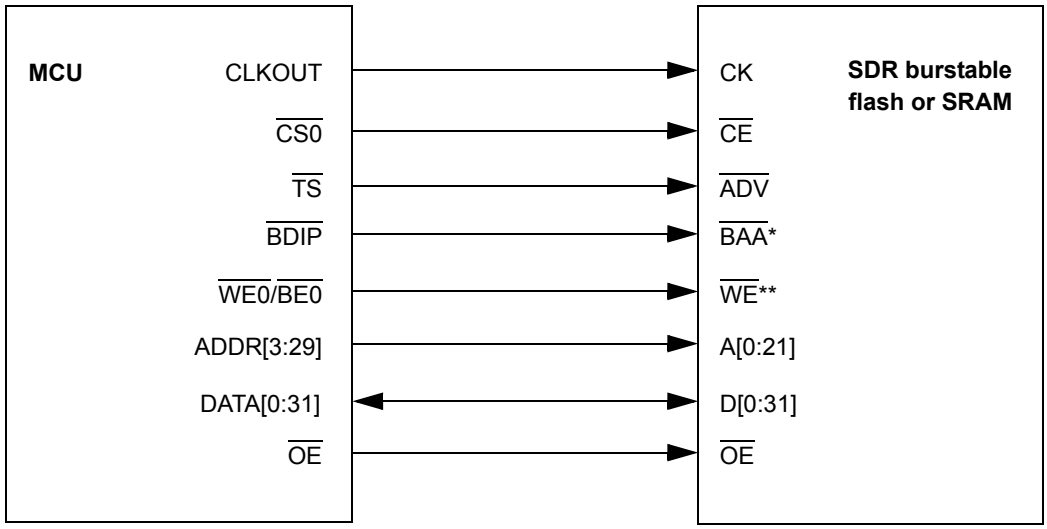
If code in external memory needs to write EBI registers, this must be done in a way that avoids modifying EBI registers while external accesses are being performed, such as the following method:

- Copy the code that is doing the register writes (plus a return branch) to internal SRAM
- Branch to internal SRAM to run this code, ending with a branch back to external flash

12.6.2 Running with SDR (single data rate) burst memories

This includes flash and SRAM memories with a compatible burst interface. $\overline{\text{BDIP}}$ is required only for some SDR memories. *Figure 111* shows a block diagram of an MCU connected to a 32-bit SDR burst memory.

Figure 111. MCU connected to SDR burst memory



* May or may not be connected, depending on the memory used

** Flash memories typically use one $\overline{\text{WE}}$ signal as shown; RAMs use two or four (16-bit or 32-bit).

Refer to *Figure 94* for an example of the timing of a typical Burst Read operation to an SDR burst memory. Refer to *Figure 83* for an example of the timing of a typical Single Write operation to SDR memory.

12.6.3 Running with asynchronous memories

The EBI also supports asychronous memories. In this case, the CLKOUT, $\overline{TS}$, and $\overline{BDIP}$ pins are not used by the memory and bursting is not supported. However, the EBI still drives these outputs, and always drives and latches all signals at posedge CLKOUT (that is, there is no “asynchronous mode” for the EBI). The data timing is controlled by setting the SCY bits in the appropriate Option Register to the proper number of wait states to work with the access time of the asynchronous memory, just as done for a synchronous memory.

12.6.3.1 Example wait state calculation

This example applies to any chip-select memory, synchronous or asynchronous.

As an example, say we have a memory with 50 ns access time, and we are running the external bus @ 66 MHz (CLKOUT period: 15.2 ns). Assume the input data specification for the MCU is 4 ns.

Number of wait states = (access time) / (CLKOUT period) + (0 or 1) (depending on setup time)

50/15.2 = 3 with 4.4 ns remaining (so we need at least three wait states, now check setup time)

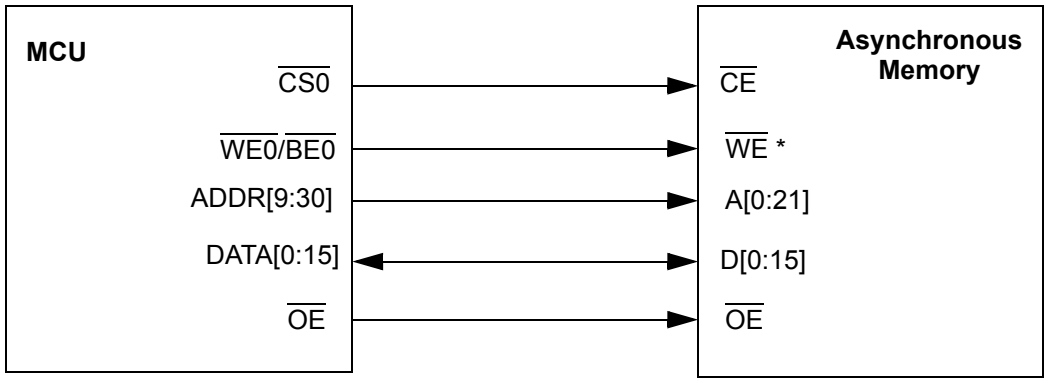
15.2 – 4.4 = 10.8 ns (this is the achieved input data setup time)

Since actual input setup (10.8 ns) is greater than the input setup specification (4.0 ns), three wait states is sufficient. If the actual input setup was less than 4.0 ns, we would have to use four wait states instead.

12.6.3.2 Timing and connections for asynchronous memories

The connections to an asynchronous memory are the same as for a synchronous memory, except that the CLKOUT, $\overline{TS}$, and $\overline{BDIP}$ signals are not used. *Figure 112* shows a block diagram of an MCU connected to an asynchronous memory.

Figure 112. MCU connected to asynchronous memory



* Flash memories typically use one $\overline{WE}$ signal as shown; RAMs use two or four (16-bit or 32-bit)

Figure 113 shows a timing diagram of a read operation to a 16-bit asynchronous memory using three wait states.

Figure 114 shows a timing diagram of a write operation to a 16-bit asynchronous memory using three wait states.

Figure 113. Read operation to asynchronous memory, three initial wait states

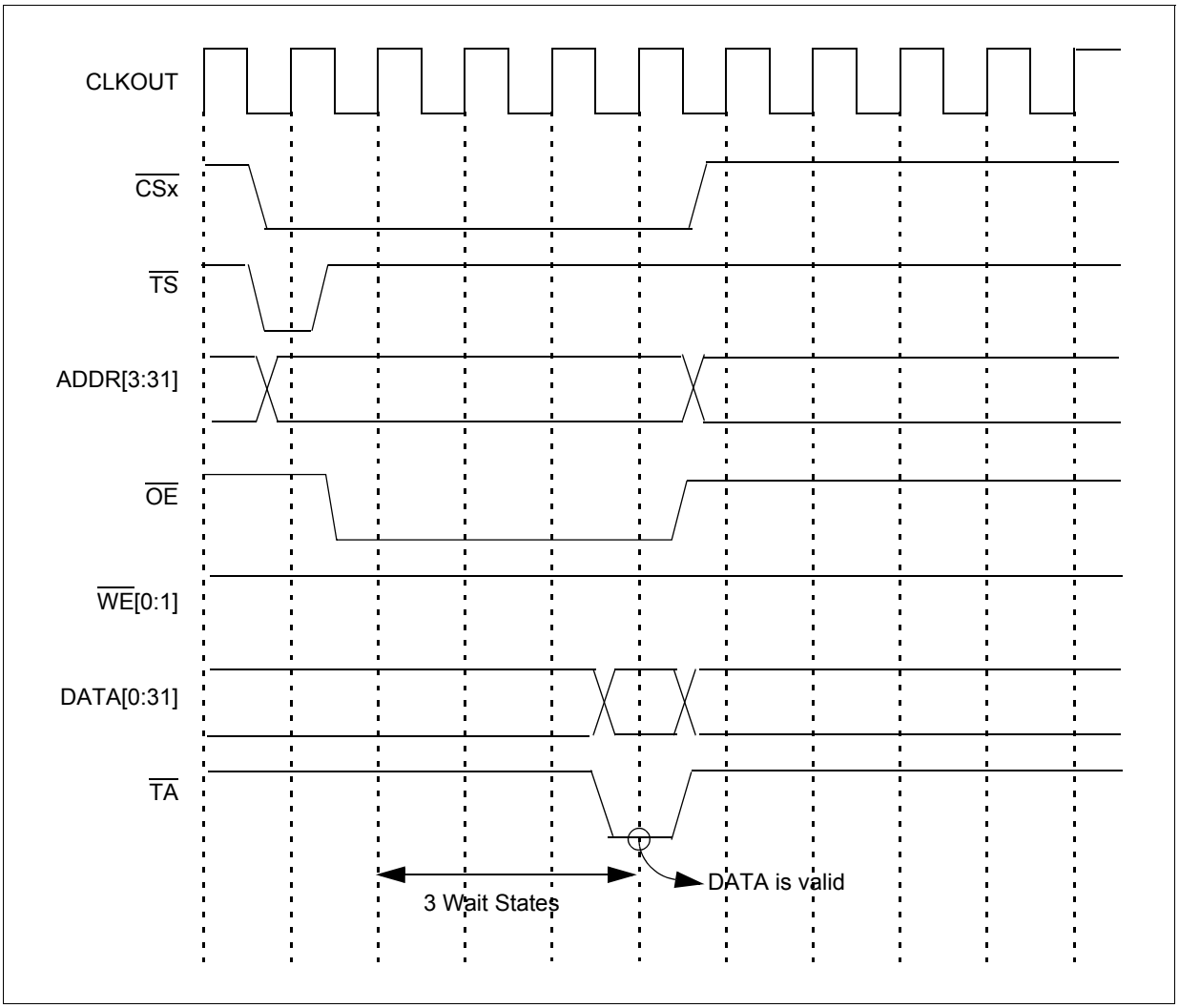
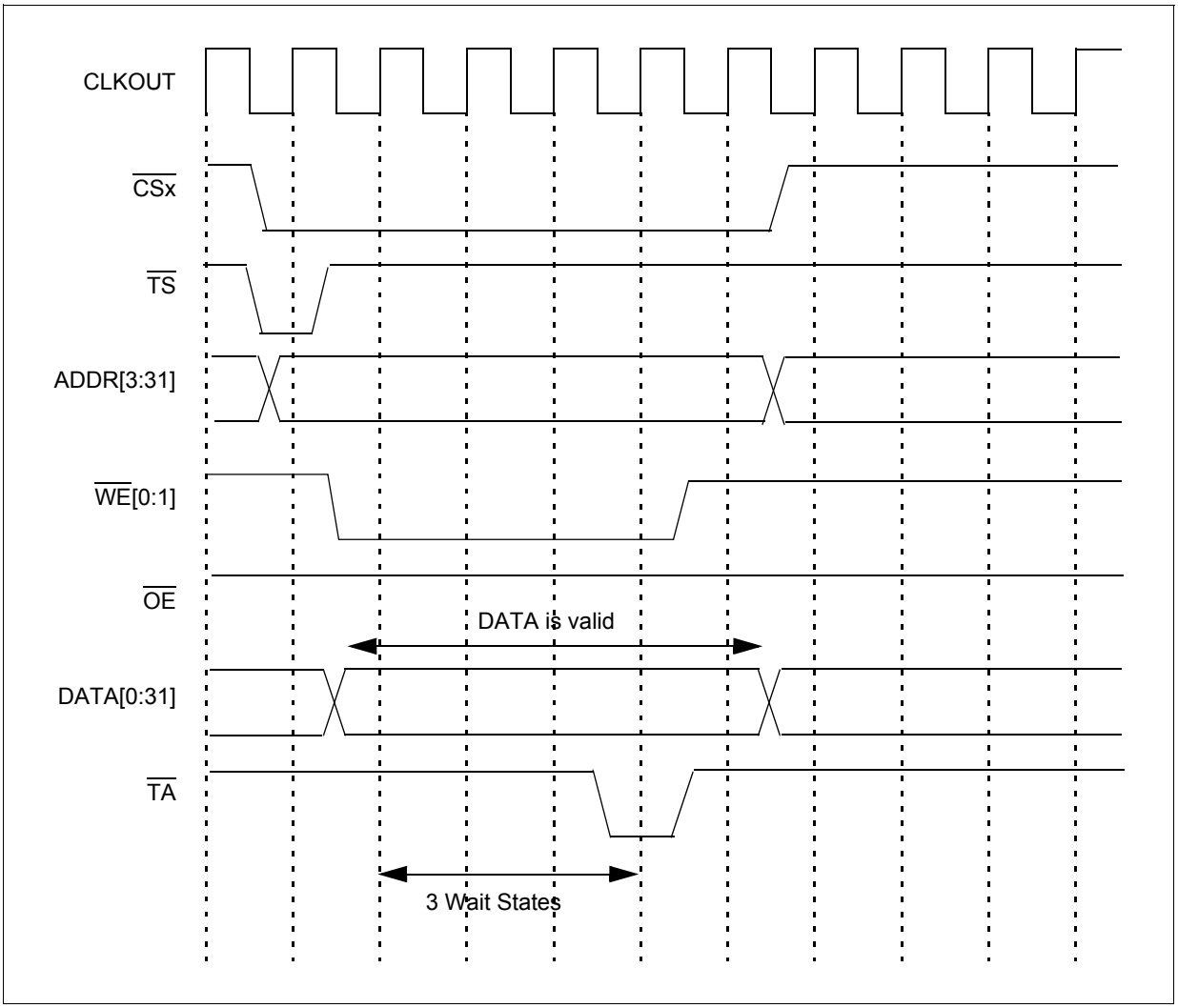


Figure 114. Write operation to asynchronous memory, three initial wait states

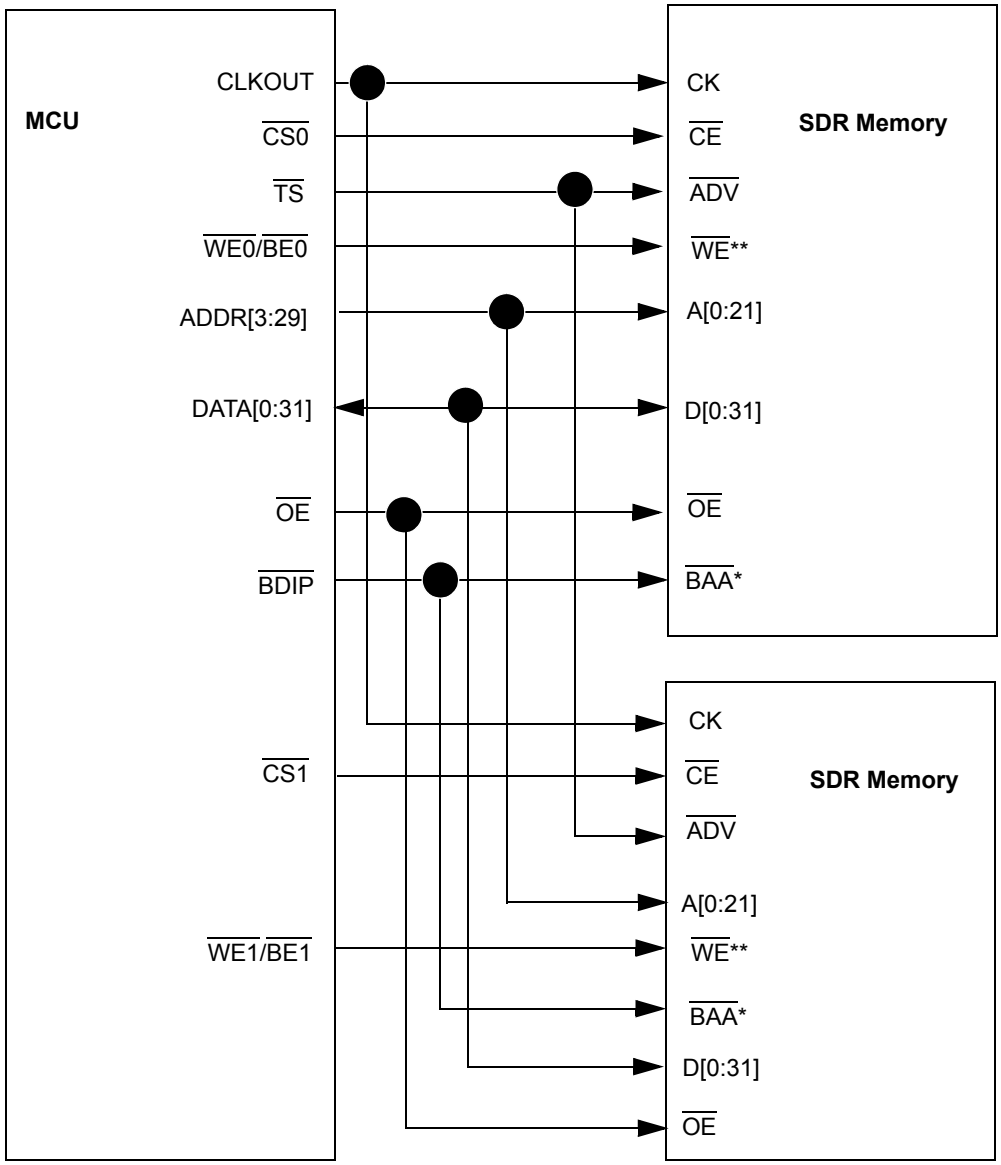


12.6.4 Connecting an MCU to multiple memories

The MCU can be connected to more than one memory at a time.

Figure 115 shows an example of how two memories could be connected to one MCU.

Figure 115. MCU connected to multiple memories



* May or may not be connected, depending on the memory used.

** Flash memories typically use one WE signal as shown, RAMs use two or four (16-bit or 32-bit)

12.6.5 EBI operation with reduced pinout MCUs

Some MCUs with this EBI may not have all the pins described in this document pinned out for a particular package. Some of the most common pins to be removed are DATA[16:31], arbitration pins (BB, BG, BR), and TSIZ[0:1]. This section describes how to configure dual-MCU systems for each of those scenarios, as well as describing limitations to EBI operation when other pins are missing (TA, TEA, BDIP). More than one section may apply if the applicable pins are not present on one or both MCUs.

12.6.5.1 Connecting 16-bit MCU to 32-bit MCU (master/master or master/slave)

This scenario is straightforward. Simply connect DATA[0:15] between both MCUs, and configure both for 16-bit Data Bus Mode operation (DBM = 1 in EBI_MCR). Note that 32-bit external memories are not supported in this scenario.

12.6.5.2 Arbitration with no arbitration pins (master/slave only)

To implement a master/slave system with an MCU that has no arbitration pins ($\overline{BB}$, $\overline{BG}$, $\overline{BR}$), the user must configure the master MCU for internal arbitration (EARB = 0 in EBI_MCR) and the slave MCU for external arbitration (EARB = 1). Internally on an MCU with no arbitration pins, the $\overline{BR}$, $\overline{BG}$, and $\overline{BB}$ signals to the EBI will be tied negated. This means that the slave MCU will never receive bus grant asserted, so it will never attempt to start an access on the external bus. The master MCU will never receive bus request or bus busy asserted, so it will maintain ownership of the bus without any arbitration delays. In the erroneous case that the slave MCU executes internal code that attempts to access external address space, that access will never get external and will eventually timeout in the slave MCU.

12.6.5.3 Transfer size with no TSIZ Pins (master/master or master/slave)

Since there are no TSIZ pins to communicate transfer size from master MCU to slave MCU, the internal SIZE field of the EBI_MCR must be used on the slave MCU (by setting SIZEEN = 1 in slave's EBI_MCR). Anytime the master MCU needs to read or write the slave MCU with a different transfer size than the current value of the slave's SIZE field, the master MCU must first write the slave's SIZE field with the correct size for the subsequent transaction.

12.6.5.4 No Transfer Acknowledge ($\overline{TA}$) pin

If an MCU has no $\overline{TA}$ pin available, this restricts the MCU to chip-select accesses only (no MCU->MCU transfers are possible). Non-chip-select accesses have no way for the EBI to know which cycle to latch the data. The EBI has no built-in protection to prevent non-chip-select accesses in this scenario; it is up to the user to make certain they set up chip-selects and external memories correctly to ensure all external accesses fall in a valid chip-select region.

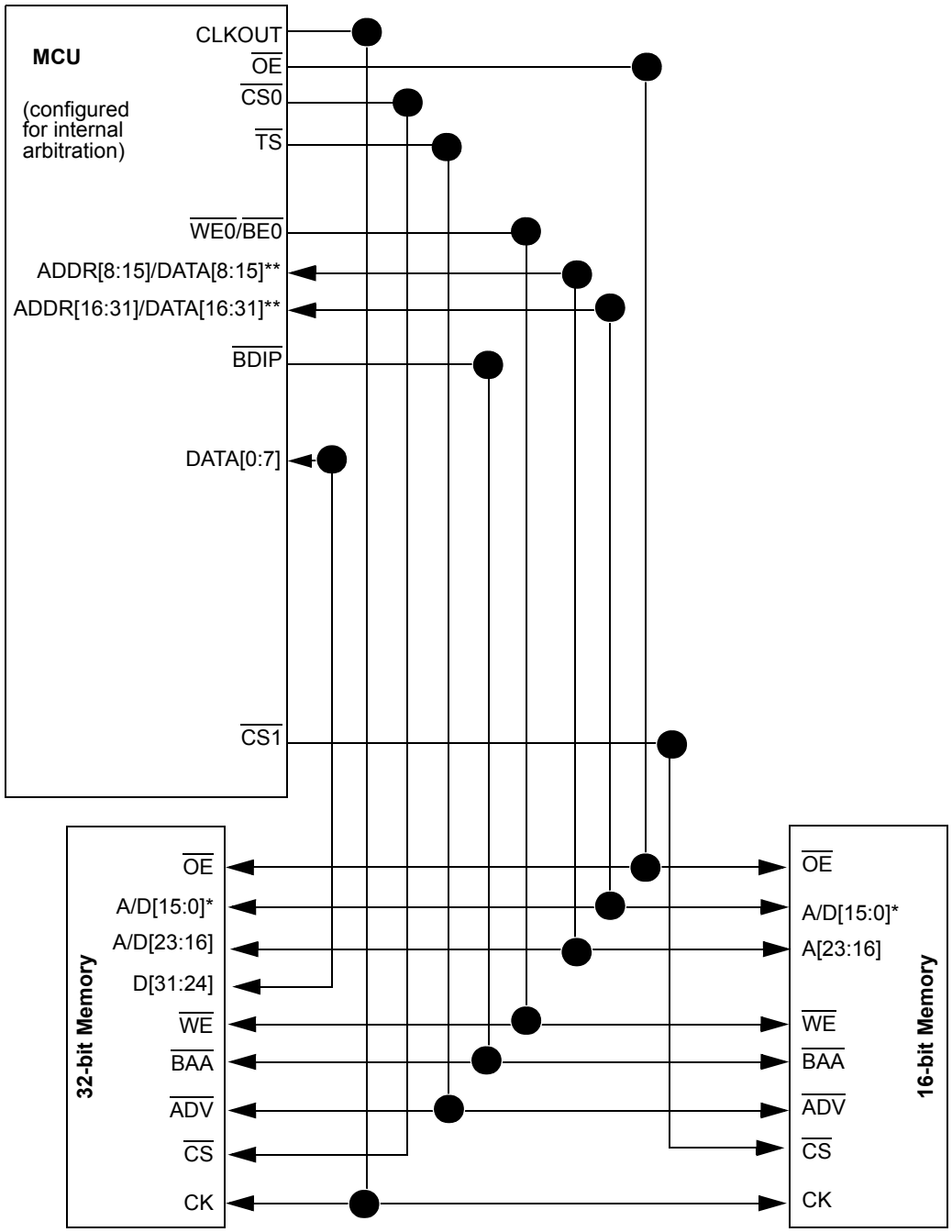
12.6.5.5 No Transfer Error ($\overline{TEA}$) pin

If an MCU has no $\overline{TEA}$ pin available, this eliminates the feature of terminating an access with $\overline{TEA}$. This means if an access times out in the EBI bus monitor, the EBI (master) will still terminate the access early, but there will be no external visibility of this termination, so the slave device might end up driving data much later, when a subsequent access is already underway. Therefore, the EBI bus monitor should be disabled when no $\overline{TEA}$ pin exists.

12.6.5.6 No Burst Data In Progress ($\overline{BDIP}$) pin

If an MCU has no $\overline{BDIP}$ pin available, this eliminates burst support only if the burstable memory being used requires $\overline{BDIP}$ to burst. Many external memories use a self-timed configurable burst mechanism that does not require a dynamic burst indicator. Check the applicable external memory specification to see if $\overline{BDIP}$ is required in your system.

Figure 116. Address/data multiplexing with both 16-bit and 32-bit memories in single master system



* Most memories have the LSB signified by bit 0, this must match the LSB of the MCU, signified by bit 31. If the memory is word addressable, other bits of the data bus can be used. For example, a 32-bit Flash might need A[2] as the LSB, where A/D1 & A/D0 are not used for address decoding.

** These refer to the DATA pins of the MCU, which are used for both address and data functions in this system.

13 Interrupt Controller (INTC)

13.0.1 Device-specific features

- 191 peripheral interrupt request sources
- 165 reserved positions
- 8 software interrupts
- 4-byte offset between vectors
- INTC base address: 0xFFF4_8000

13.0.2 Device-specific register information

- 53 Priority Select Registers for a total of 364 interrupt vectors
- 112 reserved interrupt vectors

Note: This is a single-core device. All registers and bits in this chapter pertaining to multi-core operation do not apply to the SPC563Mxx family of devices.

13.1 Introduction

13.1.1 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 is optimized for a large number of interrupt requests, up to 512. 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.

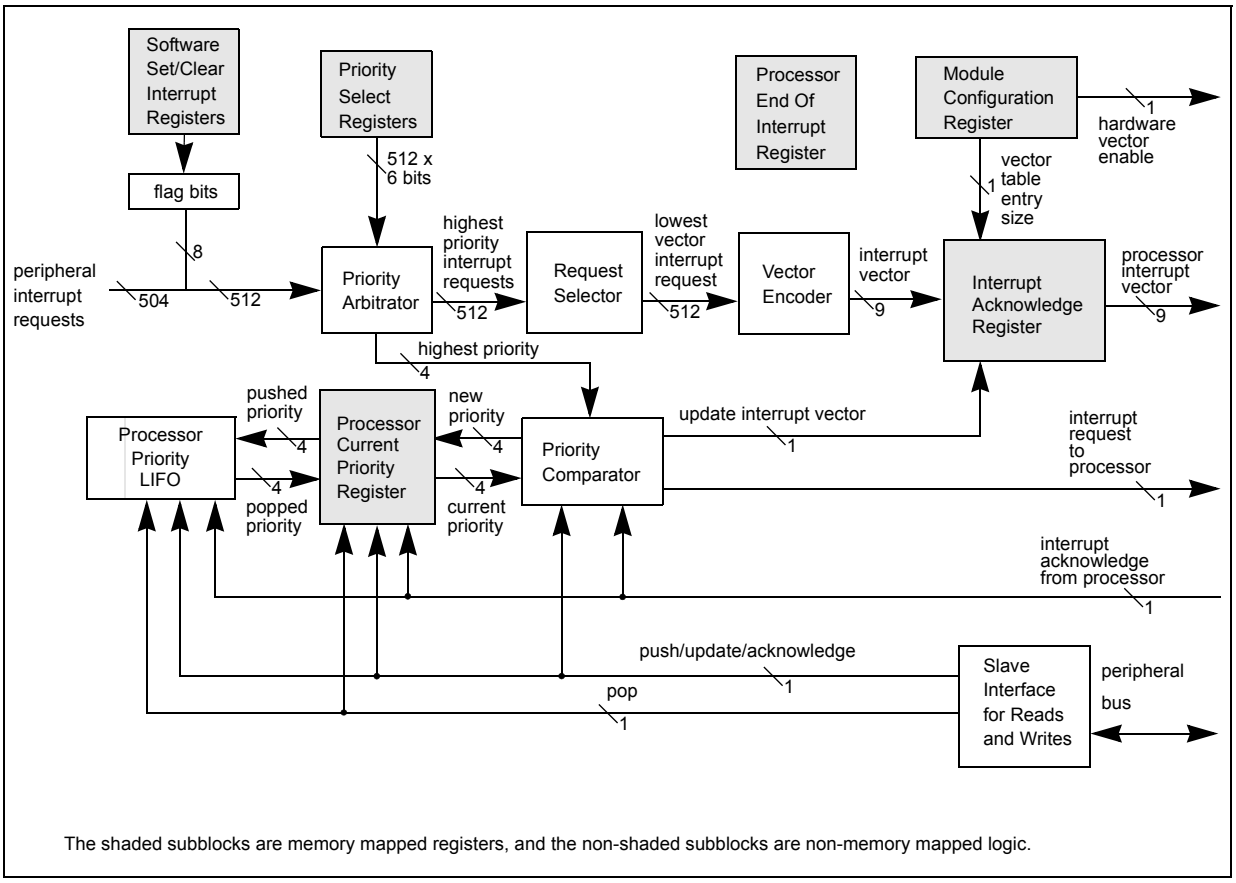
When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the Priority Ceiling Protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks which share the resource can not preempt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

Multiple processors can assert interrupt requests to each other through software setable interrupt requests. These same software setable 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 setable interrupt request to finish the servicing in a lower priority ISR. Therefore these software setable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

13.1.2 Block diagram

Figure 117 is a block diagram of the INTC.

Figure 117. INTC block diagram



The shaded subblocks are memory mapped registers, and the non-shaded subblocks are non-memory mapped logic.

13.1.3 Features

- Parameterizable up to 504 peripheral interrupt request sources.
- 8 software setable interrupt request sources
- 9-bit vector
  - Unique vector for each interrupt request source
  - Hardware connection to processor or read from register
- Each interrupt source programmable 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

13.2 Modes of operation

13.2.1 Normal mode

In normal mode, the INTC has two handshaking modes with the processor: software vector mode and hardware vector mode.

13.2.1.1 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 HVEN or HVEN_PRC1 bit in the INTC_MCR (see *Section 13.4.3: INTC Module Configuration Register (INTC_MCR)*) is negated. The hardware vector enable signal to the processor is driven as negated when its associated HVEN_PRC*x* bit is negated. The vector is read from the INTC_IACKR (see *Section 13.4.5: INTC Interrupt Acknowledge Register (INTC_IACKR)*). Reading the INTC_IACKR negates the interrupt request to the 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 (see *Section 13.4.4: INTC Current Priority Register (INTC_CPR)*) onto the LIFO and updates PRI in the INTC_CPR with the new priority.

Furthermore, the interrupt vector to the processor is driven as all ‘0’s. The interrupt acknowledge signal from the processor is ignored.

13.2.1.2 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 setable interrupt request rather than being common to all of them. The INTC will use hardware vector mode for a given processor when its associated the HVEN or HVEN_PRC1 bit in the INTC_MCR is asserted. The hardware vector enable signal to the processor is driven as asserted. When the interrupt request to the 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 setable interrupt request. The vector value matches the value of the INTVEC_PRC0 field in the INTC_IACKR.

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 PRI value in the INTC_CPR register onto the LIFO and updates the PRI in the INTC_CPR register with the new priority. This pushing of the PRI value onto the LIFO and updating PRI in the INTC_CPR does not occur when the interrupt acknowledge signal asserts and the INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7) is written at a time such that the PRI value in the INTC_CPR register would need to be pushed and the previously last pushed PRI value would need to be popped simultaneously. In this case, PRI in the INTC_CPR is updated with the new priority, and the LIFO is neither pushed or popped.

13.2.2 Debug mode

The INTC operation in debug mode is identical to its operation in normal mode.

13.2.3 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.

13.2.4 Factory test mode

All INTC registers are accessible in factory test mode.

13.3 External signal description

The INTC has no external MCU signals.

13.4 Memory map/register definition

13.4.1 Memory map

Table 106 is the INTC memory map.

Table 106. INTC memory map

Address	Use	Access	Location
INTC_BASE	INTC Module Configuration Register (INTC_MCR)	Unrestricted	*on page 345*
INTC_BASE+0x4	Reserved	—	—
INTC_BASE+0x8	INTC Current Priority Register (INTC_CPR)	Unrestricted	*on page 346*
INTC_BASE+0xC	Reserved	Unrestricted	—
INTC_BASE+0x10	*INTC Interrupt Acknowledge Register (INTC_IACKR)*	Write and non-speculative read(1)	*on page 347*
INTC_BASE+0x14	Reserved	Write and non-speculative read(1)	—
INTC_BASE+0x18	*INTC End of Interrupt Register (INTC_EOIR)*	Write only	*on page 348*
INTC_BASE+0x1C	Reserved	Write only	—
INTC_BASE+0x20 – INTC_BASE+0x24	*INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)*	Unrestricted	*on page 349*

Table 106. INTC memory map (continued)

Address	Use	Access	Location
INTC_BASE+0x28 – INTC_BASE+0x3C	Reserved	—	—
INTC_BASE+0x40 – INTC_BASE+0x23C	*INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511)*	Unrestricted	*on page 350*

1. When the HVEN bit in INTC Module Configuration Register (INTC_MCR) is asserted, a read of the INTC_INTC_IACKR has no side effects.

13.4.2 Register information

All of the registers are 32 bits in width. Any combination of accessing the 4 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 two bytes, and aligned 32 bits.

In software vector mode, the side effects of a read of INTC Interrupt Acknowledge Register (INTC_IACKR) are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to either INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7) or INTC End of Interrupt Register (INTC_EOIR) does not affect on the operation of the write.

Table 107 shows the registers and bits that are not accessible in devices with one processor. In such devices, writes to these registers and bits will have no effect and reads will return zeros.

Table 107. Registers and bits inaccessible on single-processor devices

Register name	Bits
INTC Module Configuration Register (INTC_MCR)	VTES_PRC1 and HVEN_PRC1
INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511)	PCR_SEL0–PRC_SEL511

13.4.3 INTC Module Configuration Register (INTC_MCR)

The Module Configuration Register is used to configure options of the INTC.

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	0	0	
W			VTES_PRC1					HVEN_PRC1			VTES					HVEN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 118. INTC Module Configuration Register (INTC_MCR)

Table 108. INTC_MCR field descriptions

Field	Description
VTES, VTES_PRC1	Vector Table Entry Size for Processor 0 and Processor 1 The VTES bit controls the number of ‘0’s to the right of INTVEC_PRC0 in INTC Interrupt Acknowledge Register (INTC_IACKR). The VTES_PRC1 bit controls the number of ‘0’s to the right of INTVEC_PRC1. If the contents of INTC_IACKR 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. 1 8 bytes 0 4 bytes
HVEN, HVEN_PRC1	Hardware Vector Enable for Processor 0 and Processor 1 The HVEN bit controls whether the INTC is in hardware vector mode or software vector mode. Refer to *Section 13.2.1: Normal mode* for the details of the handshaking with the processor in each mode. 1 Hardware vector mode 0 Software vector mode

13.4.4 INTC Current Priority Register (INTC_CPR)

The Current Priority Register masks any peripheral or software setable interrupt request at the same or lower priority of the current value of the PRI field in INTC_CPR from generating an interrupt request to Processor 0. When INTC Interrupt Acknowledge Register (INTC_IACKR) is read in software vector mode, or the interrupt acknowledge signal from the processor is asserted in hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7) is written, the LIFO is popped into the INTC_CPR’s PRI field. An exception case in hardware vector mode to this behavior is described in *Section 13.2.1.2: Hardware vector mode*.

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to *Section 13.6.5: 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 13.6.5.2: Ensuring coherency for example code to ensure coherency.

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 119. INTC Current Priority Register (INTC_CPR)

Table 109. INTC_CPR field descriptions

Field	Description
PRI[0:3]	Priority PRI is the priority of the currently executing ISR according to the field values defined below. 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

13.4.5 INTC Interrupt Acknowledge Register (INTC_IACKR)

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 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 does not have side effects in hardware vector mode.

INTC_BASE+0x10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VTBA (most significant 16 bits)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20(1)	21	22	23	24	25	26	27	28	29(1)	30	31
R	VTBA (least significant 5 bits)					INTVEC									0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 120. INTC Interrupt Acknowledge Register (INTC_IACKR)

1. When the VTES bit in INTC Module Configuration Register (INTC_MCR) is asserted, 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.

Table 110. INTC_IACKR field descriptions

Field	Description
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 bit in INTC_MCR is asserted.
INTVEC_PRC0[0:8]	Interrupt Vector for Processor 0 INTVEC_PRC0 is the vector of the peripheral or software setable 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.

13.4.6 INTC End of Interrupt Register (INTC_EOIR)

Writing to the End of Interrupt Register signals the end of the servicing of the interrupt request. When the INTC_EOIR is written, the priority last pushed on the LIFO is popped into INTC Current Priority Register (INTC_CPR). An exception case in hardware vector mode to this behavior is described in *Section 13.2.1.2: Hardware vector mode*. The values and size of data written to the INTC_EOIR are ignored. Those values and sizes written to this register neither update the INTC_EOIR contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all '0's to the INTC_EOIR.

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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	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

Figure 121. INTC End Of Interrupt Register (INTC_EOIR)

13.4.7 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)

The Software Set/Clear Interrupt Registers support the setting or clearing of software setable 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.

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 122. 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 123. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR0_3)

Table 111. INTC_SSCIRx field descriptions

Field	Description
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.

13.4.8 INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511)

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 setable interrupt request determines which INTC_PSR*x*_*x* is assigned to that interrupt request. The software setable 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 8–511, and their priorities are configured in INTC_PSR8_11 through INTC_PSR508_511, respectively. The number of peripheral interrupt requests is dependent on the SoC implementation. Therefore, not all versions of the INTC will have 512 sources of peripheral or software setable interrupt requests or the higher number priority select registers.

Note: The PRC_SELx or PRIx field of an INTC_PSRx_x must not be modified while its corresponding peripheral or software setable interrupt request is asserted.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRC_SEL0		0	0	PRI0				PRC_SEL1		0	0	PRI1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRC_SEL2		0	0	PRI2				PRC_SEL3		0	0	PRI3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 124. INTC Priority Select Register 0–3 (INTC_PSR0_3)

INTC_BASE+0x23C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRC_ SEL508		0	0	PRI508				PRC_ SEL509		0	0	PRI509			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRC_ SEL510		0	0	PRI510				PRC_ SEL511		0	0	PRI511			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 125. INTC Priority Select Register 508 - 511 (INTC_PSR508_511)

Table 112. INTC_PSRx field descriptions

Field	Description
PRC_SEL0[0:1] – PRC_SEL511[0:1]	Processor Select If an interrupt source is enabled, PRC_SELx[0:1] selects whether the interrupt request is to be sent to processor 0, processor 1 or both processors. 00 Interrupt request sent to Processor 0 01 Interrupt request sent to both processors 10 Reserved 11 Interrupt request sent to Processors 1 **Note:** This field is ignored because this is a single-processor device.
PRI0[0:3] – PRI511[0:3]	Priority Select PRI*x* selects the priority for the interrupt requests. Refer to the field values in *Table 109*.

13.5 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.

13.5.1 Interrupt request sources

The INTC has two types of interrupt requests: peripheral and software setable. These interrupt requests can assert on any clock cycle.

Note: The INTC has no spurious vector support. Therefore, if an asserted peripheral or software setable interrupt request, whose PRIx value in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511) is higher than the PRI value in INTC Current Priority Register (INTC_CPR), negates before the interrupt request to the processor for that peripheral or software setable interrupt request is acknowledged, the interrupt request to the processor still can assert or will remain asserted for that peripheral or software setable interrupt request. In this case, the interrupt vector will correspond to that peripheral or software setable interrupt request. Also, the PRI value in the INTC_CPR will be updated with the corresponding PRIx value in INTC_PSRx_x.

Note: 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.

13.5.1.1 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. While the INTC can support up to 504 peripheral interrupt requests, the actual number is dependent on the SoC implementation.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

Interrupt requests from devices external to the SoC are classified as peripheral interrupt requests in this chapter. An anticipated way that an SoC will support an external interrupt request is by having a flag bit in a peripheral on the SoC which records an edge on the external interrupt request. This flag bit then drives a peripheral interrupt request. If the external interrupt request is level sensitive instead of edge triggered, the flag bit is effectively on the external device and not on the SoC. In that case, the external interrupt request must have transitioned as a result of clearing the flag bit on the external device before the INTC_EOIR can be written.

13.5.1.2 Software setable interrupt requests

An interrupt request is triggered by software by writing a '1' to a SET*x* bit in INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7). 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.

13.5.1.3 Unique vector for each interrupt request source

Each peripheral and software setable interrupt request is assigned a hardwired unique 9-bit vector. Software setable 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. While the peripheral interrupt request input ports at the boundary of the INTC block are assigned specific hardwired vectors within the INTC, those input ports are not associated with any specific peripheral until the SoC integration. Those assignments are dependent on the SoC integration.

13.5.2 Priority management

The asserted interrupt requests are compared to each other based on their PRI*x* and PRC_SELx values set in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511). The result of that comparison also is compared to PRI in INTC Current Priority Register (INTC_CPR). The results of those comparisons are used to manage the priority of the ISR being executed by the processor. The LIFO also assists in managing that priority.

13.5.2.1 Current priority and preemption

The priority arbitrator, selector, encoder, and comparator subblocks shown in *Figure 117* are used to compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software setable 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 setable interrupt request is generated for the INTC Interrupt Acknowledge Register (INTC_IACKR), and if in hardware vector mode, for the interrupt vector provided to the processor.

13.5.2.1.1 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 setable. 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.

13.5.2.1.2 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 setable interrupt requests.

13.5.2.1.3 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 processor.

13.5.2.1.4 Priority comparator subblock

The priority comparator subblock compares the highest priority output from the associated priority arbitrator subblock with PRI in the INTC_CPR. If the priority comparator subblock detects that this highest priority is higher than the current priority, then it asserts the interrupt request to the processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in the INTC_CPR, or the PRI value in the INTC_CPR is lowered below this highest priority. This highest priority then becomes the new priority which will be written to PRI in the INTC_CPR 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 INTC_CPR.

13.5.2.2 LIFO

The LIFO stores the preempted PRI values from the INTC_CPR. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the INTC_CPR does not need to be loaded from the INTC_CPR and stored onto the context stack. Likewise at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the INTC_CPR.

The PRI value in the INTC_CPR is pushed onto the LIFO when the INTC_IACKR is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode. The priority is popped into PRI in the INTC_CPR whenever the INTC_EOIR is written. An exception case in hardware vector mode to this behavior is described in *Section 13.2.1.2: Hardware vector mode*.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR equal to 15 will not be preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority of 0 is not needed because of how pushing onto a full LIFO and popping an empty LIFO are treated. If the LIFO is pushed 15 or more times than it is popped, the priorities first pushed are overwritten. A priority of 0 would be an overwritten priority. However, the LIFO will pop ‘0’s if it is popped more times than it is pushed. Therefore, although a priority of 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory mapped, even in debug mode or factory test mode.

13.5.3 Handshaking with processor

13.5.3.1 Software vector mode handshaking

13.5.3.1.1 Acknowledging interrupt request to processor

The software vector mode handshaking can be used with processors that only support an interrupt request to them, or processors that support both just an interrupt request to them or an interrupt request and interrupt vector to them. The software vector mode handshaking even supports processors which always expect an interrupt vector with the interrupt request to them. Refer to *Table 113*.

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 126*. The INTC examines the peripheral and software setable interrupt requests. When it finds an asserted peripheral or software setable interrupt request with a higher priority than PRI in the INTC Current Priority Register (INTC_CPR), it asserts the interrupt

request to the processor. The INTVEC field in the INTC Interrupt Acknowledge Register (INTC_IACKR) is updated with the preempting interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The rest of the handshaking is described in *Section 13.2.1.1: Software vector mode*.

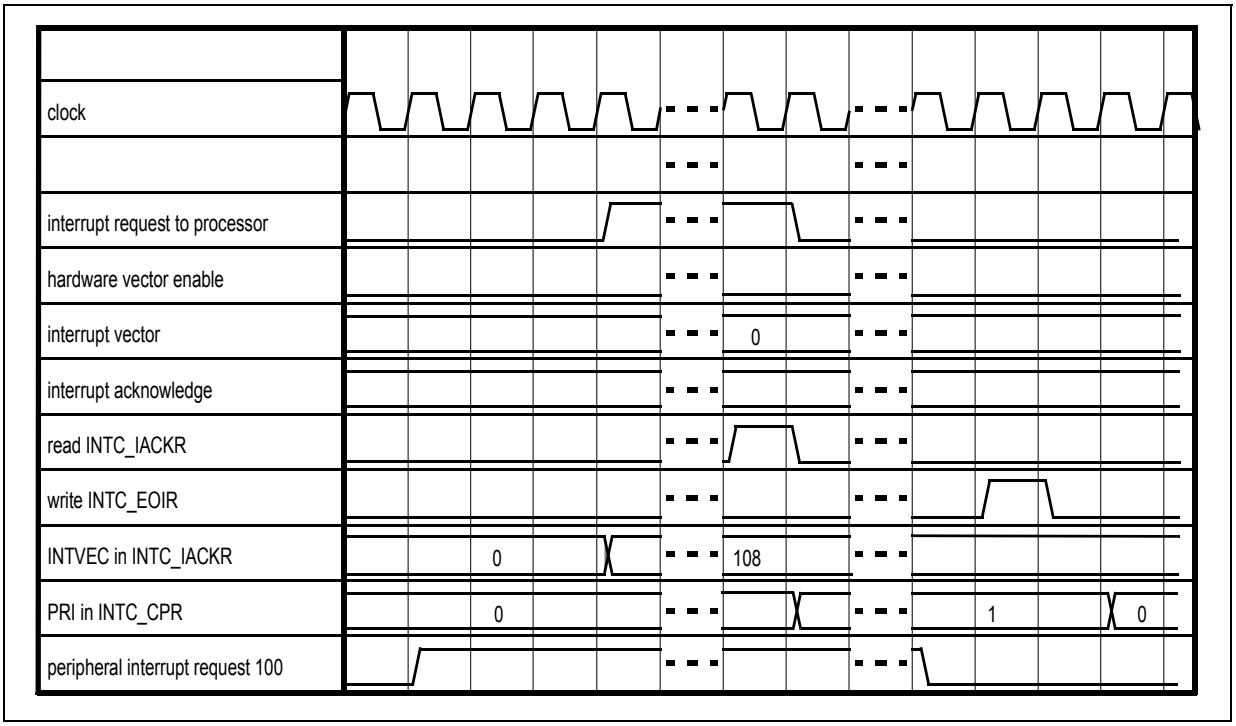
13.5.3.1.2 End of interrupt exception handler

Before the interrupt exception handling completes, INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7) must be written. When it is written, the LIFO is popped so that the preempted priority is restored into PRI of the INTC_CPR. Before it is written, the peripheral or software setable flag bit must be cleared so that the peripheral or software setable interrupt request is negated.

Note: Depending on the pipelining on an SoC implementation's pipelining capabilities and bus architecture, a store to clear the peripheral or software setable interrupt flag bit which closely precedes the store to the INTC_EOIR can result in that peripheral or software setable interrupt request being serviced again. If this scenario can happen, preventative measures can be used such as executing a Power Architecture isync instruction before the store to the INTC_EOIR as shown in Section 13.6.2.1: Software vector mode.

When returning from the preemption, the INTC does not search for the peripheral or software setable 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 INTC_CPR is lowered to the priority of the preempted ISR, the interrupt request for the preempted ISR or any other asserted peripheral or software setable 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 126. Software vector mode handshaking timing diagram

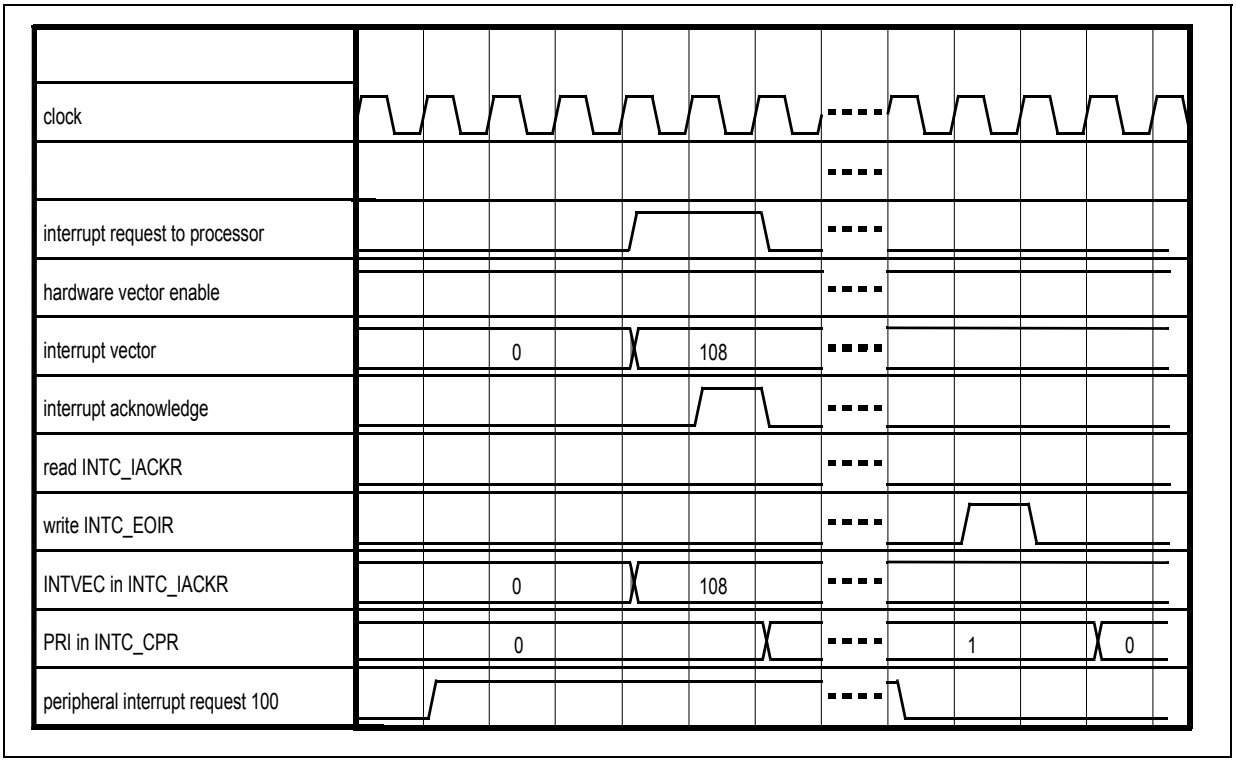


13.5.3.2 Hardware vector mode handshaking

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 127*. As in software vector mode, the INTC examines the peripheral and software setable interrupt requests, and when it finds an asserted one with a higher priority than PRI in the INTC_CPR, it asserts the interrupt request to the processor. The INTVEC field in the INTC_IACKR is updated with the preempting peripheral or software setable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. In addition, the value of the interrupt vector to the processor matches the value of the INTVEC field in the INTC_IACKR. The rest of the handshaking is described in *Section 13.2.1.2: Hardware vector mode*.

The handshaking near the end of the interrupt exception handler, that is the writing to the INTC_EOIR, is the same as in software vector mode. Refer to *Section 13.5.3.1.2: End of interrupt exception handler*.

Figure 127. Hardware vector mode handshaking timing diagram



13.5.3.3 Processor interrupt handshaking compatibility

Table 113 lists the compatibility between the INTC's software and hardware vector modes and the possible processor interrupt handshaking. The only configuration not supported is hardware vector mode with a processor that does not have an interrupt acknowledge signal.

Table 113. Processor interrupt handshaking compatibility

Processor interrupt handshaking capabilities	Vector mode compatibility	
	Software	Hardware
No interrupt vector support	Yes	—
Optionally no interrupt vector or interrupt vector with interrupt acknowledge signal	Yes	Yes
Only interrupt vector with interrupt acknowledge signal	Yes	Yes
Optionally no interrupt vector or interrupt vector without interrupt acknowledge signal	Yes	No
Only interrupt vector without interrupt acknowledge signal	Yes	No

13.5.4 Reserved spaces in memory map

The memory map has spaces reserved for alternate implementations of the INTC. These features are not in this implementation of the INTC.

13.5.4.1 Additional software setable interrupt requests

The addresses INTC_BASE+0x28 through INTC_BASE+0x3C are reserved for 24 more Software Set/Clear Interrupt Registers.

13.6 Initialization/application information

13.6.1 Initialization flow

After exiting reset, all of the PRI*x* and PRC_SEL*x* fields in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511) will be zero, and PRI in INTC Current Priority Register (INTC_CPR) will be 15. These reset values will prevent the INTC from asserting the interrupt request to the 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 setable interrupt requests to cause an interrupt request to the processor is:

```
interrupt_request_initialization:
configure VTES,VTES_PRC1,HVEN and HVEN_PRC1 in INTC_MCR
configure VTBA_PRCx in IACKR
raise the PRIx fields and set the PRC_SELx fields to the desired processor
in PSRx_x
set the enable bits or clear the mask bits for the peripheral interrupt
requests
lower PRI in INTC_CPR to zero
enable processor(s) recognition of interrupts
```

13.6.2 Interrupt exception handler

These example interrupt exception handlers use Power Architecture assembly code.

13.6.2.1 Software vector mode

```
interrupt_exception_handler:
code to save SRR0 and SRR1

lis r3,hi(IACKR)# form IACKR address
ori r3,r3,lo(IACKR)
lwz r3,0x0(r3)# load IACKR, which clears request to processor
lwz r3,0x0(r3)# load address of ISR from vector table

code to enable processor recognition of interrupts and save context
required by EABI

mtlr r3    # move IACKR contents into link register
blrl       # branch to ISR; link register updated with epilog
           # address

epilog:
```

```
lis r3,hi(EOIR)# form EOIR address
ori r3,r3,lo(EOIR)
li r4,0x0  # form 0 to write to EOIR
stw r4,0x0(r3)# store to EOIR, 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
```

13.6.2.2 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:
lis r3,hi(EOIR)# form EOIR address
ori r3,r3,lo(EOIR)
li r4,0x0  # form 0 to write to EOIR
stw r4,0x0(r3)# store to EOIR, 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
```

13.6.3 ISR, RTOS, and task hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC Current Priority Register (INTC_CPR) having a value of 0. The RTOS will execute the tasks according to whatever priority scheme that it may have, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC_CPR priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task's priority can be elevated in the INTC_CPR while the shared resource is being accessed.

An ISR whose PRI*x* in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511) has a value of 0 will not cause an interrupt request to the selected processor, even if its peripheral or software setable 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.

13.6.4 Order of execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software setable interrupt requests. However, if multiple peripheral or software setable 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 setable interrupt requests asserted.

The example in *Table 114* shows the order of execution of both ISRs with different priorities and the same priority.

Table 114. Order of ISR execution example

Step		Code executing at end of step						PRI in INTC_CPR at end of step
No.	Description	RTOS	ISR108 (1)	ISR208	ISR308	ISR408	Interrupt exception handler	
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 asserts.					X		4
5	Peripheral interrupt request 200 at priority 3 asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserts first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR.						X	0
12	RTOS continues execution.	X						0

1. ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software setable interrupt requests.

13.6.5 Priority ceiling protocol

13.6.5.1 Elevating priority

The PRI field in INTC Current Priority Register (INTC_CPR) is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol 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 to 3, the ceiling of all of the ISR priorities. After they release the resource, they must lower the PRI value in INTC_CPR 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.

13.6.5.2 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. 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 can be made by those system services with the code sequence:

```
disable processor recognition of interrupts
PRI modification
enable processor recognition of interrupts
```

13.6.6 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.

13.6.7 Software setable interrupt requests

The software setable 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.

13.6.7.1 Scheduling a lower priority portion of an ISR

A portion of an ISR needs to be executed at the PRI*x* value in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511), which becomes the PRI value in INTC Current Priority Register (INTC_CPR) 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 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7). Writing a '1' to SET*x* causes a software setable interrupt request. This software setable interrupt request, which usually will have a lower PRI*x* value in the PSR*x*_*x*, therefore will not cause priority inversion.

13.6.7.2 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 setable 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.

13.6.8 Lowering priority within an ISR

In SoC implementations without the software setable interrupt requests in 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 13.6.7.1: 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 INTC Current Priority Register (INTC_CPR) within an ISR to below the ISR's corresponding PRI value in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511) 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.

13.6.9 Negating an interrupt request outside of its ISR

13.6.9.1 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.

13.6.9.2 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.

13.6.9.3 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 setable interrupt requests for these other flag bits must be selected properly. Their PRI*x* values in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511) 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 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 13.5.3.1.2: 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 PSR*x*_*x*.

13.6.10 Examining LIFO contents

In normal mode, the user does not need to know the contents of the LIFO. He may not even know how deeply the LIFO is nested. However, if he 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 INTC Current Priority Register (INTC_CPR). The code sequence is:

```
pop_lifo:
store to EOIR
load INTC_CPR, examine PRI, and store onto stack
if PRI is not zero or value when interrupts were enabled, branch to pop_lifo
```

When the examination is complete, the LIFO can be restored using this code sequence:

```
push_lifo:
load stacked PRI value and store to INTC_CPR
load IACKR
if stacked PRI values are not depleted, branch to push_lifo
```

14 Interrupts

14.1 Introduction

This chapter focuses on the interrupt operation of the e200z335 core. Details of the operation of the interrupt controller are given in *Chapter 13: Interrupt Controller (INTC)*.

Two types of modes are used to learn the vector of the source of the interrupt request: software vector mode and hardware vector mode. Software vector mode is compliant with Power Architecture technology. The e200z335 branches to a common interrupt exception handler to service the interrupt request. The interrupt exception handler reads the INTC_IACKR to learn the vector of the source of the interrupt request. In hardware vector mode, the interrupt exception handler is unique to the vector of the source of the interrupt request.

14.2 Interrupt vectors

14.2.1 External input

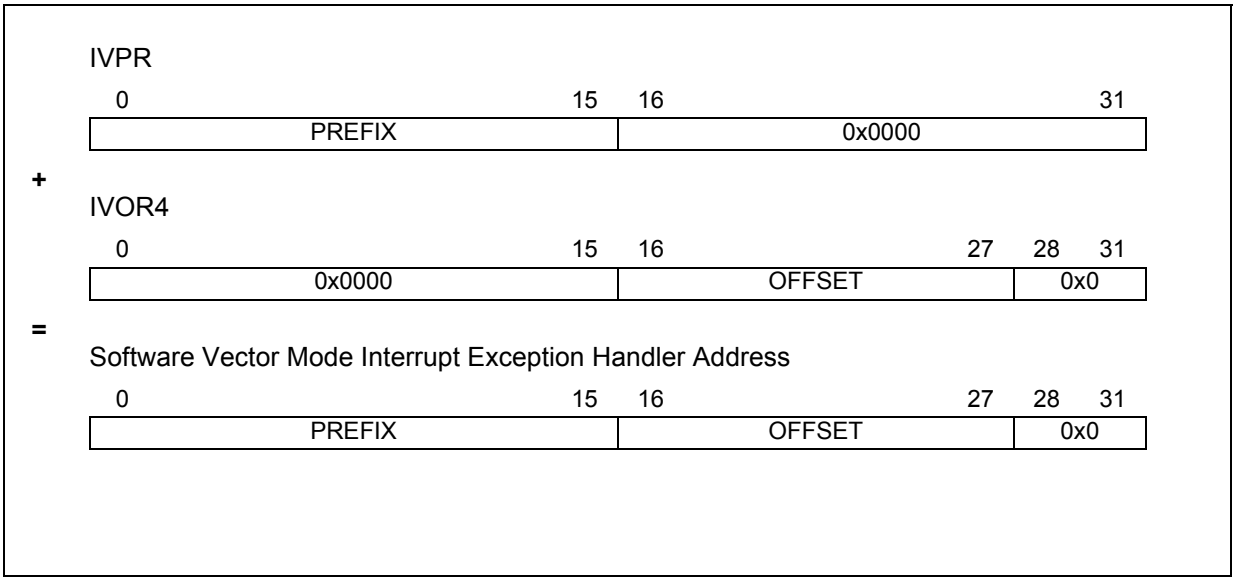
14.2.1.1 Software Vector Mode

In software vector mode, the interrupt exception handler acknowledges the interrupt request to the e200z335 from the INTC by reading the INTC_IACKR. The e200z335 is enabled again to recognize the external input by setting the EE bit of the MSR. The prolog of the interrupt exception handler must acknowledge the interrupt request before the e200z335 is enabled again to recognize the external input. Otherwise, the e200z335 will attempt to service the same source of the interrupt request.

The INTC's LIFO is popped by writing to the INTC_EOIR. The e200's recognition of the external input is disabled by clearing the EE bit of the MSR. In the epilog of the interrupt exception handler, the timing relationship between popping the LIFO and disabling recognition of the external input has no restrictions. The writes can happen in either order. However, disabling recognition of the external input before popping the LIFO eases the calculation of the maximum pipe depth at the cost of postponing the servicing of the next interrupt request.

The IVPR acts as a base register for all types of exceptions. An IVOR, unique to each type of exception, determines the offset from the IVPR. The IVPR and IVOR are added to calculate the interrupt exception handler address. In software vector mode, IVOR4 is used for the external input, that is, the interrupt request to the e200z335 from the INTC. *Figure 128* shows the software vector mode interrupt exception handler address calculation.

Figure 128. Software Vector Mode Interrupt Exception Handler Address Calculation



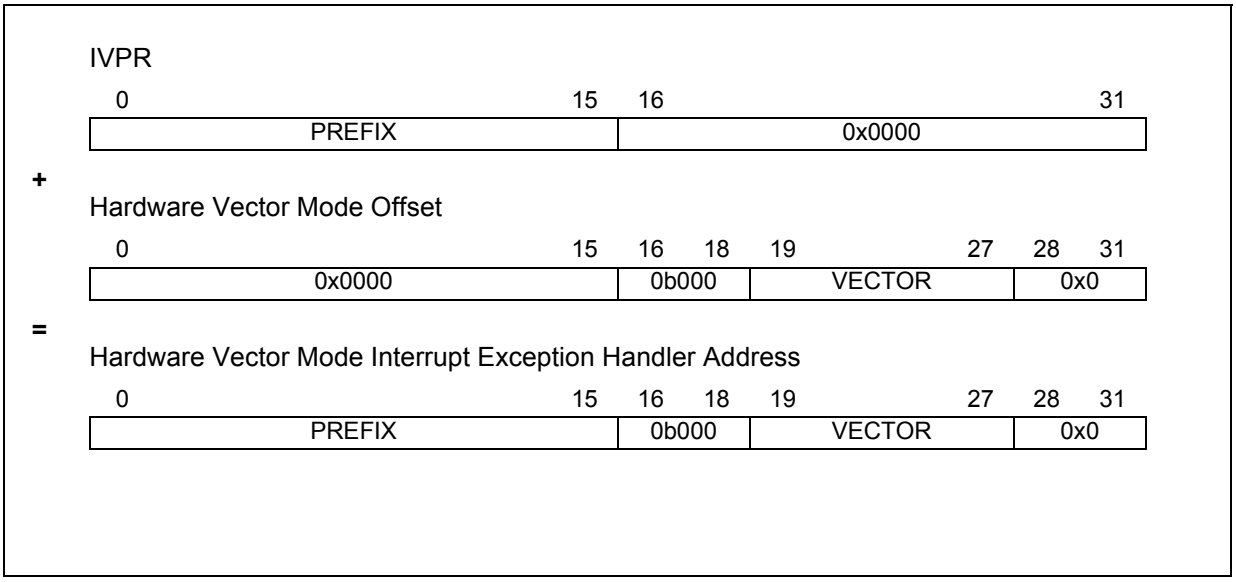
14.2.1.2 Hardware Vector Mode

In hardware vector mode, the interrupt request to the e200z335 from the INTC is acknowledged before the e200z335 starts to execute the exception handler. The INTC_IACKR does not need to be read to acknowledge the interrupt request before the e200z335 is enabled again to recognize the external input.

As in software vector mode, the timing relationship between popping the LIFO and disabling recognition of the external input has no restrictions. Also, as in software vector mode, disabling recognition of the external input before popping the LIFO eases the calculation of the maximum pipe depth at the cost of postponing the servicing of the next interrupt request.

In hardware vector mode, no IVOR is used, including IVOR4, which has no effect. The interrupt exception handler for each vector is offset from the IVPR. The vectors for each source are shown in *Table 115*. The amount of the offset is the vector times 16 bytes. *Figure 129* shows the hardware vector mode interrupt exception handler address calculation.

Figure 129. Hardware Vector Mode Interrupt Exception Handler Address Calculation



Any vector which is not reserved can have an interrupt exception handler which can be executed.

Note: In hardware vector mode, the IVORs for all other exceptions besides the external input must be configured to not use the interrupt exception handler addresses.

14.2.2 Critical input

The critical interrupt input can be triggered by the Software Watchdog Timer (SWT) or by the Non-Maskable Interrupt (NMI), provided that the NMI_SEL bit of the DMA/Interrupt Request Enable Register is asserted. The NMI_SEL bit is write-once.

This device incorporates a non-maskable interrupt pin called NMI and a new MSR bit to indicate whether it can be recovered or not. This would generate a machine check exception, set a new NMI bit in the MCSR, and update the MSR new RI (recoverable interrupt) bit. A scenario exists where the NMI could be received during processing of the critical input interrupt, and this would not be recoverable.

14.3 Interrupt summary

The assignments between the interrupt requests from the blocks to the vectors for the external input to the e200z335 are shown in *Table 115*. The Source column is written in C language syntax. The syntax is block_instance.register[bit]. The syntax '||' represents the ORing of individual interrupt requests from the block.

Table 115. Interrupt summary

Interrupt	Offset	Vector	Priority (1)	Source	Description
INTC_SSCIR0_3_CLR0	0x0000	0	—	INTC.INTC_SSCIR0_3[CLR0]	INTC software setable Clear flag 0
INTC_SSCIR0_3_CLR1	0x0010	1	—	INTC.INTC_SSCIR0_3[CLR1]	INTC software setable Clear flag 1
INTC_SSCIR0_3_CLR2	0x0020	2	—	INTC.INTC_SSCIR0_3[CLR2]	INTC software setable Clear flag 2
INTC_SSCIR0_3_CLR3	0x0030	3	—	INTC.INTC_SSCIR0_3[CLR3]	INTC software setable Clear flag 3
INTC_SSCIR4_7_CLR4	0x0040	4	—	INTC.INTC_SSCIR4_7[CLR4]	INTC software setable Clear flag 4
INTC_SSCIR4_7_CLR5	0x0050	5	—	INTC.INTC_SSCIR4_7[CLR5]	INTC software setable Clear flag 5
INTC_SSCIR4_7_CLR6	0x0060	6	—	INTC.INTC_SSCIR4_7[CLR6]	INTC software setable Clear flag 6
INTC_SSCIR4_7_CLR7	0x0070	7	—	INTC.INTC_SSCIR4_7[CLR7]	INTC software setable Clear flag 7
SWT_IR_TIF	0x0080	8	—	SWT.SWT_IR[TIF]	Software Watchdog Interrupt Flag
MCM_ESR_COMB	0x0090	9	—	MCM.ESR[RNCE] || MCM.ESR[FNCE]	MCM combined interrupt request of the Platform RAM Non-Correctable Error and Platform Flash Non-Correctable Error interrupt requests
eDMA_ERRL_ERR31_0	0x00a0	10	—	eDMA.DMAERRL[ERR31:ERR0]	eDMA channel Error flags 31–0
eDMA_INTL_INT0	0x00b0	11	—	eDMA.DMAINTL[INT0]	eDMA channel Interrupt 0
eDMA_INTL_INT1	0x00c0	12	—	eDMA.DMAINTL[INT1]	eDMA channel Interrupt 1
eDMA_INTL_INT2	0x00d0	13	—	eDMA.DMAINTL[INT2]	eDMA channel Interrupt 2
eDMA_INTL_INT3	0x00e0	14	—	eDMA.DMAINTL[INT3]	eDMA channel Interrupt 3
eDMA_INTL_INT4	0x00f0	15	—	eDMA.DMAINTL[INT4]	eDMA channel Interrupt 4
eDMA_INTL_INT5	0x0100	16	—	eDMA.DMAINTL[INT5]	eDMA channel Interrupt 5
eDMA_INTL_INT6	0x0110	17	—	eDMA.DMAINTL[INT6]	eDMA channel Interrupt 6

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
eDMA_INTL_INT7	0x0120	18	—	eDMA.DMAINTL[INT7]	eDMA channel Interrupt 7
eDMA_INTL_INT8	0x0130	19	—	eDMA.DMAINTL[INT8]	eDMA channel Interrupt 8
eDMA_INTL_INT9	0x0140	20	—	eDMA.DMAINTL[INT9]	eDMA channel Interrupt 9
eDMA_INTL_INT10	0x0150	21	—	eDMA.DMAINTL[INT10]	eDMA channel Interrupt 10
eDMA_INTL_INT11	0x0160	22	—	eDMA.DMAINTL[INT11]	eDMA channel Interrupt 11
eDMA_INTL_INT12	0x0170	23	—	eDMA.DMAINTL[INT12]	eDMA channel Interrupt 12
eDMA_INTL_INT13	0x0180	24	—	eDMA.DMAINTL[INT13]	eDMA channel Interrupt 13
eDMA_INTL_INT14	0x0190	25	—	eDMA.DMAINTL[INT14]	eDMA channel Interrupt 14
eDMA_INTL_INT15	0x01a0	26	—	eDMA.DMAINTL[INT15]	eDMA channel Interrupt 15
eDMA_INTL_INT16	0x01b0	27	—	eDMA.DMAINTL[INT16]	eDMA channel Interrupt 16
eDMA_INTL_INT17	0x01c0	28	—	eDMA.DMAINTL[INT17]	eDMA channel Interrupt 17
eDMA_INTL_INT18	0x01d0	29	—	eDMA.DMAINTL[INT18]	eDMA channel Interrupt 18
eDMA_INTL_INT19	0x01e0	30	—	eDMA.DMAINTL[INT19]	eDMA channel Interrupt 19
eDMA_INTL_INT20	0x01f0	31	—	eDMA.DMAINTL[INT20]	eDMA channel Interrupt 20
eDMA_INTL_INT21	0x0200	32	—	eDMA.DMAINTL[INT21]	eDMA channel Interrupt 21
eDMA_INTL_INT22	0x0210	33	—	eDMA.DMAINTL[INT22]	eDMA channel Interrupt 22
eDMA_INTL_INT23	0x0220	34	—	eDMA.DMAINTL[INT23]	eDMA channel Interrupt 23
eDMA_INTL_INT24	0x0230	35	—	eDMA.DMAINTL[INT24]	eDMA channel Interrupt 24
eDMA_INTL_INT25	0x0240	36	—	eDMA.DMAINTL[INT25]	eDMA channel Interrupt 25
eDMA_INTL_INT26	0x0250	37	—	eDMA.DMAINTL[INT26]	eDMA channel Interrupt 26

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
eDMA_INTL_INT27	0x0260	38	—	eDMA.DMAINTL[INT27]	eDMA channel Interrupt 27
eDMA_INTL_INT28	0x0270	39	—	eDMA.DMAINTL[INT28]	eDMA channel Interrupt 28
eDMA_INTL_INT29	0x0280	40	—	eDMA.DMAINTL[INT29]	eDMA channel Interrupt 29
eDMA_INTL_INT30	0x0290	41	—	eDMA.DMAINTL[INT30]	eDMA channel Interrupt 30
eDMA_INTL_INT31	0x02a0	42	—	eDMA.DMAINTL[INT31]	eDMA channel Interrupt 31
FMPLL_SYNSR_LOCF	0x02b0	43	—	FMPLL.SYNSR[LOCF]	FMPLL Loss Of Clock Flag
FMPLL_SYNSR_LOLF	0x02c0	44	—	FMPLL.SYNSR[LOLF]	FMPLL Loss Of Lock Flag
SIU_OSR_OVER	0x02d0	45	—	SIU.SIU_OSR[OVF15:OVF0]	SIU combined overrun interrupt request of the external interrupt Overrun Flags
SIU_EISR_EIF0	0x02e0	46	—	SIU.SIU_EISR[EIF0]	SIU External Interrupt Flag 0
SIU_EISR_EIF1	0x02f0	47	—	SIU.SIU_EISR[EIF1]	SIU External Interrupt Flag 1
SIU_EISR_EIF2	0x0300	48	—	SIU.SIU_EISR[EIF2]	SIU External Interrupt Flag 2
SIU_EISR_EIF3	0x0310	49	—	SIU.SIU_EISR[EIF3]	SIU External Interrupt Flag 3
SIU_EISR_EIF15_4	0x0320	50	—	SIU.SIU_EISR[EIF15:EIF4]	SIU External Interrupt Flags 15–4
eMIOS_FLAG_F0	0x0330	51	—	eMIOS.eMIOSFLAG[F0]	eMIOS channel 0 Flag
eMIOS_FLAG_F1	0x0340	52	—	eMIOS.eMIOSFLAG[F1]	eMIOS channel 1 Flag
eMIOS_FLAG_F2	0x0350	53	—	eMIOS.eMIOSFLAG[F2]	eMIOS channel 2 Flag
eMIOS_FLAG_F3	0x0360	54	—	eMIOS.eMIOSFLAG[F3]	eMIOS channel 3 Flag
eMIOS_FLAG_F4	0x0370	55	—	eMIOS.eMIOSFLAG[F4]	eMIOS channel 4 Flag
eMIOS_FLAG_F5	0x0380	56	—	eMIOS.eMIOSFLAG[F5]	eMIOS channel 5 Flag
eMIOS_FLAG_F6	0x0390	57	—	eMIOS.eMIOSFLAG[F6]	eMIOS channel 6 Flag
eMIOS_FLAG_F7	0x03a0	58	—	eMIOS.eMIOSFLAG[F7]	eMIOS channel 7 Flag
eMIOS_FLAG_F8	0x03b0	59	—	eMIOS.eMIOSFLAG[F8]	eMIOS channel 8 Flag
eMIOS_FLAG_F9	0x03c0	60	—	eMIOS.eMIOSFLAG[F9]	eMIOS channel 9 Flag
eMIOS_FLAG_F10	0x03d0	61	—	eMIOS.eMIOSFLAG[F10]	eMIOS channel 10 Flag

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
eMIOS_FLAG_F11	0x03e0	62	—	eMIOS.eMIOSFLAG[F11]	eMIOS channel 11 Flag
eMIOS_FLAG_F12	0x03f0	63	—	eMIOS.eMIOSFLAG[F12]	eMIOS channel 12 Flag
eMIOS_FLAG_F13	0x0400	64	—	eMIOS.eMIOSFLAG[F13]	eMIOS channel 13 Flag
eMIOS_FLAG_F14	0x0410	65	—	eMIOS.eMIOSFLAG[F14]	eMIOS channel 14 Flag
eMIOS_FLAG_F15	0x0420	66	—	eMIOS.eMIOSFLAG[F15]	eMIOS channel 15 Flag
eTPU_MCR_GE	0x0430	67	—	eTPU.eTPUMCR[MGE1] || eTPU.eTPUMCR[MGE2] || eTPU.eTPUMCR[ILF1] || eTPU.eTPUMCR[ILF2] || eTPU.eTPUMCR[SCMMISF]	eTPU Global Exception
eTPU_CISR_1_CIS0	0x0440	68	—	eTPU.eTPUCISR_1[CIS0]	eTPU_1 Channel 0 Interrupt Status
eTPU_CISR_1_CIS1	0x0450	69	—	eTPU.eTPUCISR_1[CIS1]	eTPU_1 Channel 1 Interrupt Status
eTPU_CISR_1_CIS2	0x0460	70	—	eTPU.eTPUCISR_1[CIS2]	eTPU_1 Channel 2 Interrupt Status
eTPU_CISR_1_CIS3	0x0470	71	—	eTPU.eTPUCISR_1[CIS3]	eTPU_1 Channel 3 Interrupt Status
eTPU_CISR_1_CIS4	0x0480	72	—	eTPU.eTPUCISR_1[CIS4]	eTPU_1 Channel 4 Interrupt Status
eTPU_CISR_1_CIS5	0x0490	73	—	eTPU.eTPUCISR_1[CIS5]	eTPU_1 Channel 5 Interrupt Status
eTPU_CISR_1_CIS6	0x04a0	74	—	eTPU.eTPUCISR_1[CIS6]	eTPU_1 Channel 6 Interrupt Status
eTPU_CISR_1_CIS7	0x04b0	75	—	eTPU.eTPUCISR_1[CIS7]	eTPU_1 Channel 7 Interrupt Status
eTPU_CISR_1_CIS8	0x04c0	76	—	eTPU.eTPUCISR_1[CIS8]	eTPU_1 Channel 8 Interrupt Status
eTPU_CISR_1_CIS9	0x04d0	77	—	eTPU.eTPUCISR_1[CIS9]	eTPU_1 Channel 9 Interrupt Status
eTPU_CISR_1_CIS10	0x04e0	78	—	eTPU.eTPUCISR_1[CIS10]	eTPU_1 Channel 10 Interrupt Status
eTPU_CISR_1_CIS11	0x04f0	79	—	eTPU.eTPUCISR_1[CIS11]	eTPU_1 Channel 11 Interrupt Status
eTPU_CISR_1_CIS12	0x0500	80	—	eTPU.eTPUCISR_1[CIS12]	eTPU_1 Channel 12 Interrupt Status

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
eTPU_CISR_1_CIS13	0x0510	81	—	eTPU.eTPUCISR_1[CIS13]	eTPU_1 Channel 13 Interrupt Status
eTPU_CISR_1_CIS14	0x0520	82	—	eTPU.eTPUCISR_1[CIS14]	eTPU_1 Channel 14 Interrupt Status
eTPU_CISR_1_CIS15	0x0530	83	—	eTPU.eTPUCISR_1[CIS15]	eTPU_1 Channel 15 Interrupt Status
eTPU_CISR_1_CIS16	0x0540	84	—	eTPU.eTPUCISR_1[CIS16]	eTPU_1 Channel 16 Interrupt Status
eTPU_CISR_1_CIS17	0x0550	85	—	eTPU.eTPUCISR_1[CIS17]	eTPU_1 Channel 17 Interrupt Status
eTPU_CISR_1_CIS18	0x0560	86	—	eTPU.eTPUCISR_1[CIS18]	eTPU_1 Channel 18 Interrupt Status
eTPU_CISR_1_CIS19	0x0570	87	—	eTPU.eTPUCISR_1[CIS19]	eTPU_1 Channel 19 Interrupt Status
eTPU_CISR_1_CIS20	0x0580	88	—	eTPU.eTPUCISR_1[CIS20]	eTPU_1 Channel 20 Interrupt Status
eTPU_CISR_1_CIS21	0x0590	89	—	eTPU.eTPUCISR_1[CIS21]	eTPU_1 Channel 21 Interrupt Status
eTPU_CISR_1_CIS22	0x05a0	90	—	eTPU.eTPUCISR_1[CIS22]	eTPU_1 Channel 22 Interrupt Status
eTPU_CISR_1_CIS23	0x05b0	91	—	eTPU.eTPUCISR_1[CIS23]	eTPU_1 Channel 23 Interrupt Status
eTPU_CISR_1_CIS24	0x05c0	92	—	eTPU.eTPUCISR_1[CIS24]	eTPU_1 Channel 24 Interrupt Status
eTPU_CISR_1_CIS25	0x05d0	93	—	eTPU.eTPUCISR_1[CIS25]	eTPU_1 Channel 25 Interrupt Status
eTPU_CISR_1_CIS26	0x05e0	94	—	eTPU.eTPUCISR_1[CIS26]	eTPU_1 Channel 26 Interrupt Status
eTPU_CISR_1_CIS27	0x05f0	95	—	eTPU.eTPUCISR_1[CIS27]	eTPU_1 Channel 27 Interrupt Status
eTPU_CISR_1_CIS28	0x0600	96	—	eTPU.eTPUCISR_1[CIS28]	eTPU_1 Channel 28 Interrupt Status
eTPU_CISR_1_CIS29	0x0610	97	—	eTPU.eTPUCISR_1[CIS29]	eTPU_1 Channel 29 Interrupt Status
eTPU_CISR_1_CIS30	0x0620	98	—	eTPU.eTPUCISR_1[CIS30]	eTPU_1 Channel 30 Interrupt Status
eTPU_CISR_1_CIS31	0x0630	99	—	eTPU.eTPUCISR_1[CIS31]	eTPU_1 Channel 31 Interrupt Status

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
eQADC_FISR_OVER	0x0640	100	—	eQADC.eQADC_FISR*x*[TORF] || eQADC.eQADC_FISR*x*[RFOF] || eQADC.eQADC_FISR*x*[TFUF]	eQADC combined overrun interrupt request of the Trigger Overrun, Receive FIFO Overflow, and Transmit FIFO Underflow interrupt requests from all of the FIFOs
eQADC_FISR0_NCF0	0x0650	101	—	eQADC.eQADC_FISR0[NCF0]	eQADC command FIFO 0 Non-Coherency Flag
eQADC_FISR0_PF0	0x0660	102	—	eQADC.eQADC_FISR0[PF0]	eQADC command FIFO 0 Pause Flag
eQADC_FISR0_EOQF0	0x0670	103	—	eQADC.eQADC_FISR0[EOQF0]	eQADC command FIFO 0 command queue End Of Queue Flag
eQADC_FISR0_CFFF0	0x0680	104	—	eQADC.eQADC_FISR0[CFFF0]	eQADC Command FIFO 0 Fill Flag
eQADC_FISR0_RFDF0	0x0690	105	—	eQADC.eQADC_FISR0[RFDF0]	eQADC Receive FIFO 0 Drain Flag
eQADC_FISR1_NCF1	0x06a0	106	—	eQADC.eQADC_FISR1[NCF1]	eQADC command FIFO 1 Non-Coherency Flag
eQADC_FISR1_PF1	0x06b0	107	—	eQADC.eQADC_FISR1[PF1]	eQADC command FIFO 1 Pause Flag
eQADC_FISR1_EOQF1	0x06c0	108	—	eQADC.eQADC_FISR1[EOQF1]	eQADC command FIFO 1 command queue End Of Queue Flag
eQADC_FISR1_CFFF1	0x06d0	109	—	eQADC.eQADC_FISR1[CFFF1]	eQADC Command FIFO 1 Fill Flag
eQADC_FISR1_RFDF1	0x06e0	110	—	eQADC.eQADC_FISR1[RFDF1]	eQADC Receive FIFO 1 Drain Flag
eQADC_FISR2_NCF2	0x06f0	111	—	eQADC.eQADC_FISR2[NCF2]	eQADC command FIFO 2 Non-Coherency Flag
eQADC_FISR2_PF2	0x0700	112	—	eQADC.eQADC_FISR2[PF2]	eQADC command FIFO 2 Pause Flag
eQADC_FISR2_EOQF2	0x0710	113	—	eQADC.eQADC_FISR2[EOQF2]	eQADC command FIFO 2 command queue End Of Queue Flag

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
eQADC_FISR2_CFFF2	0x0720	114	—	eQADC.eQADC_FISR2[CFFF2]	eQADC Command FIFO 2 Fill Flag
eQADC_FISR2_RFDF2	0x0730	115	—	eQADC.eQADC_FISR2[RFDF2]	eQADC Receive FIFO 2 Drain Flag
eQADC_FISR3_NCF3	0x0740	116	—	eQADC.eQADC_FISR3[NCF3]	eQADC command FIFO 3 Non-Coherency Flag
eQADC_FISR3_PF3	0x0750	117	—	eQADC.eQADC_FISR3[PF3]	eQADC command FIFO 3 Pause Flag
eQADC_FISR3_EOQF3	0x0760	118	—	eQADC.eQADC_FISR3[EOQF3]	eQADC command FIFO 3 command queue End Of Queue Flag
eQADC_FISR3_CFFF3	0x0770	119	—	eQADC.eQADC_FISR3[CFFF3]	eQADC Command FIFO 3 Fill Flag
eQADC_FISR3_RFDF3	0x0780	120	—	eQADC.eQADC_FISR3[RFDF3]	eQADC Receive FIFO 3 Drain Flag
eQADC_FISR4_NCF4	0x0790	121	—	eQADC.eQADC_FISR4[NCF4]	eQADC command FIFO 4 Non-Coherency Flag
eQADC_FISR4_PF4	0x07a0	122	—	eQADC.eQADC_FISR4[PF4]	eQADC command FIFO 4 Pause Flag
eQADC_FISR4_EOQF4	0x07b0	123	—	eQADC.eQADC_FISR4[EOQF4]	eQADC command FIFO 4 command queue End Of Queue Flag
eQADC_FISR4_CFFF4	0x07c0	124	—	eQADC.eQADC_FISR4[CFFF4]	eQADC Command FIFO 4 Fill Flag
eQADC_FISR4_RFDF4	0x07d0	125	—	eQADC.eQADC_FISR4[RFDF4]	eQADC Receive FIFO 4 Drain Flag
eQADC_FISR5_NCF5	0x07e0	126	—	eQADC.eQADC_FISR5[NCF5]	eQADC command FIFO 5 Non-Coherency Flag
eQADC_FISR5_PF5	0x07f0	127	—	eQADC.eQADC_FISR5[PF5]	eQADC command FIFO 5 Pause Flag
eQADC_FISR5_EOQF5	0x0800	128	—	eQADC.eQADC_FISR5[EOQF5]	eQADC command FIFO 5 command queue End Of Queue Flag
eQADC_FISR5_CFFF5	0x0810	129	—	eQADC.eQADC_FISR5[CFFF5]	eQADC Command FIFO 5 Fill Flag
eQADC_FISR5_RFDF5	0x0820	130	—	eQADC.eQADC_FISR5[RFDF5]	eQADC Receive FIFO 5 Drain Flag

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
DSPI_B_ISR_OVER	0x0830	131	—	DSPI_B.DSPI_ISR[TFUF] || DSPI_B.DSPI_ISR[RFOF]	DSPI_B combined overrun interrupt request of the Transmit FIFO Underflow and Receive FIFO Overflow interrupt requests
DSPI_B_ISR_EOQF	0x0840	132	—	DSPI_B.DSPI_ISR[EOQF]	DSPI_B transmit FIFO End Of Queue Flag
DSPI_B_ISR_TFFF	0x0850	133	—	DSPI_B.DSPI_ISR[TFFF]	DSPI_B Transmit FIFO Fill Flag
DSPI_B_ISR_TCF	0x0860	134	—	DSPI_B.DSPI_ISR[TCF]	DSPI_B Transfer Complete Flag
DSPI_B_ISR_RFDF	0x0870	135	—	DSPI_B.DSPI_ISR[RFDF]	DSPI_B Receive FIFO Drain Flag
DSPI_C_ISR_OVER	0x0880	136	—	DSPI_C.DSPI_ISR[TFUF] || DSPI_C.DSPI_ISR[RFOF]	DSPI_C combined overrun interrupt request of the Transmit FIFO Underflow and Receive FIFO Overflow interrupt requests
DSPI_C_ISR_EOQF	0x0890	137	—	DSPI_C.DSPI_ISR[EOQF]	DSPI_C transmit FIFO End Of Queue Flag
DSPI_C_ISR_TFFF	0x08a0	138	—	DSPI_C.DSPI_ISR[TFFF]	DSPI_C Transmit FIFO Fill Flag
DSPI_C_ISR_TCF	0x08b0	139	—	DSPI_C.DSPI_ISR[TCF]	DSPI_C Transfer Complete Flag
DSPI_C_ISR_RFDF	0x08c0	140	—	DSPI_C.DSPI_ISR[RFDF]	DSPI_C Receive FIFO Drain Flag
Reserved	0x08d0	141	—	Reserved	Reserved
Reserved	0x08e0	142	—	Reserved	Reserved
Reserved	0x08f0	143	—	Reserved	Reserved
Reserved	0x0900	144	—	Reserved	Reserved
Reserved	0x0910	145	—	Reserved	Reserved

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
eSCI_A_COMB	0x0920	146	—	eSCI_A.SCISR1[TDRE] || eSCI_A.SCISR1[TC] || eSCI_A.SCISR1[RDRF] || eSCI_A.SCISR1[IDLE] || eSCI_A.SCISR1[OR] || eSCI_A.SCISR1[NF] || eSCI_A.SCISR1[FE] || eSCI_A.SCISR1[PF] || eSCI_A.SCISR2[BERR] || eSCI_A.LINSTAT1[RXRDY] || eSCI_A.LINSTAT1[TXRDY] || eSCI_A.LINSTAT1[LWAKE] || eSCI_A.LINSTAT1[STO] || eSCI_A.LINSTAT1[PBERR] || eSCI_A.LINSTAT1[CERR] || eSCI_A.LINSTAT1[CKERR] ||	eSCI_A combined interrupt request of the eSCI Status Register 1 Transmit Data Register Empty, Transmit Complete, Receive Data Register Full, Idle line, Overrun, Noise, Frame Error, and Parity Error interrupt requests, eSCI Status Register 2 Bit Error interrupt request, LIN Status Register 1 Receive Data Ready, Transmit Data Ready, Received LIN Wakeup Signal, Slave
Reserved	0x0930	147	—	Reserved	Reserved
Reserved	0x0940	148	—	Reserved	Reserved
eSCI_B_COMB	0x0950	149	—	eSCI_B.SCISR1[TDRE] || eSCI_B.SCISR1[TC] || eSCI_B.SCISR1[RDRF] || eSCI_B.SCISR1[IDLE] || eSCI_B.SCISR1[OR] || eSCI_B.SCISR1[NF] || eSCI_B.SCISR1[FE] || eSCI_B.SCISR1[PF] || eSCI_B.SCISR2[BERR] || eSCI_B.LINSTAT1[RXRDY] || eSCI_B.LINSTAT1[TXRDY] || eSCI_B.LINSTAT1[LWAKE] || eSCI_B.LINSTAT1[STO] || eSCI_B.LINSTAT1[PBERR] || eSCI_B.LINSTAT1[CERR] || eSCI_B.LINSTAT1[CKERR] ||	eSCI_B combined interrupt request of the eSCI Status Register 1 Transmit Data Register Empty, Transmit Complete, Receive Data Register Full, Idle line, Noise, Frame Error, and Parity Error Overrun interrupt requests, eSCI Status Register 2 Bit Error interrupt request, LIN Status Register 1 Receive Data Ready, Transmit Data Ready, Received LIN Wakeup Signal, Slave TimeOut,
Reserved	0x0960	150	—	Reserved	Reserved
Reserved	0x0970	151	—	Reserved	Reserved
FLEXCAN_A_ESR_ BOFF_INT	0x0980	152	—	FLEXCAN_A.ESR[BOFF_INT]	FlexCAN_A Bus Off Interrupt
				FLEXCAN_A.ESR[TWRN_INT]	FlexCAN_A Transmit Warning Interrupt
				FLEXCAN_A.ESR[RWRN_INT]	FlexCAN_A Receive Warning Interrupt

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
FLEXCAN_A_ESR_ ERR_INT	0x0990	153	—	FLEXCAN_A.ESR[ERR_INT]	FLEXCAN_A Error Interrupt
FLEXCAN_A_WAKEUP_ INT	0x09a0	154	—	FLEXCAN_A.IPI_INT_WAKEIN	FLEXCAN_A wake up Interrupt
FLEXCAN_A_IFLAG1_ BUF0I	0x09b0	155	—	FLEXCAN_A.IFLAG1[BUF0I]	FLEXCAN_A Buffer 0 Interrupt
FLEXCAN_A_IFLAG1_ BUF1I	0x09c0	156	—	FLEXCAN_A.IFLAG1[BUF1I]	FLEXCAN_A Buffer 1 Interrupt
FLEXCAN_A_IFLAG1_ BUF2I	0x09d0	157	—	FLEXCAN_A.IFLAG1[BUF2I]	FLEXCAN_A Buffer 2 Interrupt
FLEXCAN_A_IFLAG1_ BUF3I	0x09e0	158	—	FLEXCAN_A.IFLAG1[BUF3I]	FLEXCAN_A Buffer 3 Interrupt
FLEXCAN_A_IFLAG1_ BUF4I	0x09f0	159	—	FLEXCAN_A.IFLAG1[BUF4I]	FLEXCAN_A Buffer 4 Interrupt
FLEXCAN_A_IFLAG1_ BUF5I	0x0a00	160	—	FLEXCAN_A.IFLAG1[BUF5I]	FLEXCAN_A Buffer 5 Interrupt
FLEXCAN_A_IFLAG1_ BUF6I	0x0a10	161	—	FLEXCAN_A.IFLAG1[BUF6I]	FLEXCAN_A Buffer 6 Interrupt
FLEXCAN_A_IFLAG1_ BUF7I	0x0a20	162	—	FLEXCAN_A.IFLAG1[BUF7I]	FLEXCAN_A Buffer 7 Interrupt
FLEXCAN_A_IFLAG1_ BUF8I	0x0a30	163	—	FLEXCAN_A.IFLAG1[BUF8I]	FLEXCAN_A Buffer 8 Interrupt
FLEXCAN_A_IFLAG1_ BUF9I	0x0a40	164	—	FLEXCAN_A.IFLAG1[BUF9I]	FLEXCAN_A Buffer 9 Interrupt
FLEXCAN_A_IFLAG1_ BUF10I	0x0a50	165	—	FLEXCAN_A.IFLAG1[BUF10I]	FLEXCAN_A Buffer 10 Interrupt
FLEXCAN_A_IFLAG1_ BUF11I	0x0a60	166	—	FLEXCAN_A.IFLAG1[BUF11I]	FLEXCAN_A Buffer 11 Interrupt
FLEXCAN_A_IFLAG1_ BUF12I	0x0a70	167	—	FLEXCAN_A.IFLAG1[BUF12I]	FLEXCAN_A Buffer 12 Interrupt
FLEXCAN_A_IFLAG1_ BUF13I	0x0a80	168	—	FLEXCAN_A.IFLAG1[BUF13I]	FLEXCAN_A Buffer 13 Interrupt
FLEXCAN_A_IFLAG1_ BUF14I	0x0a90	169	—	FLEXCAN_A.IFLAG1[BUF14I]	FLEXCAN_A Buffer 14 Interrupt
FLEXCAN_A_IFLAG1_ BUF15I	0x0aa0	170	—	FLEXCAN_A.IFLAG1[BUF15I]	FLEXCAN_A Buffer 15 Interrupt
FLEXCAN_A_IFLAG1_ BUF31_16I	0x0ab0	171	—	FLEXCAN_A.IFLAG1[BUF31I:BU F16I]	FLEXCAN_A Buffers 31–16 Interrupts
FLEXCAN_A_IFLAG2_ BUF63_32I	0x0ac0	172	—	FLEXCAN_A.IFLAG2 [BUF63I:BUF32I]	FLEXCAN_A Buffers 63–32 Interrupts

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
FLEXCAN_C_ESR_BOFF_INT	0x0ad0	173	—	FLEXCAN_C.ESR[BOFF_INT]	FlexCAN_C Bus Off Interrupt
				FLEXCAN_C.ESR[TWRN_INT]	FlexCAN_C Transmit Warning Interrupt
				FLEXCAN_C.ESR[RWRN_INT]	FlexCAN_C Receive Warning Interrupt
FLEXCAN_C_ESR_ERR_INT	0x0ae0	174	—	FLEXCAN_C.ESR[ERR_INT]	FLEXCAN_C Error Interrupt
FLEXCAN_C_WAKEUP_INT	0x0af0	175	—	FLEXCAN_C.IPI_INT_WAKEIN	FLEXCAN_C wake up Interrupt
FLEXCAN_C_IFLAG1_BUF0I	0x0b00	176	—	FLEXCAN_C.IFLAG1[BUF0I]	FLEXCAN_C Buffer 0 Interrupt
FLEXCAN_C_IFLAG1_BUF1I	0x0b10	177	—	FLEXCAN_C.IFLAG1[BUF1I]	FLEXCAN_C Buffer 1 Interrupt
FLEXCAN_C_IFLAG1_BUF2I	0x0b20	178	—	FLEXCAN_C.IFLAG1[BUF2I]	FLEXCAN_C Buffer 2 Interrupt
FLEXCAN_C_IFLAG1_BUF3I	0x0b30	179	—	FLEXCAN_C.IFLAG1[BUF3I]	FLEXCAN_C Buffer 3 Interrupt
FLEXCAN_C_IFLAG1_BUF4I	0x0b40	180	—	FLEXCAN_C.IFLAG1[BUF4I]	FLEXCAN_C Buffer 4 Interrupt
FLEXCAN_C_IFLAG1_BUF5I	0x0b50	181	—	FLEXCAN_C.IFLAG1[BUF5I]	FLEXCAN_C Buffer 5 Interrupt
FLEXCAN_C_IFLAG1_BUF6I	0x0b60	182	—	FLEXCAN_C.IFLAG1[BUF6I]	FLEXCAN_C Buffer 6 Interrupt
FLEXCAN_C_IFLAG1_BUF7I	0x0b70	183	—	FLEXCAN_C.IFLAG1[BUF7I]	FLEXCAN_C Buffer 7 Interrupt
FLEXCAN_C_IFLAG1_BUF8I	0x0b80	184	—	FLEXCAN_C.IFLAG1[BUF8I]	FLEXCAN_C Buffer 8 Interrupt
FLEXCAN_C_IFLAG1_BUF9I	0x0b90	185	—	FLEXCAN_C.IFLAG1[BUF9I]	FLEXCAN_C Buffer 9 Interrupt
FLEXCAN_C_IFLAG1_BUF10I	0x0ba0	186	—	FLEXCAN_C.IFLAG1[BUF10I]	FLEXCAN_C Buffer 10 Interrupt
FLEXCAN_C_IFLAG1_BUF11I	0x0bb0	187	—	FLEXCAN_C.IFLAG1[BUF11I]	FLEXCAN_C Buffer 11 Interrupt
FLEXCAN_C_IFLAG1_BUF12I	0x0bc0	188	—	FLEXCAN_C.IFLAG1[BUF12I]	FLEXCAN_C Buffer 12 Interrupt
FLEXCAN_C_IFLAG1_BUF13I	0x0bd0	189	—	FLEXCAN_C.IFLAG1[BUF13I]	FLEXCAN_C Buffer 13 Interrupt
FLEXCAN_C_IFLAG1_BUF14I	0x0be0	190	—	FLEXCAN_C.IFLAG1[BUF14I]	FLEXCAN_C Buffer 14 Interrupt

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
FLEXCAN_C_IFLAG1_BUF15I	0x0bf0	191	—	FLEXCAN_C.IFLAG1[BUF15I]	FLEXCAN_C Buffer 15 Interrupt
FLEXCAN_C_IFLAG1_BUF31_16I	0x0c00	192	—	FLEXCAN_C.IFLAG1[BUF31I:BUF16I]	FLEXCAN_C Buffers 31–16 Interrupts
Reserved	0x0c10	193	—	Reserved	Reserved
Reserved	0x0c20	194	—	Reserved	Reserved
Reserved	0x0c30	195	—	Reserved	Reserved
Reserved	0x0c40	196	—	Reserved	Reserved
DECFIL_A_In	0x0c50	197	—	ipi_int_dec_filter_in	Decimation A Input (Fill)
DECFIL_A_Out	0x0c60	198	—	ipi_int_dec_filter_out	Decimation A Output (Drain)
DECFIL_A_Err	0x0c70	199	—	ipi_int_dec_filter	Decimation A Error
STM0	0x0c80	200	—	stm_ipi_int0	STM[0]
STM1_3	0x0c90	201	—	STM1_or_STM2_or_STM3	STM[1:3]
Reserved	0x0ca0	202	—	Reserved	Reserved
Reserved	0x0cb0	203	—	Reserved	Reserved
Reserved	0x0cc0	204	—	Reserved	Reserved
Reserved	0x0cd0	205	—	Reserved	Reserved
Reserved	0x0ce0	206	—	Reserved	Reserved
Reserved	0x0cf0	207	—	Reserved	Reserved
Reserved	0x0d00	208	—	Reserved	Reserved
eMIOS_FLAG_F23	0x0d10	209	—	eMIOS.eMIOSFLAG[F23]	eMIOS channel 23 Flag
Reserved	0x0d20	210	—	Reserved	Reserved
Reserved	0x0d30	211	—	Reserved	Reserved
Reserved	0x0d40	212	—	Reserved	Reserved
Reserved	0x0d50	213	—	Reserved	Reserved
Reserved	0x0d60	214	—	Reserved	Reserved
Reserved	0x0d70	215	—	Reserved	Reserved
Reserved	0x0d80	216	—	Reserved	Reserved
Reserved	0x0d90	217	—	Reserved	Reserved
Reserved	0x0da0	218	—	Reserved	Reserved
Reserved	0x0db0	219	—	Reserved	Reserved
Reserved	0x0dc0	220	—	Reserved	Reserved
Reserved	0x0dd0	221	—	Reserved	Reserved

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
Reserved	0x0de0	222	—	Reserved	Reserved
Reserved	0x0df0	223	—	Reserved	Reserved
Reserved	0x0e00	224	—	Reserved	Reserved
Reserved	0x0e10	225	—	Reserved	Reserved
Reserved	0x0e20	226	—	Reserved	Reserved
Reserved	0x0e30	227	—	Reserved	Reserved
Reserved	0x0e40	228	—	Reserved	Reserved
Reserved	0x0e50	229	—	Reserved	Reserved
Reserved	0x0e60	230	—	Reserved	Reserved
Reserved	0x0e70	231	—	Reserved	Reserved
Reserved	0x0e80	232	—	Reserved	Reserved
Reserved	0x0e90	233	—	Reserved	Reserved
Reserved	0x0ea0	234	—	Reserved	Reserved
Reserved	0x0eb0	235	—	Reserved	Reserved
Reserved	0x0ec0	236	—	Reserved	Reserved
Reserved	0x0ed0	237	—	Reserved	Reserved
Reserved	0x0ee0	238	—	Reserved	Reserved
Reserved	0x0ef0	239	—	Reserved	Reserved
Reserved	0x0f00	240	—	Reserved	Reserved
Reserved	0x0f10	241	—	Reserved	Reserved
Reserved	0x0f20	242	—	Reserved	Reserved
Reserved	0x0f30	243	—	Reserved	Reserved
Reserved	0x0f40	244	—	Reserved	Reserved
Reserved	0x0f50	245	—	Reserved	Reserved
Reserved	0x0f60	246	—	Reserved	Reserved
Reserved	0x0f70	247	—	Reserved	Reserved
Reserved	0x0f80	248	—	Reserved	Reserved
Reserved	0x0f90	249	—	Reserved	Reserved
Reserved	0x0fa0	250	—	Reserved	Reserved
Reserved	0x0fb0	251	—	Reserved	Reserved
Reserved	0x0fc0	252	—	Reserved	Reserved
Reserved	0x0fd0	253	—	Reserved	Reserved
Reserved	0x0fe0	254	—	Reserved	Reserved
Reserved	0x0ff0	255	—	Reserved	Reserved

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
Reserved	0x1000	256	—	Reserved	Reserved
Reserved	0x1010	257	—	Reserved	Reserved
Reserved	0x1020	258	—	Reserved	Reserved
Reserved	0x1030	259	—	Reserved	Reserved
Reserved	0x1040	260	—	Reserved	Reserved
Reserved	0x1050	261	—	Reserved	Reserved
Reserved	0x1060	262	—	Reserved	Reserved
Reserved	0x1070	263	—	Reserved	Reserved
Reserved	0x1080	264	—	Reserved	Reserved
Reserved	0x1090	265	—	Reserved	Reserved
Reserved	0x10a0	266	—	Reserved	Reserved
Reserved	0x10b0	267	—	Reserved	Reserved
Reserved	0x10c0	268	—	Reserved	Reserved
Reserved	0x10d0	269	—	Reserved	Reserved
Reserved	0x10e0	270	—	Reserved	Reserved
Reserved	0x10f0	271	—	Reserved	Reserved
Reserved	0x1100	272	—	Reserved	Reserved
Reserved	0x1110	273	—	Reserved	Reserved
Reserved	0x1120	274	—	Reserved	Reserved
Reserved	0x1130	275	—	Reserved	Reserved
Reserved	0x1140	276	—	Reserved	Reserved
Reserved	0x1150	277	—	Reserved	Reserved
Reserved	0x1160	278	—	Reserved	Reserved
Reserved	0x1170	279	—	Reserved	Reserved
Reserved	0x1180	280	—	Reserved	Reserved
Reserved	0x1190	281	—	Reserved	Reserved
Reserved	0x11a0	282	—	Reserved	Reserved
Reserved	0x11b0	283	—	Reserved	Reserved
Reserved	0x11c0	284	—	Reserved	Reserved
Reserved	0x11d0	285	—	Reserved	Reserved
Reserved	0x11e0	286	—	Reserved	Reserved
Reserved	0x11f0	287	—	Reserved	Reserved
Reserved	0x1200	288	—	Reserved	Reserved
Reserved	0x1210	289	—	Reserved	Reserved

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
Reserved	0x1220	290	—	Reserved	Reserved
Reserved	0x1230	291	—	Reserved	Reserved
Reserved	0x1240	292	—	Reserved	Reserved
Reserved	0x1250	293	—	Reserved	Reserved
Reserved	0x1260	294	—	Reserved	Reserved
Reserved	0x1270	295	—	Reserved	Reserved
Reserved	0x1280	296	—	Reserved	Reserved
Reserved	0x1290	297	—	Reserved	Reserved
Reserved	0x12a0	298	—	Reserved	Reserved
Reserved	0x12b0	299	—	Reserved	Reserved
Reserved	0x12c0	300	—	Reserved	Reserved
PIT0	0x12d0	301	—	ipi_int_pit[0]	PIT[0]
PIT1	0x12e0	302	—	ipi_int_pit[1]	PIT[1]
PIT2	0x12f0	303	—	ipi_int_pit[2]	PIT[2]
PIT3	0x1300	304	—	ipi_int_pit[3]	PIT[3]
RTI	0x1310	305	—	ipi_int_rti	RTI
PMC	0x1320	306	—	ipi_int_lvi	PMC
FLASH	0x1330	307	—	ipi_int_flsh	ECC-Correction
Reserved	0x1340	308	—	Reserved	Reserved
Reserved	0x1350	309	—	Reserved	Reserved
Reserved	0x1360	310	—	Reserved	Reserved
Reserved	0x1370	311	—	Reserved	Reserved
Reserved	0x1380	312	—	Reserved	Reserved
Reserved	0x1390	313	—	Reserved	Reserved
Reserved	0x13a0	314	—	Reserved	Reserved
Reserved	0x13b0	315	—	Reserved	Reserved
Reserved	0x13c0	316	—	Reserved	Reserved
Reserved	0x13d0	317	—	Reserved	Reserved
Reserved	0x13e0	318	—	Reserved	Reserved
Reserved	0x13f0	319	—	Reserved	Reserved
Reserved	0x1400	320	—	Reserved	Reserved
Reserved	0x1410	321	—	Reserved	Reserved
Reserved	0x1420	322	—	Reserved	Reserved
Reserved	0x1430	323	—	Reserved	Reserved

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
Reserved	0x1440	324	—	Reserved	Reserved
Reserved	0x1450	325	—	Reserved	Reserved
Reserved	0x1460	326	—	Reserved	Reserved
Reserved	0x1470	327	—	Reserved	Reserved
Reserved	0x1480	328	—	Reserved	Reserved
Reserved	0x1490	329	—	Reserved	Reserved
Reserved	0x14a0	330	—	Reserved	Reserved
Reserved	0x14b0	331	—	Reserved	Reserved
Reserved	0x14c0	332	—	Reserved	Reserved
Reserved	0x14d0	333	—	Reserved	Reserved
Reserved	0x14e0	334	—	Reserved	Reserved
Reserved	0x14f0	335	—	Reserved	Reserved
Reserved	0x1500	336	—	Reserved	Reserved
Reserved	0x1510	337	—	Reserved	Reserved
Reserved	0x1520	338	—	Reserved	Reserved
Reserved	0x1530	339	—	Reserved	Reserved
Reserved	0x1540	340	—	Reserved	Reserved
Reserved	0x1550	341	—	Reserved	Reserved
Reserved	0x1560	342	—	Reserved	Reserved
Reserved	0x1570	343	—	Reserved	Reserved
Reserved	0x1580	344	—	Reserved	Reserved
Reserved	0x1590	345	—	Reserved	Reserved
Reserved	0x15a0	346	—	Reserved	Reserved
Reserved	0x15b0	347	—	Reserved	Reserved
Reserved	0x15c0	348	—	Reserved	Reserved
Reserved	0x15d0	349	—	Reserved	Reserved
Reserved	0x15e0	350	—	Reserved	Reserved
Reserved	0x15f0	351	—	Reserved	Reserved
Reserved	0x1600	352	—	Reserved	Reserved
Reserved	0x1610	353	—	Reserved	Reserved
Reserved	0x1620	354	—	Reserved	Reserved
Reserved	0x1630	355	—	Reserved	Reserved
Reserved	0x1640	356	—	Reserved	Reserved
Reserved	0x1650	357	—	Reserved	Reserved

Table 115. Interrupt summary (continued)

Interrupt	Offset	Vector	Priority (1)	Source	Description
STM1	0x1660	358	—	stm_ipi_int3	STM[1]
STM2	0x1670	359	—	stm_ipi_int2	STM[2]
STM3	0x1680	360	—	stm_ipi_int3	STM[3]
Reserved	0x1690	361	—	Reserved	Reserved
Reserved	0x1700	362	—	Reserved	Reserved
Reserved	0x1710	363	—	Reserved	Reserved

1. The priorities are selected in INTC_PSR*x*_*x*, where the specific select register is assigned according to the vector.

15 System Integration Unit (SIU)

15.1 Overview

The System Integration Unit (SIU) controls reset configuration, pad configuration, external interrupt, general purpose I/O (GPIO), internal peripheral multiplexing, and the 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 MCU I/O pins. The reset controller performs reset monitoring of internal and external reset sources, and drives the RSTOUT pin. The SIU is accessed by the e200z3 core through the peripheral bus.

15.2 Features

- System configuration
  - MCU reset configuration via external pins
  - Pad configuration control
- System reset monitoring and generation
  - Power-on reset support
  - Reset Status Register provides last reset source to software
  - Reset source identification by reset timing
  - Glitch detection on reset input
  - Software controlled reset assertion
- External interrupt
  - 11 interrupt requests
  - 1 Non-Maskable/Critical Interrupt request (NMI)
  - Rising or falling edge event detection
  - Programmable digital filter for glitch rejection
- GPIO
  - GPIO function on 80 I/O pins
  - Dedicated input and output registers for each GPIO pin
- External multiplexing
  - Allows serial and parallel chaining of DSPIs
  - Allows flexible selection of eQADC trigger inputs
  - Allows selection of interrupt requests between external pins and DSPI

15.3 Modes of operation

15.3.1 Normal Mode

In normal mode, the SIU provides the register interface and logic that controls system configuration, the reset controller, and GPIO.

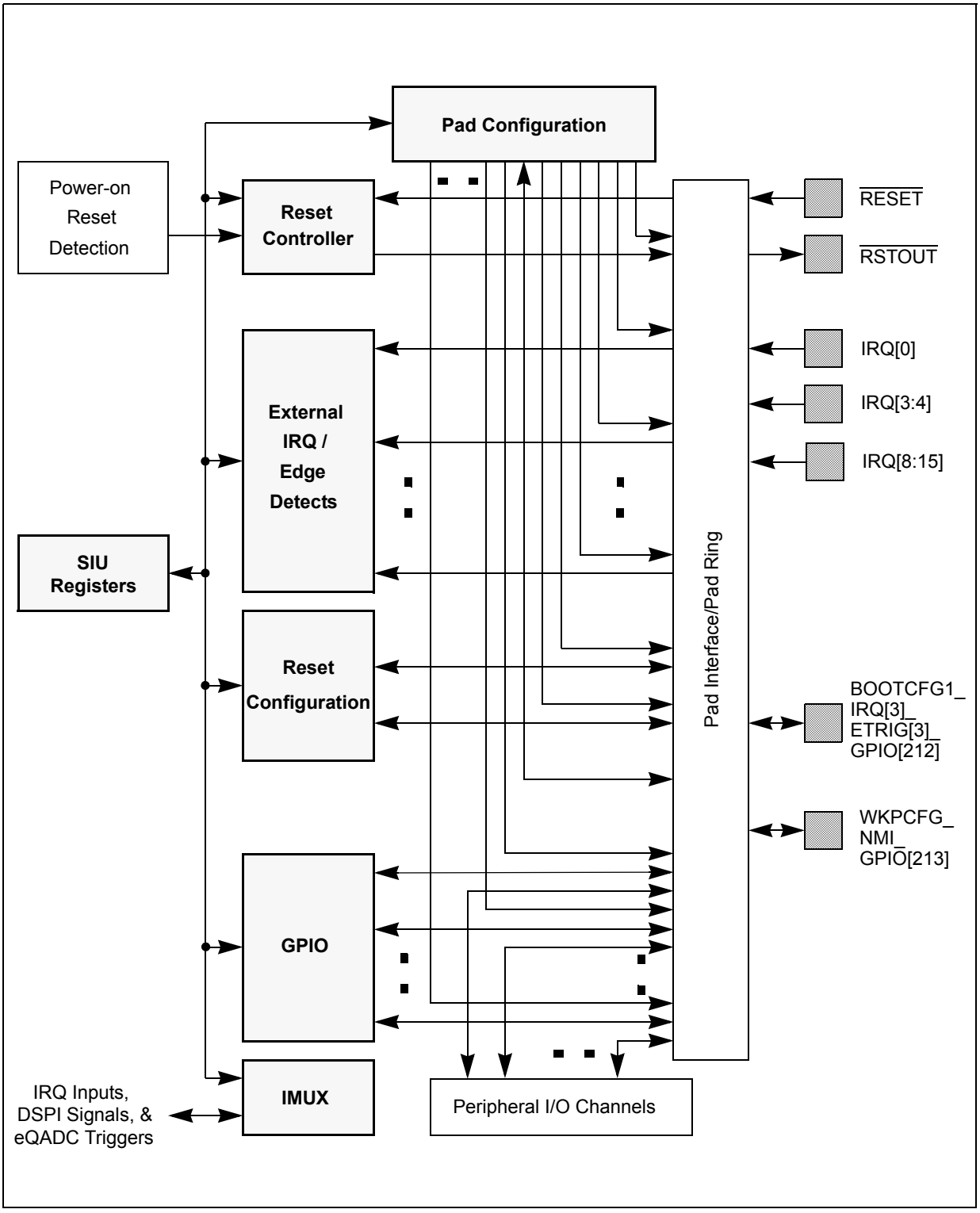
15.3.2 Debug Mode

SIU operation in debug mode is identical to operation in normal mode.

15.4 Block diagram

Figure 130 is a block diagram of the SIU. The signals shown are external pins to the device. Note that the Power-on Reset Detection block, Pad Interface/Pad Ring block, and Peripheral I/O Channels are external to the SIU.

Figure 130. SIU block diagram



15.5 Signal description

Table 116 lists the external pins used by the SIU.

Table 116. SIU signal properties

Name	I/O type	Pad type	Function	Pull up/down(1)
Resets				
RESET	Input	—	Reset Input	Up
RSTOUT	Output	Slow	Reset Output	Up
System configuration				
BOOTCFG1_ IRQ[3]_ ETRIG[3]_ GPIO[212]	Input Input Input I/O	Slow	Boot Configuration Input / External Interrupt Request / eQADC Trigger Input General Purpose I/O	Down — Up Up/Down
WKPCFG_ NMI_ GPIO[213]	Input Input I/O	Slow	Weak Pull Configuration Pin / Non-Maskable Interrupt / General Purpose I/O	Up — Up/Down
GPIO configuration				
GPIO[0:213]	I/O	Slow	General Purpose I/O	Up/Down
External interrupts				
IRQ[0, 3:4, 8:15]	Input	Slow	External Interrupt Request Input	—(2)

1. Internal weak pull up/down. The reset weak pull up/down state is given by the pull up/down state for the primary pin function. For example, the reset weak pull up/down state of the BOOTCFG1_IRQ3_ETRIG3_GPIO[212] pin is weak pull down enabled.
2. The weak pull up/down state at reset for the IRQ[0:15] depends on the pins that they are shared with. The weak pull up/down state for these pins is as follows: IRQ[0,4,12,13,14] - Up, IRQ[3,15] - Down, IRQ[8:11] - WKPCFG.

15.6 Detailed signal descriptions

15.6.1 $\overline{\text{RESET}}$ — Reset Input

The $\overline{\text{RESET}}$ pin is an active low input. The $\overline{\text{RESET}}$ pin is asserted by an external device during a power-on or external reset. The internal reset signal asserts only if the $\overline{\text{RESET}}$ pin asserts for 10 clock cycles. Assertion of the $\overline{\text{RESET}}$ pin while the device is in reset causes the reset cycle to start over. The $\overline{\text{RESET}}$ pin has a glitch detector which detects spikes greater than two clock cycles in duration that fall below the switch point of the input buffer logic of the VDDEH input pins. The switch point lies between the maximum VIL and minimum VIH specifications for the VDDEH input pins.

15.6.2 $\overline{\text{RSTOUT}}$ — Reset Output

The $\overline{\text{RSTOUT}}$ pin is an active low output that uses a push/pull configuration. The $\overline{\text{RSTOUT}}$ pin is driven to the low state by the MCU for all internal and external reset sources. There is a delay between initiation of the reset and the assertion of the RSTOUT pin. See *Section 4.3.2: RSTOUT* for details.

15.6.3 GPIO[0:213] — General Purpose I/O Pins

The GPIO pins are generally multiplexed with other I/O pin functions. Each GPIO input and output is separately controlled by an 8-bit input (GPDI) or output (GPDO) register. See *Section 15.8.13.82: Pad Configuration Register 350 – 381 (SIU_PCR350 – SIU_PCR381)*, *Section 15.8.13.83: Pad Configuration Register 382 – 389 (SIU_PCR382 – SIU_PCR389)*, *Section 15.8.13.84: Pad Configuration Register 390 – 413 (SIU_PCR390 – SIU_PCR413)*, and *Section 15.8.15: GPO Data Output Registers (SIU_GPDO350 – SIU_GPDO413)*.

15.6.4 BOOTCFG1 (BOOTCFG1_IRQ[3]_ETRIG[3]_GPIO[212]) — Boot Configuration Pin

The boot configuration pin specify the boot mode initiated by the BAM program. BOOTCFG1 is an input pin that is sampled 4 clock cycles before the negation of the RSTOUT pin, and the value latched is stored in the Reset Status Register (RSR). This occurs for all reset sources except a Debug Port Reset and a Software External Reset.

15.6.5 WKPCFG (WKPCFG_NMI_DSPI_B_SOUT_GPIO[213]) — I/O Pin Weak Pull Up Reset Configuration Pin

The WKPCFG pin is applied at the assertion of the internal reset signal (assertion of RSTOUT), and is sampled 4 clock cycles before the negation of the RSTOUT pin. The value is used to configure whether the eTPU and eMIOS pins are connected to internal weak pull up or weak pull down devices after reset. The value latched on the WKPCFG pin at reset is stored in the Reset Status Register (RSR), and is updated for all reset sources except the Debug Port Reset and Software External Reset.

15.6.6 IRQ[0:15] — External Interrupt Request Input Pins

The IRQ[0:15] pins connect to the SIU IRQ inputs. The External IRQ Input Select Register (EIISR) is used to select the IRQ[0:15] pins as inputs to the IRQs.

15.7 Memory map

Table 117 is the address map for the SIU registers.

Note: The SIU_BASE address is 0xC3F9_0000.

Table 117. SIU address map

Address	Use	Bits per register	Location
SIU_BASE	MCU ID Register 2 (SIU_MIDR2)	32	*on page 392*
SIU_BASE+0x4	MCU ID Register (SIU_MIDR)	32	*on page 394*
SIU_BASE+0x8	Reserved	—	—
SIU_BASE+0xC	Reset Status Register (SIU_RSR)	32	*on page 396*
SIU_BASE+0x10	System Reset Control Register (SIU_SRCR)	32	*on page 397*
SIU_BASE+0x14	External Interrupt Status Register (SIU_EISR)	32	*on page 399*
SIU_BASE+0x18	DMA/Interrupt Request Enable Register (SIU_DIRER)	32	*on page 399*

Table 117. SIU address map (continued)

Address	Use	Bits per register	Location
SIU_BASE+0x1C	DMA/Interrupt Request Select Register (SIU_DIRSR)	32	*on page 400*
SIU_BASE+0x20	Overrun Status Register (SIU_OSR)	32	*on page 401*
SIU_BASE+0x24	Overrun Request Enable Register (SIU_ORER)	32	*on page 402*
SIU_BASE+0x28	IRQ Rising-Edge Event Enable Register (SIU_IREER)	32	*on page 403*
SIU_BASE+0x2C	External IRQ Falling-Edge Event Enable Register (SIU_IFEER)	32	*on page 403*
SIU_BASE+0x30	External IRQ Digital Filter Register (SIU_IDFR)	32	*on page 404*
SIU_BASE+0x34 – SIU_BASE+0x3F	Reserved	—	—
SIU_BASE+0x40 – SIU_BASE+0x2F2	Pad Configuration Register 0 (SIU_PCR0) – Pad Configuration Register 345 (SIU_PCR345)(1)	16	*on page 405*
SIU_BASE+0x2FC – SIU_BASE+0x37A	Pad Configuration Register 350 (SIU_PCR350) – Pad Configuration Register 413 (SIU_PCR413)	16	*on page 405*
SIU_BASE+0x37C – SIU_BASE+0x5FF	Reserved	—	—
SIU_BASE+0x600 – SIU_BASE+0x79C	GPIO Pin Data Output Register 0 – 3 (SIU_GPDO0_3) – GPIO Pin Data Output Register 412 – 413 (SIU_GPDO412_413)(1)	8	*on page 449*
SIU_BASE+0x6D8 – SIU_BASE+0x7FF	Reserved	—	—
SIU_BASE+0x800 – SIU_BASE+0x8E8	GPIO Pin Data Input Register 0 – 3 (SIU_GPDI0_3) – GPIO Pin Data Input Register 232 – 233 (SIU_GPDI232_233)(1)	8	*on page 452*
SIU_BASE+0x8D8 – SIU_BASE+0x8FF	Reserved	—	—
SIU_BASE+0x900	eQADC Trigger Input Select Register (SIU_ETISR)(2)	32	*on page 453*
SIU_BASE+0x904	External IRQ Input Select Register (SIU_EIISR)(3)	32	*on page 455*
SIU_BASE+0x908	DSPI Input Select Register (SIU_DISR)(4)	32	*on page 458*
SIU_BASE+0x90C	IMUX Select Register 3 (SIU_ISEL3)	32	*on page 460*
SIU_BASE+0x910 – SIU_BASE+0x91C	Reserved	—	—
SIU_BASE+0x920	IMUX Select Register 8 (SIU_ISEL8)	32	*on page 466*
SIU_BASE+0x924	IMUX Select Register 9 (SIU_ISEL9)	32	*on page 468*
SIU_BASE+0x928 – SIU_BASE+0x97F	Reserved	—	—
SIU_BASE+0x980	Chip Configuration Register (SIU_CCR)	32	*on page 470*
SIU_BASE+0x984	External Clock Control Register (SIU_ECCR)	32	*on page 471*
SIU_BASE+0x988	Compare A High Register (SIU_CARH)	32	*on page 472*
SIU_BASE+0x98C	Compare A Low Register (SIU_CARL)	32	*on page 473*

Table 117. SIU address map (continued)

Address	Use	Bits per register	Location
SIU_BASE+0x990	Compare B High Register (SIU_CBRH)	32	on page 473
SIU_BASE+0x994	Compare B Low Register (SIU_CBRL)	32	on page 473
SIU_BASE+0x998	Reserved	—	—
SIU_BASE+0x9A0	System Clock Register (SIU_SYSDIV)	8	on page 474
SIU_BASE+0x9A4	Halt Register (SIU_HLT)	32	on page 475
SIU_BASE+0x9A8	Halt Acknowledge Register (SIU_HLTACK)	32	on page 477
SIU_BASE+0x9AC – SIU_BASE+0x9FF	Reserved	—	—

1. Gaps exist in this memory space where I/O pins are not available in the specified package.
2. The ETISR is sometimes referred to as ISEL0.
3. The EIISR is sometimes referred to as ISEL1.
4. The DISR is sometimes referred to as ISEL2.

15.8 Register descriptions

Note: The convention is to not use the module name prefix ("SIU_") for register names in software.

15.8.1 MCU ID Register 2 (SIU_MIDR2)

The MCU ID Register 2 contains additional configuration information about the device. *Figure 131* shows the bit mapping of MCU ID Register 2 contents and fields descriptions are given in *Table 118*.

SIU_BASE + 0x0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Function	S_F	FLASH_SIZE_1				FLASH_SIZE_2				TEMP_ RANGE		Reserved				SUPPLY
Status	C	C	C	C	C	C	C	C	C	C	C	S	S	S	S	C
Hard coded	S_F (1)	0	1	1	0	0	1	0	0			0	0	0	0	0
		6				4										

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Function	PART_NUMBER (ASCII Character)								Reserved			EE	Reserved			FR
Status	C	C	C	C	C	C	C	C	S	S	S	C	S	S	S	C
Hard coded	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0
	M = 0x4D															

1. S_F set with metal option.

Figure 131. MCU ID Register 2 (SIU_MIDR2)

Table 118. SIU_MIDR2 field description

Bit	Name	Description
0	S_F	Identifies the manufacturer 0 reserved 1 ST
1–4	FLASH_SIZE_1	Defines major flash memory size (see *Table 119* for details)
5–8	FLASH_SIZE_2	Defines flash memory size, small granularity (see *Table 120* for details)
9–10	TEMP_RANGE	Defines maximum operating range
11–14	Reserved	Reserved for future enhancements
15	SUPPLY	Defines if the part is 5V or 3V 0 5V
16–23	PART_NUMBER	Contain the ASCII representation of the character that indicates the product family
24–26	Reserved	Reserved for future enhancements
27	EE	Indicates if Data Flash is present 1 Data Flash present 0 Data Flash not present
28–30	Reserved	Reserved for future enhancements
31	FR	Indicates if Data FlexRay is present 0 FlexRay not present

Table 119. Flash memory size

FLASH_SIZE_1 field	Size
0	16 KB
1	32 KB
2	64 KB
3	128 KB
4	256 KB
5	512 KB
6	1024 KB
7	2048 KB
8	4096 KB
…	...
n	2^{4+n} KB

Table 120. Flash memory size detailed(1)

FLASH_SIZE_2 field	Size
0h	0 × (FLASH_SIZE_1) / 8
1h	1 × (FLASH_SIZE_1) / 8
2h	2 × (FLASH_SIZE_1) / 8
...	
n	n × (FLASH_SIZE_1) / 8

1. Total flash memory size = (FLASH_SIZE_1) + (FLASH_SIZE_2)

Table 121. Flash size values by part number

Device MIDR	PARTNUM	FLASH_SIZE_1	FLASH_SIZE_2
SPC563M54	5632	5	4
SPC563M60	5633	6	0
SPC563M64	5634	6	4

15.8.2 MCU ID Register (SIU_MIDR)

The MCU ID Register contains the part number and the package ID of the device. *Figure 132* shows the bit mapping of MCU ID Register contents and fields descriptions are given in *Table 122*.

SIU_BASE + 0x4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Function	PARTNUM [0–15] (4 Digits)															
Hard coded	0	1	0	1	0	1	1	0	0	0	1	1	0	0	1	1
	5				6				3				3			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Function	CSP	PKG [0–4]					Reserved		MASKNUM [0–3] - Major				MASKNUM [0–3] - Minor			
Hard Coded	-(1)	0	1	1	0	1	0	0	—	—	—	—	—	—	—	—

1. Values corresponding to device packaging; see *Table 122: SIU_MIDR field descriptions*.

Figure 132. MCU ID Register (SIU_MIDR)

Table 122. SIU_MIDR field descriptions

Field	Description
0–15 PARTNUM [0–15]	Device part number: 0x5630 Please see *Table 123* for details on memory size.
16 CSP	CSP configuration: 1 = Calibration tool package 0 = standard QFP package
17–21 PKG [0–4]	Indicate the package the die is mounted in
22–23	Reserved
24–27 Major MASKNUM [0–3]	MCU major mask number. The reset value changes with each revision of the device.
28–31 Minor MASKNUM [0–3]	MCU minor mask number. The reset value changes with each revision of the device.

Table 123. Memory size core dependency

PARTNUM field	z0, z1	z3, z4, z5
0h	Reserved	Reserved
1h	128 KB	512 KB
2h	256 KB	768 KB
3h	320 KB / 384 KB	1024 KB
4h	512 KB	1.5 MB
5h	768 KB	2 MB
6h	1024 KB	3 MB
7h	1.5 MB	4 MB
8h	2 MB	6 MB
9h	3 MB	8 MB

Table 124. CSP and PKG[0–4]

Field values						Package
CSP	PKG[0]	PKG[1]	PKG[2]	PKG[3]	PKG[4]	
0	0	1	1	0	1	144 QFP
0	1	0	0	0	1	176 QFP
1	0	1	1	0	1	144 QFP(1)
1	1	0	0	0	1	176 QFP(1)

1. Emulated by calibration tool.

15.8.3 Reset Status Register (SIU_RSR)

The Reset Status Register (SIU_RSR) reflects the most recent source, or sources, of reset. This register contains one bit for each reset source, except JTAG reset. A bit set to logic one indicates the type of reset that occurred. Simultaneous reset requests cause more than one bit to set at the same time. Once set, the reset source bits in the SIU_RSR remain set until another reset occurs. A Software External Reset causes the SERF bit to be set, but no previously set bits in the SIU_RSR will be cleared.

The SIU_RSR also contains the values latched at the last reset on the WKPCFG and BOOTCFG[1] pins and a $\overline{\text{RESET}}$ input pin glitch flag. The reset glitch flag bit (RGF) is cleared by writing a one to the bit. A write of zero has no effect on the bit state. The register can be read at all times.

SIU_BASE + 0xC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PORS	ERS	LLRS	LCRS	WDRS	CRS	SWTRS	0	0	0	0	0	0	0	SSRS	SERF
W																
Re-set(1)	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	WKPCFG	0	0	0	0	0	0	0	0	0	0	0	ABR	BOOTCFG [0–1]	RGF
W															
Reset1	U(2)	0	0	0	0	0	0	0	0	0	0	0	U(3)	U(4)	0

= Unimplemented or Reserved

1. The reset values for this register are defined for power-on reset only.
2. The reset value of this bit is determined by the value latched on the associated pin at the negation of the last reset.
3. The reset value of this bit is determined by the inverse of the value latched on the associated EVTO pin.
4. The reset value of this bit is determined by the value latched on the associated pin at the negation of the last reset.

Figure 133. Reset Status Register (SIU_RSR)

Table 125. SIU_RSR field descriptions

Field	Description
0 PORS	Power-On Reset Status 1 A Power-On Reset has occurred. 0 No Power-On Reset has occurred.
1 ERS	External Reset Status 1 An External Reset has occurred. 0 No External Reset has occurred.
2 LLRS	Loss of Lock Reset Status 1 Loss of Lock Reset has occurred. 0 No Loss of Lock Reset has occurred.

Table 125. SIU_RSR field descriptions (continued)

Field	Description
3 LCRS	Loss of Clock Reset Status 1 A Loss of Clock Reset has occurred due to a loss of the reference or failure of the FMPLL. 0 No Loss of Clock Reset has occurred.
4 WDRS	Watchdog Timer/Debug Reset Status 1 A Watchdog Timer or Debug Reset has occurred. 0 No Watchdog Timer or Debug Reset has occurred.
5 CRS	Checkstop Reset Status 1 An enabled Checkstop Reset has occurred. 0 No enabled Checkstop Reset has occurred.
6 SWTRS	Software Watchdog Timer Reset Status 1 An enabled SWT Reset has occurred. 0 No enabled SWT Reset has occurred.
7–13	Reserved, should be cleared.
14 SSRS	Software System Reset Status 1 A Software System Reset has occurred. 0 No Software System Reset has occurred.
15 SERF	Software External Reset Flag 1 A Software External Reset has occurred. 0 No Software External Reset has occurred.
16 WKPCFG	Weak Pull Configuration Pin Status 1 WKPCFG pin latched during the last reset was logical one and weak pull up is the default setting. 0 WKPCFG pin latched during the last reset was logical zero and weak pull down is the default setting.
17–27	Reserved
28 ABR	Auto Baud Rate 1 Auto baud rate enabled 0 Auto baud rate disabled
29–30 BOOTCFG[0–1]	Reset Configuration Pin Status The BOOTCFG field holds the value of the BOOTCFG[1] pin that was latched on the last negation of the RSTOUT pin. The BOOTCFG field is used by the BAM program to configure the boot process. See *Section 20.5.2: BAM program operation*.
31 RGF	RESET Glitch Flag This bit is set by the MCU when the RESET pin is asserted for more than 2 clock cycles, but less than the minimum RESET assertion time of 10 consecutive clock cycles to cause a reset. This bit is cleared by the reset controller for a valid assertion of the RESET pin or a power-on reset or a write of one to the bit. 1 A glitch was detected on the RESET pin. 0 No glitch was detected on the RESET pin.

15.8.4 System Reset Control Register (SIU_SRCR)

The System Reset Control Register (SIU_SRCR) allows software to generate either a Software System Reset or Software External Reset. The Software System Reset causes an internal reset sequence, while the Software External Reset only causes the external

RSTOUT pin to be asserted for the predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*). When written to '1', field SIU_SRCR[SER] automatically clears after the clock count expires. If the value of SIU_SRCR[SER] is '1' and a '0' is written to the bit, the bit is cleared and the RSTOUT pin is negated regardless if the clock count has expired.

Field SIU_SRCR[CRE] allows software to enable a Checkstop Reset. If enabled, a Checkstop Reset will occur if the checkstop reset input to the reset controller is asserted. The Checkstop Reset is enabled by default.

SIU_BASE + 0x10

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

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

= Unimplemented or Reserved

1. The CRE bit is reset to 1'b1 by POR. Other reset sources do not reset the bit value.

Figure 134. System Reset Control Register (SIU_SRCR)

Table 126. SIU_SRCR field descriptions

Field	Description
0 SSR	Software System Reset Writing a one to this bit causes an internal reset and assertion of the RSTOUT pin. The bit is automatically cleared by all reset sources except the Software External Reset. 1 Generate a Software System Reset. 0 Do not generate a Software System Reset.
1 SER	Software External Reset Writing a one to this bit causes a Software External Reset. The RSTOUT pin is asserted for the predetermined number of clock cycles (refer to *Section 4.3.2: RSTOUT*), but the MCU is not reset. The bit is automatically cleared when the Software External Reset completes. 1 Generate a Software External Reset. 0 Do not generate a Software External Reset.
2–15	Reserved
16 CRE	Checkstop Reset Enable Writing a one to this bit enables a Checkstop Reset when the e200z335 core enters a checkstop state. The CRE bit defaults to Checkstop Reset enabled. If this bit is cleared, it remains cleared until the next POR. 1 A reset occurs when the e200z335 core enters a checkstop state. 0 No reset occurs when the e200z335 core enters a checkstop state.
17–31	Reserved

15.8.5 External Interrupt Status Register (SIU_EISR)

The External Interrupt Status Register is used to record edge triggered events on the IRQ0 – IRQ15 inputs to the SIU.

SIU_BASE + 0x14

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF15	EIF14	EIF13	EIF12	EIF11	EIF10	EIF9	EIF8	0	0	0	EIF4	EIF3	0	EIF1	EIF0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 135. External IRQ Status Register (SIU_EISR)

Table 127. SIU_EISR field descriptions

Field	Description
0 NMI	Non-Maskable Interrupt Flag This bit is set when a NMI interrupt occurs on the NMI input pin. 1 An NMI event has occurred on the NMI input 0 No NMI event has occurred on the NMI input
1–7	Reserved
8 SWT	Software Watch Dog Timer Interrupt Flag, from platform This bit is set when a SWT interrupt occurs on the platform. 1 An SWT event has occurred 0 No SWT event has occurred
9–15	Reserved
16–23, 27–28 30–31 EIF*x*	External Interrupt Request Flag *x* This bit is set when an edge triggered event occurs on the corresponding IRQ*x* input. 1 An edge triggered event has occurred on the corresponding IRQ*x* input 0 No edge triggered event has occurred on the corresponding IRQ*x* input
24–26	Reserved
29	Reserved

15.8.6 DMA/Interrupt Request Enable Register (SIU_DIRER)

The DMA/Interrupt Request Enable Register allows the assertion of a DMA or interrupt request if the corresponding flag bit is set in *Section 15.8.5: External Interrupt Status Register (SIU_EISR)*. The External Interrupt Request Enable bits enable the interrupt or DMA request. There is only one interrupt request from the SIU to the interrupt controller. The EIRE bits allow selection of which External Interrupt Request Flag bits cause assertion of the one interrupt request signal.

SIU_BASE + 0x18

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIRE15	EIRE14	EIRE13	EIRE12	EIRE11	EIRE10	EIRE9	EIRE8	EIRE7	EIRE6	EIRE5	EIRE4	EIRE3	EIRE2	EIRE1	EIRE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 136. DMA/Interrupt Request Enable Register (SIU_DIRER)

Table 128. SIU_DIRER field descriptions

Field	Description
0 NMI_SEL	NMI Interrupt Platform Input Selection This write once bit selects NMI or CI for the SWaT timer. 1 Critical interrupt request input is selected 0 NMI interrupt request input is selected
1–7	Reserved
8 SWT_SEL	Software Watchdog Timer Interrupt/Critical Interrupt Platform Input Selection This write once bit select each platform input will be used for the SWT interrupt signal, when an edge triggered event occurs on the NMI input. 1 Critical interrupt request input is selected 0 NMI interrupt request input is selected
9–15	Reserved
16–31 EIRE*x*	External DMA/Interrupt Request Enable *x* This bit enables the assertion of a DMA or the interrupt request from the SIU to the interrupt controller when an edge triggered event occurs on the IRQ*x* inputs. 1 External interrupt request is enabled 0 External interrupt request is disabled

15.8.7 DMA/Interrupt Request Select Register (SIU_DIRSR)

The DMA/Interrupt Request Select Register allows selection between a DMA or interrupt request for events on the IRQ0 and IRQ3 inputs.

SIU_BASE + 0x1C

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

	16	17	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	DIRS3			DIRS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 137. DMA/Interrupt Request Select Register (SIU_DIRSR)

Table 129. SIU_DIRSR field descriptions

Field	Description
0–27	Reserved
28, 31 DIRS*x*	DMA/Interrupt Request Select *x* This bit selects between a DMA or interrupt request when an edge triggered event occurs on the corresponding IRQ*x* input. 1 DMA request is selected (on this device these DMA connections do not exist, causing the interrupt to be inhibit) 0 Interrupt request is selected **Note:** For this device the DMA option is not implemented and should not be selected. These bits should not be written, and must be kept in the default reset value
29–30	Reserved

15.8.8 Overrun Status Register (SIU_OSR)

The Overrun Status Register contains flag bits that record an overrun.

SIU_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	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVF15	OVF14	OVF13	OVF12	OVF11	OVF10	OVF9	OVF8	0	0	0	OVF4	OVF3	0	OVF1	OVF0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 138. Overrun Status Register (SIU_OSR)

Table 130. SIU_OSR field descriptions

Field	Description
0–15	Reserved
16–23, 27–28, 30–31 OVF*x*	Overrun Flag *x* This bit is set when an overrun occurs on the corresponding IRQ*x* input. 1 An overrun has occurred on the corresponding IRQ*x* input 0 No overrun has occurred on the corresponding IRQ*x* input
24–26	Reserved
29	Reserved

15.8.9 Overrun Request Enable Register (SIU_ORER)

The Overrun Request Enable Register contains bits to enable an overrun if the corresponding flag bit is set in the SIU_OSR. If any Overrun Request Enable bit and the corresponding flag bit is set, the single combined overrun request from the SIU to the interrupt controller is asserted.

SIU_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	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ORE15	ORE14	ORE13	ORE12	ORE11	ORE10	ORE9	ORE8	0	0	0	ORE4	ORE3	0	ORE1	ORE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 139. Overrun Request Enable Register (SIU_ORER)

Table 131. SIU_ORER field descriptions

Field	Description
0–15	Reserved
16–23, 27–28, 30–31 ORE*x*	Overrun Request Enable *x* This bit enables the corresponding overrun request when an overrun occurs on the corresponding IRQ*x* input. 1 Overrun request is enabled 0 Overrun request is disabled
24–26	Reserved
29	Reserved

15.8.10 IRQ Rising-Edge Event Enable Register (SIU_IREER)

The IRQ Rising-Edge Event Enable Register allows rising edge triggered events to be enabled on the corresponding IRQ*x* inputs. Rising and falling edge events can be enabled by setting the corresponding bits in both the SIU_IREER and SIU_IFEER.

SIU_BASE + 0x28

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IREE15	IREE14	IREE13	IREE12	IREE11	IREE10	IREE9	IREE8				IREE4	IREE3		IREE1	IREE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 140. IRQ Rising-Edge Event Enable Register (SIU_IREER)

Table 132. SIU_IREER field descriptions

Field	Description
0 NMIRE	NMI Rising-Edge Event Enable *x* This write once bit enables rising-edge triggered events on the NMI input. 1 Rising edge event is enabled 0 Rising edge event is disabled
1–7	Reserved
8 SWTRE	SWT Rising-Edge Event Enable *x* This write once bit enables rising-edge triggered events on the NMI input. 1 Rising edge event is enabled 0 Rising edge event is disabled
9–15	Reserved
16–23, 27–28, 30–31 IREE*x*	IRQ Rising-Edge Event Enable *x* This bit enables rising-edge triggered events on the corresponding IRQ*x* input. 1 Rising edge event is enabled 0 Rising edge event is disabled
24–26	Reserved
29	Reserved

15.8.11 External IRQ Falling-Edge Event Enable Register (SIU_IFEER)

The External IRQ Falling-Edge Event Enable Register allows falling edge triggered events to be enabled on the corresponding IRQ*x* inputs. Rising and falling edge events can be enabled by setting the corresponding bits in both the SIU_IREER and SIU_IFEER.

SIU_BASE + 0x2C

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IFEE15	IFEE14	IFEE13	IFEE12	IFEE11	IFEE10	IFEE9	IFEE8				IFEE4	IFEE3		IFEE1	IFEE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 141. External IRQ Falling-Edge Event Enable Register (SIU_IFEER)

Table 133. SIU_IFEER field descriptions

Field	Description
0 NMIFE	NMI Falling-Edge Event Enable *x* This write once bit enables falling-edge triggered events on the NMI input. 1 Falling edge event is enabled 0 Falling edge event is disabled
1–7	Reserved
8 SWTFE	SWT Falling-Edge Event Enable *x* This write once bit enables falling-edge triggered events on the NMI input. 1 Falling edge event is enabled 0 Falling edge event is disabled
9–15	Reserved
16–23, 27–28, 30–31 IFEE*x*	IRQ Falling-Edge Event Enable *x* This bit enables falling-edge triggered events on the corresponding IRQ*x* input. 1 Falling edge event is enabled 0 Falling edge event is disabled
24–26	Reserved
29	Reserved

15.8.12 External IRQ Digital Filter Register (SIU_IDFR)

The External IRQ Digital Filter Register specifies the amount of digital filtering on the IRQ0 – IRQ15 inputs. The Digital Filter Length field specifies the number of system clocks that define the period of the digital filter.

SIU_BASE + 0x30

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

	16	17	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	DFL[0–3]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 142. IRQ Digital Filter Register (SIU_IDFR)

Table 134. SIU_IDFR field descriptions

Field	Description
0–27	Reserved
28–31 DFL[0–3]	Digital Filter Length This field defines the digital filter period on the IRQ*x* inputs according to *Equation 1*: **Equation 1** $\text{Filter Period} = (\text{SystemClockPeriod} \times 2^{\text{DFL}}) + 1(\text{SystemClockPeriod})$ For an 80 MHz system clock, this gives a range of 25 ns to 410 µs. The minimum time of two clocks accounts for synchronization of the IRQ input pins with the system clock.

15.8.13 Pad Configuration Registers (SIU_PCR)

The following subsections define the Pad Configuration Registers for all device pins that allow configuration of the pin function, direction, and static electrical attributes. The information presented pertains to which bits and fields are active for a given pin or group of pins, and the reset state of the register. Note that the reset state of PCRs given in the following sections is that prior to execution of the BAM program. The BAM program may change certain PCRs based on the reset configuration. See *Chapter 20: Boot Assist Module (BAM)* for more details.

Figure 143 shows a sample PCR map. Please note the following:

- The register bit numbering order follows the Power Architecture standard of the most significant bit being bit 0. Field bit ranges are the opposite—the least significant bit is referred to as bit 0.
- Bits 0 through 3 and bits 12 through 13 are examples reserved bits. They are read-only and always return a value of 0.
- The PCRs are 16-bit registers but may be read or written as 32-bit values aligned on 32-bit address boundaries.

Figure 143. Sample PCR map

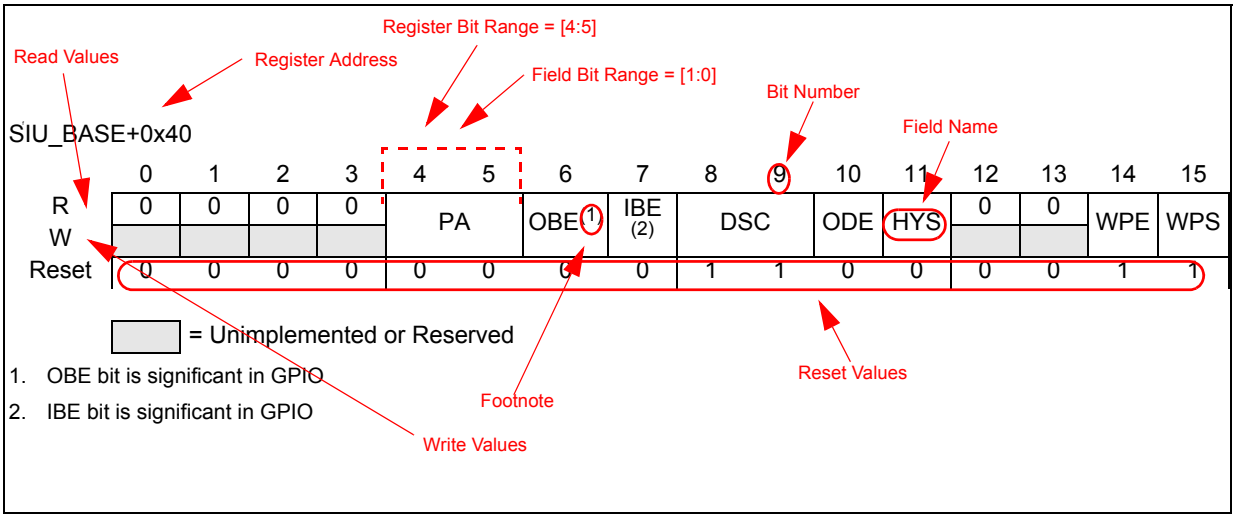


Table 135 lists and describes the fields contained in the PCRs. Not all fields appear in each PCR but each field has identical function in each register where it resides.

Table 135. SIU_PCR field descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>Reserved</td><td>Reserved fields are indicated by shading in the register maps.</td></tr>
<tr><td>PA</td><td>Pin assignment
Selects the function of a multiplexed pad.

<table>
<tr><th>PA value(1)</th><th colspan="2">Pin function</th></tr>
<tr><td>0b0001</td><td>P</td><td>Primary function</td></tr>
<tr><td>0b0010</td><td>A1</td><td>Alternate function</td></tr>
<tr><td>0b0100</td><td>A2</td><td>Alternate function 2</td></tr>
<tr><td>0b1000</td><td>A3</td><td>Alternate function 3</td></tr>
<tr><td>0b0000</td><td>G</td><td>GPIO</td></tr>
</table>
1. Depending on the register, the PA field size can vary in length. For PA fields having fewer than four bits, remove the appropriate number of leading zeroes from these values.</td></tr>
<tr><td>OBE(1),(2)</td><td>Output buffer enable
Enables the pad as an output and drives the output buffer enable signal.
0 Disable output buffer for the pad.
1 Enable output buffer for the pad.</td></tr>
</table>

Table 135. SIU_PCR field descriptions (continued)

Field	Description
IBE(1),(2)	Input buffer enable Enables the pad as an input and drives the input buffer enable signal. 0 Disable input buffer for the pad. 1 Enable input buffer for the pad is enabled. The IBE and OBE bit definitions are specific for each PCR. In cases where an I/O function is input or output only the IBE and OBE bits do not need to be set to enable the input or output respectively. In cases where an I/O function can be either an input and output, the IBE and OBE bits must be set accordingly (IBE = 1 for input, and OBE = 1 for output). For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits have no effect. For all PCRs where GPIO function is available on the pin, if the pin is configured as an output and the IBE bit is set, the actual value of the pin will be reflected in the corresponding GPDI*x*_*x* register. Negating the IBE bit when the pin is configured as an output will reduce noise and power consumption.
DSC(3)	Drive strength control Controls the pad drive strength. Drive strength control pertains to pins with the fast I/O pad type. 00 10 pF drive strength 01 20 pF drive strength 10 30 pF drive strength 11 50 pF drive strength
ODE(3)	Open drain output enable Controls output driver configuration for the pads. Either open drain or push/pull driver configurations can be selected. This feature applies to output pins only. 0 Disable open drain for the pad (push/pull driver enabled). 1 Enable open drain for the pad.
HYS(4)	Input hysteresis Controls whether hysteresis is enabled for the pad. 0 Disable hysteresis for the pad. 1 Enable hysteresis for the pad.
SRC(3)	Slew rate control Controls slew rate for the pad. Slew rate control pertains to pins with slow or medium I/O pad types, and the output signals are driven according to the value of this field. Actual slew rate depends on the pad type and load. Refer to the electrical specifications for this information. 00 Minimum slew rate 01 Medium slew rate 10 Invalid value 11 Maximum slew rate

Table 135. SIU_PCR field descriptions (continued)

Field	Description
WPE(5)	Weak pullup/down enable Controls whether the weak pullup/down devices are enabled/disabled for the pad. Pullup/down devices are enabled by default. 0 Disable weak pull device for the pad. 1 Enable weak pull device for the pad.
WPS(5)	Weak pullup/down select Controls whether weak pullup or weak pulldown devices are used for the pad when weak pullup/down devices are enabled. The WKPCFG pin determines whether pullup or pulldown devices are enabled during reset. The WPS bit determines whether weak pullup or pulldown devices are used after reset, or for pads in which the WKPCFG pin does not determine the reset weak pullup/down state. 0 Pulldown is enabled for the pad. 1 Pullup is enabled for the pad.

1. In cases where an I/O function is either input-only or output-only the IBE and OBE bits do not need to be set to enable pin I/O.
2. For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally and the IBE and OBE bits are ignored.
3. If a pin is configured as an input, the ODE, SRC, and DSC bits do not apply.
4. If a pin is configured as an output, the HYS bit does not apply.
5. When a pin is configured as an output, the weak internal pull up/down is disabled regardless of the WPE or WPS settings in the PCR.

All pin names on this device begin with the primary function, followed by the alternate functions, and then GPIO. For example, for SIU_PCR87 and the CAN_C_TX_GPIO[87] pin, CAN_C_TX is the primary function.The device is available in the packages listed in *Chapter 1: Introduction*. Some of the I/O controlled by the SIU PCRs are not available in the smaller packages. The port enable logic for these PCRs is the same for PCRs that control I/O that is available in all packages. For the smaller packages where some of the I/O are not available, the pad drivers are disabled in the pad interface logic. The user should take care not to select the unavailable functions via the PA field. See *Section 3.2: External Signal Summary* for a definition of which I/O functions are available in each package.

15.8.13.1 Pad Configuration Register 83 (SIU_PCR83)

The SIU_PCR83 register controls the pin function, direction, and static electrical attributes of the CAN_A_TX_SCI_A_TX_GPIO[83] pin.

SIU_BASE + 0xE6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. When configured as CAN_TX or SCI_TX, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
2. When configured as CAN_TX or SCI_TX, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 144. CAN_A_TX_SCI_A_TX_GPIO[83] Pad Configuration Register (SIU_PCR83)

15.8.13.2 Pad Configuration Register 84 (SIU_PCR84)

The SIU_PCR84 register controls the function, direction, and static electrical attributes of the CAN_A_RX_SCI_A_RX_GPIO[84] pin.

SIU_BASE + 0xE8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE(2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. When configured as CAN_RX or SCI_RX, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
2. When configured as CAN_RX or SCI_RX, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 145. CAN_A_TX_SCI_A_TX_GPIO[84] Pad Configuration Register (SIU_PCR84)

15.8.13.3 Pad Configuration Register 87 (SIU_PCR87)

The SIU_PCR87 register controls the function, direction, and static electrical attributes of the CAN_C_TX_GPIO[87] pin. This register allows selection of the CAN_C_TX and GPIO functions.

SIU_BASE + 0xEE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. The DSPI_D_PCS function is not available on SPC563Mxx. Do not select b'10 in the PA field.
2. When configured as CAN_TX the OBE bit has no effect. When configured as GPO, the OBE bit should be set to '1'.
3. When configured as CAN_TX or GPO, the IBE bit may be set to '1' to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to '0' reduces power consumption. When configured as GPI, the IBE bit should be set to '1'.

Figure 146. CAN_C_TX_GPIO[87] Pad Configuration Register (SIU_PCR87)

15.8.13.4 Pad Configuration Register 88 (SIU_PCR88)

The SIU_PCR88 register controls the function, direction, and static electrical attributes of the CAN_C_RX_GPIO[88] pin. This register allows selection of the CAN_C_RX and GPIO functions.

SIU_BASE + 0xF0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. The DSPI_D_PCS function is not available on this device. Do not select b'10 in the PA field.
2. When configured as CAN_RX, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to '1'.
3. When configured as CAN_RX or GPO, the IBE bit may be set to '1' to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to '0' reduces power consumption. When configured as GPI, the IBE bit should be set to '1'.

Figure 147. CAN_C_RX_GPIO[88] Pad Configuration Register (SIU_PCR88)

15.8.13.5 Pad Configuration Register 89 (SIU_PCR89)

The SIU_PCR89 register controls the function, direction, and static electrical attributes of the SCI_A_TX_eMIOS[13]_GPIO[89] pin.

SIU_BASE + 0xF2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. When configured as SCI_TX or eMIOS, the OBE bit has no effect. When configured as GPIO, the OBE bit should be set to '1'.
2. When configured as SCI_TX or eMIOS, the IBE bit has no effect. Setting the IBE bit to '0' reduces power consumption.When configured as GPO, the IBE bit may be set to '1' to reflect the pin state in the corresponding GPDI register. The IBE bit must be set to '1' for GPI when configured as inputs.

Figure 148. SCI_A_TX_eMIOS[13]_GPIO[89] Pad Configuration Register (SIU_PCR89)

15.8.13.6 Pad Configuration Register 90 (SIU_PCR90)

The SIU_PCR90 register controls the function, direction, and static electrical attributes of the SCI_A_RX_eMIOS[15]_GPIO[90] pin.

SIU_BASE + 0xF4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. When configured as SCI_RX or eMIOS, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
2. When configured as SCI_RX or eMIOS, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 149. SCI_A_RX_eMIOS[15]_GPIO[90] Pad Configuration Register (SIU_PCR90)

15.8.13.7 Pad Configuration Register 91 (SIU_PCR91)

The SIU_PCR91 register controls the function, direction, and static electrical attributes of the SCI_B_TX_GPIO[91] pin. This register allows selection of the SCI_TX_B and GPIO functions.

SIU_BASE + 0xF6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. The DSPI_D_PCS function is not available on this device. Do not select b’10 in the PA field.
2. When configured as SCI_TX, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as SCI_TX, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 150. SCI_B_TX_GPIO[91] Pad Configuration Register (SIU_PCR91)

15.8.13.8 Pad Configuration Register 92 (SIU_PCR92)

The SIU_PCR92 register controls the function, direction, and static electrical attributes of the SCI_B_RX_GPIO[92] pin. This register allows selection of the SCI_B_RX and GPIO functions.

SIU_BASE + 0xF8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. The DSPI_D_PCS function is not available on this device. Do not select b'10 in the PA field.
2. When configured as SCI_RX, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to '1'.
3. When configured as SCI_RX, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to '1' to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to '0' reduces power consumption. When configured as GPI, the IBE bit should be set to '1'.

Figure 151. SCI_B_RX_GPIO[92] Pad Configuration Register (SIU_PCR92)

15.8.13.9 Pad Configuration Registers 98 – 99 (SIU_PCR98 – SIU_PCR99)

The SIU_PCR98 – SIU_PCR99 registers control the function, direction, and static electrical attributes of the GPIO[98:99] pins.

SIU_BASE+0x104 – SIU_BASE+0x106 (2)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to '1'.
2. Setting the IBE bit to '0' reduces power consumption. When configured as GPI, the IBE bit should be set to '1'.

Figure 152. GPIO[98:99] Pad Configuration Registers (SIU_PCR98 – SIU_PCR99)

15.8.13.10 Pad Configuration Register 102 (SIU_PCR102)

The SIU_PCR102 register controls the function, direction, and static electrical attributes of the DSPI_B_SCK_DSPI_C_PCS[1]_GPIO[102] pin. This register allows selection of the DSPI_B_SCK, DSPI_C_PCS[1] and GPIO functions.

SIU_BASE + 0x10C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. When configured as SCK, the OBE bit should be set to '1' for master operation, and set to '0' for slave operation. When configured as GPO, the OBE bit should be set to '1'.
2. When configured as SCK in slave operation the IBE bit should be set to '1'. When configured as SCK in master operation or GPO, the IBE bit may be set to '1' to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to '0' reduces power consumption. When configured as GPI, the IBE bit should be set to '1'.

Figure 153. DSPI_B_SCK_DSPI_C_PCS[1]_GPIO[102] Pad Configuration Register (SIU_PCR102)

15.8.13.11 Pad Configuration Register 103 (SIU_PCR103)

The SIU_PCR103 register controls the function, direction, and static electrical attributes of the DSPI_B_SIN_DSPI_C_PCS[2]_GPIO[103] pin. This register allows selection of the DSPI_B_SIN, DSPI_C_PCS[2] and GPIO functions.

SIU_BASE + 0x10E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. When configured as SIN, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
2. When configured as SIN, the IBE has no effect. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 154. DSPI_B_SIN_DSPI_C_PCS[2]_GPIO[103] Pad Configuration Register (SIU_PCR103)

15.8.13.12 Pad Configuration Register 104 (SIU_PCR104)

The SIU_PCR104 register controls the function, direction, and static electrical attributes of the DSPI_B_SOUT_DSPI_C_PCS[5]_GPIO[104] pin. This register allows selection of the DSPI_B_SOUT, DSPI_C_PCS[5] and GPIO functions.

SIU_BASE + 0x110

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. When configured as SOUT, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
2. When configured as SOUT, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 155. DSPI_B_SOUT_DSPI_C_PCS[5]_GPIO[104] Pad Configuration Register (SIU_PCR104)

15.8.13.13 Pad Configuration Register 105 (SIU_PCR105)

The SIU_PCR105 register controls the function, direction, and static electrical attributes of the DSPI_B_PCS[0]_GPIO[105] pin. This register allows selection of the DSPI_B_PCS[0] and GPIO functions.

SIU_BASE + 0x112

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. The alternate DSPI_D_PCS function is not available on this device. Do not select b'10 in the PA field.
2. When configured as DSPI_PCS, the OBE bit should be set to '1' for master operation (CS0), and set to '0' for slave operation (SS). When configured as GPO, the OBE bit should be set to '1'. When configured as GPO, the OBE bit should be set to '1'.
3. When configured as DSPI_PCS in slave operation the IBE bit should be set to '1'. When configured as DSPI_B_PCS in master operation or GPO, the IBE bit may be set to '1' to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to '0' reduces power consumption. When configured as GPI, the IBE bit should be set to '1'.

Figure 156. DSPI_B_PCS[0]_GPIO[105] Pad Configuration Register (SIU_PCR105)

15.8.13.14 Pad Configuration Register 106 (SIU_PCR106)

The SIU_PCR106 register controls the function, direction, and static electrical attributes of the DSPI_B_PCS[1]_GPIO[106] pin. This register allows selection of the DSPI_B_PCS and GPIO functions.

SIU_BASE + 0x114

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. The alternate DSPI_D_PCS function is not available on this device. Do not select b'10 in the PA field.
2. When configured as DSPI_PCS, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to '1'.
3. When configured as DSPI_PCS, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to '1' to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to '0' reduces power consumption. When configured as GPI, the IBE bit should be set to '1'.

Figure 157. DSPI_B_PCS[1]_GPIO[106] Pad Configuration Register (SIU_PCR106)

15.8.13.15 Pad Configuration Register 107 (SIU_PCR107)

The SIU_PCR107 register controls the function, direction, and static electrical attributes of the DSPI_B_PCS[2]_DSPI_C_SOUT_GPIO[107] pin. This register allows selection of the DSPI_B_PCS, DSPI_C_SOUT and GPIO functions.

SIU_BASE + 0x116

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. The DSPI_D_PCS function is not available on this device. Do not select b'10 in the PA field.
2. When configured as PCS, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to '1'.
3. When configured as PCS, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to '1' to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to '0' reduces power consumption. When configured as GPI, the IBE bit should be set to '1'.

Figure 158. DSPI_B_PCS[2]_DSPI_C_SOUT_GPIO[107] Pad Configuration Register (SIU_PCR107)

15.8.13.16 Pad Configuration Register 108 (SIU_PCR108)

The SIU_PCR108 register controls the function, direction, and static electrical attributes of the DSPI_B_PCS[3]_DSPI_C_SIN_GPIO[108] pin. This register allows selection of the DSPI_B_PCS, DSPI_C_SIN and GPIO functions.

SIU_BASE + 0x118

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. When configured as PCS, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to '1'.
2. When configured as PCS, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to '1' to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to '0' reduces power consumption. When configured as GPI, the IBE bit should be set to '1'.

Figure 159. DSPI_B_PCS[3]_DSPI_C_SIN_GPIO[108] Pad Configuration Register (SIU_PCR108)

15.8.13.17 Pad Configuration Register 109 (SIU_PCR109)

The SIU_PCR109 register controls the function, direction, and static electrical attributes of the DSPI_B_PCS[4]_DSPI_C_SCK_GPIO[109] pin. This register allows selection of the DSPI_B_PCS, DSPI_C_SCK and GPIO functions.

SIU_BASE + 0x11A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. When configured as PCS, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
2. When configured as PCS, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 160. DSPI_B_PCS[4]_DSPI_C_SCK_GPIO[109] Pad Configuration Register (SIU_PCR109)

15.8.13.18 Pad Configuration Register 110 (SIU_PCR110)

The SIU_PCR110 register controls the function, direction, and static electrical attributes of the DSPI_B_PCS[5]_DSPI_C_PCS[0]_GPIO[110] pin. This register allows selection of the DSPI_B_PCS, DSPI_C_PCS[0] and GPIO functions.

SIU_BASE + 0x11C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. When configured as PCS, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
2. When configured as PCS, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 161. DSPI_B_PCS[5]_DSPI_C_PCS[0]_GPIO[110] Pad Configuration Register (SIU_PCR110)

15.8.13.19 Pad Configuration Register 114 (SIU_PCR114)

The SIU_PCR114 register controls the function, direction, and static electrical attributes of the eTPU_A[0]_eTPU_A[12]_eTPU_A[19]_GPIO[114] pin.

SIU_BASE + 0x124

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eTPU_A[0] and GPO[114] when configured as outputs. When configured as eTPU_A[12] or eTPU_A[19], the OBE bit has no effect.
2. The IBE bit must be set to ‘1’ for both eTPU_A[0] and GPIO when configured as inputs. When configured as PCS, the IBE bit has no effect. When configured as eTPU_A or GPIO outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the eTPU_A[0] pins is determined by the WKPCFG pin.

Figure 162. eTPU_A[0]_eTPU_A[12]_eTPU_A[19]_GPIO[114] Pad Configuration Register (SIU_PCR114)

15.8.13.20 Pad Configuration Registers 115 – 118 (SIU_PCR115 – SIU_PCR118)

The SIU_PCR115 – SIU_PCR118 registers control the functions, directions, and static electrical attributes of the eTPU_A[1:4]_eTPU_A[13:16]_GPIO[115:118] pins. Only the output channels of eTPU_A[13:16] are connected to pins. Both the input and output channels of eTPU_A[1:4] are connected to pins.

SIU_BASE + 0x126 – SIU_BASE + 0x12C (4)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eTPU_A[1:4] and GPO[115:118] when configured as outputs. When configured as eTPU_A[13:16], the OBE bit has no effect.
2. The IBE bit must be set to ‘1’ for both eTPU_A[1:4] and GPO[115:118] when configured as inputs. When configured as eTPU_A[13:16] or when eTPU_A[1:4] or GPO[114:125] are configured as outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the eTPU_A[1:4] pins is determined by the WKPCFG pin.

Figure 163. eTPU_A[1:4]_eTPU_A[13:16]_GPIO[115:118] Pad Configuration Register (SIU_PCR115 – SIU_PCR118)

15.8.13.21 Pad Configuration Register 119 (SIU_PCR119)

The SIU_PCR119 register controls the function, direction, and static electrical attributes of the eTPU_A[5]_eTPU_A[17]_DSPI_B_SCK_LVDS–_GPIO[119] pin.

It is required to program the PA field of both registers, SIU_PCR119 and SIU_PCR120, to select the SCK_LVDS alternate function, and then use the register SIU_PCR120 to program the SCK_LVDS characteristics (drive strength using the slew rate field).

SIU_BASE + 0x12E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1](3)		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(4)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eTPU_A[5] and GPO[119] when configured as outputs. When configured as eTPU_A[17], the OBE bit has no effect.
2. The IBE bit must be set to ‘1’ for both eTPU_A[5] and GPO[119] when configured as inputs. When eTPU_A[17] or when eTPU_A[5] or GPO[119] are configured as outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. On the LVDS pads these bits are used to allow the control of output voltage swing. They are connected to the lvds_opt0 and lvds_opt1 inputs of the LVDS pads (see the table “LVDS Pads Voltage Swing” in *Chapter 25: Deserial Serial Peripheral Interface (DSPI)*). In the other pad types they assume the Slew Rate Control functionality.
4. The weak pull up/down selection at reset for the eTPU_A[5] pin is determined by the WKPCFG pin.

Figure 164. eTPU_A[5]_eTPU_A[17]_DSPI_B_SCK_LVDS–_GPIO[119] Pad Configuration Register (SIU_PCR119)

15.8.13.22 Pad Configuration Register 120 (SIU_PCR120)

The SIU_PCR120 register controls the function, direction, and static electrical attributes of the eTPU_A[6]_eTPU_A[18]_DSPI_B_SCK_LVDS+_GPIO[120] pin.

It is required to program the PA field of both registers, SIU_PCR119 and SIU_PCR120, to select the DSPI_B_SCK_LVDS alternate function, and then use the register SIU_PCR120 to program the DSPI_B_SCK_LVDS characteristics (drive strength using the slew rate field).

SIU_BASE + 0x130

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1](3)		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(4)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eTPU_A[6] and GPO[120] when configured as outputs. When configured as eTPU_A[18], the OBE bit has no effect.
2. The IBE bit must be set to ‘1’ for both eTPU_A[6] and GPO[120] when configured as inputs. When eTPU_A[18] or when eTPU_A[6] or GPO[120] are configured as outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. On the LVDS pads these bits are used to allow the control of output voltage swing. They are connected to the lvds_opt0 and lvds_opt1 inputs of the LVDS pads (see the table “LVDS Pads Voltage Swing” in *Chapter 25: Deserial Serial Peripheral Interface (DSPI)*). In the other pad types they assume the Slew Rate Control functionality.
4. The weak pull up/down selection at reset for the eTPU_A[6] pin is determined by the WKPCFG pin.

Figure 165. eTPU_A[6]_eTPU_A[18]_DSPI_B_SCK_LVDS+_GPIO[120] Pad Configuration Register (SIU_PCR120)

15.8.13.23 Pad Configuration Register 121 (SIU_PCR121)

The SIU_PCR120 register controls the function, direction, and static electrical attributes of the eTPU_A[7]_eTPU_A[19]_DSPI_B_SOUT_LVDS–_eTPU_A[6]_GPIO[121] pin.

It is required to program the PA field of both registers, SIU_PCR121 and SIU_PCR122, to select the DSPI_B_SOUT_LVDS alternate function, and then use the register SIU_PCR122 to program the DSPI_B_SOUT_LVDS characteristics (drive strength using the slew rate field).

SIU_BASE + 0x132

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA[0–3]				OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1](3)		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(4)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eTPU_A[7] and GPO[121] when configured as outputs. When configured as eTPU_A[6] and eTPU_A[19], the OBE bit has no effect.
2. The IBE bit must be set to ‘1’ for both eTPU_A[7] and GPO[121] when configured as inputs. When eTPU_A[6] and eTPU_A[19] or when eTPU_A[7] or GPO[121] are configured as outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. On the LVDS pads these bits are used to allow the control of output voltage swing. They are connected to the lvds_opt0 and lvds_opt1 inputs of the LVDS pads (see the table “LVDS Pads Voltage Swing” in *Chapter 25: Deserial Serial Peripheral Interface (DSPI)*). In the other pad types they assume the Slew Rate Control functionality.
4. The weak pull up/down selection at reset for the eTPU_A[7] pin is determined by the WKPCFG pin.

Figure 166. eTPU_A[7]_eTPU_A[19]_DSPI_B_SOUT_LVDS–_eTPU_A[6]_GPIO[121] Pad Configuration Register (SIU_PCR121)

15.8.13.24 Pad Configuration Register 122 (SIU_PCR122)

The SIU_PCR122 register controls the function, direction, and static electrical attributes of the eTPU_A[8]_eTPU_A[20]_DSPI_B_SOUT_LVDS+_GPIO[122] pin.

It is required to program the PA field of both registers, SIU_PCR121 and SIU_PCR122, to select the DSPI_B_SOUT_LVDS alternate function, and then use the register SIU_PCR122 to program the DSPI_B_SOUT_LVDS characteristics (drive strength using the slew rate field).

SIU_BASE + 0x134

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1](3)		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(4)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eTPU_A[8] and GPO[122] when configured as outputs. When configured as eTPU_A[20], the OBE bit has no effect.
2. The IBE bit must be set to ‘1’ for both eTPU_A[8] and GPO[122] when configured as inputs. When eTPU_A[20] or when eTPU_A[8] or GPO[122] are configured as outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. On the LVDS pads these bits are used to allow the control of output voltage swing. They are connected to the lvds_opt0 and lvds_opt1 inputs of the LVDS pads (see the table “LVDS Pads Voltage Swing” in *Chapter 25: Deserial Serial Peripheral Interface (DSPI)*). In the other pad types they assume the Slew Rate Control functionality.
4. The weak pull up/down selection at reset for the eTPU_A[8] pin is determined by the WKPCFG pin.

Figure 167. eTPU_A[8]_eTPU_A[20]_DSPI_B_SOUT_LVDS+_GPIO[122] Pad Configuration Register (SIU_PCR122)

15.8.13.25 Pad Configuration Register 123 – 125 (SIU_PCR123 – SIU_PCR125)

The SIU_PCR123 – SIU_PCR125 registers control the function, direction, and static electrical attributes of the eTPU_A[9:11]_eTPU_A[21:23]_GPIO[123:125] pins.

SIU_BASE + 0x136 – SIU_BASE + 0x13A (3)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eTPU_A[9:11] and GPO[123:125] when configured as outputs. When configured as eTPU_A[21:23], the OBE bit has no effect.
2. The IBE bit must be set to ‘1’ for both eTPU_A[9:11] and GPO[123:125] when configured as inputs. When eTPU_A[21:23] or when eTPU_A[9:11] or GPO[123:125] are configured as outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the eTPU_A[9:11] pin is determined by the WKPCFG pin.

Figure 168. eTPU_A[9:11]_eTPU_A[21:23]_GPIO[123:125] Pad Configuration Register (SIU_PCR123 – SIU_PCR125)

15.8.13.26 Pad Configuration Register 126 (SIU_PCR126)

The SIU_PCR126 register controls the function, direction, and static electrical attributes of the eTPU_A[12]_DSPI_B_PCS[1]_GPIO[126] pin.

SIU_BASE + 0x13C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as PCS, the OBE bit has no effect. The OBE bit must be set to ‘1’ for both eTPU_A and GPIO when configured as outputs.
2. The IBE bit must be set to ‘1’ for both eTPU_A and GPO when configured as inputs. When configured as PCS, the IBE bit has no effect. When configured as eTPU_A or GPO outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the eTPU_A[12] pin is determined by the WKPCFG pin.

Figure 169. eTPU_A[12]_DSPI_B_PCS[1]_GPIO[126] Pad Configuration Register (SIU_PCR126)

15.8.13.27 Pad Configuration Register 127 (SIU_PCR127)

The SIU_PCR127 register controls the function, direction, and static electrical attributes of the eTPU_A[13]_DSPI_B_PCS[3]_GPIO[127] pin.

SIU_BASE + 0x13E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as PCS, the OBE bit has no effect. The OBE bit must be set to ‘1’ for both eTPU_A and GPO when configured as outputs.
2. The IBE bit must be set to ‘1’ for both eTPU_A and GPO when configured as inputs. When configured as PCS, the IBE bit has no effect. When configured as eTPU_A or GPO outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the eTPU_A[13] pin is determined by the WKPCFG pin.

Figure 170. eTPU_A[13]_DSPI_B_PCS[3]_GPIO[127] Pad Configuration Register (SIU_PCR127)

15.8.13.28 Pad Configuration Register 128 (SIU_PCR128)

The SIU_PCR128 register controls the function, direction, and static electrical attributes of the eTPU_A[14]_DSPI_B_PCS[4]_eTPU_A[9]_GPIO[128] pin.

SIU_BASE + 0x140

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as PCS, the OBE bit has no effect. The OBE bit must be set to ‘1’ for both eTPU_A[14] and GPO when configured as outputs.
2. The IBE bit must be set to ‘1’ for both eTPU_A[14] and GPO when configured as inputs. When configured as PCS, the IBE bit has no effect. When configured as eTPU_A[14] or GPO outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the eTPU_A[14] pin is determined by the WKPCFG pin.

Figure 171. eTPU_A[14]_DSPI_B_PCS[4]_eTPU_A[9]_GPIO[128] Pad Configuration Register (SIU_PCR128)

15.8.13.29 Pad Configuration Register 129 (SIU_PCR129)

The SIU_PCR129 register controls the function, direction, and static electrical attributes of the eTPU_A[15]_DSPI_B_PCS[5]_GPIO[129] pin.

SIU_BASE + 0x142

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as PCS, the OBE bit has no effect. The OBE bit must be set to ‘1’ for both eTPU_A and GPO when configured as outputs.
2. The IBE bit must be set to ‘1’ for both eTPU_A and GPO when configured as inputs. When configured as PCS, the IBE bit has no effect. When configured as eTPU_A or GPO outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
3. The weak pull up/down selection at reset for the eTPU_A[15] pin is determined by the WKPCFG pin.

Figure 172. eTPU_A[15]_DSPI_B_PCS[5]_GPIO[129] Pad Configuration Register (SIU_PCR129)

15.8.13.30 Pad Configuration Register 130 – 133 (SIU_PCR130 – SIU_PCR133)

The SIU_PCR130 – SIU_PCR133 registers control the function, direction, and static electrical attributes of the eTPU_A[16:19]_GPIO[130:133] pins. This registers allow selection of the eTPU_A and GPIO functions.

SIU_BASE+0x144 – SIU_BASE + 0x14A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(4)

= Unimplemented or Reserved

1. The DSPI_D_PCS function is not available on this device. Do not select b’10 in the PA field.
2. The OBE bit must be set to ‘1’ for both eTPU_A and GPO when configured as outputs.
3. The IBE bit must be set to ‘1’ for both eTPU_A and GPO when configured as inputs. When configured as eTPU_A or GPO outputs, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register.
4. The weak pull up/down selection at reset for the eTPU_A[16] pin is determined by the WKPCFG pin.

Figure 173. eTPU_A[16:19]_GPIO[130:133] Pad Configuration Registers (SIU_PCR130 – SIU_PCR133)

15.8.13.31 Pad Configuration Register 134 – 135 (SIU_PCR134 – SIU_PCR135)

The SIU_PCR134 – SIU_PCR135 registers control the functions, directions, and static electrical attributes of the eTPU_A[20:21]_IRQ[8:9]_GPIO[134:135] pins.

SIU_BASE + 0x14C – SIU_BASE + 0x14E (2)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eTPU_A[20:21] and GPIO[134:135] when configured as outputs.
2. When configured as IRQ or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for both eTPU_A[20:21] and GPIO[134:135] when configured as inputs.
3. The weak pull up/down selection at reset for the eTPU_A[20:21] pins are determined by the WKPCFG pin.

Figure 174. eTPU_A[20:21]_IRQ[8:9]_GPIO[134:135] Pad Configuration Register (SIU_PCR134 – SIU_PCR135)

15.8.13.32 Pad Configuration Register 136 (SIU_PCR136)

The SIU_PCR136 register controls the function, direction, and static electrical attributes of the eTPU_A[22]_IRQ[10]_eTPU_A[17]_GPIO[136] pin. The eTPU_A[17] is an output only function.

SIU_BASE + 0x150

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eTPU_A[22] and GPIO[136] when configured as outputs.
2. When configured as IRQ or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for both eTPU_A[22] and GPIO[136] when configured as inputs.
3. The weak pull up/down selection at reset for the eTPU_A[22] pin is determined by the WKPCFG pin.

Figure 175. eTPU_A[22]_IRQ[10]_eTPU_A[17]_GPIO[136] Pad Configuration Register (SIU_PCR136)

15.8.13.33 Pad Configuration Register 137 (SIU_PCR137)

The SIU_PCR137 register controls the function, direction, and static electrical attributes of the eTPU_A[23]_IRQ[11]_eTPU_A[21]_GPIO[137] pin.

SIU_BASE + 0x152

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eTPU_A[23] and GPIO[137] when configured as outputs.
2. When configured as IRQ or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for both eTPU_A[23] and GPIO[137] when configured as inputs.
3. The weak pull up/down selection at reset for the eTPU_A[23] pin is determined by the WKPCFG pin.

Figure 176. eTPU_A[23]_IRQ[11]_eTPU_A[21]_GPIO[137] Pad Configuration Register (SIU_PCR137)

15.8.13.34 Pad Configuration Register 138 (SIU_PCR138)

The SIU_PCR138 register controls the function, direction, and static electrical attributes of the eTPU_A[24]_IRQ[12]_DSPI_C_SCK_LVDS–_GPIO[138] pin.

It is required to program the PA field of both registers, SIU_PCR138 and SIU_PCR139, to select the DSPI_C_SCK_LVDS alternate function, and then use the register SIU_PCR139 to program the DSPI_C_SCK_LVDS characteristics (drive strength using the slew rate field).

SIU_BASE + 0x154

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1](3)		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(4)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for GPIO[138] when configured as output.
2. When configured as IRQ or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for GPIO[138] when configured as input.
3. On the LVDS pads these bits are used to allow the control of output voltage swing. They are connected to the lvds_opt0 and lvds_opt1 inputs of the LVDS pads (see the table “LVDS Pads Voltage Swing” in *Chapter 25: Deserial Serial Peripheral Interface (DSPI)*). In the other pad types they assume the Slew Rate Control functionality.
4. The weak pull up/down selection at reset for the eTPU_A[24] pin is determined by the WKPCFG pin.

Figure 177. eTPU_A[24]_IRQ[12]_DSPI_C_SCK_LVDS–_GPIO[138] Pad Configuration Register (SIU_PCR138)

15.8.13.35 Pad Configuration Register 139 (SIU_PCR139)

The SIU_PCR139 register controls the function, direction, and static electrical attributes of the eTPU_A[25]_IRQ[13]_DSPI_C_SCK_LVDS+_GPIO[139] pin.

It is required to program the PA field of both registers, SIU_PCR138 and SIU_PCR139, to select the DSPI_C_SCK_LVDS alternate function, and then use the register SIU_PCR139 to program the DSPI_C_SCK_LVDS characteristics (drive strength using the slew rate field).

SIU_BASE + 0x156

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1](3)		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(4)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for GPIO[139] when configured as output.
2. When configured as IRQ or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for GPIO[139] when configured as input.
3. On the LVDS pads these bits are used to allow the control of output voltage swing. They are connected to the lvds_opt0 and lvds_opt1 inputs of the LVDS pads (see the table “LVDS Pads Voltage Swing” in *Chapter 25: Deserial Serial Peripheral Interface (DSPI)*). In the other pad types they assume the Slew Rate Control functionality.
4. The weak pull up/down selection at reset for the eTPU_A[25] pin is determined by the WKPCFG pin.

Figure 178. eTPU_A[25]_IRQ[13]_DSPI_C_SCK_LVDS+_GPIO[139] Pad Configuration Register (SIU_PCR139)

15.8.13.36 Pad Configuration Register 140 (SIU_PCR140)

The SIU_PCR140 register controls the function, direction, and static electrical attributes of the eTPU_A[26]_IRQ[14]_DSPI_C_SOUT_LVDS–_GPIO[140] pin.

It is required to program the PA field of both registers, SIU_PCR140 and SIU_PCR141, to select the DSPI_C_SOUT_LVDS alternate function, and then use the register SIU_PCR141 to program the DSPI_C_SOUT_LVDS characteristics (drive strength using the slew rate field).

SIU_BASE + 0x158

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1](3)		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(4)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for GPIO[140] when configured as output.
2. When configured as IRQ or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for GPIO[140] when configured as input.
3. On the LVDS pads these bits are used to allow the control of output voltage swing. They are connected to the lvds_opt0 and lvds_opt1 inputs of the LVDS pads (see the table “LVDS Pads Voltage Swing” in *Chapter 25: Deserial Serial Peripheral Interface (DSPI)*). In the other pad types they assume the Slew Rate Control functionality.
4. The weak pull up/down selection at reset for the eTPU_A[26] pin is determined by the WKPCFG pin.

Figure 179. eTPU_A[26]_IRQ[14]_DSPI_C_SOUT_LVDS–_GPIO[140] Pad Configuration Register (SIU_PCR140)

15.8.13.37 Pad Configuration Register 141 (SIU_PCR141)

The SIU_PCR141 register controls the function, direction, and static electrical attributes of the eTPU_A[27]_IRQ[15]_DSPI_C_SOUT_LVDS+_DSPI_B_SOUT_GPIO[141] pin.

It is required to program the PA field of both registers, SIU_PCR140 and SIU_PCR141, to select the DSPI_B_SOUT_LVDS alternate function, and then use the register SIU_PCR141

to program the DSPI_B_SOUT_LVDS characteristics (drive strength using the slew rate field).

SIU_BASE + 0x15A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PA[0–3]				OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1](3)		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(4)

= Unimplemented or Reserved

1. When configured as eTPU_A[27] or IRQ, the OBE bit has no effect. The OBE bit must be set to ‘1’ for both eTPU_A[27] and GPIO[141] when configured as outputs
2. When configured as eTPU_A[27] or IRQ or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for both eTPU_A[27] and GPIO[141] when configured as inputs.
3. On the LVDS pads these bits are used to allow the control of output voltage swing. They are connected to the lvds_opt0 and lvds_opt1 inputs of the LVDS pads (see the table “LVDS Pads Voltage Swing” in *Chapter 25: Deserial Serial Peripheral Interface (DSPI)*). In the other pad types they assume the Slew Rate Control functionality.
4. The weak pull up/down selection at reset for the eTPU_A[27] pins is determined by the WKPCFG pin.

Figure 180. eTPU_A[27]_IRQ[15]_DSPI_C_SOUT_LVDS+_DSPI_B_SOUT_GPIO[141] Pad Configuration Register (SIU_PCR141)

15.8.13.38 Pad Configuration Register 142 – 143 (SIU_PCR142 – SIU_PCR143)

The SIU_PCR142 – SIU_PCR143 registers control the functions, directions, and static electrical attributes of the eTPU_A[28:29]_DSPI_C_PCS[1:2]_GPIO[142:143] pins. Only the output channel of eTPU_A[28:29] is connected to the pin. This register allows selection of the eTPU_A and GPIO functions.

SIU_BASE + 0x15C – SIU_BASE + 0x15E (2)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as eTPU_A, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
2. When configured as eTPU_A, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for GPIO when configured as input.
3. The weak pull up/down selection at reset for the eTPU_A[28:29] pin is determined by the WKPCFG pin.

Figure 181. eTPU_A[28:29]_DSPI_C_PCS[1:2]_GPIO[142:143] Pad Configuration Register (SIU_PCR142 – SIU_PCR143)

15.8.13.39 Pad Configuration Register 144 (SIU_PCR144)

The SIU_PCR144 register controls the function, direction, and static electrical attributes of the eTPU_A[30]_DSPI_C_PCS[3]_eTPU_A[11]_GPIO[144] pin. This register allows selection of the eTPU_A, DSPI_C_PCS and GPIO functions. The eTPU_A[11] is an output only function.

SIU_BASE + 0x160

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as eTPU_A output or GPO, the OBE bit should be set to ‘1’.
2. When configured as eTPU_A output or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for eTPU_A or GPIO when configured as input.
3. The weak pull up/down selection at reset for the eTPU_A[30] pin is determined by the WKPCFG pin.

Figure 182. eTPU_A[30]_DSPI_C_PCS[3]_eTPU_A[11]_GPIO[144] Pad Configuration Register (SIU_PCR144)

15.8.13.40 Pad Configuration Register 145 (SIU_PCR145)

The SIU_PCR145 register controls the function, direction, and static electrical attributes of the eTPU_A[31]_DSPI_C_PCS[4]_eTPU_A[13]_GPIO[145] pin. This register allows selection of the eTPU_A, DSPI_C_PCS and GPIO functions. The eTPU_A[13] is an output only function.

SIU_BASE + 0x162

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as eTPU_A output or GPO, the OBE bit should be set to ‘1’.
2. When configured as eTPU_A output or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for eTPU_A or GPIO when configured as input.
3. The weak pull up/down selection at reset for the eTPU_A[31] pin is determined by the WKPCFG pin.

Figure 183. eTPU_A[31]_DSPI_C_PCS[4]_eTPU_A[13]_GPIO[145] Pad Configuration Register (SIU_PCR145)

15.8.13.41 Pad Configuration Register 179 (SIU_PCR179)

The SIU_PCR179 register controls the function, direction, and static electrical attributes of the eMIOS[0]_eTPU_A[0]_eTPU_A[25]_GPIO[179] pin. This register allows selection of the eMIOS, eTPU_A and GPIO functions. The eTPU_A[25] is an output only function.

SIU_BASE + 0x1A6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as eMIOS output or GPO, the OBE bit should be set to ‘1’.
2. When configured as eMIOS output or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for eMIOS or GPIO when configured as input.
3. The weak pull up/down selection at reset for the eMIOS[0] pin is determined by the WKPCFG pin.

Figure 184. eMIOS[0]_eTPU_A[0]_eTPU_A[25]_GPIO[179] Pad Configuration Register (SIU_PCR179)

15.8.13.42 Pad Configuration Register 180 (SIU_PCR180)

The SIU_PCR180 register controls the function, direction, and static electrical attributes of the eMIOS[1]_eTPU_A[1]_GPIO[180] pin. This register allows selection of the eTPU_A and GPIO functions.

SIU_BASE+0x1A8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as eMIOS output or GPO, the OBE bit should be set to ‘1’.
2. When configured as eMIOS output or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for eMIOS or GPIO when configured as input.
3. The weak pull up/down selection at reset for the eMIOS[1] pin is determined by the WKPCFG pin.

Figure 185. eMIOS[1]_eTPU_A[1]_GPIO[180] Pad Configuration Register (SIU_PCR180)

15.8.13.43 Pad Configuration Register 181 (SIU_PCR181)

The SIU_PCR181 register controls the function, direction, and static electrical attributes of the eMIOS[2]_eTPU_A[2]_GPIO[181] pin. This register allows selection of the eTPU_A and GPIO functions.

SIU_BASE + 0x1AA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as eMIOS output or GPO, the OBE bit should be set to ‘1’.
2. When configured as eMIOS output or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for eMIOS or GPIO when configured as input.
3. The weak pull up/down selection at reset for the eMIOS[2] pins are determined by the WKPCFG pin.

Figure 186. eMIOS[2]_eTPU_A[2]_GPIO[181] Pad Configuration Register (SIU_PCR181)

15.8.13.44 Pad Configuration Register 183 (SIU_PCR183)

The SIU_PCR183 register controls the function, direction, and static electrical attributes of the eMIOS[4]_eTPU_A[4]_GPIO[183] pin. This register allows selection of the eTPU_A and GPIO functions.

SIU_BASE + 0x1AE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as eMIOS output or GPO, the OBE bit should be set to ‘1’.
2. When configured as eMIOS output or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for eMIOS or GPIO when configured as input.
3. The weak pull up/down selection at reset for the eMIOS[4] pins are determined by the WKPCFG pin.

Figure 187. eMIOS[4]_eTPU_A[4]_GPIO[183] Pad Configuration Register (SIU_PCR183)

15.8.13.45 Pad Configuration Register 187 (SIU_PCR187)

The SIU_PCR187 register controls the functions, directions, and static electrical attributes of the eMIOS[8]_eTPU_A[8]_SCI_B_TX_GPIO[187] pin. Both the input and output functions of eMIOS[8:9] are connected to pins. Only the output channels of eTPU_A[8:9] are connected to pins.

SIU_BASE+0x1B6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eMIOS[8] and GPIO[187] when configured as outputs.
2. When configured as eMIOS, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for both eMIOS[8] and GPIO[187] when configured as inputs.
3. The weak pull up/down selection at reset for the eMIOS[8:9] pins is determined by the WKPCFG pin.

Figure 188. eMIOS[8]_eTPU_A[8]_SCI_B_TX_GPIO[187] Pad Configuration Register (SIU_PCR187)

15.8.13.46 Pad Configuration Register 188 (SIU_PCR188)

The SIU_PCR188 register controls the functions, directions, and static electrical attributes of the eMIOS[9]_eTPU_A[9]_SCI_B_RX_GPIO[188] pin. Both the input and output functions of eMIOS[8:9] are connected to pins. Only the output channels of eTPU_A[8:9] are connected to pins.

SIU_BASE+0x1B8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eMIOS[9] and GPIO[188] when configured as outputs.
2. When configured as eMIOS, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for both eMIOS[9] and GPIO[188] when configured as inputs.
3. The weak pull up/down selection at reset for the eMIOS[8:9] pins is determined by the WKPCFG pin.

Figure 189. eMIOS[9]_eTPU_A[9]_SCI_B_RX_GPIO[188] Pad Configuration Register (SIU_PCR188)

15.8.13.47 Pad Configuration Register 189 – 190 (SIU_PCR189 – SIU_PCR190)

The SIU_PCR189 – SIU_PCR190 registers control the pin functions, directions, and static electrical attributes of the eMIOS[10:11]_GPIO[189:190] pins. Both the input and output functions of eMIOS[10:11] are connected to pins.

SIU_BASE + 0x1BA – SIU_BASE + 0x1BC (2)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for both eMIOS[10:11] and GPIO[189:190] when configured as outputs.
2. When configured as eMIOS or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for both eMIOS[10:11] and GPIO[189:190] when configured as inputs.
3. The weak pull up/down selection at reset for the eMIOS[10:11] pins is determined by the WKPCFG pin.

Figure 190. eMIOS[10:11]_GPIO[189:190] Pad Configuration Register (SIU_PCR189 – SIU_PCR190)

15.8.13.48 Pad Configuration Register 191 (SIU_PCR191)

The SIU_PCR191 register controls the function, direction, and static electrical attributes of the eMIOS[12]_DSPI_C_SOUT_eTPU_A[27]_GPIO[191] pin. This register allows selection of the eMIOS and GPIO functions.

SIU_BASE + 0x1BE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for GPIO[191] when configured as an output.
2. When configured as eMIOS or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for GPIO[191] when configured as an input.
3. The weak pull up/down selection at reset for the eMIOS[12] pin is determined by the WKPCFG pin.

Figure 191. eMIOS[12]_DSPI_C_SOUT_eTPU_A[27]_GPIO[191] Pad Configuration Register (SIU_PCR191)

15.8.13.49 Pad Configuration Register 192 (SIU_PCR192)

The SIU_PCR192 register controls the function, direction, and static electrical attributes of the eMIOS[13]_GPIO[192] pin. This register allows selection of the eMIOS and GPIO functions.

SIU_BASE+0x1C0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for GPIO[192] when configured as an output.
2. When configured as eMIOS or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for GPIO[192] when configured as an input.
3. The weak pull up/down selection at reset for the eMIOS[12] pin is determined by the WKPCFG pin.

Figure 192. eMIOS[13]_GPIO[192] Pad Configuration Register (SIU_PCR192)

15.8.13.50 Pad Configuration Register 193 (SIU_PCR193)

The SIU_PCR193 register controls the function, direction, and static electrical attributes of the eMIOS[14]_IRQ[0]_eTPU_A[29]_GPIO[193] pin. Only the output functions of eMIOS[14] are connected to this pin.

SIU_BASE + 0x1C2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2]			OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. When configured as eMIOS or IRQ, the OBE bit has no effect. The OBE bit must be set to ‘1’ for GPIO[193] when configured as outputs.
2. When configured as eMIOS or IRQ, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for GPIO[193] when configured as inputs.
3. The weak pull up/down selection at reset for the eMIOS[14] pins is determined by the WKPCFG pin.

Figure 193. eMIOS[14]_IRQ[0]_eTPU_A[29]_GPIO[193] Pad Configuration Register (SIU_PCR193)

15.8.13.51 Pad Configuration Register 194 (SIU_PCR194)

The SIU_PCR194 register controls the function, direction, and static electrical attributes of the eMIOS[15]_IRQ[1]_GPIO[194] pin. This register allows selection of the eMIOS and GPIO functions.

SIU_BASE+0x1C4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(3)

= Unimplemented or Reserved

1. The OBE bit must be set to ‘1’ for GPIO[194] when configured as an output.
2. When configured as eMIOS or GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for GPIO[194] when configured as an input.
3. The weak pull up/down selection at reset for the eMIOS[15] pin is determined by the WKPCFG pin.

Figure 194. eMIOS[15]_IRQ[1]_GPIO[194] Pad Configuration Register (SIU_PCR194)

15.8.13.52 Pad Configuration Register 202 (SIU_PCR202)

The SIU_PCR202 register controls the function, direction, and static electrical attributes of the eMIOS[23]_GPIO[202] pin. Both the input and output functions of eMIOS[23] are connected to the pin. The eTPU_B function is not available on this device.

SIU_BASE + 0x1D4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	U(4)

= Unimplemented or Reserved

1. The eTPU_B function is not available on this device. Do not select b’10 in the PA field.
2. The OBE bit must be set to ‘1’ for both eMIOS[23] and GPIO[202] when configured as outputs.
3. When configured as eMIOS, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. The IBE bit must be set to ‘1’ for both eMIOS[23] and GPIO[202] when configured as inputs.
4. The weak pull up/down selection at reset for the eMIOS[23] pins is determined by the WKPCFG pin.

Figure 195. eMIOS[23]_GPIO[202] Pad Configuration Register (SIU_PCR202)

15.8.13.53 Pad Configuration Registers 206 – 207 (SIU_PCR206 – SIU_PCR207)

The SIU_PCR206 – SIU_PCR207 registers control the function, direction, and static electrical attributes of the GPIO[206:207] pins.

SIU_BASE+0x1DC – SIU_BASE+0x1DE (2)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. When configured as GPO, the OBE bit should be set to ‘1’.
2. Setting the IBE bit to ‘0’ reduces power consumption. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 196. GPIO[206:207] Pad Configuration Registers (SIU_PCR206 – SIU_PCR207)

15.8.13.54 Pad Configuration Register 208 (SIU_PCR208)

The SIU_PCR208 register controls the function, direction, and static electrical attributes of the PLLREF_IRQ[4]_ETRIG[2]_GPIO[208] pin.

SIU_BASE + 0x1E0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2](1)			OBE (2)	IBE (3)	0	0	ODE	HYS (4)	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	1

= Unimplemented or Reserved

1. The PLLREF function applies only during reset. The PA field should be set to 0b010 for IRQ[4], set to 0b100 for ETRIG[2] and set to 0b000 for GPIO[208].
2. When configured as PLLREF or IRQ or ETRIG, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as PLLREF or IRQ or ETRIG, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. When configured as GPI, the IBE bit should be set to ‘1’.
4. When configured as IRQ, the HYS bit should be set to ‘1’.

Figure 197. PLLREF_IRQ[4]_ETRIG[2]_GPIO[208] Pad Configuration Register (SIU_PCR208)

15.8.13.55 Pad Configuration Register 212 (SIU_PCR212)

The SIU_PCR212 register controls the function, direction, and static electrical attributes of the BOOTCFG1_IRQ[3]_ETRIG[3]_GPIO[212] pin.

SIU_BASE + 0x1E8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2](1)			OBE (2)	IBE (3)	0	0	ODE	HYS (4)	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0

= Unimplemented or Reserved

1. The BOOTCFG1 function applies only during reset. The PA field should be set to 0b010 for IRQ[3], set to 0b100 for ETRIG[3] and set to 0b000 for GPIO[212].
2. When configured as BOOTCFG1 or IRQ or ETRIG, the OBE bit has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as BOOTCFG1 or IRQ or ETRIG, the IBE bit has no effect. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. Setting the IBE bit to ‘0’ reduces power consumption. When configured as GPI, the IBE bit should be set to ‘1’.
4. When configured as IRQ, the HYS bit should be set to ‘1’.

Figure 198. BOOTCFG1_IRQ[3]_ETRIG[3]_GPIO[212] Pad Configuration Register (SIU_PCR212)

15.8.13.56 Pad Configuration Register 213 (SIU_PCR213)

The SIU_PCR213 register controls the function, direction, and static electrical attributes of the WKPCFG_NMI_DSPI_B_SOUT_GPIO[213] pin.

SIU_BASE + 0x1EA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	ODE	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1

= Unimplemented or Reserved

1. WKPCFG function is only applicable during reset. The PA bit must be set to ‘0’ for GPIO operation
2. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 199. WKPCFG_NMI_DSPI_B_SOUT_GPIO[213] Pad Configuration Register (SIU_PCR213)

15.8.13.57 Pad Configuration Register 215 (SIU_PCR215)

The SIU_PCR215 register controls the function, direction, and static electrical attributes of the AN[12]_MA[0]_eTPU_A[19]_$\overline{SDS}$ pin. The AN[12] function is an analog pin on this device. This register allows selection of the MA[0] and $\overline{SDS}$ functions.

SIU_BASE + 0x1EE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2](1)			0	0	0	0	ODE	0	SRC[0–1]		0	0
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. Input and output buffers are enabled/disabled based on PA selection. Both input and output buffer disabled for AN[12] function. Output buffer only enabled for MA[0] and SDS functions.

Figure 200. AN[12]_MA[0]_eTPU_A[19]_$\overline{SDS}$ Pad Configuration Register (SIU_PCR215)

15.8.13.57.1 PA — Pin Assignment

The PA field for PCR215 is given in *Table 136*.

Table 136. PCR215 PA field definition

PA field	Pin function
000	SDS_BPA
100	eTPU_A[19]
010	MA[0]
011	AN[12]

15.8.13.58 Pad Configuration Register 216 (SIU_PCR216)

The SIU_PCR216 register controls the function, direction, and static electrical attributes of the AN[13]_MA[1]_eTPU_A[21]_SDO pin. The AN[13] function is not available on this device. This register allows selection of the MA[1] and SDO functions.

SIU_BASE + 0x1F0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2](1)			0	0	0	0	ODE	0	SRC[0–1]		0	0
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. Input and output buffers are enabled/disabled based on PA selection. Both input and output buffer disabled for AN[13] function. Output buffer only enabled for MA[1] and SDO functions.

Figure 201. AN[13]_MA[1]_eTPU_A[21]_SDO Pad Configuration Register (SIU_PCR216)

15.8.13.58.1 PA — Pin Assignment

The PA field for PCR216 is given in *Table 137*.

Table 137. PCR216 PA field definition

PA field	Pin function
000	SDO
100	eTPU_A[21]
010	MA[1]
011	AN[13]

15.8.13.59 Pad Configuration Register 217 (SIU_PCR217)

The SIU_PCR217 register controls the function, direction, and static electrical attributes of the AN[14]_MA[2]_eTPU_A[27]_SDI pin. The AN[14] function is an analog pin on this device. This register allows selection of the MA[2] and SDI functions.

SIU_BASE + 0x1F2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA[0–2](1)			0	0	0	0	ODE	HYS	SRC[0–1]		WPE (2)	WPS (3)
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. Input and output buffers are enabled/disabled based on PA selection. Both input and output buffer disabled for AN[14] function. Output buffer only enabled for MA[2] and SDI functions.
2. The WPE bit should be set to ‘0’ when configured as an analog input or MA[2], and set to ‘1’ when configured as SDI.
3. The WPS bit should be set to ‘1’ when configured as SDI.

Figure 202. AN[14]_MA[2]_eTPU_A[27]_SDI Pad Configuration Register (SIU_PCR217)

15.8.13.59.1 PA — Pin Assignment

The PA field for PCR217 is given in *Table 138*.

Table 138. PCR217 PA field definition

PA field	Pin function
000	SDI
100	eTPU_A[27]
010	MA[2]
011	AN[14]

15.8.13.60 Pad Configuration Register 218 (SIU_PCR218)

The SIU_PCR218 register controls the function, direction, and static electrical attributes of the AN[15]_FCK_eTPU_A[29] pin. The AN[15] function is an analog pin on this device. This register allows selection of the FCK function.

SIU_BASE + 0x1F4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		0	0	0	0	ODE	0	SRC[0–1]		0	0
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 203. AN[15]_FCK_eTPU_A[29] Pad Configuration Register (SIU_PCR218)

15.8.13.60.1 PA — Pin Assignment

The PA field for PCR218 is given in *Table 138*.

Table 139. PCR218 PA field definition

PA field	Pin function
00	eTPU_A[29]
10	FCK
11	AN[15]

15.8.13.61 Pad Configuration Register 219 (SIU_PCR219)

The SIU_PCR219 register controls the drive strength of the MCKO_GPIO[219] pin.

SIU_BASE + 0x1F6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	0	HYS	SRC[0–1]		WPE (4)	WPS (5)
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. The primary function is not selected by PA when the pin is a Nexus signal, instead it is activated by the Nexus controller (ipp_port_en_rpm). The only PA valid value is 00, all other values are reserved. The selection of MCKO is made regardless the PA value.
2. When configured as MCKO, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.
4. The WPE bit should be set to ‘0’ when configured as an analog input or MCKO, and set to ‘1’ when configured as GPIO.
5. The WPS bit should be set to ‘1’ when configured as GPIO.

Figure 204. MCKO_GPIO[219] Pad Configuration Register (SIU_PCR219)

15.8.13.62 Pad Configuration Register 220 (SIU_PCR220)

The SIU_PCR220 register controls the drive strength of the MDO[0]_eTPU_A[13]_GPIO[220] pin.

SIU_BASE + 0x1F8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	0	HYS	SRC[0–1]		WPE (3)	WPS (4)
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. When configured as MDO or eTPU, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
2. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.
3. The WPE bit should be set to ‘0’ when configured as an analog input or MA[2], and set to ‘1’ when configured as SDI.
4. The WPS bit should be set to ‘1’ when configured as SDI.

Figure 205. MDO[0]_eTPU_A[13]_GPIO[220] Pad Configuration Register (SIU_PCR220)

15.8.13.63 Pad Configuration Register 221 (SIU_PCR221)

The SIU_PCR221 register controls the drive strength of the MDO[1]_eTPU_A[19]_GPIO[221] pin.

SIU_BASE + 0x1FA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	0	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. The primary function is not selected by PA when the pin is a Nexus signal, instead it is activated by the Nexus controller (ipp_port_en_rpm)
2. When configured as MDO or eTPU, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 206. MDO[1]_eTPU_A[19]_GPIO[221] Pad Configuration Register (SIU_PCR221)

15.8.13.64 Pad Configuration Register 222 (SIU_PCR222)

The SIU_PCR222 register controls the drive strength of the MDO[2]_eTPU_A[21]_GPIO[222] pin.

SIU_BASE + 0x1FC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	0	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. The primary function is not selected by PA when the pin is a Nexus signal, instead it is activated by the Nexus controller (ipp_port_en_rpm)
2. When configured as MDO or eTPU, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 207. MDO[2]_eTPU_A[21]_GPIO[222] Pad Configuration Register (SIU_PCR222)

15.8.13.65 Pad Configuration Register 223 (SIU_PCR223)

The SIU_PCR223 register controls the drive strength of the MDO[3]_eTPU_A[25]_GPIO[223] pin.

SIU_BASE + 0x1FE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	0	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. The primary function is not selected by PA when the pin is a Nexus signal, instead it is activated by the Nexus controller (ipp_port_en_rpm)
2. When configured as MDO or eTPU, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 208. MDO[3]_eTPU_A[25]_GPIO[223] Pad Configuration Register (SIU_PCR223)

15.8.13.66 Pad Configuration Register 224 (SIU_PCR224)

The SIU_PCR224 register controls the drive strength of the MSEO[0]_eTPU_A[27]_GPIO[224] pin.

SIU_BASE + 0x200

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	0	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. The primary function is not selected by PA when the pin is a Nexus signal, instead it is activated by the Nexus controller (ipp_port_en_rpm)
2. When configured as MSEO or eTPU, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 209. MSEO[0]_eTPU_A[27]_GPIO[224] Pad Configuration Register (SIU_PCR224)

15.8.13.67 Pad Configuration Register 225 (SIU_PCR225)

The SIU_PCR225 register controls the drive strength of the MSEO[1]_eTPU_A[29]_GPIO[225] pin.

SIU_BASE + 0x202

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	0	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. The primary function is not selected by PA when the pin is a Nexus signal, instead it is activated by the Nexus controller (ipp_port_en_rpm)
2. When configured as MDO or eTPU, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 210. MSEO[1]_eTPU_A[29]_GPIO[225] Pad Configuration Register (SIU_PCR225)

15.8.13.68 Pad Configuration Register 227 (SIU_PCR227)

The SIU_PCR227 register controls the drive strength of the $\overline{\text{EVTO}}$_eTPU_A[4]_GPIO[227] pin.

SIU_BASE + 0x206

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	0	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1

= Unimplemented or Reserved

1. The primary function is not selected by PA when the pin is a Nexus signal, instead it is activated by the Nexus controller (ipp_port_en_rpm)
2. When configured as EVTO or eTPU, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 211. $\overline{\text{EVTO}}$_eTPU_A[4]_GPIO[227] Pad Configuration Register (SIU_PCR227)

15.8.13.69 Pad Configuration Register 228 (SIU_PCR228)

The SIU_PCR228 register controls the drive strength of the TDO_eMIOS[6]_GPIO[228] pin.

SIU_BASE + 0x208

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	0	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. The primary function is not selected by PA when the pin is a JTAG signal, instead it is activated by the JTAG controller (jtag_active)
2. When configured as TDO or eMIOS, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 212. TDO_eMIOS[6]_GPIO[228] Pad Configuration Register (SIU_PCR228)

15.8.13.70 Pad Configuration Register 229 (SIU_PCR229)

The SIU_PCR229 register controls the enabling/disabling and drive strength of the CLKOUT pin. The CLKOUT pin is enabled and disabled by setting and clearing the OBE bit. The CLKOUT pin is enabled during reset.

SIU_BASE + 0x20A

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

= Unimplemented or Reserved

Figure 213. CLKOUT Pad Configuration Register (SIU_PCR229)

15.8.13.71 Pad Configuration Register 230 (SIU_PCR230)

The SIU_PCR230 register controls the slew rate of the $\overline{\text{RSTOUT}}$ pin.

SIU_BASE + 0x20C

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

= Unimplemented or Reserved

Figure 214. $\overline{\text{RSTOUT}}$ Pad Configuration Register (SIU_PCR230)

15.8.13.72 Pad Configuration Register 231 (SIU_PCR231)

The SIU_PCR231 register controls the function and drive strength of the $\overline{\text{EVTI}}$_eTPU_A[2]_GPIO[231] pin.

SIU_BASE + 0x20E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1]		OBE (1)	IBE (2)	0	0	0	HYS	SRC[0–1]		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. When configured as eTPU, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
2. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 215. EVTI_eTPU_A[2]_GPIO[231] Pad Configuration Register (SIU_PCR231)

15.8.13.73 Pad Configuration Register 232 (SIU_PCR232)

The SIU_PCR232 register controls the drive strength of the TDI_eMIOS[5]_GPIO[232] pin.

SIU_BASE + 0x210

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA[0–1](1)		OBE (2)	IBE (3)	0	0	0	HYS	SRC[0–1]		0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. The primary function is not selected by PA when the pin is a JTAG signal, instead it is activated by the JTAG controller (jtag_active)
2. When configured as eMIOS, the OBE has no effect. When configured as GPO, the OBE bit should be set to ‘1’.
3. When configured as GPO, the IBE bit may be set to ‘1’ to reflect the pin state in the corresponding GPDI register. When configured as GPI, the IBE bit should be set to ‘1’.

Figure 216. TDI_eMIOS[5]_GPIO[232] Pad Configuration Register (SIU_PCR232)

15.8.13.74 Pad Configuration Register 336 (SIU_PCR336)

The SIU_PCR336 register controls the drive strength of the CAL_$\overline{\text{CS}}$[0] pin.

SIU_BASE + 0x2E0

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

= Unimplemented or Reserved

Figure 217. CAL_$\overline{CS}$[0] Pad Configuration Register (SIU_PCR336)

15.8.13.75 Pad Configuration Registers 338 – 339 (SIU_PCR338 – SIU_PCR339)

The SIU_PCR338 – SIU_PCR339 registers control the functions and drive strength of the CAL_$\overline{CS}$[2:3]_CAL_ADDR[10:11] pins.

SIU_BASE + 0x2E4 – SIU_BASE + 0x2E6 (2)

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

= Unimplemented or Reserved

Figure 218. CAL_$\overline{CS}$[2:3]_CAL_ADDR[10:11] Pad Configuration Registers (SIU_PCR338 – SIU_PCR339)

15.8.13.76 Pad Configuration Register 340 (SIU_PCR340)

The SIU_PCR340 register controls the drive strength of the CAL_ADDR[12:15] pins. Multiple pins are controlled by this PCR.

SIU_BASE + 0x2E8

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

= Unimplemented or Reserved

Figure 219. CAL_ADDR[12:15] Pad Configuration Register (SIU_PCR340)

15.8.13.77 Pad Configuration Register 341 (SIU_PCR341)

The SIU_PCR341 register controls the drive strength of the CAL_DATA[0:15] pins. Multiple pins are controlled by this PCR.

SIU_BASE + 0x2EA

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

= Unimplemented or Reserved

Figure 220. CAL_DATA[0:15] Pad Configuration Register (SIU_PCR341)

15.8.13.78 Pad Configuration Register 342 (SIU_PCR342)

The SIU_PCR342 register controls the drive strength of the CAL_RD_$\overline{WR}$, CAL_$\overline{WE}$[0:1] and CAL_$\overline{OE}$ pins. Multiple pins are controlled by this one PCR. The WEBS bit in the EBI Base Register selects between the write enable and byte enable functions.

SIU_BASE + 0x2EC

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

= Unimplemented or Reserved

Figure 221. CAL_RD_$\overline{WR}$, CAL_$\overline{WE}$[0:1]/BE[0:1] and CAL_$\overline{OE}$ Pad Configuration Register (SIU_PCR342)

15.8.13.79 Pad Configuration Register 343 (SIU_PCR343)

The SIU_PCR343 register controls the drive strength of the CAL_$\overline{TS}$_ALE pin.

SIU_BASE + 0x2EE

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

= Unimplemented or Reserved

Figure 222. CAL_$\overline{TS}$_ALE Pad Configuration Register (SIU_PCR343)

15.8.13.80 Pad Configuration Register 344 (SIU_PCR344)

The SIU_PCR344 register controls the drive strength of the ALT_MCKO and ALT_$\overline{EVTO}$ pins. Multiple pins are controlled by this PCR.

SIU_BASE + 0x2F0

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

= Unimplemented or Reserved

Figure 223. ALT_MCKO and ALT_$\overline{\text{EVTO}}$ Pad Configuration Register (SIU_PCR344)

15.8.13.81 Pad Configuration Register 345 (SIU_PCR345)

The SIU_PCR345 register controls the drive strength of the CAL_ADDR[16:27]_ALT_MDO[0:11], CAL_ADDR[28:29]_CAL_MSEO[0:1] and CAL_ADDR[30]_ALT_$\overline{\text{EVTI}}$ pins. Multiple pins are controlled by this PCR.

SIU_BASE + 0x2F2

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

= Unimplemented or Reserved

Figure 224. CAL_ADDR[16:27]_ALT_MDO[0:11], CAL_ADDR[28:29]_CAL_MSEO[0:1] and CAL_ADDR[30]_ALT_EVTI Pad Configuration Register (SIU_PCR345)

15.8.13.82 Pad Configuration Register 350 – 381 (SIU_PCR350 – SIU_PCR381)

The SIU_PCR350 – SIU_PCR381 registers control the muxing of the signals to the DSPI. PA field values are shown in *Table 140*.

SIU_BASE+0x2Fc – SIU_BASE+0x33a (32)

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

Figure 225. Pad Configuration Register 350 – 381 (SIU_PCR350 – SIU_PCR381)

Table 140. SIU_PCR350 – SIU_PCR381 DSPI muxing

Register	Address offset from SIU_BASE	DSPI serialize signal name	DSPI deserialize destination	PA value			
				0b01	0b11	0b10	0b00
SIU_PCR350	0x2FC	DSPI_B -DSI0	IRQ_0 (mux)	ETPU_A_23	—	EMIOS_11	GPIO350
SIU_PCR351	0x2FE	DSPI_B -DSI1	IRQ_1 (mux)	ETPU_A_22	—	EMIOS_10	GPIO351
SIU_PCR352	0x300	DSPI_B -DSI2	IRQ_2 (mux)	ETPU_A_21	—	EMIOS_9	GPIO352
SIU_PCR353	0x302	DSPI_B -DSI3	IRQ_3 (mux)	ETPU_A_20	—	EMIOS_8	GPIO353

Table 140. SIU_PCR350 – SIU_PCR381 DSPI muxing (continued)

Register	Address offset from SIU_BASE	DSPI serialize signal name	DSPI deserialize destination	PA value			
				0b01	0b11	0b10	0b00
SIU_PCR354	0x304	DSPI_B -DSI4	IRQ_4 (mux)	ETPU_A_19	—	EMIOS_6	GPIO354
SIU_PCR355	0x306	DSPI_B -DSI5	IRQ_5 (mux)	ETPU_A_18	—	EMIOS_5	GPIO355
SIU_PCR356	0x308	DSPI_B -DSI6	IRQ_6 (mux)	ETPU_A_17	—	EMIOS_4	GPIO356
SIU_PCR357	0x30A	DSPI_B -DSI7	IRQ_7 (mux)	ETPU_A_16	—	EMIOS_3	GPIO357
SIU_PCR358	0x30C	DSPI_B -DSI8	ETPU_A_29 IRQ_8 (mux)	ETPU_A_29	—	EMIOS_2	GPIO358
SIU_PCR359	0x30E	DSPI_B -DSI9	ETPU_A_28 IRQ_9 (mux)	ETPU_A_28	—	EMIOS_1	GPIO359
SIU_PCR360	0x310	DSPI_B -DSI10	ETPU_A_27 IRQ_10 (mux)	ETPU_A_27	—	EMIOS_0	GPIO360
SIU_PCR361	0x312	DSPI_B -DSI11	ETPU_A_26 IRQ_11 (mux)	ETPU_A_26	—	EMIOS_23	GPIO361
SIU_PCR362	0x314	DSPI_B -DSI12	ETPU_A_25 IRQ_12 (mux)	ETPU_A_25	—	EMIOS_15	GPIO362
SIU_PCR363	0x316	DSPI_B -DSI13	ETPU_A_24 IRQ_13 (mux)	ETPU_A_24	—	EMIOS_14	GPIO363
SIU_PCR364	0x318	DSPI_B -DSI14	EMIOS_13 IRQ_14 (mux)	ETPU_A_31	—	EMIOS_13	GPIO364
SIU_PCR365	0x31A	DSPI_B -DSI15	EMIOS_12 IRQ_15 (mux)	ETPU_A_30	—	EMIOS_12	GPIO365
SIU_PCR366	0x31C	DSPI_B -DSI16	—	ETPU_A_12	—	EMIOS_23	GPIO366
SIU_PCR367	0x31E	DSPI_B -DSI17	—	ETPU_A_13	—	EMIOS_15	GPIO367
SIU_PCR368	0x320	DSPI_B -DSI18	—	ETPU_A_14	—	EMIOS_14	GPIO368
SIU_PCR369	0x322	DSPI_B -DSI19	—	ETPU_A_15	—	EMIOS_13	GPIO369
SIU_PCR370	0x324	DSPI_B -DSI20	—	ETPU_A_0	—	EMIOS_12	GPIO370
SIU_PCR371	0x326	DSPI_B -DSI21	—	ETPU_A_1	—	EMIOS_11	GPIO371
SIU_PCR372	0x328	DSPI_B -DSI22	—	ETPU_A_2	—	EMIOS_10	GPIO372
SIU_PCR373	0x32A	DSPI_B -DSI23	—	ETPU_A_3	—	EMIOS_9	GPIO373
SIU_PCR374	0x32C	DSPI_B -DSI24	—	ETPU_A_4	—	EMIOS_8	GPIO374
SIU_PCR375	0x32E	DSPI_B -DSI25	—	ETPU_A_5	—	EMIOS_6	GPIO375
SIU_PCR376	0x330	DSPI_B -DSI26	—	ETPU_A_6	—	EMIOS_5	GPIO376
SIU_PCR377	0x332	DSPI_B -DSI27	—	ETPU_A_7	—	EMIOS_4	GPIO377
SIU_PCR378	0x334	DSPI_B -DSI28	—	ETPU_A_8	—	EMIOS_3	GPIO378
SIU_PCR379	0x336	DSPI_B -DSI29	—	ETPU_A_9	—	EMIOS_2	GPIO379
SIU_PCR380	0x338	DSPI_B -DSI30	—	ETPU_A_10	—	EMIOS_1	GPIO380
SIU_PCR381	0x33A	DSPI_B -DSI31	—	ETPU_A_11	—	EMIOS_0	GPIO381

15.8.13.83 Pad Configuration Register 382 – 389 (SIU_PCR382 – SIU_PCR389)

Note: The SIU_PCR382 – SIU_PCR413 registers control the muxing of the signals to the DSPI. See Section 3.5: DSPI Connections to eTPU_A, eMIOS and SIU for more information.

The SIU_PCR382 – SIU_PCR389 registers control the muxing of the signals to the DSPI. PA field values are shown in *Table 141*.

SIU_BASE+0x33C – SIU_BASE+0x34A (8)

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

Figure 226. Pad Configuration Register 382 – 389 (SIU_PCR382 – SIU_PCR389)

Table 141. SIU_PCR382 – SIU_PCR389 DSPI muxing

Register	Address offset from SIU_BASE	DSPI serialize signal name	DSPI deserialize destination	PA value			
				0b001	0b011	0b010	0b000
SIU_PCR382	0x33C	DSPI_C -DSI0	IRQ_15 (mux)	ETPU_A_12	EMIOS_7	EMIOS_12	GPIO382
SIU_PCR 383	0x33E	DSPI_C -DSI1	IRQ_0 (mux)	ETPU_A_13	EMIOS_16	EMIOS_13	GPIO383
SIU_PCR 384	0x340	DSPI_C -DSI2	IRQ_1 (mux)	ETPU_A_14	EMIOS_17	EMIOS_14	GPIO384
SIU_PCR 385	0x342	DSPI_C -DSI3	IRQ_2 (mux)	ETPU_A_15	EMIOS_18	EMIOS_15	GPIO385
SIU_PCR 386	0x344	DSPI_C -DSI4	IRQ_3 (mux)	ETPU_A_0	EMIOS_19	EMIOS_23	GPIO386
SIU_PCR 387	0x346	DSPI_C -DSI5	IRQ_4 (mux)	ETPU_A_1	EMIOS_20	EMIOS_0	GPIO387
SIU_PCR 388	0x348	DSPI_C -DSI6	IRQ_5 (mux)	ETPU_A_2	EMIOS_21	EMIOS_1	GPIO388
SIU_PCR 389	0x34A	DSPI_C -DSI7	IRQ_6 (mux)	ETPU_A_3	EMIOS_22	EMIOS_2	GPIO389

15.8.13.84 Pad Configuration Register 390 – 413 (SIU_PCR390 – SIU_PCR413)

The SIU_PCR390 – SIU_PCR413 registers control the muxing of the signals to the DSPI. PA field values are shown in *Table 142*.

SIU_BASE+0x34C – SIU_BASE+0x37A (24)

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

Figure 227. Pad Configuration Register 390 – 413 (SIU_PCR390 – SIU_PCR413)

Table 142. SIU_PCR390 – SIU_PCR413 DSPI muxing

Register	Address offset from SIU_BASE	DSPI serialize signal name	DSPI deserialize destination	PA value			
				0b01	0b11	0b10	0b00
SIU_PCR 390	0x34C	DSPI_C -DSI8	IRQ_7 (mux)	ETPU_A_4	—	EMIOS_3	GPIO390
SIU_PCR 391	0x34E	DSPI_C -DSI9	IRQ_8 (mux)	ETPU_A_5	—	EMIOS_4	GPIO391
SIU_PCR 392	0x350	DSPI_C -DSI10	IRQ_9 (mux)	ETPU_A_6	—	EMIOS_5	GPIO392
SIU_PCR 393	0x352	DSPI_C -DSI11	IRQ_10 (mux)	ETPU_A_7	—	EMIOS_6	GPIO393
SIU_PCR 394	0x354	DSPI_C -DSI12	IRQ_11 (mux)	ETPU_A_8	—	EMIOS_8	GPIO394
SIU_PCR 395	0x356	DSPI_C -DSI13	IRQ_12 (mux)	ETPU_A_9	—	EMIOS_9	GPIO395
SIU_PCR 396	0x358	DSPI_C -DSI14	IRQ_13 (mux)	ETPU_A_10	—	EMIOS_10	GPIO396
SIU_PCR 397	0x35A	DSPI_C -DSI15	IRQ_14 (mux)	ETPU_A_11	—	EMIOS_11	GPIO397
SIU_PCR 398	0x35C	DSPI_C -DSI16	—	ETPU_A_23	—	EMIOS_0	GPIO398
SIU_PCR 399	0x35E	DSPI_C -DSI17	—	ETPU_A_22	—	EMIOS_1	GPIO399
SIU_PCR 400	0x360	DSPI_C -DSI18	—	ETPU_A_21	—	EMIOS_2	GPIO400
SIU_PCR 401	0x362	DSPI_C -DSI19	—	ETPU_A_20	—	EMIOS_3	GPIO401
SIU_PCR 402	0x364	DSPI_C -DSI20	—	ETPU_A_19	—	EMIOS_4	GPIO402
SIU_PCR 403	0x366	DSPI_C -DSI21	—	ETPU_A_18	—	EMIOS_5	GPIO403
SIU_PCR 404	0x368	DSPI_C -DSI22	—	ETPU_A_17	—	EMIOS_6	GPIO404
SIU_PCR 405	0x36A	DSPI_C -DSI23	—	ETPU_A_16	—	EMIOS_8	GPIO405
SIU_PCR 406	0x36C	DSPI_C -DSI24	—	ETPU_A_29	—	EMIOS_9	GPIO406
SIU_PCR 407	0x36E	DSPI_C -DSI25	—	ETPU_A_28	—	EMIOS_10	GPIO407
SIU_PCR 408	0x370	DSPI_C -DSI26	—	ETPU_A_27	—	EMIOS_11	GPIO408
SIU_PCR 409	0x372	DSPI_C -DSI27	—	ETPU_A_26	—	EMIOS_12	GPIO409
SIU_PCR 410	0x374	DSPI_C -DSI28	—	ETPU_A_25	—	EMIOS_13	GPIO410
SIU_PCR 411	0x376	DSPI_C -DSI29	—	ETPU_A_24	—	EMIOS_14	GPIO411
SIU_PCR 412	0x378	DSPI_C -DSI30	—	ETPU_A_31	—	EMIOS_15	GPIO412
SIU_PCR 413	0x37A	DSPI_C -DSI31	—	ETPU_A_30	—	EMIOS_23	GPIO413

15.8.14 GPIO Pin Data Output Registers (SIU_GPDO83_86 – SIU_GPDO230_232)

The definition of the SIU_GPDO*x*_*x* registers is given in *Figure 228* and *Figure 229*. Each of the 148 PDO (71 used) bits correspond to the pin with the same GPIO pin number. For example, PDO0 is the pin data output bit for the $\overline{\text{CS}}$[0]_ADDR[8]_GPIO[0] pin, and PDO213 is the pin data output bit for the WKPCFG_GPIO[213] pin. Gaps exist in this memory space where the pin is not available in the package.

The SIU_GPDO*x*_*x* registers are written to by software to drive data out on the external GPIO pin. Each byte of a register drives a single external GPIO pin, which allows the state of the pin to be controlled independently from other GPIO pins. Writes to the SIU_GPDO*x*_*x* registers have no effect on pin states if the pins are configured as inputs by the associated Pad Configuration Registers. The SIU_GPDO*x*_*x* register values are automatically driven to

the GPIO pins without software update if the direction of the GPIO pins is changed from input to output.

Writes to the SIU_GPDO*x*_*x* registers have no effect on the state of the corresponding pins when the pins are configured for their primary function by the corresponding PCR.

SIU_BASE + 0x653

	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 83	0	0	0	0	0	0	0	PDO 84
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	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 85	0	0	0	0	0	0	0	PDO 86
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 228. GPIO Pin Data Out Register 83 – 86 (SIU_GPDO83 – SIU_GPDO86)

SIU_BASE + 0x6EA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDO230	0	0	0	0	0	0	0	PDO231
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDO232	0	0	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

Figure 229. GPIO Pin Data Out Register 230 – 232 (SIU_GPDO230 – SIU_GPDO232)

Table 143. SIU_GPDO*x* Register field descriptions

Field	Description
PDO*x*	Pin Data Out This bit stores the data to be driven out on the external GPIO pin associated with the register. 1 VOH is driven on the external GPIO pin when the pin is configured as an output. 0 VOL is driven on the external GPIO pin when the pin is configured as an output.

15.8.15 GPO Data Output Registers (SIU_GPDO350 – SIU_GPDO413)

The definition of the SIU_GPDO*x*_*x* registers is given in *Figure 230* and *Figure 231*. Each of the 148 PDO (71 used) bits correspond to the pin with the same GPIO pin number. For example, PDO0 is the pin data output bit for the CS[0]_ADDR[8]_GPIO[0] pin, and PDO213

is the pin data output bit for the WKPCFG_GPIO[213] pin. Gaps exist in this memory space where the pin is not available in the package.

The SIU_GPDO*x_x* registers are written to by software to drive data out on the external GPIO pin. Each byte of a register drives a single external GPIO pin, which allows the state of the pin to be controlled independently from other GPIO pins. Writes to the SIU_GPDO*x_x* registers have no effect on pin states if the pins are configured as inputs by the associated Pad Configuration Registers. The SIU_GPDO*x_x* register values are automatically driven to the GPIO pins without software update if the direction of the GPIO pins is changed from input to output.

Writes to the SIU_GPDO*x_x* registers have no effect on the state of the corresponding pins when the pins are configured for their primary function by the corresponding PCR.

SIU_BASE + 0x75E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDO350	0	0	0	0	0	0	0	PDO351
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDO352	0	0	0	0	0	0	0	PDO353
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 230. GPIO Pin Data Out Register 350 – 353 (SIU_GPDO350 – SIU_GPDO353)

SIU_BASE + 0x79D

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDO410	0	0	0	0	0	0	0	PDO411
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDO412	0	0	0	0	0	0	0	PDO413
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 231. GPIO Pin Data Out Register 410 – 413 (SIU_GPDO410 – SIU_GPDO413)

Table 144. SIU_GPDO*x* Register field descriptions

Field	Description
PDO*x*	Pin Data Out This bit stores the data to be driven out on the external GPIO pin associated with the register. 1 VOH is driven on the external GPIO pin when the pin is configured as an output. 0 VOL is driven on the external GPIO pin when the pin is configured as an output.

15.8.16 GPIO Pin Data Input Registers (SIU_GPDI83_86 – SIU_GPDI_232)

The definition of the SIU_GPDI*x*_*x* registers is given in *Figure 232* and *Figure 233*. Each of the 148 GPDI (71 used) bits correspond to the pin with the same GPIO pin number. For example, GPDI0 is the pin data input bit for the CS[0]_GPIO[0] pin, and PDI213 is the pin data input bit for the WKPCFG_GPIO[213] pin. Gaps exist in this memory space where the pin is not available in the package.

The SIU_GPDI*x*_*x* registers are read-only registers that allow software to read the input state of an external GPIO pin. Each byte of a register represents the input state of a single external GPIO pin. If the GPIO pin is configured as an output, and the input buffer enable (IBE) bit is set in the associated Pad Configuration Register, the SIU_GPDI*x*_*x* register reflects the actual state of the output pin.

SIU_BASE + 0x853

	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 83	0	0	0	0	0	0	0	PDI 84
W																
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	U

	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 85	0	0	0	0	0	0	0	PDI 86
W																
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	U

= Unimplemented or Reserved

Figure 232. GPIO Pin Data In Register 83 – 86 (SIU_GPDI83 – SIU_GPDI86)

SIU_BASE + 0x8EA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDI230	0	0	0	0	0	0	0	PDI231
W																
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDI232	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 233. GPIO Pin Data In Register 230 – 232 (SIU_GPDI230 – SIU_GPDI232)

Table 145. SIU_GPDI*x* Register field descriptions

Field	Description
PDI*x*	Pin Data In This bit stores the input state on the external GPIO pin associated with the register. 1 Signal on pin is greater than or equal to VIH. 0 Signal on pin is less than or equal to VIL.

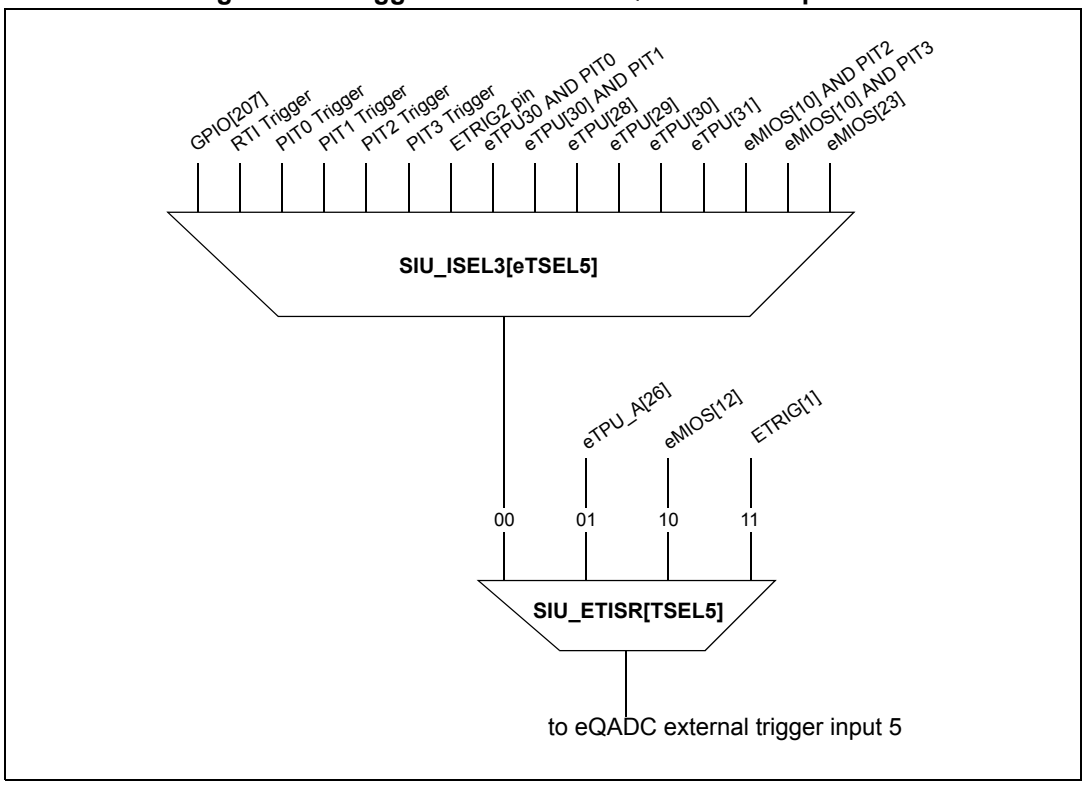
15.8.17 eQADC Trigger Input Select Register (SIU_ETISR)

The eQADC Trigger Input Select Register (SIU_ETISR) selects the source for the eQADC trigger inputs.

The fields in this register operate in conjunction with the corresponding fields in the SIU_ISEL3 register. Each field in the SIU_ETISR offers direct selection of three signals to be used as a trigger to a eQADC CFIFO queue. The TSEL5 field is used to select the trigger source for eQADC CFIFO5, the TSEL4 field selects the trigger source for eQADC CFIFO4, and so on. Additionally, each SIU_ETISR field offers selection among a group of signals using the corresponding field in the SIU_ISEL3 register.

Figure 234 illustrates the available trigger selections for eQADC CFIFO5.

Figure 234. Trigger selection for eQADC CFIFO queue 5



As shown in *Figure 234*, the TSEL5 field can be used to select the eTPU_A[26], eMIOS[12] or GPIO[207] signal or, by assigning a value of 0b00 to the TSEL5 field, a variety of other signals can be selected using the eTSEL5 field of the SIU_ISEL3 register.

SIU_BASE + 0x900

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TSEL5[0–1]		TSEL4[0–1]		TSEL3[0–1]		TSEL2[0–1]		TSEL1[0–1]		TSEL0[0–1]		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	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

Figure 235. eQADC Trigger Input Select Register (SIU_ETISR)

Table 146. SIU_ETISR field descriptions

Field	Description
0–1 TSEL5[0–1]	eQADC Trigger Input Select 5 The input for eQADC trigger 5 is specified by the TSEL5 field as follows: 00 SIU_ISEL3[eTSEL5} 01 eTPU_A[26] channel 10 eMIOS[12] channel 11 ETRIG[1] pin
2–3 TSEL4[0–1]	eQADC Trigger Input Select 4 The input for eQADC trigger 4 is specified by the TSEL4 field as follows: 00 SIU_ISEL3[eTSEL4] 01 eTPU_A[27] channel 10 eMIOS[13] channel 11 ETRIG[0] pin
4–5 TSEL3[0–1]	eQADC Trigger Input Select 3 The input for eQADC trigger 3 is specified by the TSEL3 field as follows: 00 SIU_ISEL3[eTSEL3] 01 eTPU_A[28] channel 10 eMIOS[14] channel 11 ETRIG[1] pin
6–7 TSEL2[0–1]	eQADC Trigger Input Select 2 The input for eQADC trigger 2 is specified by the TSEL2 field as follows: 00 SIU_ISEL3[eTSEL2] 01 eTPU_A[29] channel 10 eMIOS[15] channel 11 ETRIG[0] pin
8–9 TSEL1[0–1]	eQADC Trigger Input Select 1 The input for eQADC trigger 1 is specified by the TSEL1 field as follows: 00 SIU_ISEL3[eTSEL1] 01 eTPU_A[31] channel 10 eMIOS[11] channel 11 ETRIG[1] pin
10–11 TSEL0[0–1]	eQADC Trigger Input Select 0 The input for eQADC trigger 0 is specified by the TSEL0 field as follows. 00 SIU_ISEL3[eTSEL0] 01 eTPU_A[30] channel 10 eMIOS[10] channel 11 ETRIG[0] pin
12–31	Reserved

15.8.18 External IRQ Input Select Register (SIU_EIISR)

The External IRQ Input Select Register (SIU_EIISR) selects the source for the external interrupt/DMA inputs.

SIU_BASE + 0x904

	0 1	2 3	4 5	6 7	8 9	10 11	12 13	14 15
R W	ESEL15[0–1]	ESEL14[0–1]	ESEL13[0–1]	ESEL12[0–1]	ESEL11[0–1]	ESEL10[0–1]	ESEL9[0–1]	ESEL8[0–1]
Reset	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0

	16 17	18 19	20 21	22 23	24 25	26 27	28 29	30 31
R W	ESEL7[0–1]	ESEL6[0–1]	ESEL5[0–1]	ESEL4[0–1]	ESEL3[0–1]	ESEL2[0–1]	ESEL1[0–1]	ESEL0[0–1]
Reset	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0

= Unimplemented or Reserved

Figure 236. External IRQ Input Select Register (SIU_EIISR)

Table 147. SIU_EIISR field descriptions

Field	Description
0–1 ESEL15[0–1]	External IRQ Input Select 15 The IRQ[15] input is specified by the ESEL15 field as follows: 00 IRQ[15] pin 01 DSPI_B[15] serialized input 10 DSPI_C[0] serialized input 11 Reserved
2–3 ESEL14[0–1]	External IRQ Input Select 14 The IRQ[14] input is specified by the ESEL14 field as follows: 00 IRQ[14] pin 01 DSPI_B[14] serialized input 10 DSPI_C[15] serialized input 11 Reserved
4–5 ESEL13[0–1]	External IRQ Input Select 13 The IRQ[13] input is specified by the ESEL13 field as follows: 00 IRQ[13] pin 01 DSPI_B[13] serialized input 10 DSPI_C[14] serialized input 11 Reserved
6–7 ESEL12[0–1]	External IRQ Input Select 12 The IRQ[12] input is specified by the ESEL12 field as follows: 00 IRQ[12] pin 01 DSPI_B[12] serialized input 10 DSPI_C[13] serialized input 11 Reserved
8–9 ESEL11[0–1]	External IRQ Input Select 11 The IRQ[11] input is specified by the ESEL11 field as follows: 00 IRQ[11] pin 01 DSPI_B[11] serialized input 10 DSPI_C[12] serialized input 11 Reserved

Table 147. SIU_EIISR field descriptions (continued)

Field	Description
10–11 ESEL10[0–1]	External IRQ Input Select 10 The IRQ[10] input is specified by the ESEL10 field as follows. 00 IRQ[10] pin 01 DSPI_B[10] serialized input 10 DSPI_C[11] serialized input 11 Reserved
12–13 ESEL9[0–1]	External IRQ Input Select 9 The IRQ[9] input is specified by the ESEL9 field as follows: 00 IRQ[9] pin 01 DSPI_B[9] serialized input 10 DSPI_C[10] serialized input 11 Reserved
14–15 ESEL8[0–1]	External IRQ Input Select 8 The IRQ[8] input is specified by the ESEL8 field as follows: 00 IRQ[8] pin 01 DSPI_B[8] serialized input 10 DSPI_C[9] serialized input 11 Reserved
16–17 ESEL7[0–1]	External IRQ Input Select 7 The IRQ[7] input is specified by the ESEL7 field as follows: 00 IRQ[7] pin 01 DSPI_B[7] serialized input 10 DSPI_C[8] serialized input 11 Reserved
18–19 ESEL6[0–1]	External IRQ Input Select 6 The IRQ[6] input is specified by the ESEL6 value as shown below. IRQ[6] is multiplexed on the TCRCLK_B pin, which is not available in any of the packages for this device. 00 Disabled (mux output is 1’b0) 01 DSPI_B[6] serialized input 10 DSPI_C[7] serialized input 11 Reserved
20–21 ESEL5[0–1]	External IRQ Input Select 5 The IRQ[5] input is specified by the ESEL5 field as follows: 00 IRQ[5] pin 01 DSPI_B[5] serialized input 10 DSPI_C[6] serialized input 11 Reserved
22–23 ESEL4[0–1]	External IRQ Input Select 4 The IRQ[4] input is specified by the ESEL4 field as follows: 00 IRQ[4] pin 01 DSPI_B[4] serialized input 10 DSPI_C[5] serialized input 11 Reserved

Table 147. SIU_EIISR field descriptions (continued)

Field	Description
24–25 ESEL3[0–1]	External IRQ Input Select 3 The IRQ[3] input is specified by the ESEL3 field as follows: 00 IRQ[3] pin 01 DSPI_B[3] serialized input 10 DSPI_C[4] serialized input 11 Reserved
26–27 ESEL2[0–1]	External IRQ Input Select 2 The IRQ[2] input is specified by the ESEL2 field as follows: 00 IRQ[2] pin 01 DSPI_B[2] serialized input 10 DSPI_C[3] serialized input 11 Reserved
28–29 ESEL1[0–1]	External IRQ Input Select 1 The IRQ[1] input is specified by the ESEL1 field as follows: 00 IRQ[1] pin 01 DSPI_B[1] serialized input 10 DSPI_C[2] serialized input 11 eMIOS[15]_IRQ[1] pin
30–31 ESEL0[0–1]	External IRQ Input Select 0 The IRQ[0] input is specified by the ESEL0 field as follows: 00 IRQ[0] pin 01 DSPI_B[0] serialized input 10 DSPI_C[1] serialized input 11 eMIOS[14]_IRQ[0] pin

15.8.19 DSPI Input Select Register (SIU_DISR)

The DSPI Input Select Register (SIU_DISR) register specifies the source of each DSPI data input, slave select, clock input, and trigger input to allow serial and parallel chaining of the DSPI blocks.

SIU_BASE + 0x908

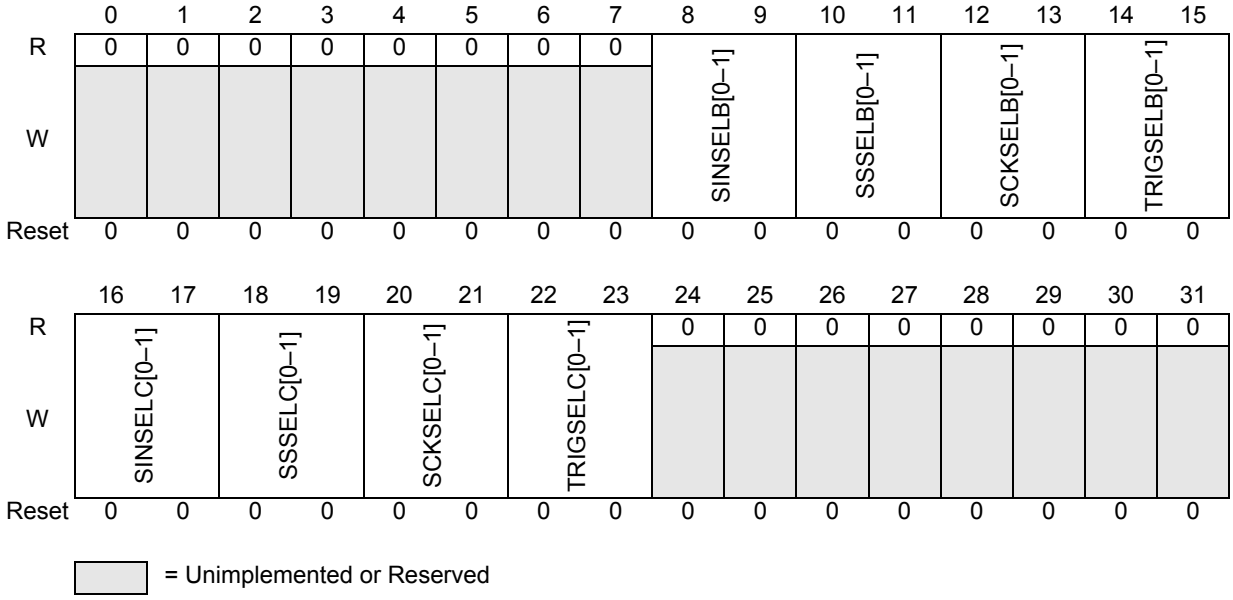


Figure 237. DSPI Input Select Register (SIU_DISR)

Table 148. SIU_DISR field descriptions

Field	Description
0–7	Reserved
8–9 SINSELB[0–1]	DSPI_B Data Input Select The source of the data input of DSPI_B is specified by the SINSELB field as follows: 00 DSPI_B_SIN_DSPI_C_PCS[2]_GPIO[103] pin 01 Reserved 10 DSPI_C_SOUT 11 SOUT_D (not available)
10–11 SSSELB[0–1]	DSPI_B Slave Select Input Select The source of the slave select input of DSPI_B is specified by the SSSELB field as follows: 00 DSPI_B_PCS[0]_GPIO[105] pin 01 Reserved 10 DSPI_C_CS[0] (Master) 11 Reserved
12–13 SCKSELB[0–1]	DSPI_B Clock Input Select The source of the clock input of DSPI_B when in slave mode is specified by the SCKSELB field as follows: 00 DSPI_B_SCK_DSPI_C_PCS[1]_GPIO[102] pin 01 Reserved 10 DSPI_C_SCK (Master) 11 Reserved

Table 148. SIU_DISR field descriptions (continued)

Field	Description
14–15 TRIGSELB[0–1]	DSPI_B Trigger Input Select The source of the trigger input of DSPI_B for master or slave mode is specified by the TRIGSELB field as follows: 00 Reserved 01 Reserved 10 DSPI_C_CS[4] 11 Reserved
16–17 SINSELC[0–1]	DSPI_C Data Input Select The source of the data input of DSPI_C is specified by the SINSELC field as follows: 00 DSPI_A_PCS[2]_DSPI_C_SIN_GPIO[108] pin 01 Reserved 10 DSPI_B_SOUT 11 Reserved
18–19 SSSELC[0–1]	DSPI_C Slave Select Input Select The source of the slave select input of DSPI_C is specified by the SSSELC field as follows: 00 DSPI_B_PCS[5]_DSPI_C_PCS[0]_GPIO[110] pin 01 Reserved 10 DSPI_B_CS[0] (Master) 11 Reserved
20–21 SCKSELC[0–1]	DSPI_C Clock Input Select The source of the clock input of DSPI_C when in slave mode is specified by the SCKSELC field as follows: 00 DSPI_B_PCS[4]_DSPI_C_SCK_GPIO[109] pin 01 Reserved 10 DSPI_B_SCK (Master) 11 Reserved
22–23 TRIGSELC[0–1]	DSPI_C Trigger Input Select The source of the trigger input of DSPI_C for master or slave mode is specified by the TRIGSELC field as follows: 00 Reserved 01 Reserved 10 DSPI_B_CS[4] 11 Reserved
24–31	Reserved

15.8.20 IMUX Select Register 3 (SIU_ISEL3)

The SIU_ISEL3 register selects the source for the external eQADC trigger inputs.

Note: This register operates in conjunction with the SIU_ETISR. See Section 15.8.17: eQADC Trigger Input Select Register (SIU_ETISR) for details.

SIU_BASE + 0x90C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	Reserved		ETSEL5[0–4]					ETSEL4[0–4]					ETSEL3[0–3]			
Reset	0	0	0	0	0	0	0	0	0	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	ETSEL3[4]		ETSEL2[0–4]					ETSEL1[0–4]					ETSEL0[0–4]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 238. IMUX Select Register 3 (SIU_ISEL3)

Table 149. SIU_ISEL3 Register Field Descriptions

Field	Description
0–1	Reserved
2–6 ETSEL5[0-4]	00000 GPIO207 (eTRIG1) 00001 RTI Trigger 00010 PIT0 Trigger 00011 PIT1 Trigger 00100 PIT2 Trigger 00101 PIT3 Trigger 00110 Reserved 00111 PLLREF (eTRIG2) 01000 eTPU30 AND PIT0 01001 eTPU30 AND PIT1 01010 Reserved 01011 Reserved 01100 eTPU28 01101 eTPU29 01110 eTPU30 01111 eTPU31 10000 Reserved 10001 Reserved 10010 Reserved 10011 Reserved 10100 eMIOS10 AND PIT2 10101 eMIOS10 AND PIT3 10110 Reserved 10111 Reserved 11000 Reserved 11001 Reserved 11010 Reserved 11011 Reserved 11100 Reserved 11101 Reserved 11110 Reserved 11111 eMIOS23

Table 149. SIU_ISEL3 Register Field Descriptions (continued)

Field	Description
7–11 ETSEL4[0-4]	00000 GPIO206 (eTRIG0) 00001 RTI Trigger 00010 PIT0 Trigger 00011 PIT1 Trigger 00100 PIT2 Trigger 00101 PIT3 Trigger 00110 Reserved 00111 BOOTCFG[1] (eTRIG3) 01000 eTPU30 AND PIT0 01001 eTPU30 AND PIT1 01010 Reserved 01011 Reserved 01100 eTPUA28 01101 eTPUA29 01110 eTPUA30 01111 eTPUA31 10000 Reserved 10001 Reserved 10010 Reserved 10011 Reserved 10100 eMIOS10 AND PIT2 10101 eMIOS10 AND PIT3 10110 Reserved 10111 Reserved 11000 Reserved 11001 Reserved 11010 Reserved 11011 Reserved 11100 Reserved 11101 Reserved 11110 Reserved 11111 eMIOS23

Table 149. SIU_ISEL3 Register Field Descriptions (continued)

Field	Description
12–16 ETSEL3[0-3]	00000 GPIO207 (eTRIG1) 00001 RTI Trigger 00010 PIT0 Trigger 00011 PIT1 Trigger 00100 PIT2 Trigger 00101 PIT3 Trigger 00110 Reserved 00111 PLLREF (eTRIG2) 01000 eTPU30 AND PIT0 01001 eTPU30 AND PIT1 01010 Reserved 01011 Reserved 01100 eTPUA28 01101 eTPUA29 01110 eTPUA30 01111 eTPUA31 10000 Reserved 10001 Reserved 10010 Reserved 10011 Reserved 10100 eMIOS10 AND PIT2 10101 eMIOS10 AND PIT3 10110 Reserved 10111 Reserved 11000 Reserved 11001 Reserved 11010 Reserved 11011 Reserved 11100 Reserved 11101 Reserved 11110 Reserved 11111 eMIOS23

Table 149. SIU_ISEL3 Register Field Descriptions (continued)

Field	Description
17–21 ETSEL2[0-4]	00000 GPIO206 (eTRIG0) 00001 RTI Trigger 00010 PIT0 Trigger 00011 PIT1 Trigger 00100 PIT2 Trigger 00101 PIT3 Trigger 00110 Reserved 00111 BOOTCFG[1] (eTRIG3) 01000 eTPU30 AND PIT0 01001 eTPU30 AND PIT1 01010 Reserved 01011 Reserved 01100 eTPUA28 01101 eTPUA29 01110 eTPUA30 01111 eTPUA31 10000 Reserved 10001 Reserved 10010 Reserved 10011 Reserved 10100 eMIOS10 AND PIT2 10101 eMIOS10 AND PIT3 10110 Reserved 10111 Reserved 11000 Reserved 11001 Reserved 11010 Reserved 11011 Reserved 11100 Reserved 11101 Reserved 11110 Reserved 11111 eMIOS23

Table 149. SIU_ISEL3 Register Field Descriptions (continued)

Field	Description
22–26 ETSEL1[0-4]	00000 GPIO207 (eTRIG1) 00001 RTI Trigger 00010 PIT0 Trigger 00011 PIT1 Trigger 00100 PIT2 Trigger 00101 PIT3 Trigger 00110 Reserved 00111 PLLREF (eTRIG2) 01000 eTPU31 AND PIT0 01001 eTPU31 AND PIT1 01010 Reserved 01011 Reserved 01100 eTPUA28 01101 eTPUA29 01110 eTPUA30 01111 eTPUA31 10000 Reserved 10001 Reserved 10010 Reserved 10011 Reserved 10100 eMIOS11 AND PIT2 10101 eMIOS11 AND PIT3 10110 Reserved 10111 Reserved 11000 Reserved 11001 Reserved 11010 Reserved 11011 Reserved 11100 Reserved 11101 Reserved 11110 Reserved 11111 eMIOS23

Table 149. SIU_ISEL3 Register Field Descriptions (continued)

Field	Description
27–31 ETSEL0[0-4]	00000 GPIO206 (eTRIG0) 00001 RTI Trigger 00010 PIT0 Trigger 00011 PIT1 Trigger 00100 PIT2 Trigger 00101 PIT3 Trigger 00110 Reserved 00111 BOOTCFG[1] (eTRIG3) 01000 eTPU30 AND PIT0 01001 eTPU30 AND PIT1 01010 Reserved 01011 Reserved 01100 eTPUA28 01101 eTPUA29 01110 eTPUA30 01111 eTPUA31 10000 Reserved 10001 Reserved 10010 Reserved 10011 Reserved 10100 eMIOS10 AND PIT2 10101 eMIOS10 AND PIT3 10110 Reserved 10111 Reserved 11000 Reserved 11001 Reserved 11010 Reserved 11011 Reserved 11100 Reserved 11101 Reserved 11110 Reserved 11111 eMIOS23

Note: For options 0b01000, 0b01001, 0b10100, 0b10101 for each queue, two trigger sources are logically ANDed together. The PIT is intended to provide the regular cyclic trigger, while the eTPU or eMIOS channels 'gate' that cyclic trigger. This way, the ADC can make regular samples but only during a given time or angle window.

15.8.21 IMUX Select Register 8 (SIU_ISEL8)

The SIU_ISEL 8 register is used to multiplex the ETPU[24:29] inputs. These six eTPU channels can come from the output of the DSPI or the corresponding pad.

SIU_BASE+0x920

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R												ESEL5				ESEL4
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R				ESEL3				ESEL2				ESEL1				ESEL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 239. IMUX Select Register 8 (SIU_ISEL8)

Table 150. SIU_ISEL8 Register field descriptions

Field	Description
0–10	Reserved
11 ESEL5	eTPU[29] input channel 1 eTPU channel 29 0 DSPI_B Serialized input 8
12–14	Reserved
15 ESEL4	eTPU[28] input channel 1 eTPU channel 28 0 DSPI_B Serialized input 9
16–18	Reserved
19 ESEL3	eTPU[27] input channel 1 eTPU channel 27 0 DSPI_B Serialized input 10
20–22	Reserved
23 ESEL2	eTPU[26] input channel 1 eTPU channel 26 0 DSPI_B Serialized input 11
24–26	Reserved
27 ESEL1	eTPU[25] input channel 1 eTPU channel 25 0 DSPI_B Serialized input 12
28–29	Reserved
31 ESEL0	eTPU[24] input channel 1 eTPU channel 24 0 DSPI_B Serialized input 13

15.8.22 IMUX Select Register 9 (SIU_ISEL9)

The eQADC has a new mode of operation called ‘Streaming’. This mode requires a second trigger for queue0. The source pin for this trigger can come from eTPU, eMIOS or PIT channels. A mux select register is required to select the source of this new queue0 trigger.

SIU_BASE+0x924

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R												eTSEL0A				
W																
Reset	0	0	0	0	0	0	0	0	*	*	*	*	**	0	0	0

= Unimplemented or Reserved

* Reset state determined during reset configuration

** Reset state determined during reset configuration

Figure 240. IMUX Select Register 9 (SIU_ISEL9)

Table 151. SIU_ISEL9 Register field descriptions

Field	Description
0–26	Reserved
27–31 eTSEL0A	00000 Reserved 00001 RTI Trigger 00010 PIT0 Trigger 00011 PIT1 Trigger 00100 PIT2 Trigger 00101 PIT3 Trigger 00110 Reserved 00111 Reserved 01000 eTPU30 AND PIT0 01001 eTPU30 AND PIT1 01010 Reserved 01011 Reserved 01100 eTPUA28 01101 eTPUA29 01110 eTPUA30 01111 eTPUA31 10000 Reserved 10001 Reserved 10010 Reserved 10011 Reserved 10100 eMIOS10 AND PIT2 10101 eMIOS10 AND PIT3 10110 Reserved 10111 Reserved 11000 Reserved 11001 Reserved 11010 Reserved 11011 Reserved 11100 Reserved 11101 Reserved 11110 Reserved 11111 eMIOS23

15.8.23 Chip Configuration Register (SIU_CCR)

SIU_BASE + 0x980

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MATCH	DISNEX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U(1)	U(2)

	16	17	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	CRSE	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. During reset the comparison is performed and result is uncertain.
2. The value after reset is uncertain.

Figure 241. Chip Configuration Register (SIU_CCR)

Table 152. SIU_CCR Register field descriptions

Field	Description
0–13	Reserved
14 MATCH	Compare Register Match The MATCH bit is a read only bit that indicates the result of a 64-bit comparison between the values in the SIU_CBRH and SIU_CBRL registers and the censorship word stored in the shadow block of flash. The match input is asserted if the values are equal. 1 Match input signal is asserted 0 Match input signal is negated
15 DISNEX	Disable Nexus The DISNEX bit is a read only bit that holds the value of the Nexus disable input signal to the SIU. When system reset negates, the value in this bit depends on the censorship control word and the boot configuration bits. 1 Nexus disable input signal is asserted 0 Nexus disable input signal is negated
16–29	Reserved

Table 152. SIU_CCR Register field descriptions (continued)

Field	Description
30 CRSE	Calibration Reflection Suppression Enable The CRSE bit enables the suppression of reflections from the EBI's calibration bus onto the non-calibration bus. The EBI drives some outputs to both the calibration and non-calibration buses. When CRSE is asserted, the values driven onto the calibration bus pins will not be reflected onto the non-calibration bus pins. When CRSE is negated, the values driven onto the calibration bus pins will be reflected onto the non-calibration bus pins. CRSE only enables reflection suppression for non-calibration bus pins which do not have a negated state to which the pins return at the end of the access. CRSE does not enable reflection suppression for the non-calibration bus pins which have a negated state to which the pins return at the end of an access. Those reflections always are suppressed. Furthermore, the suppression of reflections from the non-calibration bus onto the calibration bus is not enabled by CRSE. Those reflections are also always suppressed. 1 Calibration reflection suppression is enabled 0 Calibration reflection suppression is disabled
31	Reserved

15.8.24 External Clock Control Register (SIU_ECCR)

The SIU_ECCR controls the timing relationship between the system clock and the external clock CLKOUT. All bits and fields in the SIU_ECCR are read/write and are reset by the IP Green-Line asynchronous reset signal.

SIU_BASE + 0x984

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	1	0	0	0	0	0	0	0	0	EBTS	0	EBDF[0–1]	
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

= Unimplemented or Reserved

Figure 242. External Clock Control Register (SIU_ECCR)

Table 153. SIU_ECCR Register field descriptions

Field	Description
0–27	Reserved
28 EBTS	External Bus Tap Select The EBTS bit changes the phase relationship between the system clock and CLKOUT. Changing the phase relationship so that CLKOUT is advanced in relation to system clock increases the output hold time of the external bus signals to a non-zero value. It also increases the output delay times, increases the input hold times to non-zero values, and decreases the input setup times. Refer to the device data sheet for how the EBTS bit affects the electrical specifications. 1 External bus signals have non-zero output hold times. 0 External bus signals have zero output hold times. **Note:** The EBTS bit must not be modified while an external bus transaction is in progress. The field [16–24] is reserved and should not be used.
29	Reserved
30–31 EBDF[0–1]	External Bus Division Factor The EBDF field specifies the frequency ratio between the system clock and CLKOUT. The EBDF field must not be changed during an external bus access or while an access is pending. The CLKOUT frequency is divided from the system clock frequency as follows: 00 External Bus Division Factor = 1 01 External Bus Division Factor = 2 10 Reserved 11 External Bus Division Factor = 4 **Note:** The reset value of the EBDF field is divide-by-2. After reset, if EBDF is changed to divided-by-1, no glitches occurs on the CLKOUT signal, but if EBDF is changed back to divide-by-2 or divide-by-4, there is no guarantee that the switch will be glitchless.

15.8.25 Compare A High Register (SIU_CARH)

Note: This register is currently not used.

SIU_BASE + 0x988

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

= Unimplemented or Reserved

Figure 243. Compare A High Register (SIU_CARH)

15.8.26 Compare A Low Register (SIU_CARL)

Note: This register is currently not used.

SIU_BASE + 0x98C

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

= Unimplemented or Reserved

Figure 244. Compare A Low Register (SIU_CARL)

15.8.27 Compare B High Register (SIU_CBRH)

The SIU_CBRH register is used in conjunction with the SIU_CBRL register as part of the JTAG censorship mechanism. During bootup, the concatenated contents of the SIU_CBRH and SIU_CBRL registers are compared to the censorship password value stored in the flash shadow block. The results of the match are indicated by the SIU_CCR[MATCH] bit.

SIU_BASE + 0x990

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

= Unimplemented or Reserved

Figure 245. Compare B High Register (SIU_CBRH)

15.8.28 Compare B Low Register (SIU_CBRL)

The SIU_CBRL register is used in conjunction with the SIU_CBRH register as part of the JTAG censorship mechanism. During bootup, the concatenated contents of the SIU_CBRH and SIU_CBRL registers are compared to the censorship password value stored in the flash shadow block. The results of the match are indicated by the SIU_CCR[MATCH] bit.

SIU_BASE + 0x994

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

= Unimplemented or Reserved

Figure 246. Compare B Low Register (SIU_CBRL)

15.8.29 System Clock Register (SIU_SYSDIV)

SIU_BASE + 0x9A0

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																
W												BYPASS	SYSCLKDIV[0–1]			
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

= Unimplemented or Reserved

Figure 247. System Clock Register (SIU_SYSDIV)

Table 154. SIU_SYSDIV Register field descriptions

Field	Description
0–26	Reserved
27 BYPASS	Bypass bit 0 System clock divider is not bypassed 1 System clock divider is bypassed

Table 154. SIU_SYSDIV Register field descriptions (continued)

Field	Description
28–29 SYSCLKDIV[0–1]	System Clock Divide The SYSCLKDIV bits select the divider value for the system clock, that is, the output of the FMPLL. Note that the SYSCLKDIV divider is required in addition to the RFD to allow the other sources for the system clock, i.e., output of the FMPLL, to be divided down to slowest frequencies to improve power. The output of the clock divider is nominally a 50% duty cycle. 00 Divide by 2 01 Divide by 4 10 Divide by 8 11 Divide by 16
30–31	Reserved

15.8.30 Halt Register (SIU_HLT)

The SIU_HLT register is used to put various modules into stop mode to save power. Each bit drives a separate stop request signal to a different module. When the module acknowledges the stop request, the clock to that module is halted. To remove the module from stop mode, the corresponding bit in the SIU_HLT register must be cleared. In the case of the CPU, stop mode in entered when the corresponding bit in SIU_HLT is set and a WAIT instruction is executed. The CPU exits stop mode upon reception of any interrupt request.

SIU_BASE + 0x9A4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CPUSTP	0	0	0	NDEDISTP	TPUSTP	NPCSTP	EBISTP	ADCSTP	0	MIOSSTP	DFILSTP	0	PITSTP	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	CNCSTP	0	CNASTP	0	SPICSTP	SPIBSTP	0	0	0	0	0	0	0	SCIBSTP	SCIASTP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 248. Halt Register (SIU_HLT)

Table 155. SIU_HLT field descriptions

Field	Description
0 CPUSTP	CPU stop request When asserted, a stop request is sent to the following modules: CPU, crossbar, peripheral bridge, system RAM, Flash, STM, DMA. 1: Stop mode request 0: Normal operation
1–2	Reserved

Table 155. SIU_HLT field descriptions (continued)

Field	Description
3 SWTSTP	SWT stop request When asserted, a stop request is sent to the Software Watchdog Timer. 1: Stop mode request 0: Normal operation
4	Reserved
5 TPUSTP	eTPU stop request When asserted, a stop request is sent to the eTPU module. 1: Stop mode request 0: Normal operation
6 NPCSTP	Nexus stop request When asserted, a stop request is sent to the Nexus Controller. 1: Stop mode request 0: Normal operation
7 EBISTP	EBI stop request When asserted, a stop request is sent to the external bus controller which handles the calibration interface. 1: Stop mode request 0: Normal operation
8 ADCSTP	eQADC stop request When asserted, a stop request is sent to the eQADC module. 1: Stop mode request 0: Normal operation
9	Reserved
10 MIOSSTP	eMIOS stop request When asserted, a stop request is sent to the eMIOS module. 1: Stop mode request 0: Normal operation
11 DFILSTP	Decimation filter stop request When asserted, a stop request is sent to the decimation filter module. 1: Stop mode request 0: Normal operation
12	Reserved
13 PITSTP	PIT stop request When asserted, a stop request is sent to the periodic interrupt timer module. 1: Stop mode request 0: Normal operation
14–16	Reserved
17 CNCSTP	FlexCAN_C stop request When asserted, a stop request is sent to the FlexCAN module C. 1: Stop mode request 0: Normal operation

Table 155. SIU_HLT field descriptions (continued)

Field	Description
18	Reserved
19 CNASTP	FlexCAN_A stop request When asserted, a stop request is sent to the FlexCAN module A. 1: Stop mode request 0: Normal operation
20	Reserved
21 SPICSTP	DSPI C stop request When asserted, a stop request is sent to the DSPI C. 1: Stop mode request 0: Normal operation
22 SPIBSTP	DSPI B stop request When asserted, a stop request is sent to the DSPI B. 1: Stop mode request 0: Normal operation
23–29	Reserved
30 SCIBSTP	eSCI B stop request When asserted, a stop request is sent to the eSCI module B. 1: Stop mode request 0: Normal operation
31 SCIASTP	eSCI A stop request When asserted, a stop request is sent to the eSCI module A. 1: Stop mode request 0: Normal operation

15.8.31 Halt Acknowledge Register (SIU_HLTACK)

The bits in the SIU_HLTACK register indicate that the module requested to halt via the SIU_HLT register has completed the halt process and has entered a halted state with the module clocks disabled. This register is read-only.

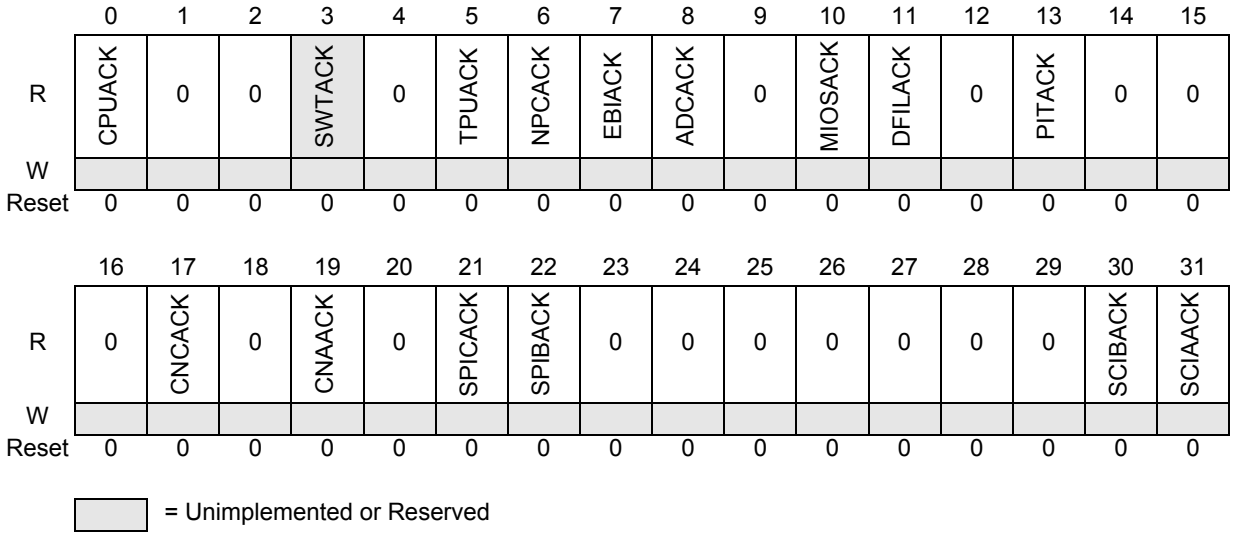


Figure 249. Halt Acknowledge Register (SIU_HLTACK)

Table 156. SIU_HLTACK field descriptions

Field	Description
0 CPUACK	CPU stop acknowledge When asserted, indicates that a stop acknowledge was received from the following modules: CPU, crossbar, peripheral bridge, system RAM, Flash, STM, DMA. 1: Stop mode request 0: Normal operation
1–2	Reserved
3 SWTACK	SWT stop acknowledge When asserted, indicates that a stop acknowledge was received from the Software Watchdog Timer 1: Stop mode request 0: Normal operation
4	Reserved
5 TPUACK	eTPU stop acknowledge When asserted, indicates that a stop acknowledge was received from the eTPU module. 1: Stop mode request 0: Normal operation
6 NPCACK	Nexus stop acknowledge When asserted, indicates that a stop acknowledge was received from the Nexus Controller. 1: Stop mode request 0: Normal operation
7 EBIACK	EBI stop acknowledge When asserted, indicates that a stop acknowledge was received from the external bus controller which handles the calibration interface. 1: Stop mode request 0: Normal operation

Table 156. SIU_HLTACK field descriptions (continued)

Field	Description
8 ADCACK	eQADC stop acknowledge When asserted, indicates that a stop acknowledge was received from the eQADC module. 1: Stop mode request 0: Normal operation
9	Reserved
10 MIOSACK	eMIOS stop acknowledge When asserted, indicates that a stop acknowledge was received from the eMIOS module. 1: Stop mode request 0: Normal operation
11 DFILACK	Decimation filter stop acknowledge When asserted, indicates that a stop acknowledge was received from the decimation filter module. 1: Stop mode request 0: Normal operation
12	Reserved
13 PITACK	PIT stop acknowledge When asserted, indicates that a stop acknowledge was received from the periodic interrupt timer module. 1: Stop mode request 0: Normal operation
14–16	Reserved
17 CNCACK	FlexCAN_C stop acknowledge When asserted, indicates that a stop acknowledge was received from the FlexCAN module C. 1: Stop mode request 0: Normal operation
18	Reserved
19 CNAACK	FlexCAN_A stop acknowledge When asserted, indicates that a stop acknowledge was received from the FlexCAN module A. 1: Stop mode request 0: Normal operation
20	Reserved
21 SPICACK	DSPI C stop acknowledge When asserted, indicates that a stop acknowledge was received from the DSPI C. 1: Stop mode request 0: Normal operation
22 SPIBACK	DSPI B stop acknowledge When asserted, indicates that a stop acknowledge was received from the DSPI B. 1: Stop mode request 0: Normal operation
23–29	Reserved

Table 156. SIU_HLTACK field descriptions (continued)

Field	Description
30 SCIBACK	eSCI B stop acknowledge When asserted, indicates that a stop acknowledge was received from the eSCI module B. 1: Stop mode request 0: Normal operation
031 SCIAACK	eSCI A stop acknowledge When asserted, indicates that a stop acknowledge was received from the eSCI module A. 1: Stop mode request 0: Normal operation

15.9 Functional description

The following sections provide an overview of the SIU operation features.

15.9.1 System configuration

15.9.1.1 Boot configuration

The BAM program uses the BOOTCFG bit of the Reset Status Register (RSR) to determine whether an internal or an external (serial) boot will happen. For additional details on the BAM program operation see *Section 20.5.2: BAM program operation*.

15.9.1.2 Pad configuration

The Pad Configuration Registers (PCR) in the SIU allow software control of the static electrical characteristics of external pins. The multiplexed function of a pin, selection of pull up or pull down devices, the slew rate of I/O signals, open drain mode for output pins, hysteresis on input pins, and the drive strength for bus signals can be specified through the PCRs.

15.9.2 Reset control

The reset controller logic is located in the SIU. See *Chapter 4: Resets* for details on reset operation.

15.9.3 External interrupt

There are sixteen external interrupt inputs IRQ0–IRQ15 to the SIU, but only eleven are mapped on this device. The IRQ*x* inputs can be configured for rising or falling edge events or both. Each IRQ*x* input has a corresponding flag bit in SIU_ESR (see *Section 15.8.5: External Interrupt Status Register (SIU_EISR)*.) The flag bits for the IRQ[4:15] inputs are ORed together to form one interrupt request to the interrupt controller (OR function performed in the integration glue logic). The flag bits for the IRQ[0] and IRQ[3] inputs can generate either an interrupt request to the interrupt controller or a DMA transfer request to the DMA controller. *Figure 250* shows the DMA and interrupt request connections to the interrupt and DMA controllers. The non used interrupts sources are tied to ‘0’.

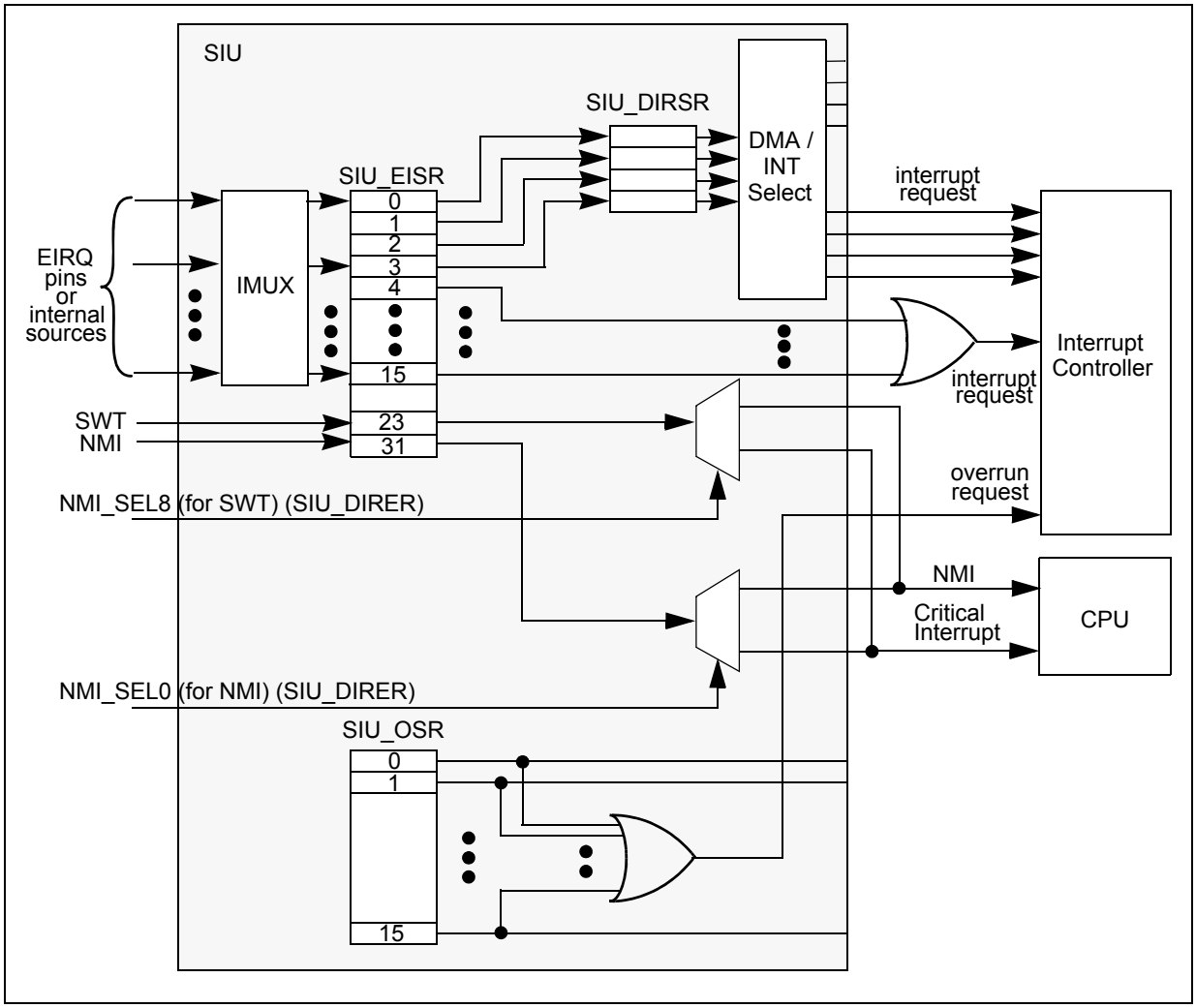
The SIU contains an overrun request for each IRQ and one combined overrun request which is the logical OR of the individual overrun requests. Only the combined overrun request is used in this device, and the individual overrun requests are not connected.

Each IRQ pin has a programmable filter for rejecting glitches on the IRQ signals. The filter length for the IRQ pins is specified in the External IRQ Digital Filter Register (SIU_IDFR).

The NMI and SWT interrupts are ORed and the write once signal NMI_SEL selects which platform input, NMI input or Critical Interrupt input, are driven by this logic.

For this device the SIU outputs to eDMA are not connected. If the SIU_DIRSR selects the eDMA output, the effect is to disable the interrupts.

Figure 250. SIU DMA/Interrupt request diagram



Note: If either the SWT or the NMI inputs to the core in the Interrupt Rising Edge Event Enable register {IREE[8] and IREE[0]} or Interrupt Falling edge Event Enable register {IFEE[8] and IFEE[0]}, then the NMI pin (GPIO 213) is disabled as an output.

15.9.4 GPIO operation

All GPIO functionality is provided by the SIU for this device. Each device pin that has GPIO functionality has an associated Pin Configuration Register in the SIU where the GPIO function is selected for the pin. In addition, each device pin with GPIO functionality has an input data register (SIU_GPDI*x*_*x*) and an output data register (SIU_GPDO*x*_*x*).

15.9.5 Internal multiplexing

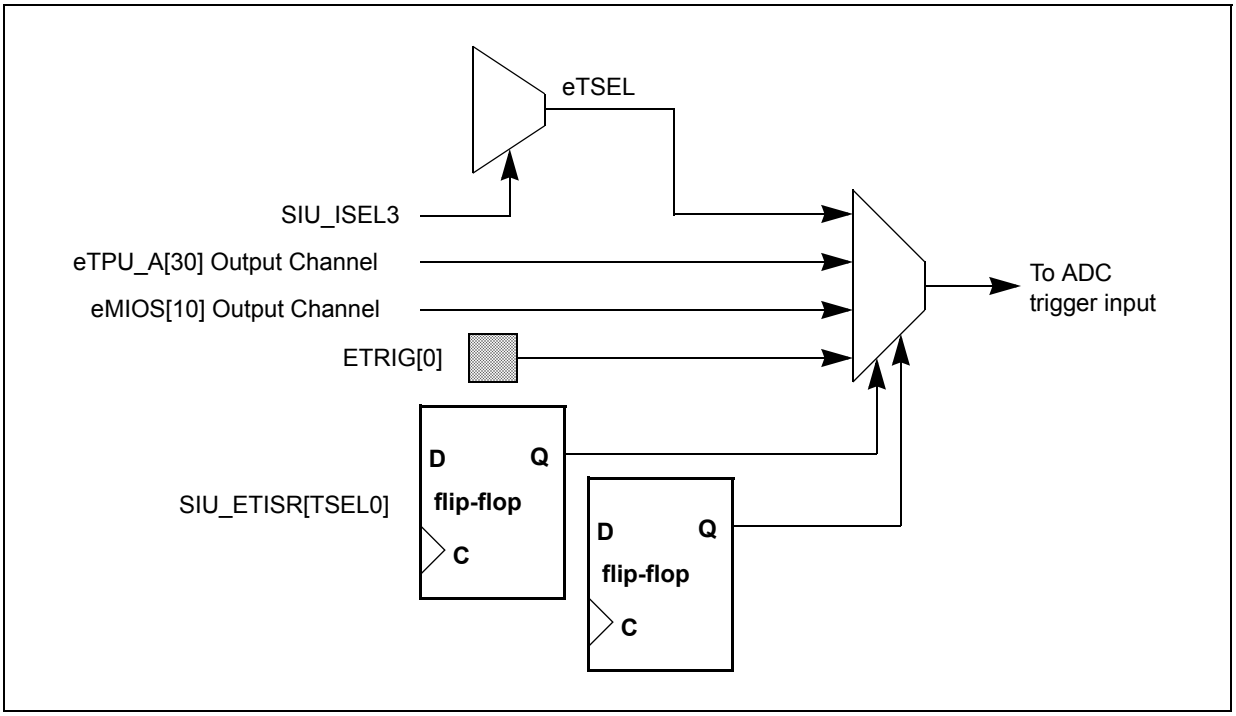
The following registers are used to provide selection of the eQADC external trigger inputs sources, the SIU external interrupts, and the DSPI signals that are used in the serial and parallel chaining of DSPI blocks:

- eQADC Trigger Input Select Register (SIU_ETISR)
- External IRQ Input Select Register (SIU_EIISR)
- DSPI Input Select Register (SIU_DISR)
- IMUX Select Register 3 (SIU_ISEL3)

15.9.5.1 eQADC external trigger input multiplexing

The six eQADC external trigger inputs can be connected to either an external pin, an eTPU channel, or an eMIOS channel. The input source for each eQADC external trigger is individually specified in eQADC Trigger Input Select Register (SIU_ETISR). An example of the multiplexing of an eQADC external trigger input is given in *Figure 251*. As shown in the figure, the ETRIG[0] input of the eQADC can be connected to either the eTPU_A[30] channel or the eMIOS[10] channel. The remaining ETRIG inputs are multiplexed in the same manner. Note that if an ETRIG input is connected to an eTPU or eMIOS channel, the external pin used by that channel can be used by the alternate function on that pin.

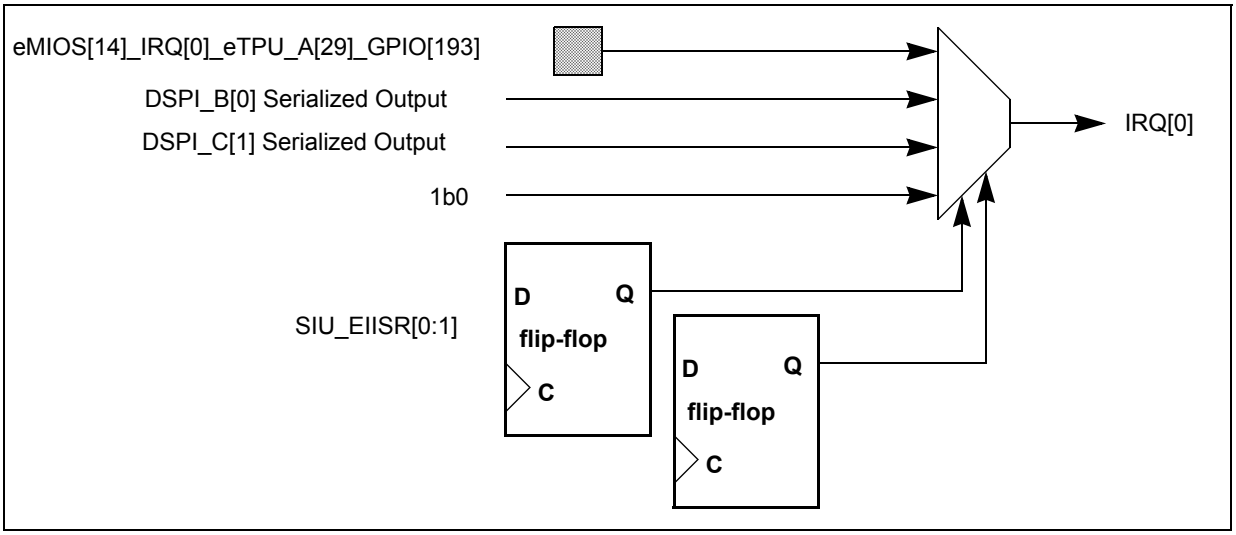
Figure 251. eQADC external trigger input multiplexing



15.9.5.2 SIU external interrupt input multiplexing

The sixteen SIU external interrupt inputs can be connected to either external pins or to deserialized output signals from a DSPI block. The input source for each SIU external interrupt is individually specified in the External IRQ Input Select Register (SIU_EIISR). An example of the multiplexing of an SIU external interrupt input is given in *Figure 252*. As shown in the figure, the IRQ[0] input of the SIU can be connected to either the eMIOS[14]_IRQ[0]_eTPU_A[29]_GPIO[193] pin, the DSPI_B[0] deserialized output signal, or the DSPI_C[1] deserialized output signal. The remaining IRQ inputs are multiplexed in the same manner. The inputs to the IRQ from each DSPI block are offset by one so that if more than one DSPI block is connected to the same external device type, a separate interrupt can be generated for each device. This also applies to DSPI blocks connected to external devices of different type that have status bits in the same bit location of the deserialized information.

Figure 252. SIU external interrupt input multiplexing



15.9.5.3 Multiplexed inputs for DSPI multiple transfer operation

To support multiple DSPIs transfer operations, an input multiplexor is required for the SIN, SS, SCK IN, and Trigger signals of each DSPI. These DSPIs input sources can be a pin or respectively the SOUT, PCS[0], SCK OUT, PCSS of any other DSPI. They are individually specified in the DSPI Input Select Register (SIU_DISR).

See *Section 25.4.3.6: Multiple Transfer Operation (MTO)* for more information on Multiple Transfer Operation.

16 Frequency-modulated phase-locked loop (FMPLL)

16.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

16.1.1 Device-specific features

- Allowed frequency range for the crystal oscillator: 4 MHz to 20 MHz
- The ESYNCR1[EMODE] bit is not tied off, meaning that both legacy and enhanced mode are supported. Its reset value is zero, meaning that the default state is “legacy”.
- Base address for FMPLL is 0xC3F8_0000

16.1.2 Device-specific register field reset values

Table 157 shows the reset values for several register fields on this device.

Table 157. Register field reset values

Field	Reset value	Description
FMPLL_SYNCR[PREDIV]	0b111	Inhibits the clock to the phase detector
FMPLL_SYNCR[MFD]	0b00100	Divide-by-8
FMPLL_SYNCR[RFD]	0b010	Divide-by-4
FMPLL_ESYNCR1[EMODE]	0b0	Allows legacy mode to be used
FMPLL_ESYNCR1[EPREDIV]	0b1111	Inhibits the clock to the phase detector
FMPLL_ESYNCR1[EMFD]	0b0100000	Divide-by-32
FMPLL_ESYNCR2[ERFD]	0b11	Divide-by-16

16.2 Introduction

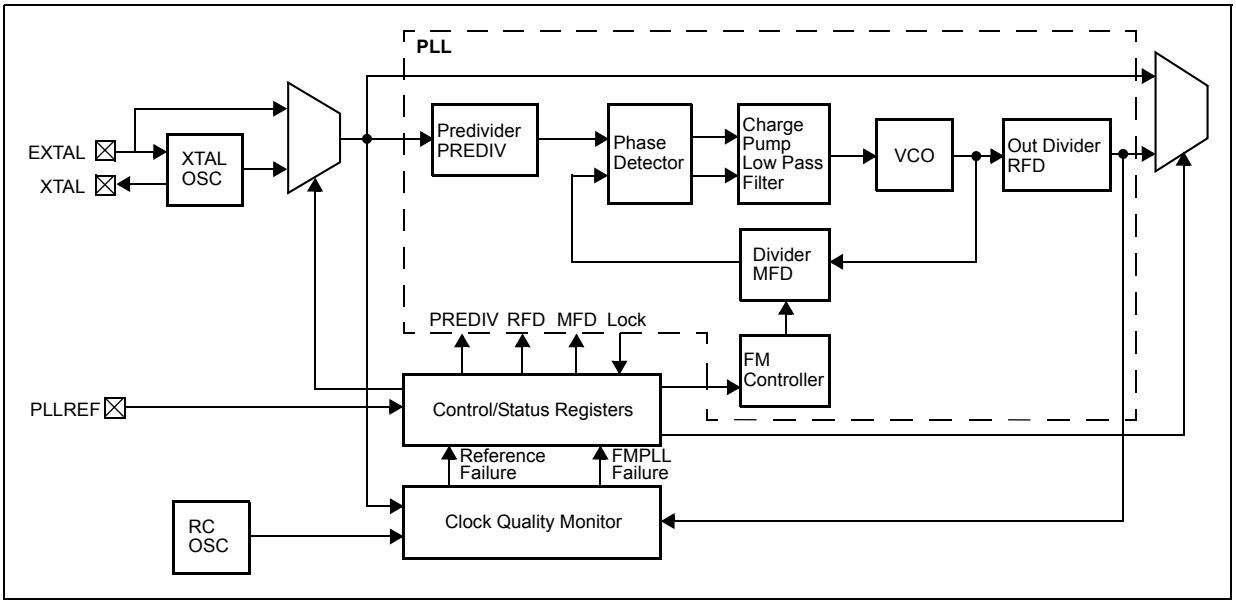
This chapter describes the features and functions of the FMPLL module.

16.2.1 Overview

The frequency-modulated phase-locked loop (FMPLL) allows the user to generate high speed system clocks from a crystal oscillator or from an external clock generator. Furthermore, the FMPLL supports programmable frequency modulation of the system clock. The FMPLL multiplication factor, reference clock predivider factor, output clock divider ratio, modulation depth and multiplication rate are all controllable through programmable registers.

Figure 253 shows the block diagram of the FMPLL.

Figure 253. FMPLL block diagram



16.2.2 Features

The FMPLL has the following features:

- Reference clock predivider for finer frequency synthesis resolution
- Reduced frequency divider for reducing the FMPLL output clock frequency without forcing the FMPLL to relock
- Input clock frequency range from 4 MHz to 20 or 40 MHz(i) before the predivider, and from 4 MHz to 16 MHz after the predivider
- Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz
- VCO free-running frequency range from 25 MHz to 125 MHz
- 4 bypass modes: crystal or external reference with PLL on or off
- 2 normal modes: crystal or external reference
- Programmable frequency modulation
  - Triangle wave modulation
  - Register programmable modulation frequency and depth
- Lock detect circuitry reports when the FMPLL has achieved frequency lock and continuously monitors lock status to report loss of lock conditions
  - User-selectable ability to generate an interrupt request upon loss of lock
  - User-selectable ability to generate a system reset upon loss of lock
- Clock quality monitor (CQM) module provides loss-of-clock detection for the FMPLL reference and output clocks
  - User-selectable ability to generate an interrupt request upon loss of clock
  - User-selectable ability to generate a system reset upon loss of clock
  - Backup clock (reference clock or FMPLL free-running) can be applied to the system in case of loss of clock

16.2.3 Modes of operation

Upon reset, the operational mode is bypass with PLL running, and the source of the reference clock, either the crystal oscillator or external clock, is determined by the reset value of the CLKCFG[2] bit of the FMPLL_ESYNCR1. The reset state of this bit comes from an external signal to the module connected to a package pin called PLLREF. After reset, a different operational mode can be selected by writing to FMPLL_ESYNCR1[CLKCFG]. The available modes are specified in *Table 158*.

Table 158. Clock mode selection

CLKCFG[0]	CLKCFG[1](1)	CLKCFG[2](2)	Clock mode
0	0	0	Bypass mode with external reference and PLL off
0	0	1	Bypass mode with crystal reference and PLL off
0	1	0	Bypass mode with external reference and PLL running
0	1	1	Bypass mode with crystal reference and PLL running
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Normal mode with external reference
1	1	1	Normal mode with crystal reference

1. CLKCFG[1] is not writable to zero while CLKCFG[0]=1.
2. The reset state of this bit is determined by the logical state applied to the PLLREF pin.

At reset the FMPLL is enabled, but the reset value of the predivider may be set by the SoC integration to inhibit the clock to the PLL, making the VCO run within its free-running frequency range of 25 MHz to 125 MHz, unconnected from the system clock (since bypass is the default mode at reset). If using crystal reference, after power-on reset the Clock Quality Monitor (CQM) will inhibit the system clock and keep system reset asserted while the crystal oscillator has not stabilized. The PLLREF input must be kept stable during the whole period while system reset is asserted.

16.2.3.1 Bypass mode with crystal reference

In the bypass mode with crystal reference, the FMPLL is completely bypassed and the system clock is driven from the crystal oscillator. The user must supply a crystal that is within the appropriate frequency range, the crystal manufacturer recommended external support circuitry, and short signal route from the MCU to the crystal.

In bypass mode the PLL itself may or may not be running, depending on the state of the CLKCFG[1] bit of FMPLL_ESYNCR1, but the PLL output is not connected to the system clock. Consequently, frequency modulation is not available. The predivider is also bypassed.

Bypass mode with crystal reference is the default mode at reset if the PLLREF input is driven high. After reset, this mode can be entered by programming FMPLL_ESYNCR1[CLKCFG] as shown in *Table 158*.

i. See *Section 16.1: Information specific to this device* for information on crystal frequencies supported.

16.2.3.2 Bypass mode with external reference

The bypass mode with external reference functions the same as bypass mode with crystal reference, except that the system clock is driven by an external clock generator connected to the EXTAL pin, rather than a crystal oscillator. The input frequency range is the same and frequency modulation is not available.

Bypass mode with external reference is the default mode at reset if the PLLREF input is driven low. After reset, this mode can be entered by programming FMPLL_ESYNCR1[PLLCFG] as shown in *Table 158*.

16.2.3.3 Normal mode with crystal reference

In the normal mode with crystal reference, the FMPLL receives an input clock frequency from the crystal oscillator and the predivider, and multiplies the frequency to create the FMPLL output clock. The user must supply a crystal that is within the appropriate frequency range, the crystal manufacturer recommended external support circuitry, and short signal route from the MCU to the crystal.

In normal mode with crystal reference, the FMPLL can generate a frequency-modulated clock or a non-modulated clock (locked on a single frequency). The modulation rate, modulation depth, output divider (RFD) and whether the FMPLL is modulating or not can be programmed by writing to the FMPLL registers.

16.2.3.4 Normal mode with external reference

The normal mode with external reference functions the same as normal mode with crystal reference, except that the input clock reference to the FMPLL is driven by an external clock generator connected to the EXTAL pin, rather than a crystal oscillator. The input frequency range is the same and frequency modulation is available.

16.3 External signal description

Table 159 lists external signals used by the FMPLL during normal operation.

Table 159. Signal properties

Name	Function	I/O	Pull
PLLREF	Configures the FMPLL clock reference at reset	I/O	Up
XTAL	Output drive for external crystal	O	—
EXTAL_EXTCLK	Crystal/external clock input	I/O	—
VDDPLL	Analog power supply (1.2V +/– 10%)	Power	—
VSSPLL	Analog ground	Ground	—

16.3.1 Detailed signal descriptions

Table 160 describes the external signals used by the FMPLL.

Table 160. IFMPLL detailed signal descriptions

Signal	I/O	Description	
PLLREF	I/O	PLL reference—Determines the reset state of the CLKCFG[2] bit in FMPLL_ESYNCR1. The PLLREF pin must be kept stable during system reset. After reset, this pin has no effect on the PLL configuration, therefore it can be assigned to another function such as GPIO.	
		State Meaning	Asserted—Indicates that the reference clock comes from the crystal oscillator. Negated—Indicates that the reference clock comes from the external clock generator.
		Timing	Assertion or negation—must be done at the beginning of the reset cycle and then kept stable for the whole reset duration.
XTAL	O	Crystal oscillator—Output for an external crystal oscillator.	
EXTAL_EXTCLK	I/O	Crystal oscillator/external clock input—This pin is the input for an external crystal oscillator or an external clock source. The function of this pin is determined by the CLKCFG[2] bit in FMPLL_ESYNCR1, whose reset value is determined by the PLLREF pin.	
VDDPLL / VSSPLL	—	PLL power supply—These are the 1.2V supply and ground for the FMPLL.	

16.4 Memory map and register definition

This section provides the memory map and detailed descriptions of all registers of the FMPLL.

16.4.1 Memory map

Table 161 shows the memory map. Addresses are given as offsets of the module base address.

Table 161. FMPLL memory map

Offset	Register	Access	Reset value(1)	Location
0x0000	Synthesizer Control Register (SYNCR)	R/W	0x****_0000	*on page 489*
0x0004	Synthesizer Status Register (SYNSR)	Special	0x0000_00*0	*on page 491*
0x0008	Enhanced Synthesizer Control Register 1 (ESYNCR1)	R/W	0x*00*_00**	*on page 494*
0x000C	Enhanced Synthesizer Control Register 2 (ESYNCR2)	R/W	0x0000_000*	*on page 494*
0x0010	Reserved	—	—	—
0x014	Reserved	—	—	—
0x018	Synthesizer FM Modulation Register (SYNFMMR)	R/W	0x0000_0000	*on page 497*

1. The symbol * means that the reset value is defined at SoC integration or by an external pin.

16.4.2 Register descriptions

This section contains the register descriptions in ascending address order. Two different programming models are selectable through FMPLL_ESYNCR1[EMODE]:

1. Legacy model—The FMPLL is controlled by the Synthesizer Control Register (SYNCR). In this model, the FMPLL operating mode changes automatically to normal mode when the register is written in the first time. There is no way to switch back to bypass mode once the operating mode has switched to normal.
2. Enhanced model—The PLL is controlled by the Enhanced Synthesizer Control Registers 1–2 (ESYNCR1/ESYNCR2). In this model, it is possible to change the FMPLL operating mode back and forth between bypass and normal modes by programming FMPLL_ESYNCR1[CLKCFG].

The reset value of FMPLL_ESYNCR1[EMODE] is determined by the SoC integration. This bit is write once. After it is set to '1', further write attempts to this bit will have no effect.

16.4.2.1 Synthesizer Control Register (SYNCR)

This register is provided for backwards compatibility with previous devices . New applications should use ESYNCR1/ESYNCR2 instead of SYNCR.

Offset 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	PREDIV			MFD					0	RFD			LOC EN	LOL RE	LOC RE
W																
Reset	0	–[1]	–[1]	–[1]	–[1]	–[1]	–[1]	–[1]	–[1]	0	–[1]	–[1]	–[1]	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	LOL IRQ	LOC IRQ	0	0	0	0	0	0	0	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 Reset value is determined by the SoC integration.

Figure 254. Synthesizer Control Register (SYNCR)

Table 162. SYNCR field descriptions

Field	Description
0	Reserved, should be cleared.
1–3 PREDIV	Predivider This 3-bit field controls the value of the divider on the input clock. The output of the predivider circuit generates the reference clock to the FMPLL analog loop. The value 111 causes the input clock to be inhibited. 000 Divide by 1 001 Divide by 2 010 Divide by 3 011 Divide by 4 100 Divide by 5 101 Divide by 6 110 Divide by 7 111 Clock inhibit

Table 162. SYNCR field descriptions (continued)

Field	Description
4–8 MFD	Multiplication factor divider This 5-bit field controls the value of the divider in the FMPLL feedback loop. The division factor programmed here is an artificial value intended to mimic legacy behavior. The actual feedback divider programmed into the PLL is given by 4*(MFD+4). The mapping between MFD and EMFD is: 00100->0100000, 00101->0100100, ..., 10100->1100000. 000xx Invalid 00100 Divide by 8 00101 Divide by 9 00110 Divide by 10 ... 10011 Divide by 23 10100 Divide by 24 10101 Invalid 1011x Invalid 11xxx Invalid
9	Reserved, should be cleared.
10–12 RFD	Reduced frequency divider This 3-bit field controls the value of the divider at the output of the FMPLL. The division factor programmed here is an artificial value intended to mimic legacy behavior. The actual divider programmed into the PLL is given by (2**RFD)*4. The mapping between RFD and ERFD is: 000->01, 001->10, 010->11. 000 Divide by 1 001 Divide by 2 010 Divide by 4 011 Invalid 1xx Invalid
13 LOCEN	Loss-of-clock enable The LOCEN bit determines if the loss-of-clock function is operational. This bit only has effect in normal mode. In bypass mode, the loss-of-clock function is always enabled, regardless of the state of the LOCEN bit. Furthermore, the LOCEN bit has no effect on the loss-of-lock detection circuitry. 0 Loss of clock disabled 1 Loss of clock enabled
14 LOLRE	Loss-of-lock reset enable The LOLRE bit determines whether system reset is asserted or not upon a loss-of-lock indication. When operating in normal mode, the FMPLL must be locked before setting the LOLRE bit, otherwise reset is immediately asserted. Note that once reset is asserted, the operating mode is switched to bypass mode, and once in bypass, a loss-of-lock condition does not generate reset, regardless of the value of the LOLRE bit. See *Section 16.5.3: Lock detection*. 0 Ignore loss-of-lock. Reset not asserted. 1 Assert reset on loss-of-lock when operating in normal mode.
15 LOCRE	Loss-of-clock reset enable The LOCRE bit determines whether system reset is asserted or not upon a loss-of-clock condition when LOCEN=1. LOCRE has no effect when LOCEN=0. If the LOCF bit in the SYNSR indicates a loss-of-clock condition, setting the LOCRE bit causes immediate reset. In bypass mode with crystal reference, reset will occur if the reference clock fails, even if LOCRE=0 or even if LOCEN=0. The LOCRE bit has no effect in bypass mode with external reference. In this mode, the reference clock is not monitored at all. See *Section 16.5.4.2: Loss-of-clock reset*. 0 Ignore loss-of-clock. Reset not asserted. 1 Assert reset on loss-of-clock.

Table 162. SYNCR field descriptions (continued)

Field	Description
16	Reserved, should be cleared.
17 LOLIRQ	Loss-of-lock interrupt request The LOLIRQ bit enables a loss-of-lock interrupt request when the LOLF flag is set. If either LOLF or LOLIRQ is negated, the interrupt request is negated. When operating in normal mode, the FMPLL must be locked before setting the LOLIRQ bit, otherwise an interrupt is immediately asserted. The interrupt request only happens in normal mode, therefore the LOLIRQ bit has no effect in bypass mode. See *Section 16.5.3: Lock detection*. 0 Ignore loss-of-lock. Interrupt not requested. 1 Enable interrupt request upon loss-of-lock.
18 LOCIRQ	Loss-of-clock interrupt request. The LOCIRQ bit enables a loss-of-clock interrupt request when the LOCF flag is set. If either LOCF or LOCIRQ is negated, the interrupt request is negated. If loss-of-clock is detected while in bypass mode, a system reset is generated. Therefore, LOCIRQ has no effect in bypass mode. See *Section 16.5.4.3: Loss-of-clock interrupt request*. 0 Ignore loss-of-clock. Interrupt not requested. 1 Enable interrupt request upon loss-of-clock.
19–31	Reserved, should be cleared.

16.4.2.2 Synthesizer Status Register (SYNSR)

Offset 0x0004 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	LOLF	LOC	MODE	PLLSEL	PLLREF	LOCKS	LOCK	LOCF	0	0
W							w1c							w1c		
Reset	0	0	0	0	0	0	0	0	0	0	–[1]	0	0	0	0	0

[1] Reset value is determined by the state of the PLLREF pin.

Figure 255. Synthesizer Status Register (SYNSR)

Table 163. SYNSR field descriptions

Field	Description
0–21	Reserved, should be cleared.
22 LOLF	Loss-of-lock flag This bit provides the interrupt request flag for the loss-of-lock. To clear the flag, software must write a 1 to the bit. Writing 0 has no effect. This flag bit is sticky in the sense that if lock is reacquired, the bit will remain set until cleared by either writing 1 or asserting reset. It will not be asserted when lock is lost due to system reset, write to the SYNCR in legacy mode which modifies the PREDIV or MFD fields, or write to ESYNCR1 in enhanced mode which modifies the EMODE, EPREDIV, EMFD or CLKCFG[1:0] fields. Furthermore, it is not asserted if the loss-of-lock condition was detected while the FMPLL is in bypass mode. Nevertheless, going from normal to bypass will not automatically clear the flag if it was asserted while the FMPLL was in normal mode. See *Section 16.5.3: Lock detection*. 0 No loss of lock detected. Interrupt service not requested. 1 Loss of lock detected. Interrupt service requested.
23 LOC	Loss-of-clock This bit is an indication of whether a loss-of-clock condition is present. If LOC=0, the system clocks are operating normally. If LOC=1, the system clocks have failed due to a reference or VCO failure. If a loss-of-clock condition occurs which sets this bit and the clocks later return to normal, this bit will be cleared. A loss-of-clock condition can only be detected if LOCEN=1. Furthermore, the LOC bit is not asserted when the FMPLL is in bypass mode (because, in bypass, the VCO clock is not monitored and a loss-of-clock on the reference clock causes reset). See *Section 16.5.4: Loss-of-clock detection*. 0 No loss-of-clock detected. Clocks are operating normally. 1 Loss-of-clock detected. Clocks are not operating normally.
24 MODE	Mode of operation This bit indicates whether the FMPLL is working in bypass mode or normal mode. The reset value indicates bypass mode. In legacy mode (FMPLL_ESYNCR1[EMODE] negated), the MODE bit will change to normal mode at the first time the FMPLL_SYNCR register is written. In enhanced mode (FMPLL_ESYNCR1[EMODE] asserted), the MODE bit reflects the value of the CLKCFG[0] bit of the FMPLL_ESYNCR1. 0 Bypass mode 1 Normal mode
25 PLLSEL	Mode select Dual controller mode is not supported, therefore in legacy mode this bit resets to ‘0’ (bypass), but changes to ‘1’ (normal mode) at the first time the FMPLL_SYNCR is written. In enhanced mode, the MODE bit reflects the value of the CLKCFG[1] bit of the FMPLL_ESYNCR1. 0 Legacy mode: bypass or dual controller; enhanced mode: PLL off 1 Legacy mode: normal; enhanced mode: PLL on
26 PLLREF	FMPLL reference source This bit indicates whether the FMPLL reference is from a crystal oscillator or from an external clock generator. The reset value is determined by the state of the PLLREF pin. In legacy mode, the reset value captured from the PLLREF pin cannot be changed anymore after reset. In enhanced mode, the PLLREF bit reflects the value of the CLKCFG[2] bit of the FMPLL_ESYNCR1. 0 External clock reference 1 Crystal oscillator reference

Table 163. SYNSR field descriptions (continued)

Field	Description
27 LOCKS	Sticky FMPLL lock status bit This bit is set by the lock detect circuitry when the FMPLLL acquires lock after one of the following: – A system reset – A write to the FMPLL_SYNCR in legacy mode which changes the PREDIV or MFD fields – A write to the FMPLL_ESYNCR1 in enhanced mode which changes the EMODE, EPREDIV, EMFD or CLKCFG[1:2] fields Whenever the FMPLL loses lock, LOCKS is cleared. LOCKS remains cleared even after the FMPLL relocks, until one of the three previously stated conditions occurs. Coming in bypass mode from system reset, LOCKS is asserted as soon as the FMPLL has locked, even if normal mode was not entered yet. If the FMPLL is locked, going from normal to bypass mode does not clear the LOCKS bit. 0 FMPLL has lost lock since last system reset or last write to PLL registers which affect the lock status. 1 FMPLL has not lost lock.
28 LOCK	FMPLL lock status bit Indicates whether the FMPLL has acquired lock. FMPLL lock occurs when the synthesized frequency matches to within approximately 4% of the programmed frequency. The FMPLL loses lock when a frequency deviation of greater than approximately 16% occurs. The flag is also immediately negated when the PREDIV or MFD fields of the FMPLL_SYNCR are changed in legacy mode, or when EMODE, EPREDIV, EMFD or CLKCFG[1:2] are changed in enhanced mode, and then asserted again when the PLL regains lock. If operating in bypass mode, the LOCK bit is still asserted or negated when the FMPLL acquires or loses lock. 0 FMPLL is unlocked. 1 FMPLL is locked.
29 LOCF	Loss-of-clock flag This bit provides the interrupt request flag for the loss-of-clock. To clear the flag, software must write a 1 to the bit. Writing 0 has no effect. This flag bit is sticky in the sense that if clocks return to normal, the bit will remain set until cleared by either writing 1 or asserting reset. The LOCF flag is not asserted while the FMPLL is in bypass mode. See *Section 16.5.4: Loss-of-clock detection* for information on which operating modes and conditions can this flag be asserted. 0 No loss of clock detected. Interrupt service not requested. 1 Loss of clock detected. Interrupt service requested.
30–31	Reserved, should be cleared.

16.4.2.3 Enhanced Synthesizer Control Register 1 (ESYNCR1)

Offset 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMODE	CLKCFG			0	0	0	0	0	0	0	0	EPREDIV			
W																
Reset	–[1]	0	1	–[2]	0	0	0	0	0	0	0	0	–[1]	–[1]	–[1]	–[1]

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

1 Reset value is determined by the SoC integration.

2 Reset value determined by an input signal external to the module, typically connected to the PLLREF pin.

Figure 256. Enhanced Synthesizer Control Register 1 (ESYNCR1)

Table 164. ESYNCR1 field descriptions

Field	Description
0 EMODE	Enhanced mode enable This bit determines whether the FMPLL will be controlled by SYNCR or ESYNCR1/ESYNCR2. At SoC integration, a signal tie will dictate the default state that the PLL operates. If the SoC integration ties the FMPLL to run in enhanced mode, the EMODE bit will reflect this by reading a logic 1. Additionally, software writes to this bit to revert to legacy mode will not be allowed. If the signal is tied to select legacy mode as the default state, the EMODE bit will reflect this by reading a logic 0. In this case, software writes to this bit to enable enhanced mode is allowed, but it is a write once operation. After written to one, further write attempts to this bit will have no effect. 0 Legacy mode. FMPLL controlled by SYNCR. 1 Enhanced mode. FMPLL controlled by ESYNCR1/ESYNCR2.
1–3 CLKCFG	Clock configuration This 3-bit field is used to change the operating mode of the FMPLL. Bit 2 is not writable to ‘0’ while bit 1 is ‘1’. The reset state of bit 3 is determined by the state of the PLLREF pin. 000 Bypass mode with external reference and PLL off 001 Bypass mode with crystal reference and PLL off 010 Bypass mode with external reference and PLL running 011 Bypass mode with crystal reference and PLL running 100 Reserved 101 Reserved 110 Normal mode with external reference 111 Normal mode with crystal reference
4–11	Reserved, should be cleared.

Table 164. ESYNCR1 field descriptions (continued)

Field	Description
12–15 EPREDIV	Enhanced predivider This 4-bit field controls the value of the divider on the input clock. The output of the predivider circuit generates the reference clock to the PLL analog loop. The PREDIV value 1111 causes the input clock to be inhibited. 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
16–24	Reserved, should be cleared.
25–31 EMFD	Enhanced multiplication factor divider This 7-bit field controls the value of the divider in the FMPLL feedback loop. The value specified by the EMFD bits establishes the multiplication factor applied to the reference frequency. The valid range of multiplication factors is 32 (010_0000) to 96 (110_0000). Values outside this range are invalid and will cause the FMPLL to produce unpredictable clock output. 00x_xxxx Invalid 010_0000 Divide by 32 010_0001 Divide by 33 ... 101_1111 Divide by 95 110_0000 Divide by 96 110_0001 Invalid ... 111_1111 Invalid

16.4.2.4 Enhanced Synthesizer Control Register 2 (ESYNCR2)

Offset 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	LOC EN	LOL RE	LOC RE	LOL IRQ	LOC IRQ	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	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	ERFD	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	–[1]	–[1]

1 Reset value is determined by the SoC integration.

Figure 257. Enhanced Synthesizer Control Register 2 (ESYNCR2)

Table 165. ESYNCR2 field descriptions

Field	Description
0–7	Reserved, should be cleared.
8 LOCEN	Loss-of-clock enable The LOCEN bit determines if the loss-of-clock function is operational. This bit only has effect in normal mode. In bypass mode, the loss-of-clock function is always enabled, regardless of the state of the LOCEN bit. Furthermore, the LOCEN bit has no effect on the loss-of-lock detection circuitry. 0 Loss of clock disabled. 1 Loss of clock enabled.
9 LOLRE	Loss-of-lock reset enable The LOLRE bit determines whether system reset is asserted or not upon a loss-of-lock indication. When operating in normal mode, the FMPLL must be locked before setting the LOLRE bit, otherwise reset is immediately asserted. Note that once reset is asserted, the operating mode is switched to bypass mode, and once in bypass, a loss-of-lock condition does not generate reset, regardless of the value of the LOLRE bit. See *Section 16.5.3: Lock detection*. 0 Ignore loss-of-lock. Reset not asserted. 1 Assert reset on loss-of-lock when operating in normal mode.
10 LOCRE	Loss-of-clock reset enable The LOCRE bit determines whether system reset is asserted or not upon a loss-of-clock condition when LOCEN=1. LOCRE has no effect when LOCEN=0. If the LOCF bit in the SYNSR indicates a loss-of-clock condition, setting the LOCRE bit causes immediate reset. In bypass mode with crystal reference, reset will occur if the reference clock fails, even if LOCRE=0 or even if LOCEN=0. The LOCRE bit has no effect in bypass mode with external reference. In this mode, the reference clock is not monitored at all. See *Section 16.5.4.2: Loss-of-clock reset*. 0 Ignore loss-of-clock. Reset not asserted. 1 Assert reset on loss-of-clock.

Table 165. ESYNCR2 field descriptions (continued)

Field	Description
11 LOLIRQ	Loss-of-lock interrupt request The LOLIRQ bit enables a loss-of-lock interrupt request when the LOLF flag is set. If either LOLF or LOLIRQ is negated, the interrupt request is negated. When operating in normal mode, the FMPLL must be locked before setting the LOLIRQ bit, otherwise an interrupt is immediately asserted. The interrupt request only happens in normal mode, therefore the LOLIRQ bit has no effect in bypass mode. See *Section 16.5.3: Lock detection*. 0 Ignore loss-of-lock. Interrupt not requested. 1 Enable interrupt request upon loss-of-lock.
12 LOCIRQ	Loss-of-clock interrupt request The LOCIRQ bit enables a loss-of-clock interrupt request when the LOCF flag is set. If either LOCF or LOCIRQ is negated, the interrupt request is negated. If loss-of-clock is detected while in bypass mode, a system reset is generated. Therefore, LOCIRQ has no effect in bypass mode. See *Section 16.5.4.3: Loss-of-clock interrupt request*. 0 Ignore loss-of-clock. Interrupt not requested. 1 Enable interrupt request upon loss-of-clock.
13–29	Reserved, should be cleared.
30–31 ERFD	Enhanced reduced frequency divider This 2-bit field controls a divider at the output of the FMPLL. The value specified by the ERFD bits establishes the division factor applied to the FMPLL frequency. 00 Divide by 2 01 Divide by 4 10 Divide by 8 11 Divide by 16

16.4.2.5 Synthesizer FM Modulation Register (SYNFMMR)

This register controls the frequency modulation features of the FMPLL. FM modulation is not backwards compatible with previous devices. Therefore, this register must be used in enhanced mode. It can only be programmed when the FMPLL is locked. Writing to this register while the FMPLL is unlocked has no effect. Furthermore, when the PLL loses lock, FM modulation is disabled and the FMPLL_SYNFMMR is reset.

Offset 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BSY	MOD EN	MOD SEL	MODPERIOD												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 258. Synthesizer FM Modulation Register (SYNFMMR)

Table 166. SYNFMMR field descriptions

Field	Description
0 BSY	Busy This bit is asserted soon after a write access to the FMPLL_SYNFMMR, and remains asserted while the FMPLL processes the new FM modulation programming. The CPU must wait until this bit is negated before attempting another write access to this register. Any write attempt while the BSY flag is set will have no effect. 0 Write to the FMPLL_SYNFMMR is allowed. 1 The FMPLL is still busy processing the previous change on the FMPLL_SYNFMMR; write access to the register is not possible.
1 MODEN	Modulation enable This bit enables the frequency modulation. 0 Frequency modulation disabled 1 Frequency modulation enabled
2 MODSEL	Modulation selection This bit selects whether modulation will be centered around the nominal frequency or spread below the nominal frequency. 0 Modulation centered around nominal frequency 1 Modulation spread below nominal frequency
3–15 MODPERIOD	Modulation period This 13-bit field is the binary equivalent of the modperiod variable derived from the formula: $\text{modperiod} = \text{round}(\frac{f_{ref}}{4 \times f_{mod}})$ where f_{ref} represents the frequency of the feedback divider, and f_{mod} represents the modulation frequency.
16	Reserved, should be cleared.
17–31 INCSTEP	Increment step This 14-bit field is the binary equivalent of the incstep variable derived from the formula: $\text{incstep} = \text{round}(\frac{(2^{15} - 1) \times MD \times EMFD}{100 \times 5 \times \text{modperiod}})$ where MD represents the peak modulation depth in percentage (+/–MD for centered modulation, –2 * MD for modulation below nominal frequency), and EMFD represents the nominal value of the feedback loop divider.

Note: The product of INCSTEP and MODPERIOD cannot be larger than $(2^{15} - 1)$.

16.5 Functional description

This section explains the FMPLL operation and configuration.

16.5.1 Input clock frequency

The FMPLL is designed to operate over an input clock frequency range as determined by the operating mode. The operating ranges for each mode are given in *Table 167*.

Table 167. Input clock frequency at the predivider input

Mode	Input frequency range
Bypass mode with crystal reference Normal mode with crystal reference	4 MHz–20/40 MHz(1)
Bypass mode with external reference Normal mode with external reference	0 Hz–f_{sys}(2)

1. See *Section 16.1: Information specific to this device* for information on crystal frequencies supported.
2. f_{sys} is the system frequency of the MCU. The predivider ratio has to be chosen such that the input to the PLL itself (after the predivider) does not exceed 16 MHz.

16.5.2 Clock configuration

In legacy mode, the relationship between the output frequency f_{sys} and input frequency f_{ref} is determined by the PREDIV, MFD and RFD values programmed in the FMPLL_SYNCR, according to the following equation:

$$f_{sys} = f_{ref} \times \frac{MFD + 4}{(PREDIV + 1) \times 2^{RFD}}$$

In legacy mode, the relationship between the VCO frequency (f_{VCO}) and the output frequency (f_{SYS}) is determined by the value of the RFD value programmed in the FMPLL_SYNCR, according to the following equation:

$$f_{VCO} = 4 \times f_{sys} \times 2^{RFD}$$

In enhanced mode, the relationship between input and output frequency is determined by the EPREDIV, EMFD and ERFD values programmed in FMPLL_ESYNCR1 and FMPLL_ESYNCR2, according to the following equation:

$$f_{sys} = f_{ref} \times \frac{EMFD}{(EPREDIV + 1) \times 2^{(ERFD + 1)}}$$

In enhanced mode, the relationship between the VCO frequency (f_{VCO}) and the output frequency (f_{SYS}) is determined by the programmed value of FMPLL_ESYNCR2[ERFD], according to the following equation:

$$f_{VCO} = f_{sys} \times 2^{(ERFD + 1)}$$

When programming the FMPLL, be sure not to violate the maximum system clock frequency or max/min VCO frequency specification. Furthermore, the PREDIV or EPREDIV values must not be set to any value that causes the input frequency to the phase detector to go below 4 MHz.

The LOCK flag is immediately negated after any of the following events:

1. In legacy mode, the PREDIV or MFD fields of the FMPLL_SYNCR are changed
2. In enhanced mode, the EMODE, EPREDIV, EMFD of CLKCFG[1:2] fields of the FMPLL_ESYNCR1 are changed[j]

Upon any of these events an internal timer is initialized to count 64 cycles of the PLL input clock. During this period, the LOCK flag is held negated. After the timer expires, the LOCK flag reflects the value coming from the PLL lock detection circuitry. To prevent an immediate reset, the LOLRE bit must be cleared before doing any of the above operations.

Changing RFD or ERFD does not affect the FMPLL, hence no relock delay is incurred. Resulting changes in clock frequency are synchronized to the next falling edge of the current system clock. However, RFD or ERFD should only be changed when the LOCK bit is set, to avoid exceeding the allowable system operating frequency.

Coming out of reset, the FMPLL will be enabled (on), but running in bypass mode. The recommended procedure to program the FMPLL and engage normal mode is:

1. Assert the EMODE bit and program the EPREDIV and EMFD fields of FMPLL_ESYNCR1 and the RFD field of FMPLL_ESYNCR2.
2. Poll FMPLL_SYNSR[LOCK] until it asserts.
3. If required, program the FMPLL_SYNFMMR with desired frequency modulation parameters, poll the BSY bit until it negates, then enable FM by asserting the MODEN bit.
4. Engage normal mode by writing to FMPLL_ESYNCR1[CLKCFG].

16.5.3 Lock detection

A pair of counters monitor the reference and feedback clocks to determine when the system has acquired frequency lock. Once the FMPLL has locked, the counters continue to monitor the reference and feedback clocks and will report if/when the FMPLL has lost lock. The FMPLL registers provide the flexibility to select whether to generate an interrupt, assert system reset or do nothing in the event that the FMPLL loses lock.

Loss-of-lock reset and interrupt are only generated when the FMPLL is operating in normal mode. The LOCF bit is not asserted by a loss-of-lock condition detected during bypass, although going to bypass mode from normal mode does not automatically clear the flag if it was asserted while the FMPLL was in normal mode.

16.5.4 Loss-of-clock detection

The FMPLL reference and output clocks may be continuously monitored by a module called Clock Quality Monitor (CQM), shown in *Figure 259*. The intent of the CQM is to assure that the system bus clock is created from good clock sources. Whether the clocks are monitored or not is determined by the clock operating mode and control bits in the FMPLL registers, as shown in *Table 168*.

In bypass mode with crystal reference, the reference clock is always monitored, regardless of the state of the LOCEN bit. In bypass mode with external reference, the reference clock is not monitored, regardless of the state of the LOCEN bit. This is done so that the whole device frequency range can be sourced from the external clock generator when using

j. Note that changing only the CLKCFG[0] bit to move from bypass to normal or vice-versa, and keeping the values of the other FMPLL_ESYNCR1 fields unchanged, will not cause the PLL to lose lock or the lock flag to be cleared.

external reference mode. The FMPLL output may only monitored in normal mode, depending on the state of the LOCEN bit.

The clock quality monitor uses a 16 MHz RC oscillator plus a digital divider to obtain a 4 MHz clock to be used by the clock quality monitor. This clock is used as a reference time base to measure the frequency of the crystal oscillator and the FMPLL output. The frequency of these clocks are expected to be within the following frequency ranges:

- Reference clock must be within the crystal frequency range[k]
- PLL output must be above 1.5 MHz (minimum VCO free-running frequency divided by the maximum ERFD)

In the event either of the clocks fall outside the expected window, a loss of clock condition is reported. The FMPLL can be programmed to switch the system clock to a backup clock in the event of such a failure. Additionally, the user may select to have the system enter reset, assert an interrupt request, or do nothing if/when the FMPLL reports this condition.

Figure 259. Clock quality monitor

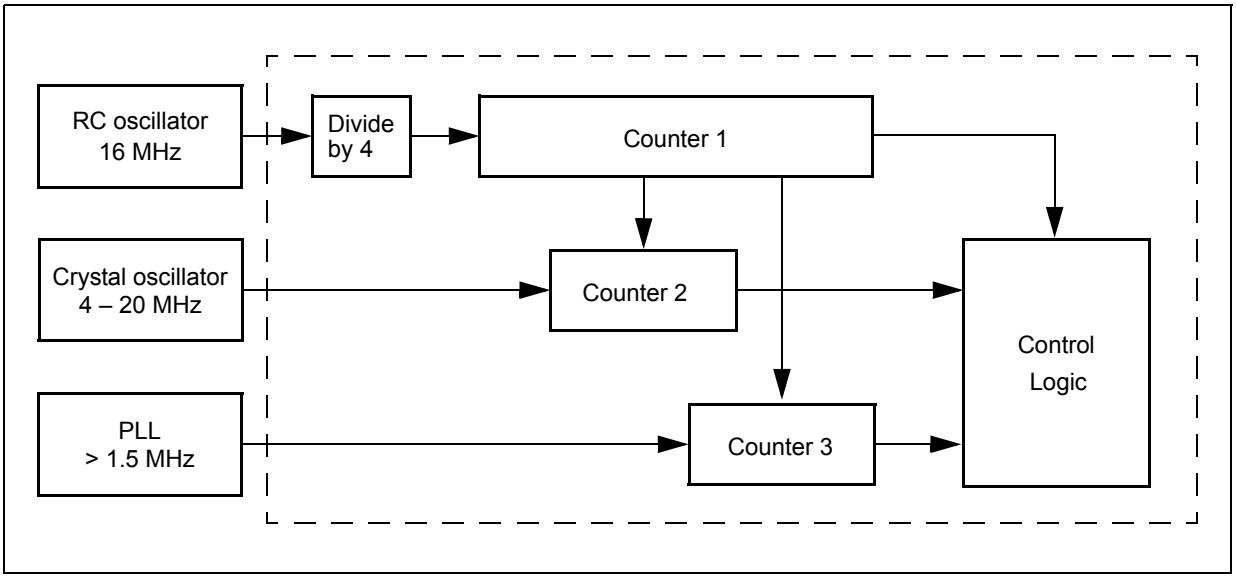


Table 168. Loss-of-clock monitoring

Operating mode	LOCEN(1)	Reference clock monitored?	FMPLL output monitored?
Bypass mode with external reference and PLL off	—	No	No
Bypass mode with crystal reference and PLL off	—	Yes	No
Bypass mode with external reference and PLL running	—	No	No
Bypass mode with crystal reference and PLL running	—	Yes	No
Normal mode with external reference	0	No	No
	1	No	Yes

k. See *Section 16.1: Information specific to this device* for information on crystal frequencies supported.

Table 168. Loss-of-clock monitoring (continued)

Operating mode	LOCEN(1)	Reference clock monitored?	FMPLL output monitored?
Normal mode with crystal reference	0	No	No
	1	Yes	Yes

1. LOCEN is the loss-of-clock enable bit in either FMPLL_SYNCR or FMPLL_ESYNCR2, depending on FMPLL_ESYNCR1[EMODE].

16.5.4.1 Alternate/backup clock selection

If loss-of-clock detection is enabled by LOCEN, the FMPLL is operating in normal mode and the Clock Quality Monitor detects a failure at the FMPLL output clock, then a backup clock selection feature automatically connects the system clock to the reference clock input (either external or crystal reference). After this happens, the system clock remains connected to the reference clock until next system reset, even if the FMPLL regains itself and relocks. If, however, the reference clock also fails, either simultaneously or after the FMPLL failure, the system clock is connected back to the FMPLL output.

If the reference fails in normal mode, then no backup clock selection occurs, and the FMPLL output continues to be the system clock. If the reference stops, the FMPLL will operate in free-running mode.

In bypass mode with crystal reference, a reference fail will force a reset. In bypass mode with external reference, no backup clock selection occurs if the reference fails.

16.5.4.2 Loss-of-clock reset

When a loss-of-clock condition is recognized, a system reset may be asserted depending on the clock operating mode and control bits in the FMPLL registers, as shown in *Table 169*. FMPLL_SYNSR[LOCF] and FMPLL_SYNSR[LOC] are cleared after reset, therefore, another means must be used externally to determine that a loss-of-clock condition occurred.

Table 169. Loss-of-clock reset

Operating mode	LOCEN (1)	LOCRE (2)	Reset	
			Reference failure	FMPLL failure
Bypass mode with external reference and PLL off	—	—	No	No
Bypass mode with crystal reference and PLL off	—	—	Yes	No
Bypass mode with external reference and PLL running	—	—	No	No
Bypass mode with crystal reference and PLL running	—	—	Yes	No
Normal mode with external reference	0	—	No	No
	1	0	No	No
	1	1	No	Yes
Normal mode with crystal reference	0	—	No	No
	1	0	No	No
	1	1	Yes	Yes

1. LOCEN is the loss-of-clock enable bit in either FMPLL_SYNCR or FMPLL_ESYNCR2, depending on FMPLL_ESYNCR1[EMODE].

2. LOCRE is the loss-of-clock reset enable bit in either FMPLL_SYNCR or FMPLL_ESYNCR2, depending on FMPLL_ESYNCR1[EMODE].

LOCEN and LOCRE have no effect in bypass mode. If the reference fails while the FMPLL is in bypass mode with crystal reference, a system reset is asserted regardless of the state of LOCEN and LOCRE. Since bypass is the FMPLL reset mode, the crystal oscillator must be present and functioning properly to exit reset when PLLREF = 1. When PLLREF = 0, the reference clock is not checked for loss-of-clock, so exit from reset can happen regardless the state of the reference clock. Exit from reset is not affected by the state of the FMPLL output because the FMPLL clock is not monitored in bypass mode.

16.5.4.3 Loss-of-clock interrupt request

When a loss-of-clock condition is recognized, an interrupt request may be asserted depending on the clock operating mode and control bits in the FMPLL registers, as shown in *Table 170*.

LOCEN and LOCIRQ have no effect in bypass mode. If the reference fails in bypass mode with crystal reference, a system reset is asserted instead of an interrupt request. If the reference fails in bypass with external reference, no reset or interrupts are generated. Furthermore, no reset or interrupts are generated when lock is lost due to a write to the FMPLL_SYNCR in legacy mode which modifies the PREDIV or MFD fields, or a write to FMPLL_ESYNCR1 in enhanced mode which modifies the EMODE, EPREDIV, EMFD or CLKCFG[1:0] fields.

Table 170. Loss-of-clock interrupt request

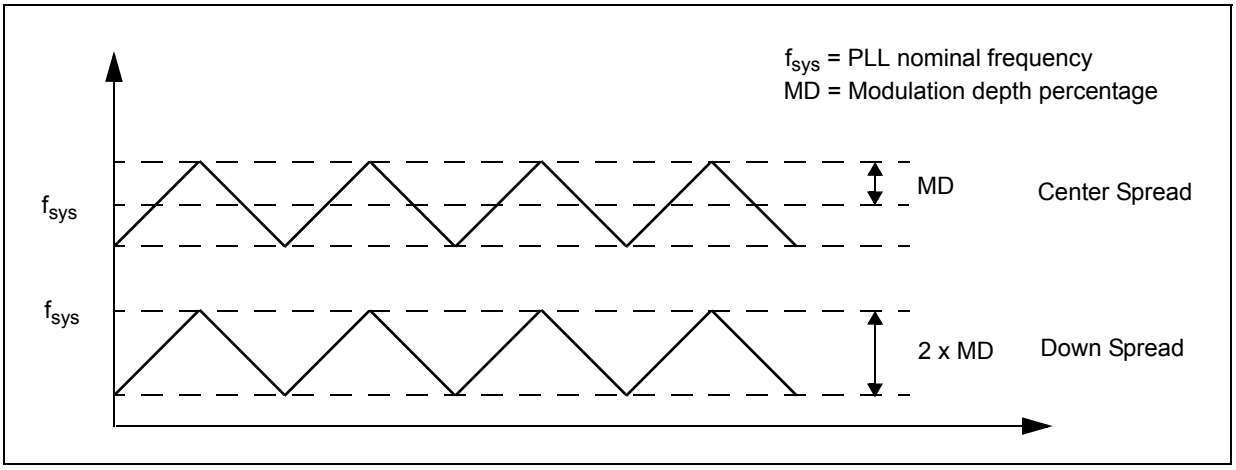
Operating mode	LOCEN (1)	LOCIRQ (2)	Interrupt request	
			Reference failure	FMPLL failure
Bypass mode with external reference and PLL off	—	—	—	—
Bypass mode with crystal reference and PLL off	—	—	No	—
Bypass mode with external reference and PLL running	—	—	—	—
Bypass mode with crystal reference and PLL running	—	—	No	—
Normal mode with external reference	0	—	—	No
	1	0	—	No
	1	1	—	Yes
Normal mode with crystal reference	0	—	No	No
	1	0	No	No
	1	1	Yes	Yes

1. LOCEN is the loss-of-clock enable bit in either FMPLL_SYNCR or FMPLL_ESYNCR2, depending on FMPLL_ESYNCR1[EMODE].
2. LOCIRQ is the loss-of-clock interrupt enable bit in either FMPLL_SYNCR or FMPLL_ESYNCR2, depending on FMPLL_ESYNCR1[EMODE].

16.5.5 Frequency modulation

Frequency modulation uses a triangular profile as shown in *Figure 260*. The modulation frequency and depth are set using the MODPERIOD and INCSTEP fields of the FMPLL_SYNFMMR.

Figure 260. Triangular frequency modulation



The following equations define how to calculate MODPERIOD and INCSTEP based on the frequency of the feedback divider (f_{ref}), the modulation frequency (f_{mod}) and the modulation depth percentage (MD):

$$\text{MODPERIOD} = \text{round}\left(\frac{f_{ref}}{4 \times f_{mod}}\right)$$

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

MODPERIOD and INCSTEP are subject to the following restriction:

$$(\text{MODPERIOD} \times \text{INCSTEP}) < 2^{15}$$

Because of the above rounding operations, the effective modulation depth applied to the FMPLL is given by the following formula:

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

As an example, suppose the following configuration:

- Input frequency: 4 MHz
- Loop divider (EMFD): 64
- Input divider: 1
- VCO frequency: 4 MHz × 64 = 256 MHz
- PLL output frequency: 256 MHz / ERFD
- Center spread (MODSEL = 0)
- Modulation frequency: 24 kHz
- Modulation depth: +/– 2.0% (4 % peak-to-peak)
- MODPERIOD = Round [$(4 \times 10^6) / (4 \times 24 \times 10^3)$] = Round [41.66] = 42
- INCSTEP = Round [$((2^{15} - 1) \times 2 \times 64) / (100 \times 5 \times 42)$] = Round [199.722] = 200
- MODPERIOD × INCSTEP = 42 × 200 = 8400 (which is less than 2^{15})
- MD (quantized) = $((42 \times 200 \times 100 \times 5) / ((2^{15} - 1) \times 64)$ = 2.00278%

In this example, the modulation depth error is 0.00278%.

The FM parameters can only be changed, and FM can only be enabled, when the PLL is locked. Writing to the FMPLL_SYNFMMR while the PLL is unlocked has no effect. Furthermore, when the PLL loses lock, the FM modulation parameters are reset and the modulation is disabled until the PLL relocks and the FMPLL_SYNFMMR is programmed again.

After programming FM parameters, it takes some time until these parameters get propagated to the PLL analog circuitry. During this time, the BSY bit gets asserted. The modulation must only be enabled when the FM parameters have already propagated to the analog circuitry. Therefore, the sequence for programming FM is:

1. Poll FMPLL_SYNSR[LOCK] until it asserts.
2. Program the MODSEL, MODPERIOD and INCSTEP fields of the FMPLL_SYNFMMR.
3. Poll FMPLL_SYNFMMR[BSY] until it negates.
4. Assert FMPLL_SYNFMMR[MODEN].

17 Error Correction Status Module (ECSM)

17.1 Overview

The Error Correction Status Module (ECSM) provides registers for capturing information on memory errors reported by error-correcting codes.

17.2 Module memory map

Table 171 is a 32-bit view of the ECSM's memory map.

Table 171. ECSM 32-bit memory map

ECSM offset	Register			
0x00 – 0x3C	Reserved			
0x40	Reserved			ECC Configuration Register (ECSM_ECR) *on page 507*
0x44	Reserved			ECC Status Register (ECSM_ESR) *on page 508*
0x48	Reserved		ECC Error Generation Register (ECSM_EEGR) *on page 510*	
0x4C	Reserved			
0x50	Flash ECC Address Register (ECSM_FEAR) *on page 513*			
0x54	Reserved		Flash ECC Master Number Register (ECSM_FEMR) *on page 513*	Flash ECC Attributes (ECSM_FEAT) Register *on page 514*
0x58	Flash ECC Data Register High (ECSM_FEDRH) *on page 515*			
0x5C	Flash ECC Data Register Low (ECSM_FEDRL) *on page 515*			
0x60	RAM ECC Address Register (ECSM_REAR) *on page 516*			
0x64	Reserved	RAM ECC Syndrome Register (ECSM_PRESR) *on page 516*	RAM ECC Master Number Register (ECSM_REMR) *on page 520*	RAM ECC Attributes Register (ECSM_REAT) *on page 520*
0x68	RAM ECC Data Register High (ECSM_REDRH) *on page 521*			
0x6C	RAM ECC Data Register Low (ECSM_REDRL) *on page 521*			

17.3 Register descriptions

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.

17.3.1 ECC registers

There are a number of program-visible registers for the sole purpose of reporting and logging of memory failures. These registers are listed in *Table 171*.

The details on the ECC registers are provided in the subsequent sections.

17.3.1.1 ECC Configuration Register (ECSM_ECR)

The ECC Configuration Register is an 8-bit control register for specifying which types of memory errors are reported. 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 can be useful for subsequent failure analysis.

Register address: ECSM Base + 0x43

	0	1	2	3	4	5	6	7
R	0	0	ER1BR	EF1BR	0	0	ERNCR	EFNCR
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented

Figure 261. ECC Configuration Register (ECSM_ECR)

Table 172. ECSM_ECR field descriptions

Name	Description
ER1BR	Enable 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 RAM correction generates a ECSM ECC interrupt request as signalled by the assertion of ECSM_ESR[R1BC]. The address, attributes and data are also captured in the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers.
EF1BR	Enable Flash 1-bit Reporting 0 = Reporting of single-bit platform flash corrections is disabled. 1 = Reporting of single-bit platform flash corrections is enabled. This bit can only be set if the SoC-configurable input enable signal is asserted. The occurrence of a single-bit flash correction generates a ECSM ECC interrupt request as signalled by the assertion of ECSM_ESR[F1BC]. The address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers.
ERNCR	Enable Platform RAM Non-Correctable Reporting 0 = Reporting of non-correctable RAM errors is disabled. 1 = Reporting of non-correctable RAM errors is enabled. The occurrence of a non-correctable multi-bit RAM error generates a ECSM ECC interrupt request as signalled by the assertion of ECSM_ESR[RNCE]. The faulting address, attributes and data are also captured in the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers.
EFNCR	Enable Platform Flash Non-Correctable Reporting 0 = Reporting of non-correctable flash errors is disabled. 1 = Reporting of non-correctable flash errors is enabled. The occurrence of a non-correctable multi-bit flash error generates a ECSM ECC interrupt request as signalled by the assertion of ECSM_ESR[FNCE]. The faulting address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers.

17.3.1.2 ECC Status Register (ECSM_ESR)

The ECC Status Register is an 8-bit control register for signaling which types of properly-enabled ECC events have been detected. The ECSM_ESR signals the last, properly-enabled memory event to be detected. An ECC interrupt request is asserted if any flag bit is asserted and its corresponding enable bit is asserted.

The ECSM allows a maximum of one bit of the ECSM_ESR to be asserted at any given time. This preserves the association between the ECSM_ESR and the corresponding address and attribute registers, which are loaded on each occurrence of a properly-enabled ECC event. If there is a pending ECC interrupt and another properly-enabled ECC event occurs, the ECSM hardware automatically handles the ECSM_ESR reporting, clearing the previous data and loading the new state and thus guaranteeing that only a single flag is asserted.

To maintain the coherent software view of the reported event, the following sequence in the ECSM error interrupt service routine is suggested:

1. Read the ECSM_ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ECSM_ESR and verify the current contents matches the original contents. If the two values are different, go back to step 1 and repeat.
4. When the values are identical, write a 1 to the asserted ECSM_ESR flag to negate the interrupt request.

Register address: ECSM Base + 0x47

	0	1	2	3	4	5	6	7
R	0	0	R1BC	F1BC	0	0	RNCE	FNCE
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented

Figure 262. ECC Status Register (ECSM_ESR)

Table 173. ECSM_ESR field descriptions

Name	Description
R1BC	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 ECSM_ECR[ER1BR] is asserted. The occurrence of a properly-enabled single-bit RAM correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.
F1BC	Platform Flash 1-bit Correction 0 = No reportable single-bit platform flash correction has been detected. 1 = A reportable single-bit platform flash correction has been detected. This bit can only be set if ECSM_ECR[EF1BR] is asserted. The occurrence of a properly-enabled single-bit flash correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.
RNCE	RAM Non-Correctable Error 0 = No reportable non-correctable RAM error has been detected. 1 = A reportable non-correctable RAM error has been detected. The occurrence of a properly-enabled non-correctable RAM error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.
FNCE	Flash Non-Correctable Error 0 = No reportable non-correctable flash error has been detected. 1 = A reportable non-correctable flash error has been detected. The occurrence of a properly-enabled non-correctable flash error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR 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 highest priority RNCE, and finally FNCE.

17.3.1.3 ECC Error Generation Register (ECSM_EEGR)

The ECC Error Generation Register is a 16-bit control register used to force the generation of single- and double-bit data inversions in the memories with ECC, most notably the RAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for “injecting” errors into the memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

It should be noted that while the ECSM_EEGR is associated with the RAM, similar capabilities exist for the flash, that is, the ability to program the non-volatile memory with single- or double-bit errors is supported for the same two reasons previously identified.

For both types of memories (RAM and flash), the intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit non-correctable errors that are terminated with an error response.

The enabling of these error generation modes requires the same SoC-configurable input enable signal (as that used to enable single-bit correction reporting) be asserted.

Register address: ECSM Base + 0x004A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	FRC1BI(1)	FR11BI(1)	0	0	FRCNCI	FR1NCI	ERRBIT[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 263. ECC Error Generation Register (ECSM_EEGR)

1. This field is writable only in test mode in cut 1.0 devices.

Table 174. ECSM_EEGR field descriptions

Name	Description
FRC1BI(1)	Force RAM Continuous 1-bit Data Inversions 0 = No RAM continuous 1-bit data inversions are generated. 1 = 1-bit data inversions in the RAM are continuously generated. The assertion of this bit forces the RAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT[6:0], continuously on every write operation. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to correctly re-enable the error generation logic.

Table 174. ECSM_EEGR field descriptions (continued)

Name	Description
FR11BI[(1)]	Force RAM One 1-bit Data Inversion 0 = No RAM single 1-bit data inversion is generated. 1 = One 1-bit data inversion in the RAM is generated. The assertion of this bit forces the RAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT[6:0], on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic.
FRCNCI	Force RAM Continuous Non-Correctable Data Inversions 0 = No RAM continuous 2-bit data inversions are generated. 1 = 2-bit data inversions in the RAM are continuously generated. The assertion of this bit forces the RAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, continuously on every write operation. After this bit has been enabled to generate another continuous non-correctable data inversion, it must be cleared before being set again to properly re-enable the error generation logic. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM.
FR1NCI	Force RAM One Non-Correctable Data Inversions 0 = No RAM single 2-bit data inversions are generated. 1 = One 2-bit data inversion in the RAM is generated. The assertion of this bit forces the RAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM. After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic.

Table 174. ECSM_EEGR field descriptions (continued)

Name	Description
ERRBIT [7:0]	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 RAM controller follows a vector bit ordering scheme where LSB=0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the RAM width. For example, consider a 64-bit RAM implementation and ECC organized on a 32-bit boundary. The 32-bit ECC approach requires 7 code bits for each 32-bit word. For RAM data width of 64 bits, the actual SRAM is 2 × (32 bits data + 7 bits 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 95, no bit position is inverted. To accommodate address bus inversions, the ERRBIT values start at 96 as defined: if ERRBIT = 96, then ADDR[0] is inverted if ERRBIT = 97, then ADDR[1] 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 RAM controller.

1. This field is writable only in test mode in cut 1.0 devices.

If an attempt to force a non-correctable inversion (by asserting ECSM_EEGR[FRCNCI] or ECSM_EEGR[FRC1NCI]) and ECSM_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.

17.3.1.4 Flash ECC Address Register (ECSM_FEAR)

The ECSM_FEAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register is read-only; any attempted write is ignored.

Register address: ECSM Base + 0x50

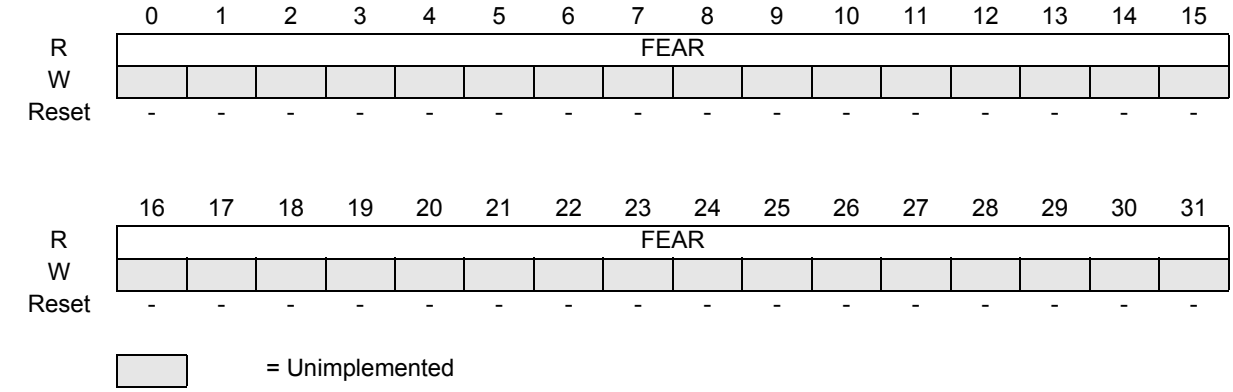


Figure 264. Flash ECC Address Register (ECSM_FEAR)

Table 175. ECSM_FEAR field descriptions

Name	Description
FEAR	Flash ECC Address Register This 32-bit register contains the faulting access address of the last, properly-enabled Flash ECC event.

17.3.1.5 Flash ECC Master Number Register (ECSM_FEMR)

The ECSM_FEMR is a 4-bit register for capturing the XBAR bus master number of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers, and the appropriate flag (FNCE) in the ECC Status Register to be asserted.

This register is read-only; any attempted write is ignored.

Register address: ECSM Base + 0x56

	0	1	2	3	4	5	6	7
R	0	0	0	0	FEMR			
W								
Reset	0	0	0	0	-	-	-	-

= Unimplemented

Figure 265. Flash ECC Master Number Register (ECSM_FEMR)

Table 176. ECSM_FEMR field descriptions

Name	Description
FEMR	Flash ECC Master Number Register This 4-bit register contains the XBAR bus master number of the faulting access of the last, properly-enabled flash ECC event.

17.3.1.6 Flash ECC Attributes (ECSM_FEAT) Register

The ECSM_FEAT register is an 8-bit register for capturing the XBAR bus master attributes of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers, and the appropriate flag (FNCE) in the ECC Status Register to be asserted.

This register is read-only; any attempted write is ignored.

Register address: ECSM Base + 0x57

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
Reset	-	-	-	-	-	-	-	-

= Unimplemented

Figure 266. Flash ECC Attributes (ECSM_FEAT) Register

Table 177. ECSM_FEAT field descriptions

Name	Description
WRITE	AMBA-AHB HWRITE 0 = AMBA-AHB read access 1 = AMBA-AHB write access

Table 177. ECSM_FEAT field descriptions (continued)

Name	Description
SIZE	AMBA-AHB HSIZE 0b000 = 8-bit AMBA-AHB access 0b001 = 16-bit AMBA-AHB access 0b010 = 32-bit AMBA-AHB access 0b011 = 64-bit AMBA-AHB access 0b1xx = Reserved
PROTECTION	AMBA-AHB HPROT 0b0xxx = Non-cacheable,0b1xxx = Cacheable 0bx0xx = Non-bufferable, 0bx1xx = Bufferable 0bxx0x = User mode, 0bxx1x = Supervisor mode 0bxxx0 = I-Fetch, 0bxxx1 = Data

17.3.1.7 Flash ECC Data Register (ECSM_FEDRH, ECSM_FEDRL)

The ECSM_FEDR is a 64-bit register for capturing the data associated with the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers, and the appropriate flag (FNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored.

Register address: ECSM Base + 0x58, +0x5c

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

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

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
R	FEDRL															
W																
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
R	FEDRL															
W																
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

= Unimplemented

Figure 267. Flash ECC Data Register (ECSM_FEDR)

Table 178. ECSM_FEDR field descriptions

Name	Description
FEDR	Flash ECC Data Register This 64-bit register contains the data associated with the faulting access of the last, properly-enabled flash ECC event. The register contains the data value taken directly from the data bus.

17.3.1.8 RAM ECC Address Register (ECSM_REAR)

The ECSM_REAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (RNCE) in the ECC Status Register to be asserted.

This register is read-only; any attempted write is ignored.

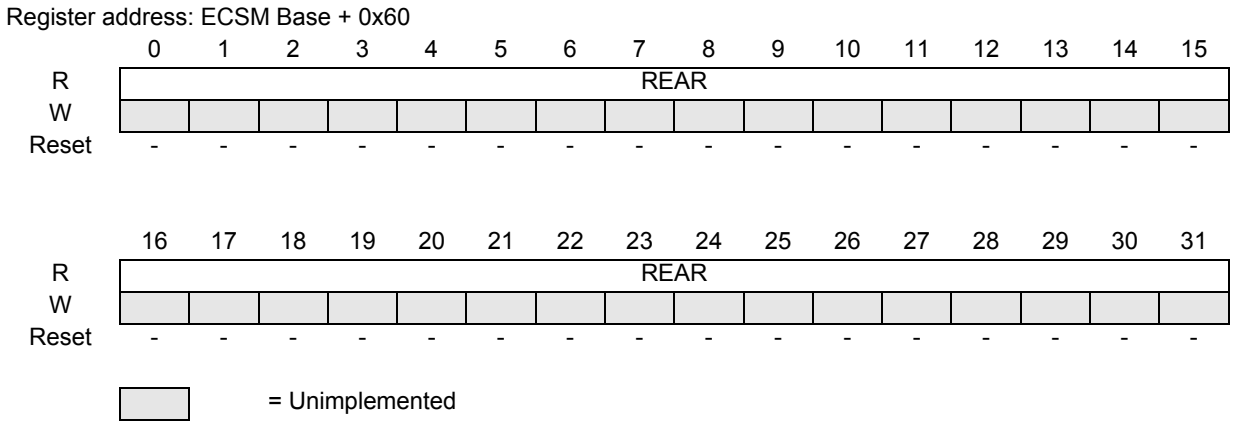


Figure 268. RAM ECC Address Register (ECSM_REAR)

Table 179. ECSM_REAR field descriptions

Name	Description
REAR	RAM ECC Address Register This 32-bit register contains the faulting access address of the last, properly-enabled RAM ECC event.

17.3.1.9 RAM ECC Syndrome Register (ECSM_PRESR)

The ECSM_PRESR is an 8-bit register for capturing the error syndrome of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (RNCE) in the ECC Status Register to be asserted.

The e200z3 core on this device is a cacheless processor. In order to improve performance on 32-bit write accesses, the RAM ECC is calculated on a 32-bit boundary versus the 64-bit

organization used on other devices based on the e200z6 processor. For the cacheless e200z3 processor, most RAM writes will be 32 bits or smaller in size. It was estimated that implementation of a 32-bit RAM ECC provides 7–8% overall performance improvement over a 64-bit organization that would force read-modify-write cycles to write 32-bit data. Each 32-bit word requires a 7-bit ECC code, so the RAM array is organized in two banks of 39 bits each.

In order to support the new 32-bit boundary ECC structure and maintain compatibility with other devices, the device 7-bit syndrome and bank select (even, odd) were remapped into the equivalent 8-bit syndrome of 64-bit organized memories (see *Table 181*).

The ECSM_PRESR register is read-only; any attempted write is ignored.

Register address: ECSM Base + 0x65

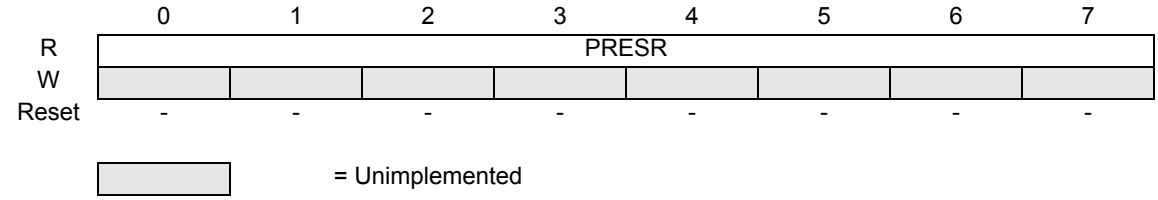


Figure 269. RAM ECC Syndrome Register (ECSM_PRESR)

Table 180. ECSM_PRESR field descriptions

Name	Description
PRESR	RAM ECC Syndrome Register This 8-bit field contains a report syndrome obtained by mapping the raw 7-bit ECC syndrome from each 32-bit bank (even, odd) to an 8-bit syndrome that is backwards compatible with 64-bit organized memories. Each 7-bit raw syndrome is formed by 6 bits of Hamming decoded parity plus an odd-parity bit for the entire 39-bit (32-bit data + 7 ECC) code word. The mapping shown in *Table 181* associates the raw syndrome of each back to the reported syndrome obtainable from the ECSM_PRESR register.

Table 181 shows the mapping between the raw ECC syndrome calculated for each 32-bit bank (even, odd) and the syndrome reported on PRES[7:0]. This table follows the bit vectoring notation where the LSB=0.

Table 181. RAM syndrome mapping for single-bit correctable errors

Bank	Data bit in error	Raw syndrome	PRES[7:0]
One-bit correctable errors			
Odd	31	0x06	0x4E
Odd	30	0x0A	0x50
Odd	29	0x0C	0x52
Odd	28	0x0E	0x54
Odd	27	0x12	0x56

Table 181. RAM syndrome mapping for single-bit correctable errors (continued)

Bank	Data bit in error	Raw syndrome	PRES[7:0]
Odd	26	0x14	0x58
Odd	25	0x16	0x5A
Odd	24	0x18	0x5C
Odd	23	0x1A	0x5E
Odd	22	0x1C	0x60
Odd	21	0x50	0x62
Odd	20	0x22	0x64
Odd	19	0x24	0x66
Odd	18	0x26	0x68
Odd	17	0x28	0x6A
Odd	16	0x2A	0x6C
Odd	15	0x2C	0x6E
Odd	14	0x58	0x70
Odd	13	0x30	0x72
Odd	12	0x32	0x74
Odd	11	0x34	0x76
Odd	10	0x64	0x78
Odd	9	0x38	0x7A
Odd	8	0x62	0x7C
Odd	7	0x70	0x7E
Odd	6	0x60	0x82
Odd	5	0x42	0x84
Odd	4	0x44	0x86
Odd	3	0x46	0x88
Odd	2	0x48	0x8A
Odd	1	0x4A	0x8C
Odd	0	0x4C	0x8E
Even	31	0x06	0x06
Even	30	0x0A	0x0A
Even	29	0x0C	0x0C
Even	28	0x0E	0x0E
Even	27	0x12	0x12
Even	26	0x14	0x14
Even	25	0x16	0x16
Even	24	0x18	0x18

Table 181. RAM syndrome mapping for single-bit correctable errors (continued)

Bank	Data bit in error	Raw syndrome	PRES[7:0]
Even	23	0x1A	0x1A
Even	22	0x1C	0x1C
Even	21	0x50	0x1E
Even	20	0x22	0x22
Even	19	0x24	0x24
Even	18	0x26	0x26
Even	17	0x28	0x28
Even	16	0x2A	0x2A
Even	15	0x2C	0x2C
Even	14	0x58	0x2E
Even	13	0x30	0x30
Even	12	0x32	0x32
Even	11	0x34	0x34
Even	10	0x64	0x36
Even	9	0x38	0x38
Even	8	0x62	0x3A
Even	7	0x70	0x3C
Even	6	0x60	0x3E
Even	5	0x42	0x42
Even	4	0x44	0x44
Even	3	0x46	0x46
Even	2	0x48	0x48
Even	1	0x4A	0x4A
Even	0	0x4C	0x4C
Multi-bit non-correctable errors			
Odd		0x00	0x00
Odd		0x02	0x02
Odd		0x04	0x04
Odd		0x08	0x08
Odd		0x10	0x10
Odd		0x20	0x20
Odd		0x40	0x40
Even		0x00	0x80
Even		0x02	0xC2

Table 181. RAM syndrome mapping for single-bit correctable errors (continued)

Bank	Data bit in error	Raw syndrome	PRES[7:0]
Even		0x04	0xC4
Even		0x08	0xC8
Even		0x10	0xD0
Even		0x20	0xE0
Even		0x40	0xF0

17.3.1.10 RAM ECC Master Number Register (ECSM_REMR)

The ECSM_REMR is a 4-bit register for capturing the XBAR bus master number of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (RNCE) in the ECC Status Register to be asserted.

This register is read-only; any attempted write is ignored.

Register address: ECSM Base + 0x66

	0	1	2	3	4	5	6	7
R	0	0	0	0	REMR			
W								
Reset	0	0	0	0	-	-	-	-

= Unimplemented

Figure 270. RAM ECC Master Number Register (ECSM_REMR)

Table 182. ECSM_REMR field descriptions

Name	Description
REMR	RAM ECC Master Number Register This 4-bit register contains the XBAR bus master number of the faulting access of the last, properly-enabled RAM ECC event.

17.3.1.11 RAM ECC Attributes Register (ECSM_REAT)

The ECSM_REAT register is an 8-bit register for capturing the XBAR bus master attributes of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (RNCE) in the ECC Status Register to be asserted. The ECSM_REAT is read only.

Register address: ECSM Base + 0x67

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
Reset	-	-	-	-	-	-	-	-

= Unimplemented

Figure 271. RAM ECC Attributes (ECSM_REAT) Register

Table 183. ECSM_REAT field descriptions

Name	Description
WRITE	AMBA-AHB HWRITE 0 = AMBA-AHB read access 1 = AMBA-AHB write access
SIZE	AMBA-AHB HSIZE 0b000 = 8-bit AMBA-AHB access 0b001 = 16-bit AMBA-AHB access 0b010 = 32-bit AMBA-AHB access 0b011 = 64-bit AMBA-AHB access 0b1xx = Reserved
PROTECTION	AMBA-AHB HPROT 0b0xxx = Non-cacheable, 0b1xxx = Cacheable 0bx0xx = Non-bufferable, 0bx1xx = Bufferable 0bxx0x = User mode, 0bxx1x = Supervisor mode 0bxxx0 = I-Fetch, 0bxxx1 = Data

17.3.1.12 RAM ECC Data Register (ECSM_REDRH, ECSM_REDRL)

The ECSM_REDR is a 64-bit register for capturing the data associated with the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_PRESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (RNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored.

Register address: ECSM Base + 0x68, +0x6c

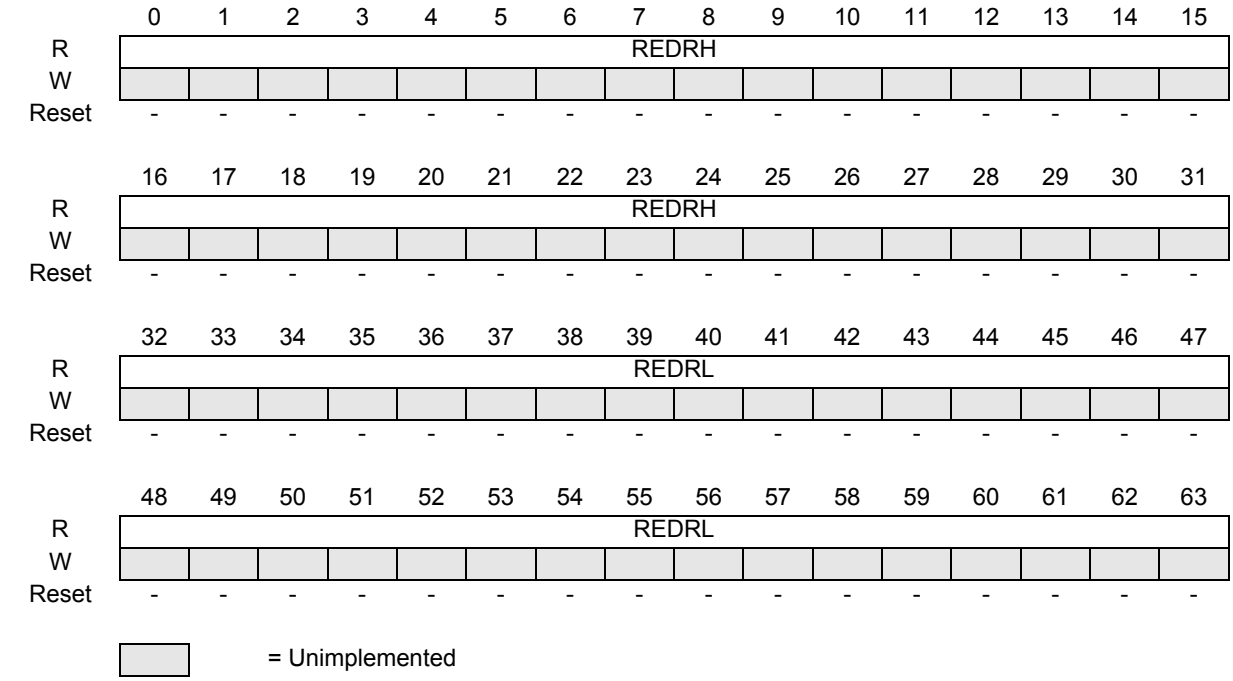


Figure 272. RAM ECC Data Register (ECSM_REDR)

Table 184. ECSM_REDR field descriptions

Name	Description
REDR	RAM ECC Data Register This 64-bit register contains the data associated with the faulting access of the last, properly-enabled RAM ECC event. The register contains the data value taken directly from the data bus.

18 System Timer Module (STM)

18.1 Introduction

18.1.1 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).

18.1.2 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

18.1.3 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.

18.2 External signal description

The STM does not have any external interface signals.

18.3 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.

18.3.1 Memory map

The STM memory map is shown in *Table 185*.

Table 185. STM memory map

Address offset	Register	Location
0x0000	STM Control Register (STM_CR)	*on page 524*
0x0004	STM Counter Value (STM_CNT)	*on page 525*
0x0008	Reserved	
0x000C	Reserved	

Table 185. STM memory map (continued)

Address offset	Register	Location
0x0010	STM Channel 0 Control Register (STM_CCR0)	on page 525
0x0014	STM Channel 0 Interrupt Register (STM_CIR0)	on page 526
0x0018	STM Channel 0 Compare Register (STM_CMP0)	on page 526
0x001C	Reserved	
0x0020	STM Channel 1 Control Register (STM_CCR1)	on page 525
0x0024	STM Channel 1 Interrupt Register (STM_CIR1)	on page 526
0x0028	STM Channel 1 Compare Register (STM_CMP1)	on page 526
0x002C	Reserved	
0x0030	STM Channel 2 Control Register (STM_CCR2)	on page 525
0x0034	STM Channel 2 Interrupt Register (STM_CIR2)	on page 526
0x0038	STM Channel 2 Compare Register (STM_CMP2)	on page 526
0x003C	Reserved	
0x0040	STM Channel 3 Control Register (STM_CCR3)	on page 525
0x0044	STM Channel 3 Interrupt Register (STM_CIR3)	on page 526
0x0048	STM Channel 3 Compare Register (STM_CMP3)	on page 526
0x004C - 0x3FFF	Reserved	

18.3.2 Register descriptions

The following sections detail the individual registers within the STM programming model.

Figure 273 shows the conventions used in the register figures.

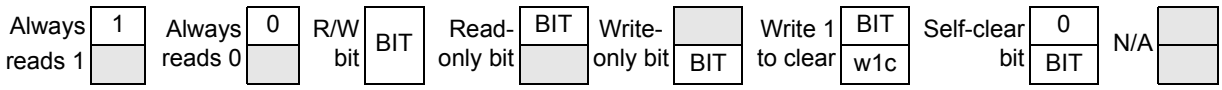


Figure 273. Key to Register Fields

18.3.2.1 STM Control Register (STM_CR)

The STM Control Register (STM_CR) includes the prescale value, freeze control and timer enable bits.

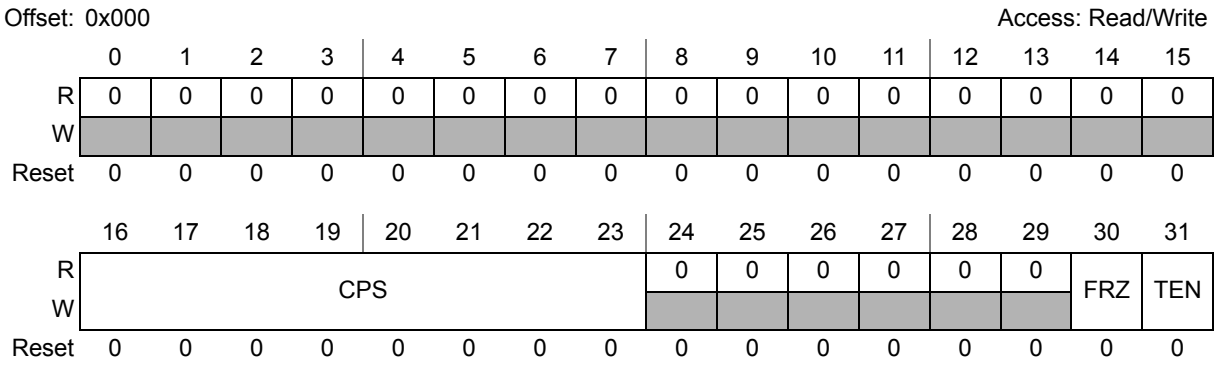


Figure 274. STM Control Register (STM_CR)

Table 186. STM_CR field descriptions

Field	Description
16:23 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
30 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.
31 TEN	Timer Counter Enabled. 0 Counter is disabled. 1 Counter is enabled.

18.3.2.2 STM Count Register (STM_CNT)

The STM Count Register (STM_CNT) holds the timer count value.

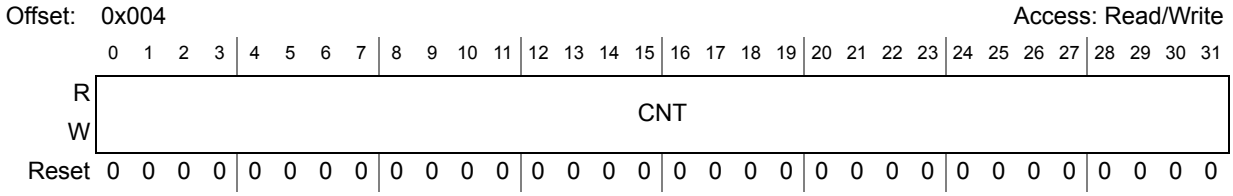


Figure 275. STM Count Register (STM_CNT)

Table 187. STM_CNT field descriptions

Field	Description
0:31 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.

18.3.2.3 STM Channel Control Register (STM_CCR*n*)

The STM Channel Control Register (STM_CCR*n*) has the enable bit for channel n of the timer.

Offset: 0x10+0x10*n Access: Read/Write

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

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

Figure 276. STM Channel Control Register (STM_CCRn)

Table 188. STM_CCR*n* field descriptions

Field	Description
31 CEN	Channel Enable. 0 The channel is disabled. 1 The channel is enabled.

18.3.2.4 STM Channel Interrupt Register (STM_CIRn)

The STM Channel Interrupt Register (STM_CIR*n*) has the interrupt flag for channel n of the timer.

Offset: 0x14+0x10*n Access: Read/Write

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

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

Figure 277. STM Channel Interrupt Register (STM_CIRn)

Table 189. STM_CIR*n* field descriptions

Field	Description
31 CIF	Channel Interrupt Flag 0 No interrupt request. 1 Interrupt request due to a match on the channel.

18.3.2.5 STM Channel Compare Register (STM_CMP*n*)

The STM channel compare register (STM_CMP*n*) holds the compare value for channel n.

Offset: 0x18+0x10*n Access: Read/Write

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31
R W	CMP							
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 278. STM Channel Compare Register (STM_CMPn)

Table 190. STM_CMP*n* field descriptions

Field	Description
0:31 CMP	Compare value for channel n. If the STM_CCR*n*[CEN] bit is set and the STM_CMP*n* register matches the STM_CNT register, a channel interrupt request is generated and the STM_CIR*n*[CIF] bit is set.

18.4 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_CCR*n*), a channel interrupt register (STM_CIR*n*) and a channel compare register (STM_CMP*n*). The channel is enabled by setting the STM_CCR*n*[CEN] bit. When enabled, the channel will set the STM_CIR*n*[CIF] bit and generate an interrupt request when the channel compare register matches the timer counter. The interrupt request is cleared by writing a 1 to the STM_CIR*n*[CIF] bit. A write of 0 to the STM_CIR*n*[CIF] bit has no effect.

Note: STM counter does not advance when the system clock is stopped.

19 Software Watchdog Timer (SWT)

19.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

19.1.1 Device-specific features

This device includes one SWT module.

The SWT can be clocked by either the system clock or crystal clock. Both clocks are connected to the SWT and can be stopped by a bit on the SIU_HLT register.

The SWT can be configured to generate an interrupt upon watchdog timeout. The SWT interrupt is 'ORed' with the NMI signal from the SIU and routed to the NMI and Critical Interrupt inputs of the CPU; see the SIU chapter for details.

The SWT includes an interrupt status bit so the ISR software can determine if the NMI request came from the SWT or the external NMI pin.

19.1.2 Reset assertion

The SWT can assert a reset when the watchdog timer expires. This reset will cause a system reset equivalent to assertion of the RESET pin. Bit 6 of the Reset Status Register in the SIU indicates the SWT as the source of the last reset.

19.1.3 Default configuration

On this device, the SWT comes out of reset with the following default values:

- CPU master access only, (MAPn = 0xFF); all 0xFF means that all masters have access to write the SWT
- Reset on Invalid (SWT_CR[RIA] = 1)
- No window mode (SWT_CR[WND] = 0)
- To cause a reset on timeout (SWT_CR[ITR] = 0)
- No hard lock (SWT_CR[HLK] = 0)
- Soft locked (SWT_CR[SLK] = 0)
- Clocked from the crystal clock (SWT_CR[CSL] = 1)
- Continues to run in STOP mode (SWT_CR[STP] = 0)
- Stopped in debug mode (SWT_CR[FRZ] = 1)
- Enabled (SWT_CR[WEN] = 1); out of reset, SWT_CR[WEN] = 1 by default but this can be controlled by the Reset Configuration Half Word
- 32.7 ms timeout for 8 MHz crystal
  WATCHDOG_PERIOD timeout for 20 MHz crystal

  Could be replaced with:

  SWT_TO reset value = 0x0013FDE0 giving a timeout period of 32.7 ms for 8 MHz crystal and 13.1 ms for 20 MHz crystal

19.2 Introduction

19.2.1 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 timeout 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 timeout, a reset is always generated on a second consecutive timeout.

19.2.2 Features

The SWT has the following features:

- 32-bit timeout register to set the timeout period
- Programmable selection of system or oscillator clock for timer operation
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial timeout
- Programmable selection of fixed or keyed servicing
- Master access protection
- Hard and soft configuration lock bits

19.2.3 Modes of operation

The SWT supports three device modes of operation:

- Normal—When the SWT is enabled in normal mode, its counter runs continuously.
- Debug—In debug mode, operation of the counter is controlled by the bit SWT_CR[FRZ]. If SWT_CR[FRZ] is set, the counter is stopped in debug mode, otherwise it continues to run.
- Stop—In stop mode, operation of the counter is controlled by the bit SWT_CR[STP]. If SWT_CR[STP] is set, the counter is stopped in stop mode, otherwise it continues to run.

19.3 External signal description

The SWT module does not have any external interface signals.

19.4 Memory map and register definition

The SWT programming model has seven 32-bit registers that 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 bit SWT_CR[RIA] 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, and SWT_SK registers are read-only.

19.4.1 Memory map

The SWT memory map is shown in *Table 191*.

Table 191. SWT memory map

Address offset (from SWT module base address FFF3_8000)	Register	Size (bits)	Access	Location
0x0000	SWT Control Register (SWT_CR)	32	R/W(1)	*on page 530*
0x0004	SWT Interrupt Register (SWT_IR)	32	R/W	*on page 532*
0x0008	*SWT Timeout Register (SWT_TO)*	32	R/W(1)	*on page 532*
0x000C	*SWT Window Register (SWT_WN)*	32	R/W(1)	*on page 533*
0x0010	*SWT Service Register (SWT_SR)*	32	R/W	*on page 533*
0x0014	*SWT Counter Output Register (SWT_CO)*	32	R	*on page 534*
0x0018	*SWT Service Key Register (SWT_SK)*	32	R/W(1)	*on page 534*
0x001C – 0x3FFF	Reserved	—	—	—

1. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

19.4.2 Register descriptions

The following sections detail the individual registers within the SWT programming model.

19.4.2.1 SWT Control Register (SWT_CR)

The SWT_CR contains fields for configuring and controlling the SWT.

Offset 0x0000 Access: Read/Write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MAP0	MAP1	MAP2	MAP3	MAP4	MAP5	MAP6	MAP7	0	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	KEY	RIA	WND	ITR	HLK	SLK	CSL	STP	FRZ	WEN
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1

Figure 279. SWT Control Register (SWT_CR)

1. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Table 192. SWT_CR field descriptions

Field	Description
MAPn	Master Access Protection for Master n The reset value for all MAPn bits is 1, giving access for all masters. See *Table 45* in *Chapter 8: Multi-Layer AHB Crossbar Switch (XBAR)* for details on master ports. Default after reset is 1 for all MAPn bits. 0 Access for the master is not enabled 1 Access for the master is enabled
KEY	Keyed Service Mode Default after reset is 0. 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 Default after reset is 1. 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 Default after reset is 0. 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 Default after reset is 0. 0 Generate a reset on a timeout 1 Generate an interrupt on an initial timeout, reset on a second consecutive timeout
HLK	Hard Lock This bit is only cleared at reset. Default after reset is 0. 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. Default after reset is 0. 0 SWT_CR, SWT_TO SWT_WN and SWT_SK are read/write registers if HLK = 0 1 SWT_CR, SWT_TO, SWT_WN and SWT_SK are read-only registers
CSL	Clock Selection Selects the clock that drives the internal timer. Default after reset is 1. 0 System clock 1 Oscillator clock
STP	Stop Mode Control Allows the watchdog timer to be stopped when the device enters stop mode. Default after reset is 0. 0 SWT counter continues to run in stop mode 1 SWT counter is stopped in stop mode

Table 192. SWT_CR field descriptions (continued)

Field	Description
FRZ	Debug Mode Control Allows the watchdog timer to be stopped when the device enters debug mode. Default after reset is 1. 0 SWT counter continues to run in debug mode 1 SWT counter is stopped in debug mode
WEN	Watchdog Enabled Default after reset is 1. 0 SWT is disabled 1 SWT is enabled

19.4.2.2 SWT Interrupt Register (SWT_IR)

The SWT_IR contains the timeout interrupt flag.

Offset 0x0004 Access: Read/Write

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

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

Figure 280. SWT Interrupt Register (SWT_IR)

Table 193. SWT_IR field descriptions

Field	Description
TIF	Timeout 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 timeout

19.4.2.3 SWT Timeout Register (SWT_TO)

The SWT_TO register contains the 32-bit timeout period.

Offset 0x008 Access: Read/Write(1)

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

Figure 281. SWT Timeout Register (SWT_TO)

1. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Table 194. SWT_TO field descriptions

Field	Description
WTO	Watchdog timeout 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.

19.4.2.4 SWT Window Register (SWT_WN)

The SWT_WN register contains the 32-bit window start value. This register is cleared on reset.

Offset 0x00C Access: Read/Write(1)

	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 282. SWT Window Register (SWT_WN)

1. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Table 195. SWT_WN field descriptions

Field	Description
WST	Window start value When window mode is enabled, the service sequence can only be written when the internal down counter is less than this value.

19.4.2.5 SWT Service Register (SWT_SR)

The SWT_SR is the target for service operation writes used to reset the watchdog timer.

Offset 0x010 Access: Read/Write

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

Figure 283. SWT Service Register (SWT_SR)

Table 196. 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 19.5: Functional description* for details. Otherwise, the sequence 0xA602 followed by 0xB480 is written to the WSC field. To clear the soft lock bit (SWT_CR[SLK]), the value 0xC520 followed by 0xD928 is written to the WSC field.

19.4.2.6 SWT Counter Output Register (SWT_CO)

The SWT_CO register is a read-only register that shows the value of the internal down counter when the SWT is disabled.

Offset 0x014 Access: Read Only

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

Figure 284. SWT Counter Output Register (SWT_CO)

Table 197. SWT_CO field descriptions

Field	Description
CNT	Watchdog Count When the watchdog is disabled (SWT_CR[WEN] = 0) this field shows the value of the internal down counter. When the watchdog is enabled the value of this field is 0x0000_0000. Values in this field can lag behind the internal counter value for 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.

19.4.2.7 SWT Service Key Register (SWT_SK)

The SWT_SK register holds the previous (or initial) service key value.

Offset 0x018 Access: Read/Write(1)

	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 285. SWT Service Register (SWT_SR)

1. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Table 198. SWT_SR field descriptions

Field	Description
SK	Service Key This field is the previous (or initial) service key value used in keyed service mode. If SWT_CR[KEY] is set, the next key value to be written to the SWT_SR is (17 * SK + 3) mod 2^{16}.

19.5 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 timeout 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 timeout period in clock cycles unless the value is less than 0x100 in which case the timeout period is set to 0x100. This timeout period is loaded into an internal 32-bit down counter when the SWT is enabled and each time a valid service operation is performed. The SWT_CR[CSL] bit selects which clock (system or oscillator) is used to drive the down counter. The reset value of the SWT_TO register is device specific.

The configuration of the SWT can be locked through use of either a soft lock or a hard lock. In either case, when locked the SWT_CR, SWT_TO, SWT_WN and SWT_SK registers are read-only. The hard lock is enabled by setting the SWT_CR[HLK] bit which can only be cleared by a reset. The soft lock is enabled by setting the SWT_CR[SLK] bit and is cleared by writing the unlock sequence to the service register. The unlock sequence is a write of 0xC520 followed by a write of 0xD928 to the SWT_SR[WSC] field. There is no timing requirement between the two writes. The unlock sequence logic ignores service sequence writes and recognizes the 0xC520, 0xD928 sequence regardless of previous writes. The unlock sequence can be written at any time and does not require the SWT_CR[WEN] bit to be set.

When enabled, the SWT requires periodic execution of a servicing operation which consists of writing two values to the SWT_SR. Writing the proper sequence of values loads the internal down counter with the timeout 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 *Equation 2* . 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.

Equation 2

$$SK_{n+1} = (17*SK_n+3) \bmod 2^{16}$$

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 timeout 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 timeout 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 timeout occurs. If the SWT_CR[ITR] bit is not set, a reset is generated immediately on a timeout. If the SWT_CR[ITR] bit is set, an initial timeout causes the SWT to generate an interrupt and load the down counter with the timeout period. If the service sequence is not written before the second consecutive timeout, the SWT generates a system reset. The interrupt is indicated by the timeout 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 timeout 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.

20 Boot Assist Module (BAM)

20.1 Overview

The Boot Assist Module (BAM) is a 4 KB block of read-only memory (ROM) containing the boot program code for this device.

The BAM program supports four different boot modes:

- Boot from internal Flash
- Serial boot via SCI or CAN interface
- Serial boot via SCI or CAN interface with baud rate detection

The BAM program is executed by the e200z335 core just after a device reset. Depending on the boot mode, the program initializes appropriate minimum device resources to start user code application.

20.2 Features

- Initial e200z335 core MMU setup with minimum address translation for all internal device resources
- MMU configuration to boot user application, compiled as Power Architecture code or as VLE code
- Passes control to user application code in the internal flash memory
- Automatic switch to Serial Boot mode if internal flash is blank or invalid
- Serial boot by loading user program via CAN bus or eSCI to the internal SRAM
  - User programmable 64-bit password protection
  - Optional automatic detection of the host SCI or CAN speed
- Boot from an external memory device, connected to the calibration bus
- Controls e200z335 core Watchdog Timer or/and the Software Watchdog Timer (SWT)

20.3 Modes of operation

20.3.1 Normal Mode

The BAM program is executed immediately following the negation of reset.

20.3.2 Debug Mode

The BAM program is not executed when the device comes out of reset in OnCE debug mode. The user must provide the required device initialization using the development tool before accessing the device resources.

20.3.3 Internal Boot Mode

This mode of operation is intended for systems that boot from internal flash memory. The internal flash memory is used for all code and all boot configuration data.

20.3.4 Serial Boot Mode

This mode of operation is intended to load a user program into internal SRAM, using either the eSCI or CAN serial interface, then to execute that program. That program can then be used to control the download of data and erasing/programming of the internal or external flash memory.

20.3.5 Calibration Bus Boot Mode

This boot mode is intended for parts packaged in CSPs.

20.4 Memory map

The BAM occupies 16 KB of memory space, 0xFFFF_C000 to 0xFFFF_FFFF. The actual code size of the BAM program is less than 4 KB and starts at 0xFFFF_F000, repeating itself down every 4 kilobytes in the BAM address space. The CPU starts the BAM program execution at its reset vector from address 0xFFFF_FFFC.

The BAM exits to the user code at 0xFFFF_FFF8 address. The last assembly-language instruction executed by the BAM is BLR (branch link register). The link register is preloaded with the user application start address. When booting from internal or external flash, the start address is taken from next to valid RCHW 32-bit word. When the device boots serially the start address is set according the serial boot protocol.

Table 199 shows the BAM address map.

Table 199. BAM memory map

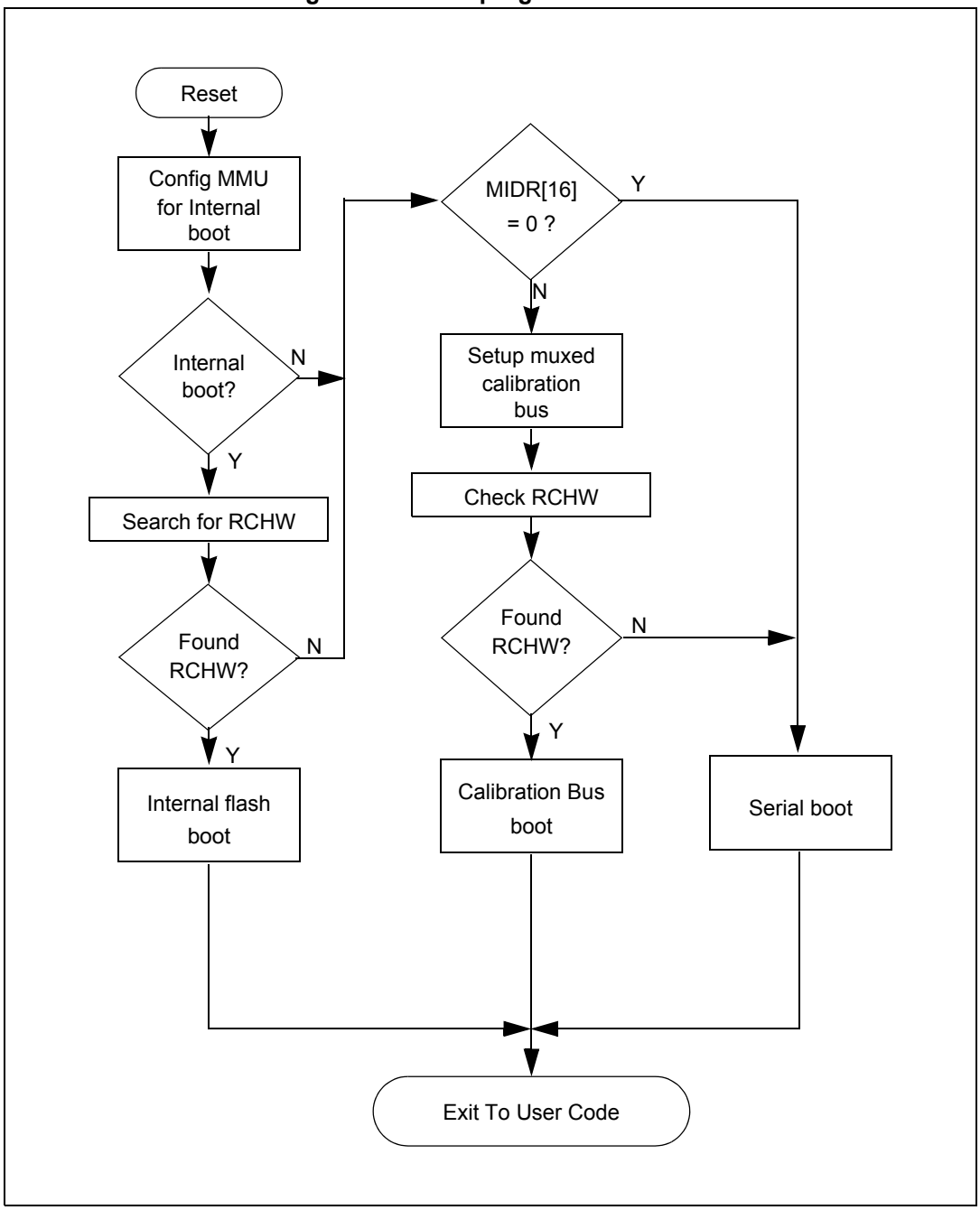
Address	Description
0xFFFF_C000–0xFFFF_EFFF	BAM program mirrored
0xFFFF_F000–0xFFFF_FFFF	BAM program
0xFFFF_FFFC	Device reset vector
0xFFFF_FFF8	BAM last executed instruction

20.5 Functional description

20.5.1 BAM program flow chart

The BAM program flow chart is shown in *Figure 286*.

Figure 286. BAM program flow chart



20.5.2 BAM program operation

The BAM is accessed by the device core after the negation of $\overline{\text{RSTOUT}}$, before user code starts. First, the BAM program configures the e200z335 core MMU to allow access to all device internal resources, according to *Table 200*. This MMU setup remains the same for internal Flash boot mode.

Table 200. MMU configuration for internal flash boot

TLB entry	Region	Logical base address	Physical base address	Size	Attributes
0	Peripheral Bridge B[1] and BAM	0xFFF0_0000	0xFFF0_0000	1 MB	Guarded Big endian Global PID
1	Internal Flash	0x0000_0000	0x0000_0000	16 MB	Not guarded Big endian Global PID
2	Calibration EBI	0x2000_0000	0x0000_0000	16 MB	Not guarded Big endian Global PID
3	Internal SRAM	0x4000_0000	0x4000_0000	256 KB	Not guarded Big endian Global PID
4	Peripheral Bridge A[1]	0xC3F0_0000	0xC3F0_0000	1 MB	Guarded Big endian Global PID

1. This device has only a single peripheral bridge, but to match the memory map of other devices the peripherals are mapped to appear as if they are on two different peripheral bridges.

The MMU regions are mapped with logical address the same as physical address except for the external calibration bus interface (EBI). The logical EBI address space is mapped to physical address space of the internal Flash memory. This allows code, written to run from external memory, to be executed from internal Flash.

After the MMU configuration, the BAM program checks the BOOTCFG field of the reset status register (SIU_RSR) and the appropriate boot sequence is started as shown in *Table 201*.

Depending on the values stored in the censorship word and serial boot control word in the shadow row of the internal Flash memory, the internal Flash memory can be enabled or disabled, the Nexus port can be enabled or disabled, the password received in the serial boot mode is compared with the fixed public password or compared to a user programmable password in the internal Flash memory.

Table 201. Boot modes

Boot mode name	BOOTCFG	Censorship control 0x00FF_FDE0	Serial boot control 0x00FF_FDE2	Internal flash state	Nexus state	Serial password
Internal - Censored	0	Any value except 0x55AA	Don't care	Enabled	Disabled	Flash
Internal - Public		0x55AA		Enabled	Enabled	Public
Serial - Flash Password	1	Don't care	0x55AA	Enabled	Disabled	Flash
Serial - Public Password			Any value except 0x55AA	Disabled	Enabled	Public

The censorship word is a 32-bit word of data stored in the shadow row of internal flash memory. This memory location is read and interpreted by hardware as part of the boot process and is used in conjunction with the BOOTCFG pin to enable/disable the internal flash memory and the Nexus interface. The address of the Censorship word is 0x00FF_FDE0. The censorship word consists of two fields: censorship control and serial boot control. The censorship word is programmed during manufacturing to be 0x55AA_55AA. This results in a device that is not censored and uses a Flash-based password for serial boot mode.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
Censorship Control - showing an uncensored part (Factory Default)															

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
Serial Boot Control - showing the use of the flash based password (Factory Default)															

Figure 287. Censorship Word

The BAM program uses the state of bit SIU_CCR[DISNEX] to determine whether the serial password received in serial boot mode, should be compared to a public password (fixed value of the 0xFEED_FACE_CAFE_BEEF) or needs to be compared to a Flash password - 64-bit data, stored in the shadow row of internal flash at address 0x00FF_FDD8. If the bit is set, the BAM uses the Flash serial password; if the bit is cleared, it uses the public password.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1
Serial Boot Password (0x00FF_FDD8) - 0xFEED (Factory Default)															

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0
Serial Boot Password (0x00FF_FDDA - 0xFACE (Factory Default)															

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0
Serial Boot Password (0x00FF_FDDC) - 0xCAFE (Factory Default)															

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1	0	1	1	1	1	1	0	1	1	1	0	1	1	1	1
Serial Boot Password (0x00FF_FDDF) - 0xBEEF (Factory Default)															

Figure 288. Serial Boot Flash Password

A valid serial password must be always programmed, regardless the boot mode used. This provides capability to “rescue” the part using the serial boot mode, if the flash content becomes corrupted for whatever reason.

20.5.3 Reset Configuration Half Word (RCHW)

The Reset Configuration Half Word defines boot options and has to be programmed by the user to predefined locations in the internal flash or at the beginning of the external flash device. The next 32-bit word after the RCHW has to be programmed with a starting address of the user application. The BAM program uses this location to fetch the address, where it passes control to.

Table 204 provides possible RCHW locations in the internal flash. When booting from the external flash device, the RCHW should reside in the very first 16-bit half word of the flash.

Figure 289 shows the fields of the RCHW.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				SWT	WTE	PS0	VLE	0	1	0	1	1	0	1	0
								Boot Identifier = 0x5A							

Figure 289. Reset Configuration Half Word

Table 202. RCHW field descriptions

Field	Description
bits 0–3	Reserved These bit values are ignored when the half-word is read. Write to 0 for future compatibility.
SWT	Watchdog timer enable This bit determines if the software watchdog timer is enabled after passing control to the user application code. 0 Disable software watchdog timer 1 Software watchdog timer maintains its default state out of reset, i.e. enabled. The timeout period is programmed to be 261600 system clocks.
WTE	Device core Watchdog timer enable This bit determines if the core software watchdog timer is enabled after passing control to the user application code. 0 Disable core software watchdog timer 1 Software watchdog timer maintains its default state out of reset, i.e. enabled. The timeout period is programmed to be 2.5×2^{17} system clocks.
PS0	Port size Defines the width of the data bus connected to the memory on $\overline{CS0}$. After system reset, $\overline{CS0}$ is changed to a 16-bit port by the BAM, which fetches the RCHW from either 16- or 32-bit external memories. Then the BAM reconfigures the EBI as a 16-bit bus or a 32-bit bus, according to the settings of this bit. 0 32-bit $\overline{CS0}$ port size 1 16-bit $\overline{CS0}$ port size **Note:** Used in cal bus boot mode only. Do not set the port to 32-bits if the device only has a 16-bit data bus.

Table 202. RCHW field descriptions (continued)

Field	Description
VLE	VLE Code Indicator This bit is used to configure the MMU entries 1–3 coded as either Power Architecture instructions or as VLE instructions. 0 User code executes as Power Architecture code 1 User code executes as VLE code
BOOTID	Boot identifier This field serves two functions. First, it is used to indicate which block in flash memory contains the boot program. Second, it identifies whether the flash memory is programmed or invalid. The value of a valid boot identifier is 0x5A.

The watchdog timeout periods, when the watchdogs are controlled by RCHW, are shown in *Table 203*.

Table 203. Watchdog timeouts

Crystal frequency (MHz)	Timeout (ms)	
	Core watchdog	Software watchdog timer
8	40.96	32.7
12	27.3	21.8
16	20.48	16.35
20	16.38	13.08

20.5.3.1 Reset boot vector

The boot vector, shown in *Figure 290*, has to be programmed by user to the user application code memory device (internal or external flash) to next 32-bit word after the RCHW. The value from this location is used by BAM program as a start address of the user application to switch to.

BOOT_BLOCK_ADDRESS + 0x0000_0004

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31

Figure 290. Reset boot vector

20.5.4 Internal Boot Mode

When the BAM program detects internal Flash boot mode, it sets up a machine check exception handler because it will be accessing Flash memory locations that may be corrupted and cause a bus error. Then the BAM program tries to find a valid RCHW in six

predefined locations. If a valid RCHW is not found, the BAM program proceeds to check of possibility of booting from the calibration bus and/or to the serial boot mode.

20.5.4.1 Finding Reset Configuration Half Word

The BAM searches the internal Flash memory for a valid reset configuration half word (RCHW). Possible RCHW locations are shown in *Table 204*.

Table 204. Possible RCHW locations in the internal flash

Block	Address
0	0x0000_0000
1	0x0000_4000
2	0x0001_0000
3	0x0001_C000
4	0x0002_0000
5	0x0003_0000

BOOT_BLOCK_ADDRESS is the address from *Table 204* where the BAM finds a valid RCHW. If the BAM program finds a valid RCHW, the core watchdog is enabled if the RCHW[WTE] bit is programmed high, the SWT is disabled if the RCHW[SWT] bit is programmed low, the BAM program fetches the reset vector from the address of the BOOT_BLOCK_ADDRESS + 0x4, and branches to the reset boot vector (shown in *Figure 290*). A user application should have a valid instruction at the reset boot vector address.

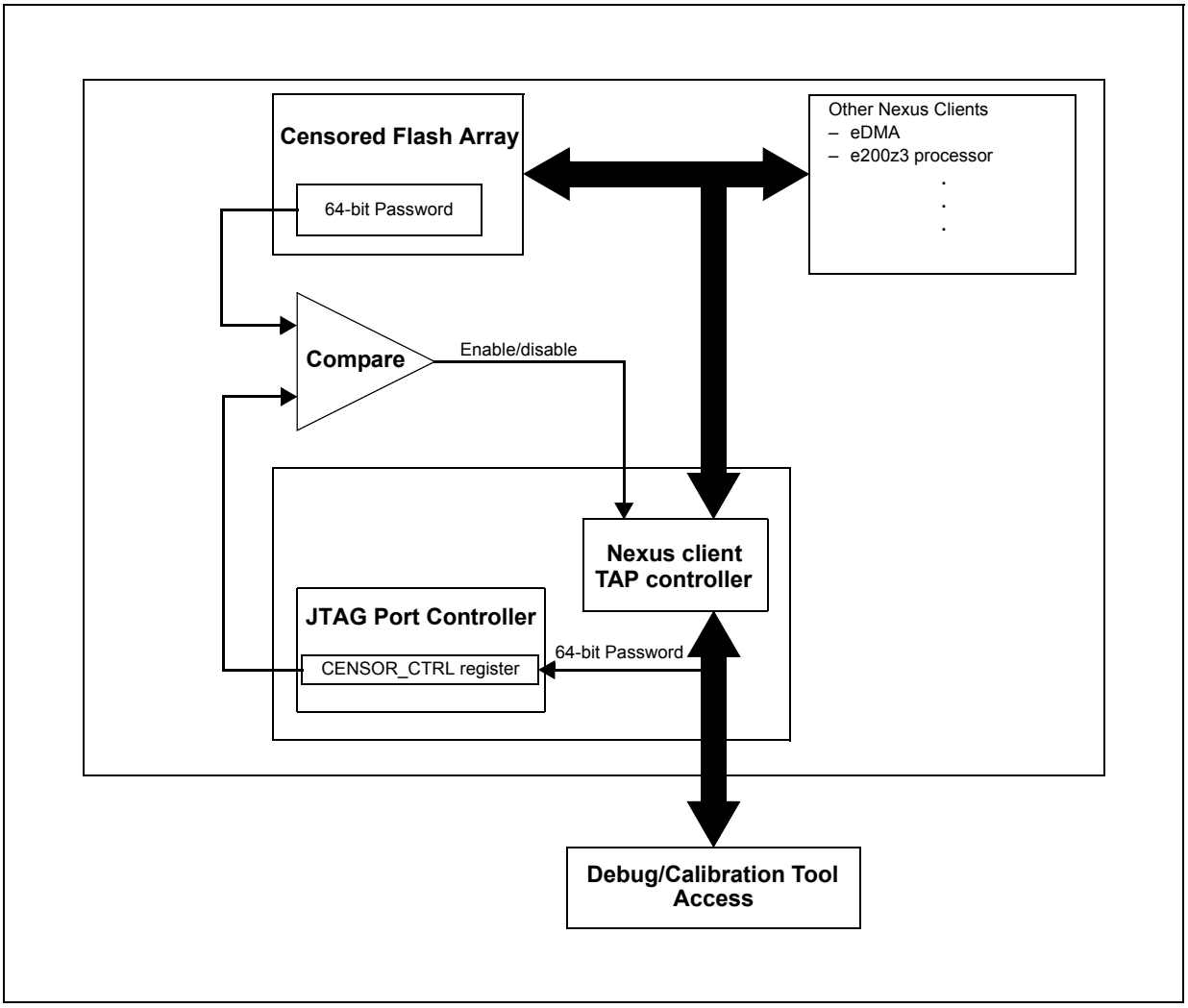
20.5.4.2 Enabling debug of a censored device

When a device is in a censored state, the debug port (JTAG/Nexus) is disabled and only JTAG BSDL commands can be used. Access to the Nexus/JTAG clients on a censored device requires inputting the proper password into the JTAG Censorship Control Register during reset.

Note: When the debug port is enabled on a censored device, it is enabled only until the next reset.

Figure 291 shows the logic that enables access to Nexus clients in a censored device using the JTAG port.

Figure 291. Enabling JTAG/Nexus port access on a censored device



The steps to enable the debug port on a censored device are as follows:

1. After the $\overline{\text{RSTOUT}}$ pin has is negated, hold the device in system reset state using a debugger or other tool.
2. While the device is being held in system reset state shift the 64-bit password into the CENSOR_CTRL register (see *Section 30.4.1.4: CENSOR_CTRL Register* in the device reference manual) via the JTAG port using the JTAG ENABLE_CENSOR_CTRL instruction. The JTAG serial password is compared against the serial boot flash password from the flash shadow block.
3. If there is a match the Nexus client TAP controller enters normal operation mode and the DISNEX flag in the SIU_CCR register is negated, indicating Nexus is enabled. Upon negation of reset the debug / calibration tool is able to access the device via NEXUS port and JTAG. If the JTAG serial password does not match the serial boot flash password or the serial boot flash password is an illegal password then the debug / calibration tool is not able to access the device.

   After the debug port is enabled, the tool can access the censored device and can erase and reprogram the shadow flash block in order to uncensor the device.

Note: If the shadow flash block is erased without reprogramming a new valid password before a reset it will contain an illegal password and the debug port will be inaccessible.

4. Subsequent resets will clear the JTAG censor password register and the Nexus client TAP controller will hold in reset again. Therefore, the tool must resend the JTAG serial password, as described above, in order to enable the Nexus client TAP controller again.

20.5.5 Serial Boot Mode

When the BAM program transitions to the Serial Boot mode, unused message buffers in CAN_A are used for the BAM program stack and variables, and the SWT watchdog is reprogrammed with a longer than default timeout period.

The MMU setup depends on how the BAM entered the serial boot mode. If a calibration bus boot was executed, the MMU is set up for that mode (see *Table 209*). If the part is not in a CSP package, the MMU setup matches *Table 200*.

The serial boot mode can run in either of two modes of operation:

- Standard serial boot mode using fixed baud rates derived from the crystal oscillator used
- Baud Rate Detection serial boot mode, which allows communication with adaptable speed, based on measured input signal

The Fixed Baud Rate mode or Baud Rate Detection mode are selected based on the state of the EVTO pin, recorded in the SIU_RSR[ABR] bit. If the bit is set, the Baud Rate Detection mode is selected if the bit is cleared, the Fixed Baud Rate is selected,

The SIU_RSR[ABR] bit reflects the inverted state of the EVTO pin, thus to select Baud Rate Detection mode, the EVTO pin needs to be driven low.

When the Fixed Baud Rate mode is selected, the BAM program configures the SCI_A_RX pin to be the input of the eSCI_A module, CAN_A_RX pin as an input and CAN_A_TX as an output of the CAN_A module.

When Baud Rate Detection Mode is selected, the BAM program configures SCI_A_RX and CAN_A_RX pins as GPI inputs for polling their state by the CPU.

The SCI and CAN controllers pin configuration summary is shown in *Table 205*.

Table 205. CAN/eSCI pin configuration for CAN/eSCI fixed baud rate boot modes

Pins	Reset function	Initial Serial Boot Mode		Serial Boot Mode after a valid CAN message received		Serial Boot Mode after a valid eSCI message received	
		Function	Pad configuration	Function	Pad configuration	Function	Pad configuration
CAN_A_TX	GPIO	CAN_A_TX	Push/Pull output, with medium slew rate	CAN_A_TX	Push/Pull output, with medium slew rate	GPIO	—
CAN_A_RX	GPIO	CAN_A_RX	Input with pull-up and hysteresis	CAN_A_RX	Input with pull-up and hysteresis	GPIO	—

Table 205. CAN/eSCI pin configuration for CAN/eSCI fixed baud rate boot modes (continued)

Pins	Reset function	Initial Serial Boot Mode		Serial Boot Mode after a valid CAN message received		Serial Boot Mode after a valid eSCI message received	
		Function	Pad configuration	Function	Pad configuration	Function	Pad configuration
SCI_A_TX	GPIO	GPIO	—	GPIO	—	SCI_A_TX	Push/Pull output, with medium slew rate
SCI_A_RX	GPIO	SCI_A_RX	Input with pull-up and hysteresis(1)	GPIO	—	SCI_A_RX	Input with pull-up and hysteresis

1. BAM disables the internal pull-up resistor during serial boot mode operation. Connect SCI_A_RX to an external 10K pull-up resistor to ensure that the pin is driven high to enable CAN boot mode.

The BAM configures the communication modules for reception with fixed baud rates as shown in *Table 206* and waits for data reception.

Table 206. Serial Boot Mode - baud rate & watchdog summary

Crystal frequency (MHz)	System clock frequency (MHz)	Desired eSCI baud rate (baud)	Actual eSCI baud rate (baud)	eSCI error (%)	CAN baud rate (baud)	Core Watchdog(1) timeout period (s)	SWT timeout period during serial boot (s)
f_{xtal}	$f_{sys} = f_{xtal}$	f_{sys} / 833.33	f_{sys} / 832	—	f_{sys} / 40	$2.5 \times 2^{27} / f_{sys}$	223696213 / f_{sys}
8	8	9600	9615.4	0.16	200K	42	27.96
12	12	14400	14423. 0	0.16	300K	28	18.64
16	16	19200	19230.8	0.16	400K	21	13.98
20	20	24000	24038.5	0.16	500K	16.8	11.18

1. The SWT is used as a watchdog during serial boot mode, but the core watchdog is enabled just before switching to the user application to provide compatibility with earlier parts.

If a message with 0x11 ID, containing 8 bytes, is received by the CAN controller first, the BAM program transitions to the Serial CAN Boot sub-mode, disabling eSCI, and reconfiguring SCI_A_RX pin to its reset state.

If a message from eSCI is received first, the BAM program transitions to the Serial SCI Boot sub-mode, disables CAN_A module and configures its pins to their reset state.

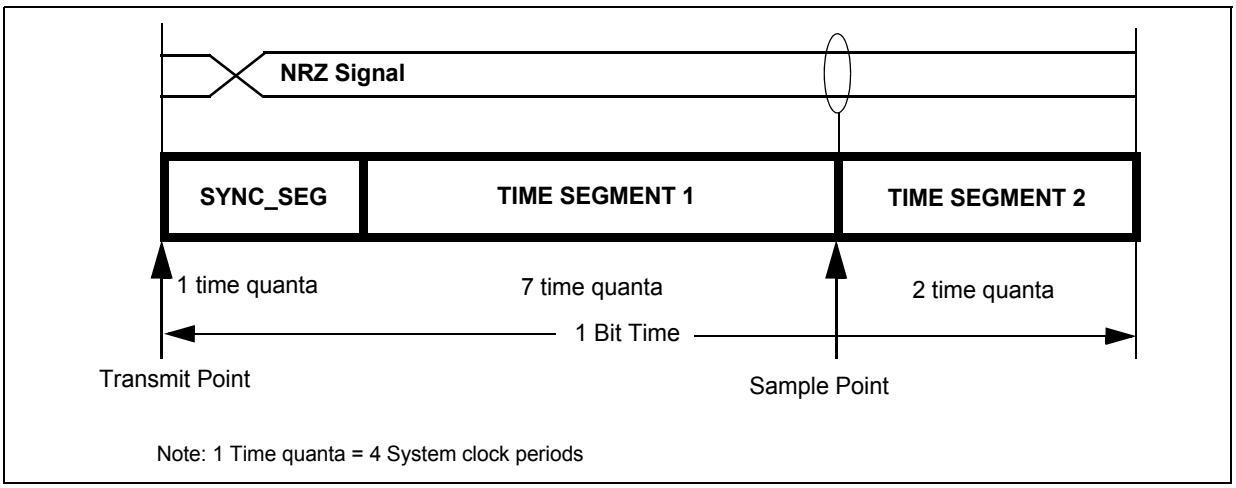
Then the BAM program transitions to the serial download protocol execution.

20.5.5.1 CAN controller configuration in the Fixed Baud Rate Mode

The CAN controller is configured to operate at a baud rate equal to system frequency divided by 40, using the standard 11-bit identifier format detailed in CAN 2.0A specification. See *Table 206* for examples of baud rates. Only one message buffer 0 is used for all communications.

The bit timing is configured as shown in *Figure 292*.

Figure 292. CAN bit timing



The BAM program ignores CAN errors and all received data is assumed to be good and is echoed out on the CAN_A_TX signal. It is the responsibility of the host computer to compare the “echoes” with the sent data and restart the process if an error is detected.

20.5.5.2 SCI Controller Configuration in Fixed Baud Rate Mode

The eSCI is configured for 1 start bit, 8 data bits, no parity, 1 stop bit and to operate at a baud rate equal to system clock divided by 832. See *Table 206* for examples of baud rates.

The BAM program ignores the eSCI errors: All data received will be assumed to be good and will be echoed out on the TXD signal. It is the responsibility of the host computer to compare the echoes with the sent data and restart the process if an error is detected.

20.5.5.3 Serial Boot Mode Download Protocol

The download protocol follows four steps:

1. Host sends 64-bit password.
2. Host sends start address, size of download code in bytes, and VLE bit.
3. Host sends the application code data.
4. The device switches to the loaded code at the start address.

The communication is done in half-duplex manner, any transmission from host is followed by the device transmission. The host computer should not send data until it receives echo from the device. All multi byte data structures have to be sent most significant byte (MSB) first.

When the CAN is used for serial download, the data is packed into standard CAN messages in the following manner:

- A message with 0x11 ID and 8-byte length is used to send the password. The device transmits the same data, but the message ID is set to 0x1.
- A message with 0x12 ID and 8-byte length is used to send the start address, length, and the VLE mode bit. The device transmits back the same data, but with ID set to 0x2.
- Messages with 0x13 ID are used to send the downloaded data. The device transmits back received data with message ID of 0x3.

When the SCI is used for serial download, the data has to be sent on a byte-by-byte basis. the device transmits back the received data.

20.5.5.4 Download protocol execution

The BAM program executes the serial boot as following:

1. Download 64-bit password.

   The received 8-byte password is checked for validity. For a password to be valid, none of its four 16-bit half words must equal 0x0000 or 0xFFFF.

   The BAM program then checks the censorship status of the device by checking the bit SIU_CCR[DISNEX]. If Nexus is disabled, the device is considered to be censored and the password is compared with a password stored in the shadow row in internal flash memory.

   If Nexus is enabled, the device is not considered to be censored and the password is compared to the fixed value = 0xFEED_FACE_CAFE_BEEF.

   If the password check fails, the device stops responding. To get the device out of that state, the RESET signal must be asserted.

   If the password check passes, the BAM transitions to the next step in the protocol.

2. Download start address, size of download, and VLE bit.

   The next 8 bytes received by the device are considered to contain a 32-bit start address, the VLE mode bit, and a 31-bit code length (see *Figure 293*).

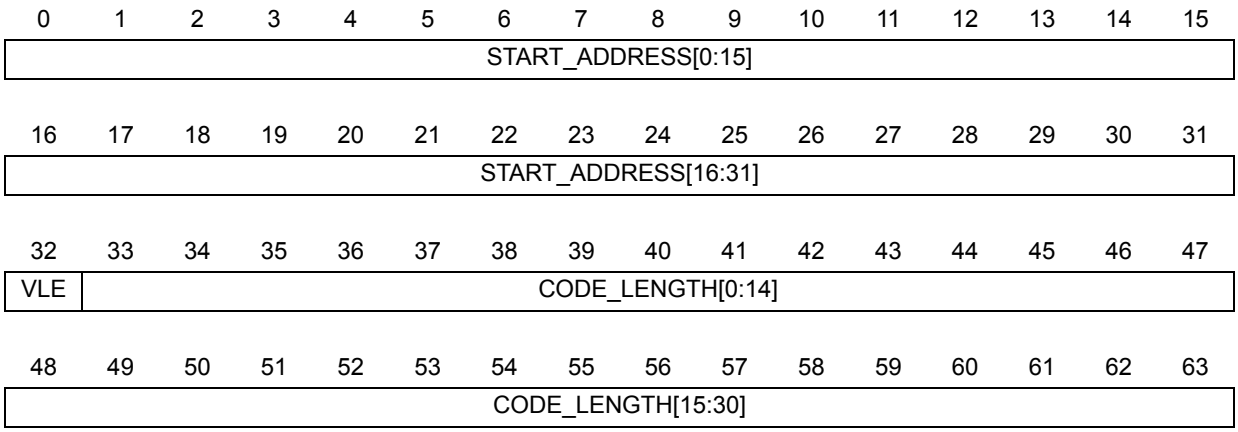


Figure 293. Start Address, VLE Bit and Download Size in Bytes

The start address defines where the received data will be stored and where the device will branch after the download is finished. The two least significant 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 to be loaded.

The VLE mode bit instructs the device to program MMU entries 1–3 with VLE attribute. If it is 1, the downloaded code must be compiled to VLE instructions, if it is 0 the code contains Power Architecture technology instructions.

3. Download data.

   Each byte of data received is stored in the device memory, starting at the address specified in the previous protocol step, and incrementing through memory until the

number of bytes of data received and stored in memory matches the number specified in the previous protocol step.

BAM program buffers incoming data, collecting up to eight bytes. The buffered data is written to the RAM with 64-bit writes to prevent ECC errors, which may happen if the device RAM is protected by 64-bit ECC code.

Once the buffered data is written to the RAM the BAM program refreshes the SWT watchdog

Note: Only system RAM supports 64-bit writes; therefore, attempting to download data to other RAM apart from system RAM will cause errors.

If the start address of the downloaded data is not on an 8-byte boundary, the BAM will write 0x0 to the memory locations from the proceeding 8-byte boundary to the start address (maximum 4 bytes). The BAM also writes 0x0 to all memory locations from the last byte of data downloaded to the following 8 byte boundary (maximum 7 bytes) and additional 8 zero bytes to prevent possible ECC errors may be caused by the CPU prefetching.

BAM writes an additional two zero double word to the system RAM, after loading code, to prevent possible ECC errors, which can happen due to the CPU speculatively pre-fetching data after the last loaded instruction, where the RAM cannot be initialized. The last loaded code address must not exceed 0x4003_FFF0 (the upper allowed RAM address by MMU settings minus two zero double words, written by BAM at the end of code download).

4. Switch to the loaded code.

   The BAM program waits for the last echo message transmission to complete, then the active communication controller is disabled. Its pins revert to GPIO inputs.

   To provide compatibility with older devices, the BAM writes the e200z335 core time base registers (TBU and TBL) with 0x0 and enables the e200z335 core watchdog to cause a reset after a timeout period of 2.5×2^{27} system clock cycles and disables SWT watchdog. See *Table 206* for examples of time out periods.

   The BAM code passes control to the loaded code at start address, which was received in step 2 of the protocol.

Note: The loaded code must periodically refresh the core watchdog timer or change the timeout period to a value that will not cause resets during normal operation.

20.5.5.5 Baud rate detection procedure

To improve baud rate detection accuracy the baud rate detection routine is copied to the beginning of the system RAM from the BAM ROM. Then the CPU branches to the RAM.

The device configures the CAN_A_RX and SCI_A_RX pins as general purpose inputs and starts to poll them until one of them goes low.

If the CAN_A_RX pin transitions first, the BAM program starts CAN baud rate detection routine, ignoring SCI_A_RX. After detecting the CAN baud rate, the BAM program transitions to the CAN download protocol routine described above.

If the SCI_A_RX pin transitions first, the SCI baud rate detection and download protocol routines are called, ignoring any further CAN pins activity.

20.5.5.5.1 SCI baud rate detection

The host has to send a zero byte to allow the device to detect the serial link baud rate. The host transmits 1 start bit, 8 zero data bits and 1 stop bit. The device does not echo it.

The device polls the SCI_A_RX pin for high to low transition and starts the core Time Base counter (TBU and TBL). Then the device polls for low to high transition on the pin and when it happens, the device turns off the TB counter. The TB content is used to calculate incoming signal baud rate. The SCI baud rate is equal to the TB content divided by 144 (measured over 9 bits with 16 system clocks per bit).

20.5.5.6 CAN baud rate detection

The host transmits a zero length message with zero 11-bit ID and the device measures time over 40 bits, polling CAN_A_RX pin for high and low, according to the sent data. The device does not acknowledge this message.

The CAN baud rate depends on the number of quanta per bit and serial clock frequency, which is defined by a prescaler. The CAN baud rate detection routine selects these parameters to maximize number of quanta per bit and achieve minimum difference between measured value and duration of the 40 CAN bits, to be programmed with selected pair of the parameters.

The CAN controller can be programmed with 8 to 25 number of quanta per bit. The bit timing parameters, selected by the baud rate detection routine, are shown in *Table 207*. (See FlexCAN chapter for the parameters definition).

Table 207. Lookup table for CAN bit timing

Time quanta per bit	Time segment 1		Time segment 2	RJW
	PROPSEG	PSEG1	PSEG2	
8	1	3	3	2
9	2	3	3	2
10	3	3	3	2
11	4	3	3	2
12	3	4	4	2
13	4	4	4	3
14	5	4	4	3
15	6	4	4	4
16	7	4	4	3
17	8	4	4	3
18	7	5	5	4
19	8	5	5	4
20	7	6	6	4
21	8	6	6	4
22	7	6	6	4
23	8	6	6	4
24	7	7	7	4
25	8	7	7	4

Maximum and minimum speeds of the serial communication modules are defined by the device system frequency and shown in *Table 208*.

Table 208. Maximum and minimum detectable baud rates

$f_{sys} = f_{xtal}$ [MHz]	Max baud rate for CAN ($f_{sys}/8$)(1) [bit/s]	Min CAN baud rate ($f_{sys}/25/256$) [bit/s]	Max baud rate for SCI ($f_{sys}/160$) [bit/s	Min baud rate for SCI ($f_{sys}/16/2^{16}$) [bit/s]
8	1M	1250	50K	7.6
12	1M	1875	75K	11.5
16	1M	2500	100K	15.2
20	1M	3125	125K	19

1. Limited to 1 Mbit/s by CAN standard

20.5.6 Booting from the calibration bus

For devices packed in the chip scale packages (CSP), there is an option to boot from a memory device on the calibration bus. The BOOTCFG pin needs to be driven high, selecting serial boot mode and the user needs to connect a boot memory device with a programmed valid RCHW to the calibration bus.

The BAM program first checks that the device is in the CSP package, by reading the SIU_MIDR. If the device is in that package, the BAM program sets up the MMU entries for EBI and Internal Flash (see *Table 209*), EBI and Calibration bus pins and tries to read RCHW from logical address 0x2000_0000.

If the valid RCHW is read from that address, the BAM program reads user application code start address from logical address 0x2000_0004, parses RCHW, sets up watchdogs, updates EBI, SRAM and Internal Flash MMU entries, according the RCHW[VLE] bit and passes control to the user code.

The RCHW[PS0] bit has to be programmed to ‘1’, since the Calibration Bus does not support a 32-bit port size.

If no valid RCHW was read, BAM switches to the serial boot mode.

Table 209. MMU Configuration for Calibration Bus Boot and Serial Boot modes

TLB entry	Region	Logical base address	Physical base address	Size	Attributes
1	Internal Flash	0x0000_0000	0x2000_0000	16 MB	Not guarded Big endian Global PID
2	Calibration EBI	0x2000_0000	0x2000_0000	16 MB	Not guarded Big endian Global PID

20.5.6.1 EBI configuration for Calibration Bus Boot Mode

The BAM program sets up EBI related registers as shown in *Table 210*.

Table 210. Calibration bus/EBI register settings

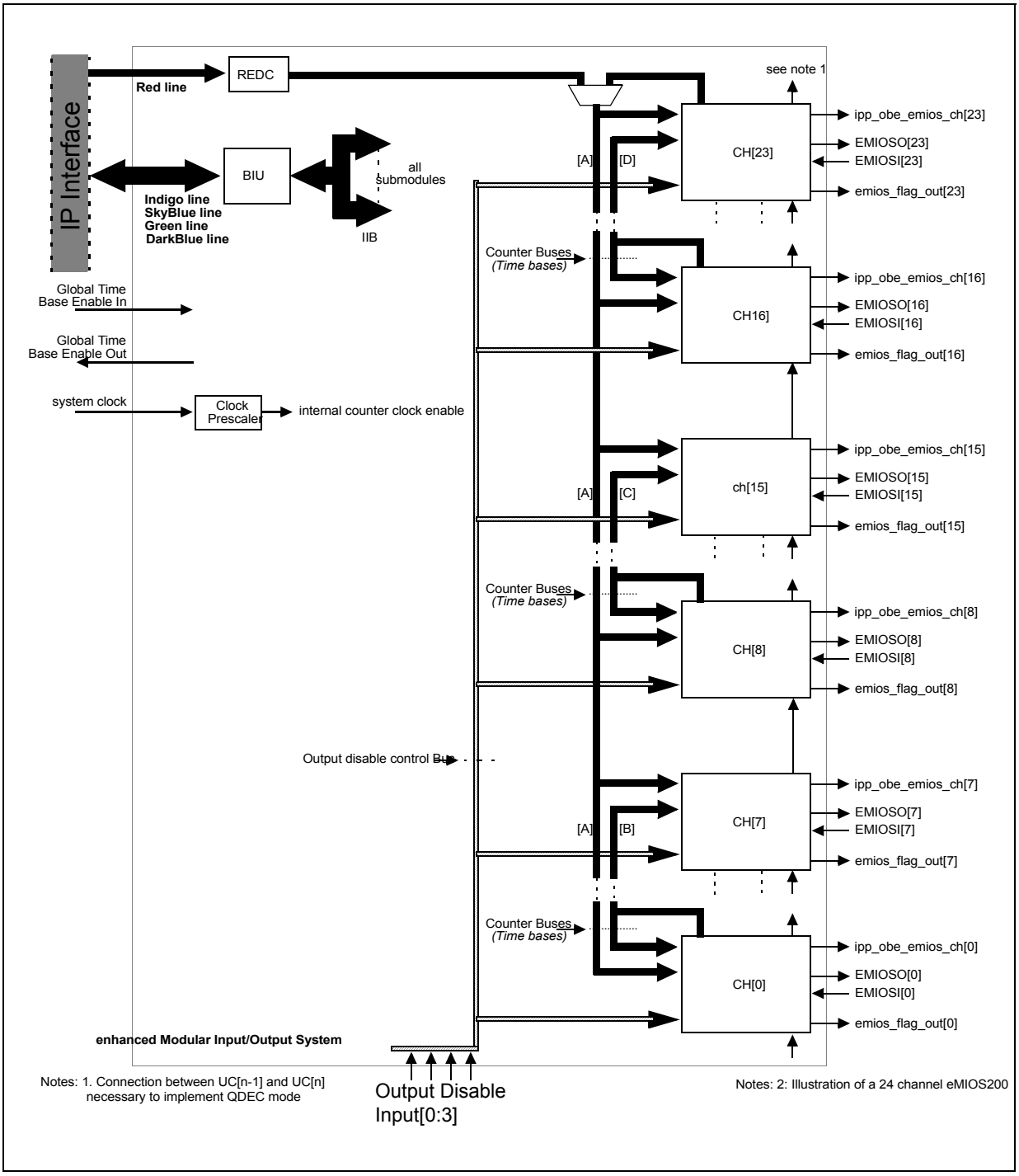
Register	Value	Comments
EBI_MCR	0x0000_0802	Sets the AD_MUX bit
EBI_CAL_BR0	0x2000_0883	Sets the AD_MUX bit, 16-bit wide bus, burst inhibit
EBI_CAL_OR0	0xFF80_00F0	Sets 15 wait states, 8 MB
SIU_PCR340 for CAL_ADDR[12:15]	0x0440	Set pads to 20 pF drive strength
SIU_PCR345 for CAL_ADDR[16:30]		
SIU_PCR341 for CAL_DATA[0:15]		
SIU_PCR336 for CAL_CS[0]		
SIU_PCR342 for CAL_WE[0], CAL_RD/WR & CAL_OE		
SIU_PCR343 for CAL_TS		Selects TS function, sets pads to 20 pF drive strength

21 Configurable Enhanced Modular IO Subsystem (eMIOS200)

21.1 Introduction

Figure 294 shows the block diagram of the configurable eMIOS200 block.

Figure 294. eMIOS200 block diagram



21.1.1 Overview

The configurable enhanced Modular Input/Output Subsystem (eMIOS200) provides functionality to generate or measure time events. It is the parameterized version of the eMIOS block keeping full functional backwards compatibility. Its overall architecture resembles that of the MIOS. The predecessor MIOS timer block provides a framework

where a set of sublocks with different timer functions are assembled to attend the specific needs of a SoC. The eMIOS200 builds on this concept by using a Unified Channel module that provides a superset of the functionality of all the individual MIOS channels, while providing a consistent user interface. This allows more flexibility as each Unified Channel can be programmed for different functions in different applications of the SoC. Besides that, eMIOS200 architecture allows using Dedicated Channels which perform specific functions not included in MIOS inheritance.

21.1.2 Features

The basic features of the eMIOS200 are the following:

- Up to 32 channels chosen among Unified or Dedicated Channels not necessarily numbered in a continuous sequence (this document shows a 24 Unified Channels implementation)
- Data registers of either 8-, 16-, 24- or 32-bit width (this document shows 24-bit wide data registers)
- Counter buses B, C, D, E can be driven by Unified Channel 0, 8, 16, 24 respectively
- Counter bus A can be driven by the Unified Channel #23 or by the external shared timer bus (Red-Line Client)
- Each channel has its own time base, alternative to the counter buses
- 1 global prescaler
- 1 prescaler per channel (CP)
- Shared timebases through the counter buses
- 1 STAC bus Client (REDC)
- Control and Status bits grouped in a single register
- Synchronization among timebases
- Shadow FLAG register
- State of the UC can be frozen for debug purposes
- Motor control capability

21.1.3 Modes of operation

The Unified Channels can be configured to operate in the following modes:

- General purpose input/output
- Single Action Input Capture
- Single Action Output Compare
- Input Pulse Width Measurement
- Input Period Measurement
- Double Action Output Compare
- Pulse/Edge Accumulation
- Pulse/Edge Counting
- Quadrature Decode
- Windowed Programmable Time Accumulation
- Modulus Counter
- Modulus Counter Buffered
- Output Pulse Width and Frequency Modulation
- Output Pulse Width and Frequency Modulation Buffered
- Center Aligned Output Pulse Width Modulation with dead time insertion
- Center Aligned Output Pulse Width Modulation with dead time insertion Buffered
- Output Pulse Width Modulation
- Output Pulse Width Modulation Buffered
- Output Pulse Width Modulation with Trigger

These modes are described in *Section 21.4.1.1: UC modes of operation*.

Each channel can have a specific set of modes implemented, according to devices requirements.

If an unimplemented mode is selected the results are unpredictable such as writing a reserved value to MODE[0:6] in *Section 21.3.2.8: eMIOS200 UC Control Register (EMIOS_CCR[n])*.

21.2 External signal description

21.2.1 Overview

Each channel has one external input and one external output signal, as described in *Figure 211*. Depending on the chip integration, the input and output signals can be connected to two separate pins, or to a single bidirectional pin.

21.2.2 Detailed signal descriptions

Table 211. External signals

Signal	Direction	Function	Reset state	Pull up
emiosi[n]	Input	eMIOS200 Channel n input	—	Chip dependent
emioso[n]	Output	eMIOS200 Channel n output	0/ Hi-Z(1)	Chip dependent

Table 211. External signals (continued)

Signal	Direction	Function	Reset state	Pull up
emios_flag_out[n]	Output	eMIOS200 Channel n flag	0	Chip dependent

1. Value “0” refers to the reset value of the signal. Hi-Z refers to the state of the external pin if a tristate output buffer is controlled by the corresponding ipp_obe_emios_ch[n] signal.

21.2.2.1 emiosi[n] - eMIOS200 Channel Input Signal

emiosi[n] is synchronized and filtered by the input programmable filter (IPF). The output of the IPF is then used by the channel logic and is available to be read by the MCU through the UCIN bit of the EMIOS_CSR[n] register.

21.2.2.2 emioso[n] - eMIOS200 Channel Output Signal

emioso[n] is a registered output and is available for reading by the MCU through the UCOUT bit of the EMIOS_CSR[n] register. Whilst the channel is operating in input modes the signal state is unknown.

21.2.2.3 emios_flag_out[n] - eMIOS200 Channel Flag Signal

emios_flag_out[n] outputs the state of F[n] bit of EMIOS_GFR register.

21.3 Memory map/register definition

21.3.1 Memory map

The overall address map organization is shown in *Table 212*.

Whenever an access to either an absent register or absent channel is performed the eMIOS200 responds asserting Transfer Error signal from the slave bus, as well as for access to reserved address.

Table 212. eMIOS200 memory map

eMIOS200[n] base address	Description	Location
0x000 0x003	eMIOS200 Module Configuration Register (EMIOS_MCR)	*on page 559*
0x004 0x007	eMIOS200 Global FLAG Register (EMIOS_GFR)	*on page 561*
0x008 0x00B	eMIOS200 Output Update Disable (EMIOS_OUDR)	*on page 562*
0x00C 0x00F	eMIOS200 Disable Channel (EMIOS_UCDIS)	*on page 563*
0x010 0x01F	Reserved	—
0x020 0x11F	Channel [0](1) to Channel [7]	

Table 212. eMIOS200 memory map (continued)

eMIOS200[n] base address	Description	Location
0x120 0x21F	Channel [8](1) to Channel [15]	
0x220 0x31F	Channel [16](1) to Channel [23](1)	
0x320 0xFFF	Reserved	—

1. It is recommended to allocate Unified Channels at slots 0, 8, 16, 23, 24 as these slots drive internal time buses.

21.3.1.1 Unified Channel memory map

Addresses of Unified Channel registers are specified as offsets from the channel's base address, otherwise the eMIOS200 base address is used as reference.

Table 213 describes the Unified Channel memory map.

Table 213. Unified Channel memory map

UC[n] base address	Description	Location
0x00	eMIOS200 UC A Register (EMIOS_CADR[n])	*on page 563*
0x04	eMIOS200 UC B Register (EMIOS_CBDR[n])	*on page 564*
0x08	eMIOS200 UC Counter Register (EMIOS_CCNTR[n])	*on page 565*
0x0C	eMIOS200 UC Control Register (EMIOS_CCR[n])	*on page 566*
0x10	eMIOS200 UC Status Register (EMIOS_CSR[n])	*on page 572*
0x14	eMIOS200 UC Alternate A Register (EMIOS_ALTA[n])	*on page 573*
0x18 – 0x1F	Reserved	—

21.3.2 Register descriptions

All control registers are 32-bit wide. This document illustrates the eMIOS200 with 24 Unified Channels and 24-bit wide data registers.

21.3.2.1 eMIOS200 Module Configuration Register (EMIOS_MCR)

EMIOS_MCR contains Global Control bits for the eMIOS200 block.

EMIOS_MCR address: eMIOS200 base address +0x00

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DOZEEN	MDIS	FRZ	GTBE	ETB	GPREN	0	0	0	0	0	0	SRV[0:3]			
W																
RESET:	0	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	GPRE[0:7]								0	0	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

Figure 295. eMIOS200 Module Configuration Register (EMIOS_MCR)

Table 214. EMIOS_MCR field descriptions

Field	Description
DOZEEN	Doze Enable bit Enable the eMIOS200 to enter low power mode when Doze Mode is requested at MCU level. The doze mode is used to stop the clock of the block, except the access to registers EMIOS_MCR, EMIOS_OUDR and EMIOS_UCDIS. 1: Enable to enter low power mode when Doze Mode is requested 0: Not enable to enter low power mode when Doze Mode is requested
MDIS	Module Disable bit MDIS - Module Disable bit Puts the eMIOS200 in low power mode. The MDIS bit is used to stop the clock of the block, except the access to registers EMIOS_MCR, EMIOS_OUDR and EMIOS_UCDIS. 1: Enter low power mode 0: Clock is running
FRZ	Freeze bit Enable the eMIOS200 to freeze the registers of the Unified Channels when Debug Mode is requested at MCU level. Each Unified Channel should have FREN bit set in order to enter freeze state. While in Freeze state, the eMIOS200 continues to operate to allow the MCU access to the Unified Channels registers. The Unified Channel will remain frozen until the FRZ bit is written to zero or the MCU exits Debug mode or the Unified Channel FREN bit is cleared. 1: Stops Unified Channels operation when in Debug mode and the FREN bit is set in the EMIOS_CCR[n] register 0: Exit freeze state
GTBE	Global Time Base Enable bit The GTBE bit is used to export a Global Time Base Enable from the module and provide a method to start time bases of several blocks simultaneously. 1: Global Time Base Enable Out signal asserted 0: Global Time Base Enable Out signal negated **Note:** The Global Time Base Enable input pin controls the internal counters. When asserted, Internal counters are enabled. When negated, Internal counters disabled.

Table 214. EMIOS_MCR field descriptions (continued)

Field	Description
ETB	External Time Base bit The ETB bit selects the time base source that drives counter bus[A]. 1: Counter bus[A] assigned to REDC 0: Counter bus[A] assigned to Unified Channel
GPREN	Global Prescaler Enable bit The GPREN bit enables the prescaler counter. 1: Prescaler enabled 0: Prescaler disabled (no clock) and prescaler counter is cleared SRV[0:3] — Server Time Slot bits The SRV[0:3] bits select the address of a specific Red Line server to which the REDC is assigned (refer to *Section 21.4.3: STAC Bus Client submodule (REDC)* for details).
GPRE[0:7]	Global Prescaler bits The GPRE[0:7] bits select the clock divider value for the global prescaler, as shown below. 00000000: 1 00000001: 2 00000010: 3 00000011: 4 . . . . 11111110: 255 11111111: 256

21.3.2.2 eMIOS200 Global FLAG Register (EMIOS_GFR)

The EMIOS_GFR is a read-only register that groups the FLAG bits from all channels. This organization improves interrupt handling on simpler devices. Each bit relates to one channel.

For Unified Channels these bits are mirrors of the FLAG bits in the EMIOS_CSR[n] register.

EMIOS_GFR address: eMIOS200 base address +0x04

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	F23	F22	F21	F20	F19	F18	F17	F16
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 296. eMIOS200 Global FLAG Register (EMIOS_GFR)

Table 215. EMIOS_GFR field descriptions

Field	Description
F[n]	Channel [n] Flag bit

21.3.2.3 eMIOS200 Output Update Disable (EMIOS_OUDR)

EMIOS_OUDR address: eMIOS200 base address +0x08

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	OU23	OU22	OU21	OU20	OU19	OU18	OU17	OU16
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OU15	OU14	OU13	OU12	OU11	OU10	OU9	OU8	OU7	OU6	OU5	OU4	OU3	OU2	OU1	OU0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 297. eMIOS200 Output Update Disable Register (EMIOS_OUDR)

Table 216. EMIOS_OUDR field descriptions

Field	Description
OU[n]	Channel [n] Output Update Disable bit When running MC, MCB or an output mode, values are written to registers A2 and B2. OU[n] bits are used to disable transfers from registers A2 to A1 and B2 to B1. Each bit controls one channel. 1 Transfers disabled 0 Transfer enabled. Depending on the operation mode, transfer may occur immediately or in the next period. Unless stated otherwise, transfer occurs immediately.

21.3.2.4 eMIOS200 Disable Channel (EMIOS_UCDIS)

EMIOS UCDIS address: eMIOS200 base address +0x0C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	CHDIS23	CHDIS22	CHDIS21	CHDIS20	CHDIS19	CHDIS18	CHDIS17	CHDIS16
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CHDIS15	CHDIS14	CHDIS13	CHDIS12	CHDIS11	CHDIS10	CHDIS9	CHDIS8	CHDIS7	CHDIS6	CHDIS5	CHDIS4	CHDIS3	CHDIS2	CHDIS1	CHDIS0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 298. eMIOS200 Enable Channel Register (EMIOS_UCDIS)

Table 217. EMIOS_UCDIS field descriptions

Field	Description
CHDIS[n]	Enable Channel [n] bit The CHDIS[n] bit is used to disable each of the channels by stopping its respective clock. 1 = Channel [n] disabled 0 = Channel [n] enabled

21.3.2.5 eMIOS200 UC A Register (EMIOS_CADR[n])

Depending on the mode of operation, internal registers A1 or A2, used for matches and captures, can be assigned to address EMIOS_CADR[n]. Both A1 and A2 are cleared by reset. *Table 218* summarizes the EMIOS_CADR[n] writing and reading accesses for all operation modes. For more information see *Section 21.4.1.1: UC modes of operation.*

EMIOS_CADR[n] address: UC[n] base address + 0x00

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	A[8:23]															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 299. eMIOS200 UC A Register (EMIOS_CADR[n])

21.3.2.6 eMIOS200 UC B Register (EMIOS_CBDR[n])

Depending on the mode of operation, internal registers B1 or B2 can be assigned to address EMIOS_CBDR[n]. Both B1 and B2 are cleared by reset. *Table 218* summarizes the EMIOS_CBDR[n] writing and reading accesses for all operation modes. For more information see *Section 21.4.1.1: UC modes of operation.*

Depending on the channel configuration it may have EMIOS_CBDR register or not. EMIOS_CBDR register is required for the following modes: DAOC, IPM, IPWM, OPWM, OPWMB, OPWFM, OPWMT, OPWFMB, OPWMC, OPWMCB, MC, MCB, PEA, PEC, WPTA. It means that if no mode requiring EMIOS_CBDR register is implemented then the register can be removed during synthesis through proper parameterization.

It is possible that, for particular reasons, EMIOS_CBDR be available in one device even if the respective channel does not feature any mode that requires it.

EMIOS_CBDR[n] address: UC[n] base address + 0x04

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	B[8:23]															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 300. eMIOS200 UC B register (EMIOS_CBDR[n])

Table 218. EMIOS_CADR[n], EMIOS_CBDR[n] and EMIOS_ALTA[n] values assignment

Operation mode	Register access					
	write	read	write	read	alt write	alt read
GPIO	A1, A2	A1	B1,B2	B1	A2	A2
SAIC(1)	—	A2	B2	B2	—	—
SAOC(1)	A2	A1	B2	B2	—	—
IPWM	—	A2	—	B1	—	—
IPM	—	A2	—	B1	—	—
DAOC	A2	A1	B2	B1	—	—
PEA	A1	A2	—	B1	—	—
PEC(1)	A1	A1	B1	B1	—	A2
QDEC(1)	A1	A1	B2	B2	—	—
WPTA	A1	A1	B1	B1	—	A2
MC(1)	A2	A1	B2	B2	—	—

Table 218. EMIOS_CADR[n], EMIOS_CBDR[n] and EMIOS_ALTA[n] values assignment

Operation mode	Register access					
	write	read	write	read	alt write	alt read
OPWFM	A2	A1	B2	B1	—	—
OPWMC	A2	A1	B2	B1	—	—
OPWM	A2	A1	B2	B1	—	—
OPWMT	A1	A1	B2	B1	A2	A2
MCB(1)	A2	A1	B2	B2	—	—
OPWFMB	A2	A1	B2	B1	—	—
OPWMCB	A2	A1	B2	B1	—	—
OPWMB	A2	A1	B2	B1	—	—

1. In these modes, the register EMIOS_CBDR[n] is not used, but B2 can be accessed.

21.3.2.7 eMIOS200 UC Counter Register (EMIOS_CCNTR[n])

The EMIOS_CCNTR[n] register contains the value of the internal counter. When GPIO mode is selected or the channel is frozen, the EMIOS_CCNTR[n] register is read/write. For all others modes, the EMIOS_CCNTR[n] is a read-only register. When entering some operation modes, this register is automatically cleared (refer to *Section 21.4.1.1: UC modes of operation* for details).

Depending on the channel configuration it may have EMIOS_CCNTR register or not. EMIOS_CCNTR register is required for the following modes: OPWFM, OPWFMB, OPWMC, OPWMCB, MC, MCB, PEA, PEC, WPTA, QDEC. It means that if no mode requiring EMIOS_CCNTR register is implemented then the register can be removed during synthesis through proper parameterization.

It is possible that, for particular reasons, EMIOS_CCNTR be available in one device even if the respective channel does not feature any mode that requires it.

EMIOS_CCNTR[n] address: UC[n] base address + 0x08

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	C[0:7]							
W(1)																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	C[8:23]															
W1																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or reserved

Figure 301. eMIOS200 UC Counter Register (EMIOS_CCNTR[n])

1. In GPIO mode or Freeze action, this register is writable.

21.3.2.8 eMIOS200 UC Control Register (EMIOS_CCR[n])

The Control register gathers bits reflecting the status of the UC input/output signals and the overflow condition of the internal counter, as well as several read/write control bits.

EMIOS_CCR[n] address: UC[n] base address + 0x0C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FREN	ODIS	ODISSL[0:1]		UCPRE[0:1]		UCPREN	DMA	0	IF[0:3]				FCK	FEN	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	BSL[0:1]		EDSEL	EDPOL	MODE[0:6]						
W			FORCMA	FORCMB												
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or reserved

Figure 302. eMIOS200 UC Control Register (EMIOS_CCR[n])

Table 219. EMIOS_CCR[n] field descriptions

Field	Description
FREN	Freeze Enable bit The FREN bit, if set and validated by FRZ bit in EMIOS_MCR register allows the channel to enter freeze state, freezing all registers values when in debug mode and allowing the MCU to perform debug functions. 1 Freeze UC registers values 0 Normal operation
ODIS	Output Disable bit The ODIS bit allows disabling the output pin when running any of the output modes with the exception of GPIO mode. 1 If the selected Output Disable Input signal is asserted, the output pin goes to EDPOL for OPWFMB and OPWMB modes and to the complement of EDPOL for other output modes, but the Unified Channel continues to operate normally, i.e., it continues to produce FLAG and matches. When the selected Output Disable Input signal is negated, the output pin operates normally 0 The output pin operates normally
ODISSL[0:1]	Output Disable select bits The ODISSL[0:1] bits select one of the four output disable input signals, as shown below. 00 Output Disable Input 0 01 Output Disable Input 1 10 Output Disable Input 2 11 Output Disable Input 3

Table 219. EMIOS_CCR[n] field descriptions (continued)

Field	Description
UCPRE[0:1]	Prescaler bits The UCPRE[0:1] bits select the clock divider value for the internal prescaler of Unified Channel, as shown below. 00 1 01 2 10 3 11 4
UCPREN	Prescaler Enable bit The UCPREN bit enables the prescaler counter. 1 Prescaler enabled 0 Prescaler disabled (no clock)
DMA	Direct Memory Access bit The DMA bit selects if the FLAG generation will be used as an interrupt or as a DMA request. 1 Flag/overrun assigned to DMA request 0 Flag/overrun assigned to Interrupt request
IF[0:3]	Input Filter bits The IF[0:3] bits control the programmable input filter, selecting the minimum input pulse width that can pass through the filter, as shown below. For output modes, these bits have no meaning. 0000 bypassed (The input signal is synchronized before arriving to the digital filter.) 0001 02 0010 04 0100 08 1000 16 all others reserved **Note:** Filter latency is 3 clock edges.
FCK	Filter Clock select bit The FCK bit selects the clock source for the programmable input filter. 1 main clock 0 prescaled clock
FEN	FLAG Enable bit The FEN bit allows the Unified Channel FLAG bit to generate an interrupt signal or a DMA request signal (The type of signal to be generated is defined by the DMA bit). 1 = Enable (FLAG will generate an interrupt or DMA request) 0 = Disable (FLAG does not generate an interrupt or DMA request)
FORCMA	Force Match A bit For output modes, the FORCMA bit is equivalent to a successful comparison on comparator A (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operation mode which uses comparator A, otherwise it has no effect. 1 Force a match at comparator A 0 Has no effect **Note:** For input modes, the FORCMA bit is not used and writing to it has no effect.

Table 219. EMIOS_CCR[n] field descriptions (continued)

Field	Description
FORCMB	Force Match B bit For output modes, the FORCMB bit is equivalent to a successful comparison on comparator B (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operation mode which uses comparator B, otherwise it has no effect. 1 Force a match at comparator B 0 Has not effect **Note:** For input modes, the FORCMB bit is not used and writing to it has no effect.
BSL[0:1]	Bus Select bits The BSL[0:1] bits are used to select either one of the counter buses or the internal counter to be used by the Unified Channel. Refer below for details. 00 All channels: counter bus[A] 01 Channels 0 to 7: counter bus[B] Channels 8 to 15: counter bus[C] Channels 16 to 23: counter bus[D] 10 reserved 11 All channels: internal counter
EDSEL	Edge Selection bit For input modes, the EDSEL bit selects whether the internal counter is triggered by both edges of a pulse or just by a single edge as defined by the EDPOL bit. When not shown in the mode of operation description, this bit has no effect. 1 Both edges triggering 0 Single edge triggering defined by the EDPOL bit For GPIO in mode, the EDSEL bit selects if a FLAG can be generated. 1 No FLAG is generated 0 A FLAG is generated as defined by the EDPOL bit For SAOC mode, the EDSEL bit selects the behavior of the output flip-flop at each match. 1 The output flip-flop is toggled 0 The EDPOL value is transferred to the output flip-flop

Table 219. EMIOS_CCR[n] field descriptions (continued)

Field	Description
EDPOL	Edge Polarity bit For input modes (except QDEC mode), the EDPOL bit asserts which edge triggers either the internal counter or an input capture or a FLAG. When not shown in the mode of operation description, this bit has no effect. 1 Trigger on a rising edge 0 Trigger on a falling edge For QDEC (MODE[6] cleared), the EDPOL bit selects the count direction according to *direction* signal (UC[n] input). 1 counts up when UC[n] is asserted 0 counts down when UC[n] is asserted **Note:** UC[n-1] EDPOL bit selects which edge clocks the internal counter of UC[n] 1 Trigger on a rising edge 0 Trigger on a falling edge For QDEC (MODE[6] set), the EDPOL bit selects the count direction according to the phase difference. 1 internal counter increments if phase_A is ahead phase_B signal 0 internal counter decrements if phase_A is ahead phase_B signal **Note:** In order to operate properly, EDPOL bit must contain the same value in UC[n] and UC[n-1] For output modes, the EDPOL bit is used to select the logic level on the output pin. 1 A match on comparator A sets the output flip-flop, while a match on comparator B clears it 0 A match on comparator A clears the output flip-flop, while a match on comparator B sets it

Table 219. EMIOS_CCR[n] field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>MODE[0:6]</td><td>Mode selection bits
The MODE[0:6] bits select the mode of operation of the Unified Channel, as shown below.

<table>
<tr><th>MODE[0:6] (1)</th><th>mode of operation</th></tr>
<tr><td>0000000</td><td>General purpose Input/Output mode (input)</td></tr>
<tr><td>0000001</td><td>General purpose Input/Output mode (output)</td></tr>
<tr><td>0000010</td><td>Single Action Input Capture</td></tr>
<tr><td>0000011</td><td>Single Action Output Compare</td></tr>
<tr><td>0000100</td><td>Input Pulse Width Measurement</td></tr>
<tr><td>0000101</td><td>Input Period Measurement</td></tr>
<tr><td>0000110</td><td>Double Action Output compare (with FLAG set on B match)</td></tr>
<tr><td>0000111</td><td>Double Action Output compare (with FLAG set on both match)</td></tr>
<tr><td>0001000</td><td>Pulse/Edge Accumulation (continuous)</td></tr>
<tr><td>0001001</td><td>Pulse/Edge Accumulation (single shot)</td></tr>
<tr><td>0001010</td><td>Pulse/Edge Counting (continuous)</td></tr>
<tr><td>0001011</td><td>Pulse/Edge Counting (single shot)</td></tr>
<tr><td>0001100</td><td>Quadrature Decode (for count & direction encoders type)</td></tr>
<tr><td>0001101</td><td>Quadrature Decode (for phase_A & phase_B encoders type)</td></tr>
<tr><td>0001110</td><td>Windowed Programmable Time Accumulation</td></tr>
<tr><td>0001111</td><td>Reserved</td></tr>
<tr><td>001000b</td><td>Modulus Counter (Up counter with clear on match start)</td></tr>
<tr><td>001001b</td><td>Modulus Counter (Up counter with clear on match end)</td></tr>
<tr><td>00101bb</td><td>Modulus Counter (Up/Down counter)</td></tr>
<tr><td>00110b0</td><td>Output Pulse Width and Frequency Modulation (immediate update)</td></tr>
<tr><td>00110b1</td><td>Output Pulse Width and Frequency Modulation (next period update)</td></tr>
<tr><td>00111b0</td><td>Center Aligned Output Pulse Width Modulation (with trail edge dead-time)</td></tr>
<tr><td>00111b1</td><td>Center Aligned Output Pulse Width Modulation (with lead edge dead-time)</td></tr>
<tr><td>01000b0</td><td>Output Pulse Width Modulation (immediate update)</td></tr>
<tr><td>01000b1</td><td>Output Pulse Width Modulation (next period update)</td></tr>
</table>
1. b = adjust parameters for the mode of operation. Refer to Section 21.4.1.1: UC modes of operation for details.

Note: If a reserved value is written to mode the results are unpredictable.</td></tr>
</table>

Table 219. EMIOS_CCR[n] field descriptions (continued)

Field	Description
MODE[0:6] (cont.)	

MODE[0:6] (1)	mode of operation
0100100 through 0100101	Reserved
0100110	Output Pulse Width Modulation with Trigger
0100111 through 1001111	Reserved
101000b	Modulus Counter Buffered (Up counter)
101001b	Reserved
10101bb	Modulus Counter Buffered (Up/Down counter)
10110b0	Output Pulse Width and Frequency Modulation Buffered
10110b1	Reserved
10111b0	Center Aligned Output Pulse Width Modulation Buffered (with trail edge dead-time)
10111b1	Center Aligned Output Pulse Width Modulation Buffered (with lead edge dead-time)
11000b0	Output Pulse Width Modulation Buffered
1100001 through 1111111	Reserved

1. b = adjust parameters for the mode of operation. Refer to *Section 21.4.1.1: UC modes of operation* for details.

Note: If a reserved value is written to mode the results are unpredictable.

21.3.2.9 eMIOS200 UC Status Register (EMIOS_CSR[n])

EMIOS_CSR[n] address: UC[n] base address + 0x10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OVR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	OVRC															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVFL	0	0	0	0	0	0	0	0	0	0	0	0	UCIN	UCOUT	FLAG
W	OVFLC															FLAGC
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or reserved

Figure 303. eMIOS200 UC Status Register (EMIOS_CSR[n])

Table 220. EMIOS_CSR[n] field descriptions

Field	Description
OVR	Overrun bit The OVR bit indicates that FLAG generation occurred when the FLAG bit was already set. 1 Overrun has occurred 0 Overrun has not occurred
OVRC	Overrun Clear bit The OVR bit can be cleared either by clearing the FLAG bit or by writing a 1 to the OVRC bit. 1 Clear OVR bit 0 Do not change OVR bit
OVFL	Overflow bit The OVFL bit indicates that an overflow has occurred in the internal counter. OVFL must be cleared by software writing a 1 to the OVFLC bit. 1 An overflow had occurred 0 No overflow
OVFLC	Overflow Clear bit The OVFL bit must be cleared by writing a 1 to the OVFLC. 1 Clear OVFL bit 0 Do not change OVFL bit
UCIN	Unified Channel Input pin bit The UCIN bit reflects the input pin state after being filtered and synchronized.
UCOUT	Unified Channel Output pin bit The UCOUT bit reflects the output pin state.

Table 220. EMIOS_CSR[n] field descriptions (continued)

Field	Description
FLAG	FLAG bit The FLAG bit is set when an input capture or a match event in the comparators occurred. 1 FLAG set event has occurred 0 FLAG cleared **Note:** emios_flag_out reflects the FLAG bit value. When DMA bit is set, the FLAG bit can be cleared by the DMA controller.
FLAGC	FLAG Clear bit The FLAG bit must be cleared by writing a 1 to FLAGC. 1 Clear FLAG bit 0 Do not change FLAG bit

21.3.2.10 eMIOS200 UC Alternate A Register (EMIOS_ALTA[n])

The EMIOS_ALTA[n] register provides an alternate address to access A2 channel registers in restricted modes (GPIO, PEC, WPTA, OPWMT) only. If EMIOS_CADR[n] register is used along with EMIOS_ALTA[n], both A1 and A2 registers can be accessed in these modes. *Figure 218* summarizes the EMIOS_ALTA[n] writing and reading accesses for all operation modes. Please see *Section 21.4.1.1.1: General purpose Input/Output (GPIO) Mode*, *Section 21.4.1.1.8: Pulse/Edge Counting (PEC) Mode*, *Section 21.4.1.1.10: Windowed Programmable Time Accumulation (WPTA) Mode*, and *Section 21.4.1.1.19: Output Pulse Width Modulation with Trigger (OPWMT) Mode* for a more detailed description of the use of EMIOS_ALTA[n] register.

EMIOS_ALTA[n] address: UC[n] base address + 0x14

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

= Unimplemented or Reserved

Figure 304. eMIOS200 UC Alternate A register (EMIOS_ALTA[n])

21.4 Functional description

The eMIOS200 provides independent channels that can be configured and accessed by a host MCU. Up to four time bases can be shared by the channels through four counter buses and each channel can generate its own time base. Optionally one of the counter buses can be driven by an external time base imported through the Red-Line interface.

The eMIOS200 module is based on a multi-bus timer architecture in which several timer channels are used to drive counter buses that are shared among the channels. There are 5 counter buses in the module: one global counter bus, shared by all channels and 4 local counter buses, each one dedicated to a slice of 8 channels. Counter bus A is referred to as the global counter bus. Counter buses B,C,D and E are the local counter buses.

The eMIOS200 counter buses are driven by channels in specific locations. The global counter bus is driven by the channel in channel slot [23]. Counter buses B,C,D and E are driven by channels in slots [0],[8],[16] and [24] respectively. Counter bus A drives all channels. Counter bus B drives channels in slots from [0] through [7]. Counter bus C drives channels in slots from [8] through [15]. Counter bus D drives channels in slots from [16] through [23]. Counter bus E drives channels from [24] through [31]. Note that the first channel in a 8-channel slice drives the local counter bus for that slice, therefore this channel should not be assigned to be driven by the same counter bus, otherwise a loop occurs. The eMIOS200 Interrupt request signal, DMA transfer request signal among others, are wired to a specific channel, thus the chip integrator should connect those signals having the eMIOS200 channel configuration in mind.

The eMIOS200 block is reset asynchronously. All registers are cleared on reset.

Figure 305 describes an eMIOS200 block configured with 32 Unified Channels. Note that the STAC bus is also present. Note also that independent of the configuration the channels are fixed in their slots, thus for example if channel [2] is not required this location will be empty, meaning that the other channels locations are not affected. In this case the application software should not access any register located in the channel[2] memory. Any attempt to access those registers will return no meaningful data and a Transfer Error will be generated.

Figure 305. eMIOS200 Full Channel Configuration using Unified Channels only

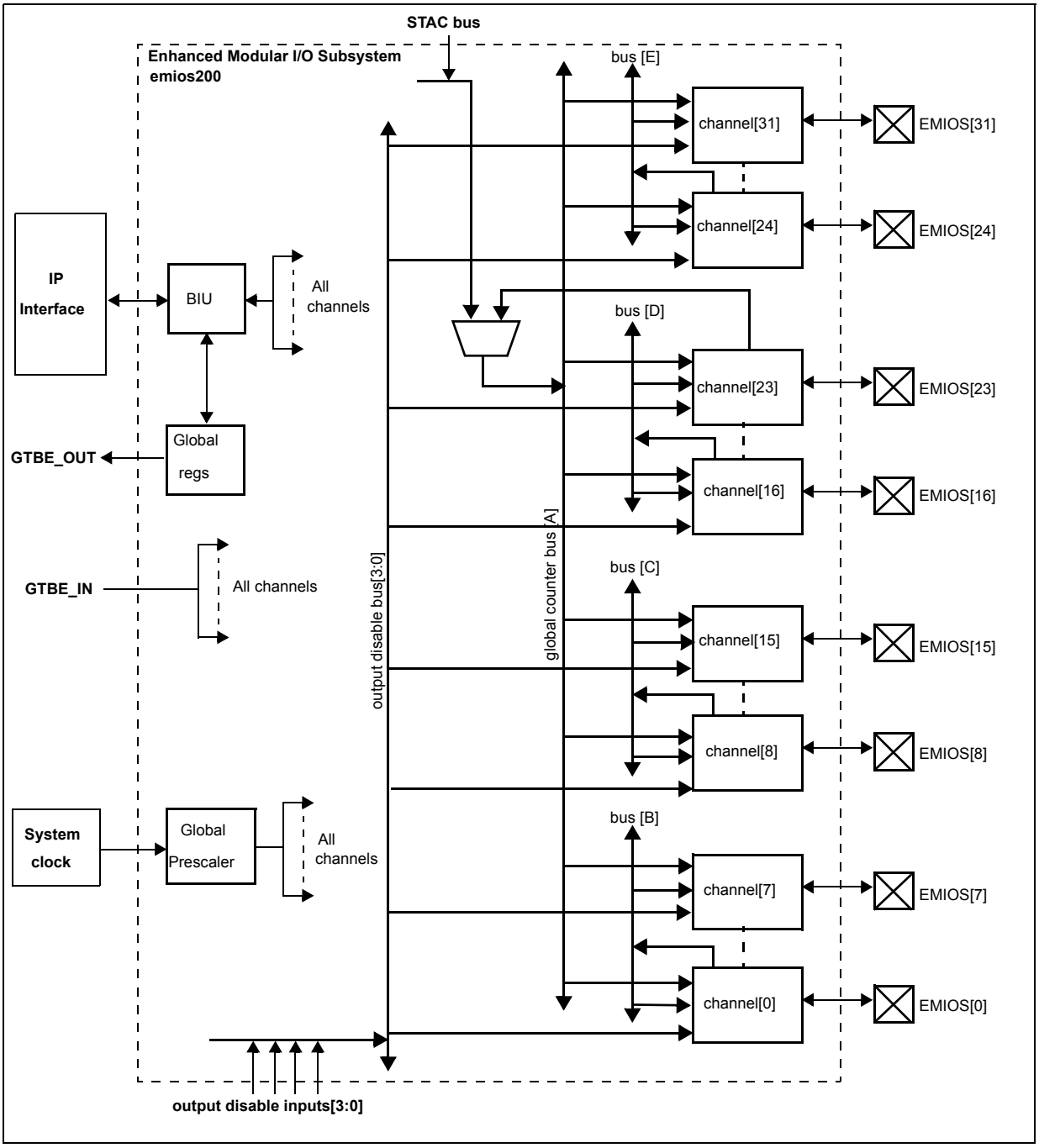
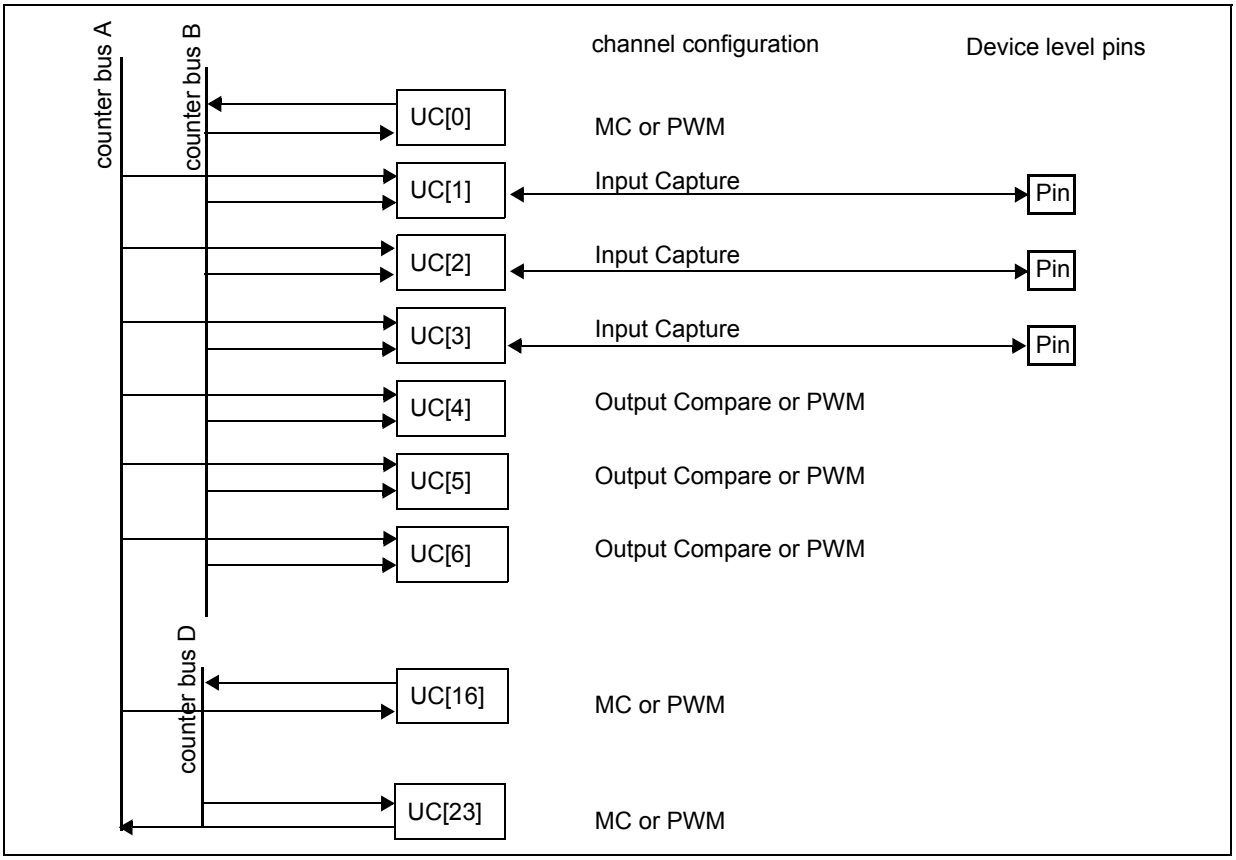


Figure 306 shows an eMIOS200 block. In this example configuration each one of the channels can be driven by the respective local bus, B for the [7:0] slice and D for the [23:16] slice, or by the global counter bus A. Note that there are unified Channels with no external pin associated to it. Those channels are used to trigger internal SoC signals such as Interrupt requests or DMA transfer requests. Since the channel Flags are also connected to the block top level pins they can be used as trigger signals for other blocks at chip level.

Figure 306. eMIOS200 example configuration



21.4.1 Unified Channel (UC)

Figure 307 shows the Unified Channel block diagram. Each Unified Channel consists of:

- Counter bus selector, which selects the time base to be used by the channel for all timing functions
- A programmable clock prescaler
- Two double buffered data registers A and B that allow up to two input capture and/or output compare events to occur before software intervention is needed.
- Two comparators (equal only) A and B, which compares the selected counter bus with the value in the data registers
- Internal counter, which can be used as a local time base or to count input events
- Programmable input filter, which ensures that only valid pin transitions are received by channel
- Programmable input edge detector, which detects the rising, falling or either edges
- An output flip-flop, which holds the logic level to be applied to the output pin
- eMIOS200 Status and Control register
- An Output Disable Input selector, which selects the Output Disable Input signal that will be used as output disable

Figure 307. Unified Channel block diagram

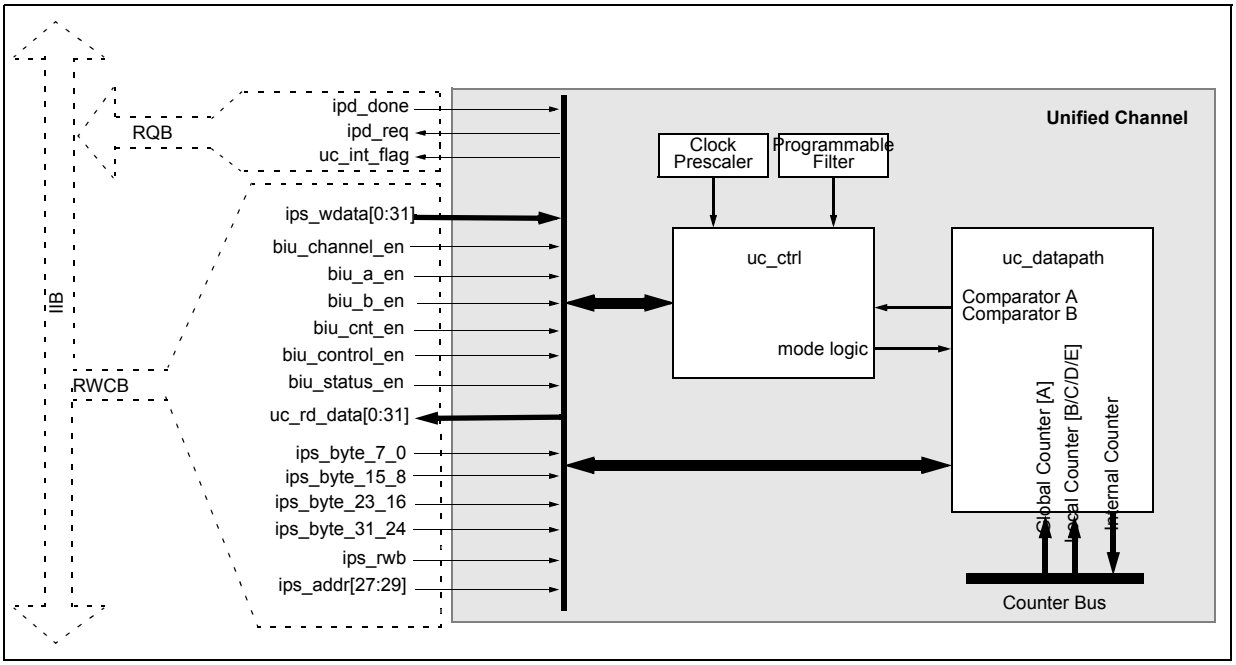
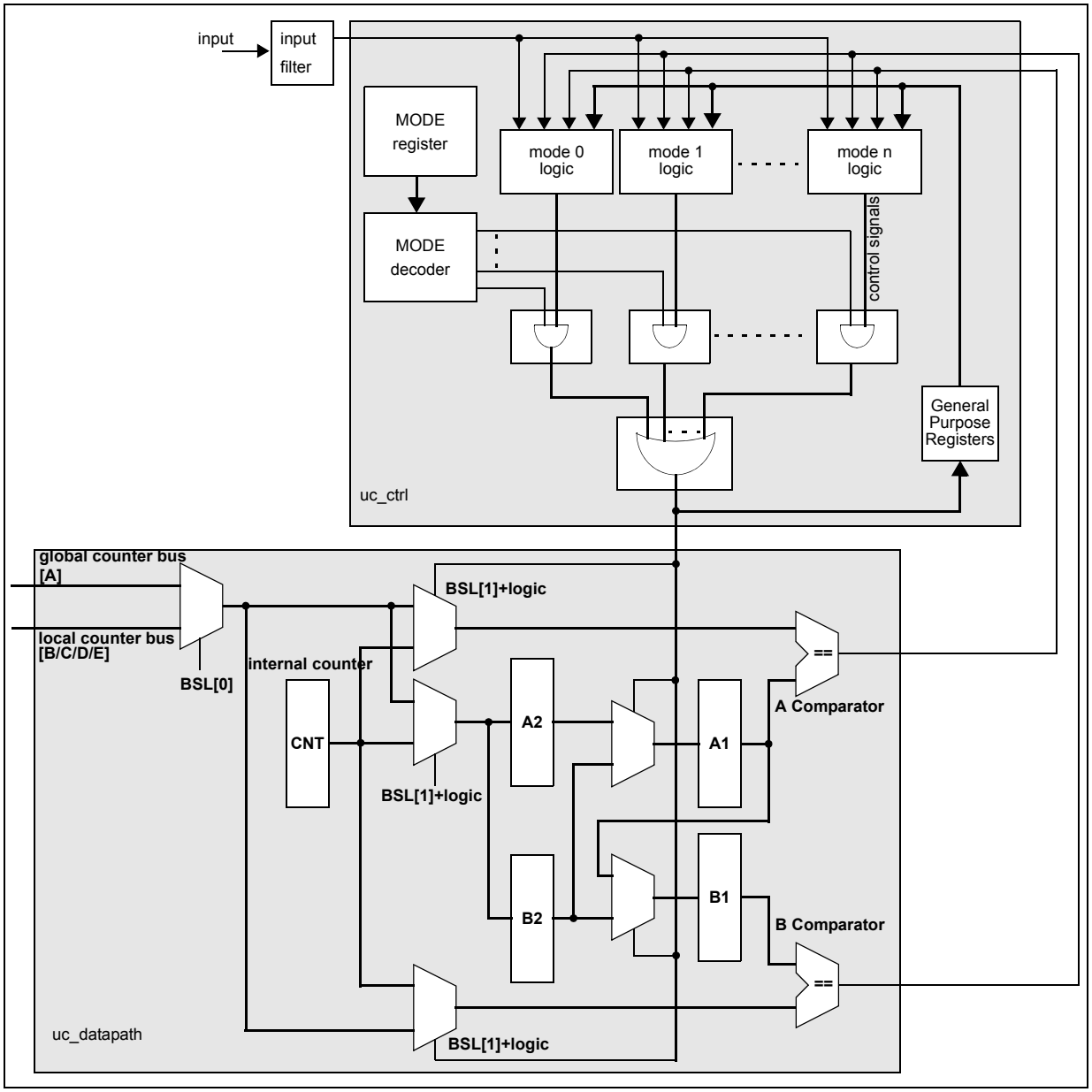


Figure 308 shows both the Unified Channel Control and Datapath block diagram. The Control block is responsible for the generation of signals to control the multiplexes in the Datapath sub-block. Each mode is implemented by a dedicated logic independent from others modes, thus allowing to optimize the logic by disabling the mode and therefore its associated logic. The unused gates are removed during the synthesis phase. Targeting the logic optimization a set of registers is shared by the modes thus providing sequential events to be stored.

The Datapath block provides the channel A and B registers, the internal time base and comparators. Multiplexors select the input of comparators and data for the registers inputs, thus configuring the datapath in order to implement the channel modes. The outputs of A and B comparators are connected to the uc_ctrl control block.

Figure 308. Unified Channel Control and Datapath Block Diagrams



21.4.1.1 UC modes of operation

The mode of operation of the Unified Channel is determined by the mode select bits MODE[0:6] in the EMIOS_CCR[n] register (see *Table 219* for details).

When entering an output mode (except for GPIO mode), the output flip-flop is set to disabled state according to ODIS bit in the EMIOS_CCR[n] register.

As the internal counter EMIOS_CCNTR[n] continues to run in all modes (except for GPIO mode), it is possible to use this as a time base if the resource is not used in the current mode.

In order to provide smooth waveform generation even if A and B registers are changed on the fly, it is available the MCB, OPWFMB, OPWMB and OPWMCB modes. In these modes

A and B registers are double buffered. They are presented in separate sections since there are several basic differences between these and the MC, OPWFM, OPWM and OPWMC modes, respectively.

21.4.1.1.1 General purpose Input/Output (GPIO) Mode

In GPIO mode, all input capture and output compare functions of the UC are disabled, the internal counter (EMIOS_CCNTR[n] register) is cleared and disabled. All control bits remain accessible. In order to prepare the UC for a new operation mode, writing to registers EMIOS_CADR[n] or EMIOS_CBDR[n] stores the same value in registers A1/A2 or B1/B2, respectively. Writing to register EMIOS_ALTA[n] stores a value only in register A2.

MODE[6] bit selects between input (MODE[6] = 0) and output (MODE[6] = 1) modes.

It is required that when changing MODE[0:6], the application software goes to GPIO mode first in order to reset the UC's internal functions properly. Failure to do this could lead to invalid and unexpected output compare or input capture results or the FLAGs being set incorrectly.

In GPIO input mode (MODE[0:6]=0000000), the FLAG generation is determined according to EDPOL and EDSEL bits and the input pin status can be determined by reading the UCIN bit.

In GPIO output mode (MODE[0:6]=0000001), the Unified Channel is used as a single output port pin and the value of the EDPOL bit is permanently transferred to the output flip-flop.

21.4.1.1.2 Single Action Input Capture (SAIC) Mode

In SAIC mode (MODE[0:6]=0000010), when a triggering event occurs on the input pin, the value on the selected time base is captured into register A2. The FLAG bit is set along with the capture event to indicate that an input capture has occurred. Register EMIOS_CADR[n] returns the value of register A2. As soon as the SAIC mode is entered coming out from GPIO mode the channel is ready to capture events. The events are captured as soon as they occur thus reading register A always returns the value of the latest captured event. Subsequent captures are enabled with no need of further reads from EMIOS_CADR[n] register. The FLAG is set at any time a new event is captured.

The input capture is triggered by a rising, falling or either edges in the input pin, as configured by EDPOL and EDSEL bits in EMIOS_CCR[n] register.

Figure 309 and *Figure 310* show how the Unified Channel can be used for input capture.

Figure 309. Single Action Input Capture with rising edge triggering example

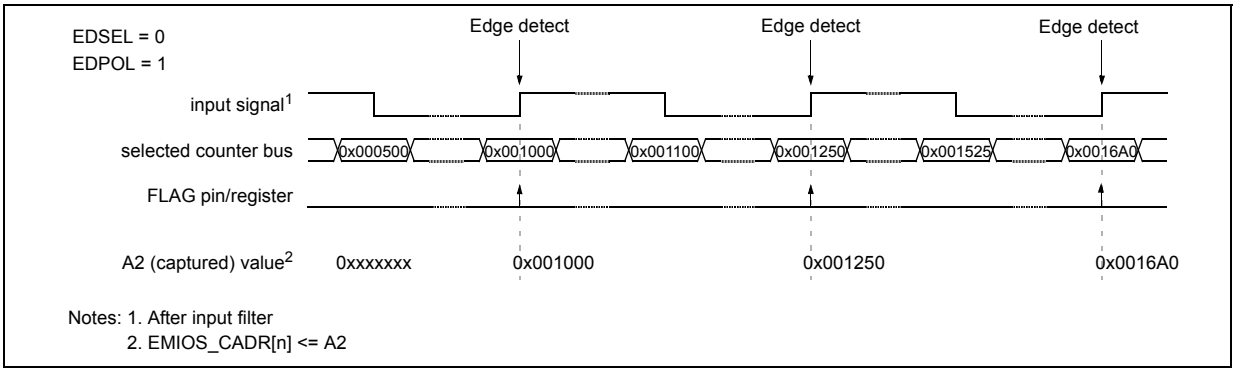
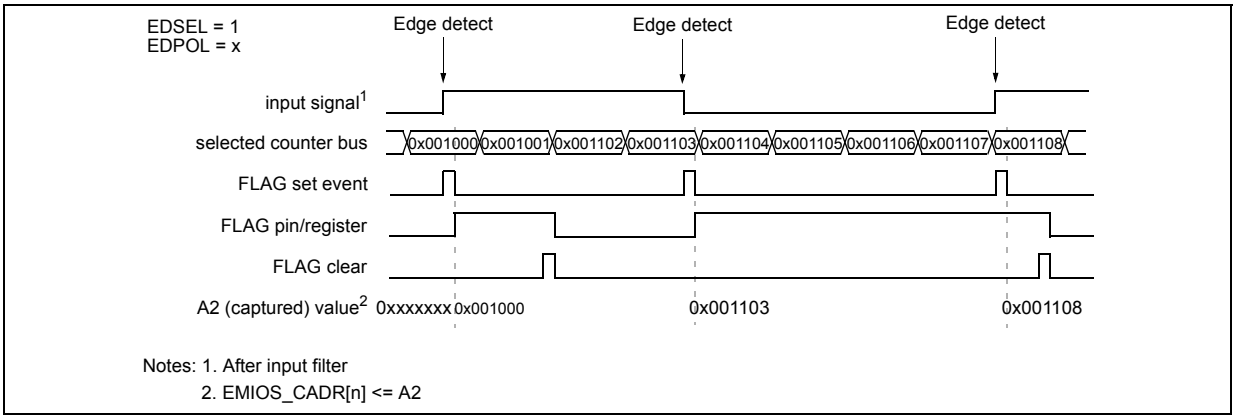


Figure 310. Single Action Input Capture with both edges triggering example



21.4.1.1.3 Single Action Output Compare (SAOC) Mode

In SAOC mode (MODE[0:6]=0000011) a match value is loaded in register A2 and then immediately transferred to register A1 to be compared with the selected time base. When a match occurs, the EDSEL bit selects whether the output flip-flop is toggled or the value in EDPOL is transferred to it. Along with the match the FLAG bit is set to indicate that the output compare match has occurred. Writing to register EMIOS_CADR[n] stores the value in register A2 and reading to register EMIOS_CADR[n] returns the value of register A1.

An output compare match can be simulated in software by setting the FORCMA bit in EMIOS_CCR[n] register. In this case, the FLAG bit is not set.

When SAOC mode is entered coming out from GPIO mode the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[n] register.

Counter bus can be either internal or external and is selected through BSL[0:1] bits.

Figure 311 and *Figure 312* show how the Unified Channel can be used to perform a single output compare with EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively. Note that once in SAOC mode the matches are enabled thus the desired match value on register A1 must be written before the mode is entered. A1 register can be updated at any time thus modifying the match value which will reflect in the output signal generated by the channel. Subsequent matches are enabled with no need of further writes to EMIOS_CADR[n] register. The FLAG is set at the same time a match occurs (see *Figure 313*).

Note: The channel internal counter in SAOC mode is free-running. It starts counting as soon as the SAOC mode is entered.

Figure 311. SAOC example with EDPOL value being transferred to the output flip-flop

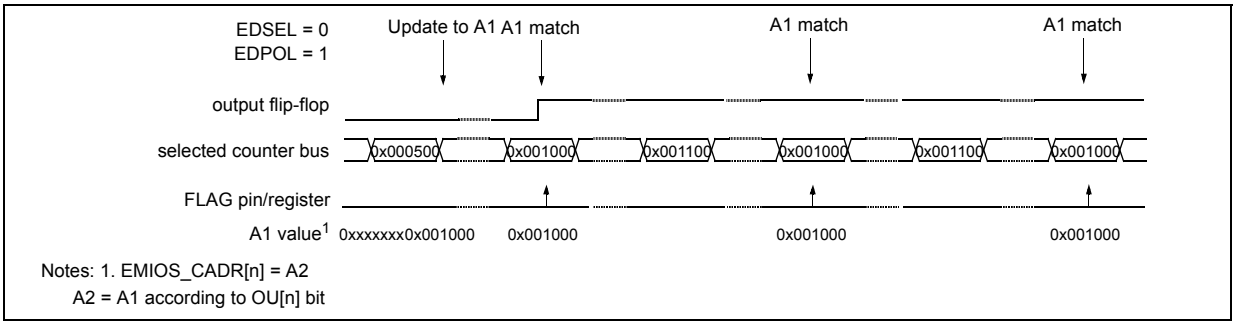


Figure 312. SAOC example toggling the output flip-flop

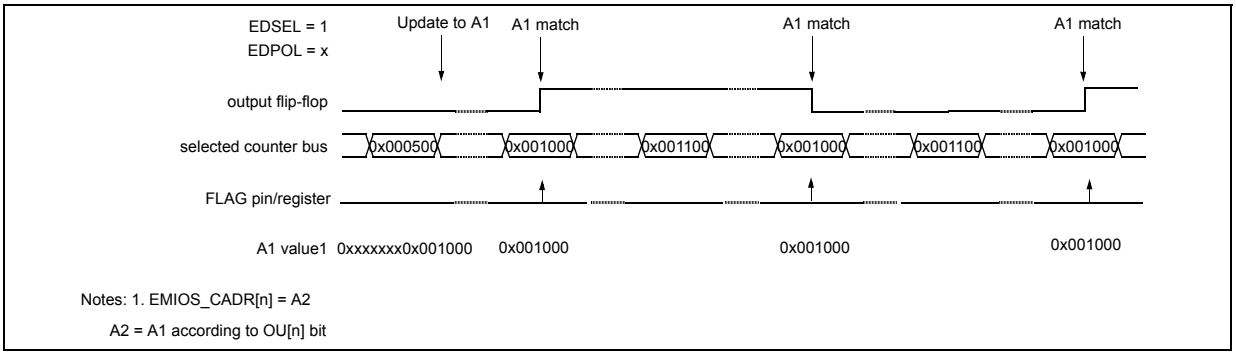
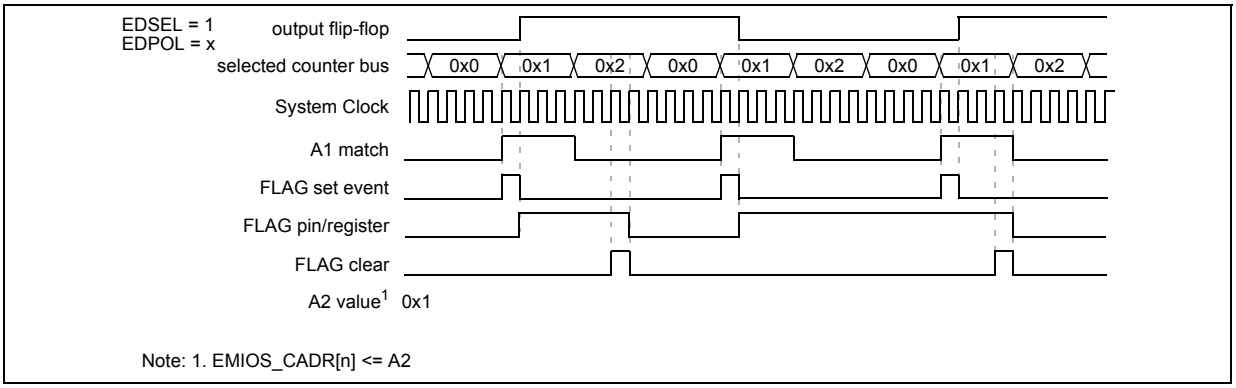


Figure 313. SAOC example with flag behavior



21.4.1.1.4 Input Pulse Width Measurement (IPWM) Mode

The IPWM mode (MODE[0:6]=0000100) allows the measurement of the width of a positive or negative pulse by capturing the leading edge on register B1 and the trailing edge on register A2. Successive captures are done on consecutive edges of opposite polarity. The leading edge sensitivity (i.e., pulse polarity) is selected by EDPOL bit in the EMIOS_CCR[n] register. Registers EMIOS_CADR[n] and EMIOS_CBDR[n] return the values in register A2 and B1, respectively.

The capture function of register A2 remains disabled until the first leading edge triggers the first input capture on register B2. When this leading edge is detected, the count value of the selected time base is latched into register B2; the FLAG bit is not set. When the trailing edge is detected, the count value of the selected time base is latched into register A2 and, at the same time, the FLAG bit is set and the content of register B2 is transferred to register B1 and to register A1.

If subsequent input capture events occur while the corresponding FLAG bit is set, registers A2, B1 and A1 will be updated with the latest captured values and the FLAG will remain set. Registers EMIOS_CADR[n] and EMIOS_CBDR[n] return the value in registers A2 and B1, respectively.

In order to guarantee coherent access, reading EMIOS_CADR[n] forces B1 be updated with the content of register A1. At the same time transfers between B2 and B1 are disabled until the next read of EMIOS_CBDR[n] register. Reading EMIOS_CBDR[n] register forces B1 be updated with A1 register content and re-enables transfers from B2 to B1, to take effect at the next trailing edge capture. Transfers from B2 to A1 are not blocked at any time.

The input pulse width is calculated by subtracting the value in B1 from A2.

Figure 314 shows how the Unified Channel can be used for input pulse width measurement.

Figure 314. Input Pulse Width Measurement example

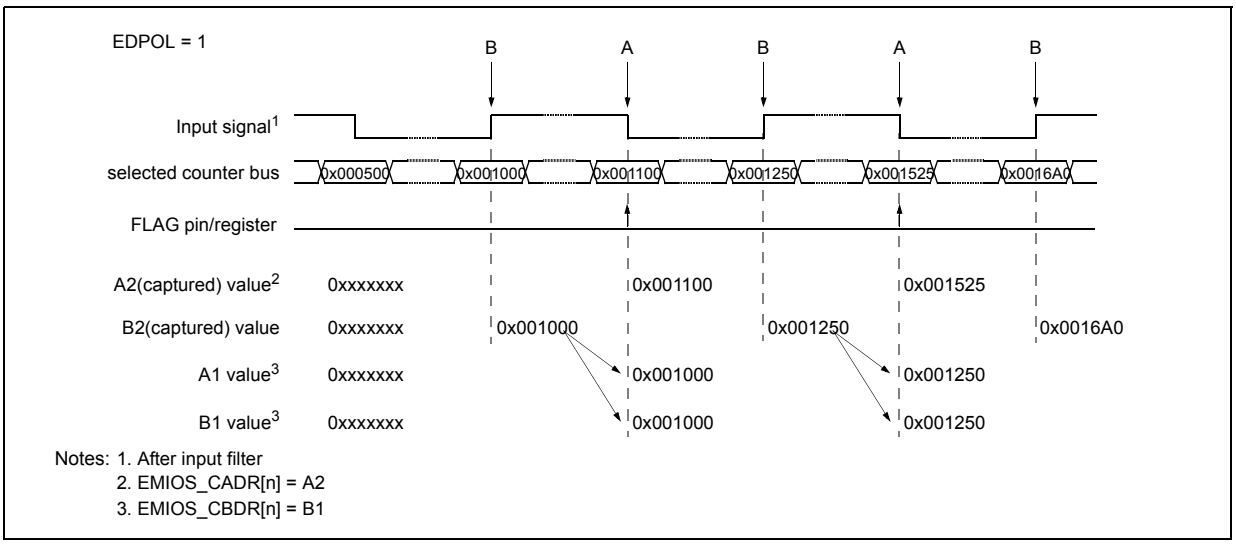
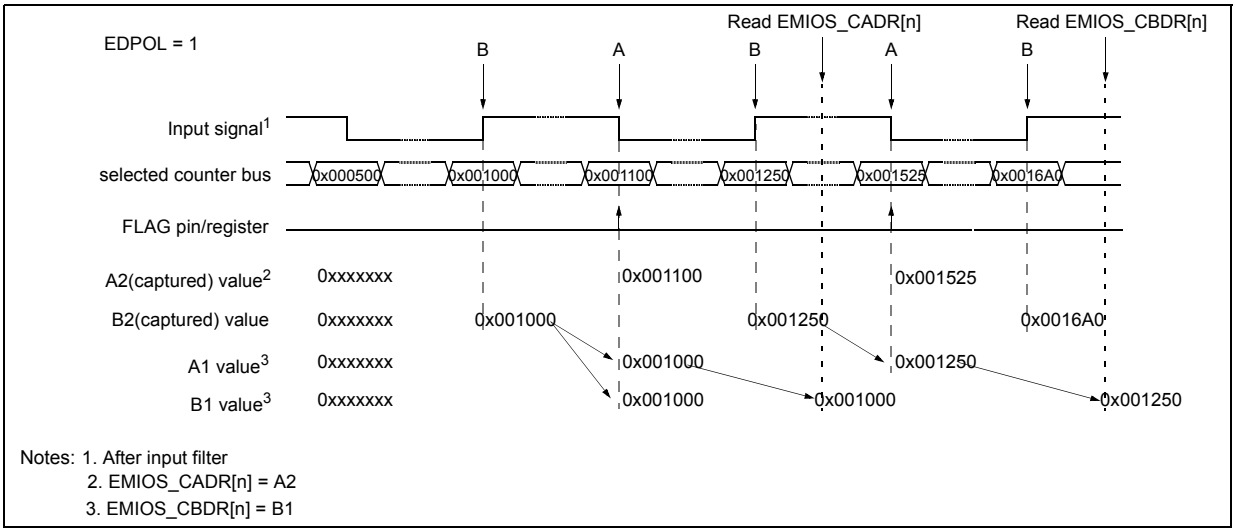


Figure 315 shows the A1 and B1 updates when EMIOS_CADR[n] and EMIOS_CBDR[n] register reads occur. Note that A1 register has always coherent data related to A2 register. Note also that when EMIOS_CADR[n] read is performed B1 register is loaded with A1 register content. This guarantee that the data in register B1 has always the coherent data related to the last EMIOS_CADR[n] read. The B1 register updates remains locked until EMIOS_CBDR[n] read occurs. If EMIOS_CADR[n] read is performed B1 is updated with A1 register content even if B1 update is locked by a previous EMIOS_CADR[n] read operation.

Figure 315. B1 and A1 updates at EMIOS_CADR[n] and EMIOS_CBDR[n] reads



Reading EMIOS_CADR[n] followed by EMIOS_CBDR[n] always provide coherent data. If not coherent data is required for any reason, the sequence of reads should be inverted, therefore EMIOS_CBDR[n] should be read prior to EMIOS_CADR[n] register. Note that even in this case B1 register updates will be blocked after EMIOS_CADR[n] read, thus a second EMIOS_CBDR[n] is required in order to release B1 register updates.

21.4.1.1.5 Input Period Measurement (IPM) Mode

The IPM mode (MODE[0:6]=0000101) allows the measurement of the period of an input signal by capturing two consecutive rising edges or two consecutive falling edges. Successive input captures are done on consecutive edges of the same polarity. The edge polarity is defined by the EDPOL bit in the EMIOS_CCR[n] register.

When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set, and the values in registers B1 is meaningless. On the second and subsequent captures, the FLAG line is set and data in register B2 is transferred to register B1.

When the second edge of the same polarity is detected, the counter bus value is latched into registers A2 and B2, the data previously held in register B2 is transferred to data register B1 and to register A1. The FLAG bit is set to indicate the start and end points of a complete period have been captured. This sequence of events is repeated for each subsequent capture. Registers EMIOS_CADR[n] and EMIOS_CBDR[n] return the values in register A2 and B1, respectively.

In order to allow coherent data, reading EMIOS_CADR[n] forces A1 content be transferred to B1 register and disables transfers between B2 and B1. These transfers are disabled until the next read of the EMIOS_CBDR[n] register. Reading EMIOS_CBDR[n] register forces A1 content to be transferred to B1 and re-enables transfers from B2 to B1, to take effect at the next edge capture.

The input pulse period is calculated by subtracting the value in B1 from A2.

Figure 316 shows how the Unified Channel can be used for input period measurement.

Figure 316. Input Period Measurement example

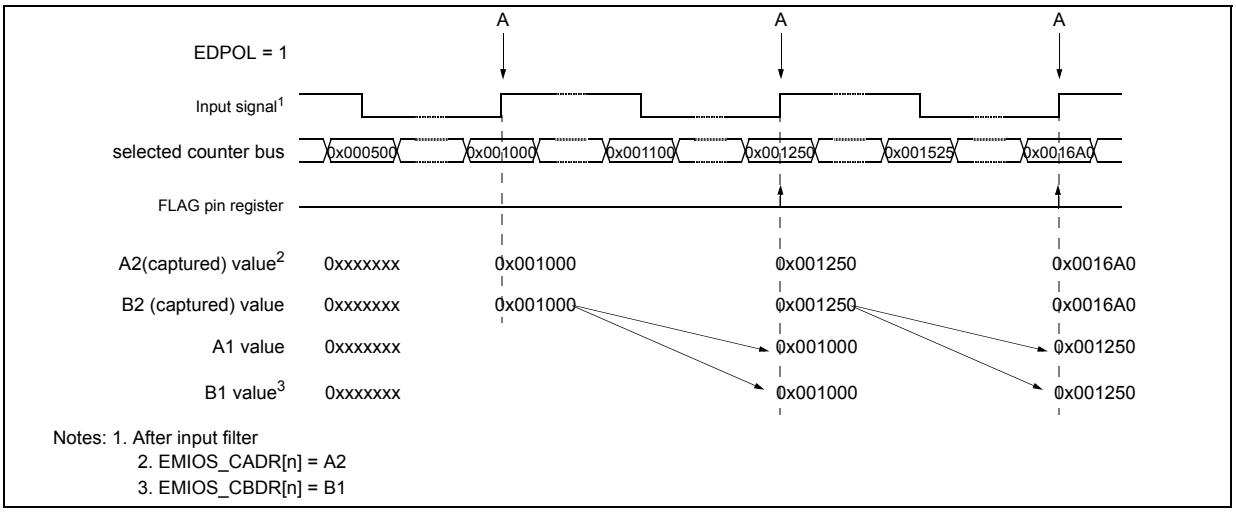
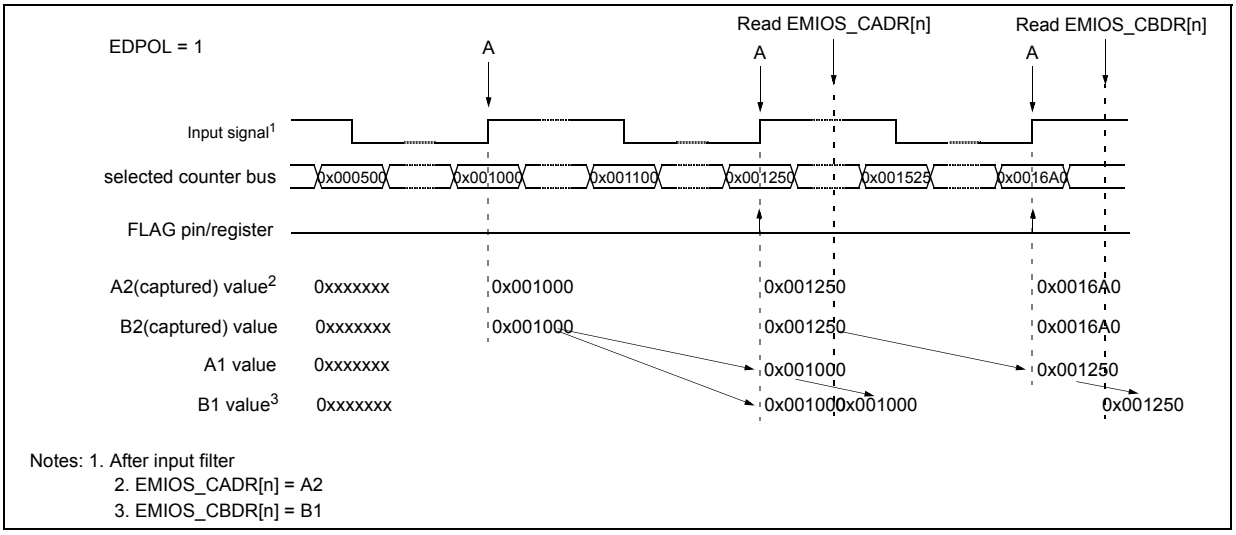


Figure 317 describes the A1 and B1 register updates when EMIOS_CADR[n] and EMIOS_CBDR[n] read operations are performed. When EMIOS_CADR[n] read occurs the content of A1 is transferred to B1 thus providing coherent data in A2 and B1 registers. Transfers from B2 to B1 are then blocked until EMIOS_CBDR[n] is read. After EMIOS_CBDR[n] is read, register A1 content is transferred to register B1 and the transfers from B2 to B1 are re-enabled to occur at the transfer edges, which is the leading edge in the *Figure 317* example.

Figure 317. A1 and B1 updates at EMIOS_CADR[n] and EMIOS_CBDR[n] reads



21.4.1.1.6 Double Action Output Compare (DAOC) Mode

In the DAOC mode the leading and trailing edges of the variable pulse width output are generated by matches occurring on comparators A and B. There is no restriction concerning the order in which A and B matches occur.

When the DAOC mode is entered, coming out from GPIO mode both comparators are disabled and the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[n] register.

Data written to A2 and B2 are transferred to A1 and B1, respectively, on the next system clock cycle if OU[n] bit of EMIOS_OUDR register is cleared (see *Figure 320*). The transfer is blocked if OU[n] bit is set. Comparator A is enabled only after the transfer to A1 register occurs and is disabled on the next A match. Comparator B is enabled only after the transfer to B1 register occurs and is disabled on the next B match. Comparators A and B are enabled and disabled independently.

The output flip-flop is set to the value of EDPOL when a match occurs on comparator A and to the complement of EDPOL when a match occurs on comparator B.

MODE[6] controls if the FLAG is set on both matches (MODE[0:6]=0000111) or just on the B match (MODE[0:6]=0000110). FLAG bit assertion depends on comparator enabling.

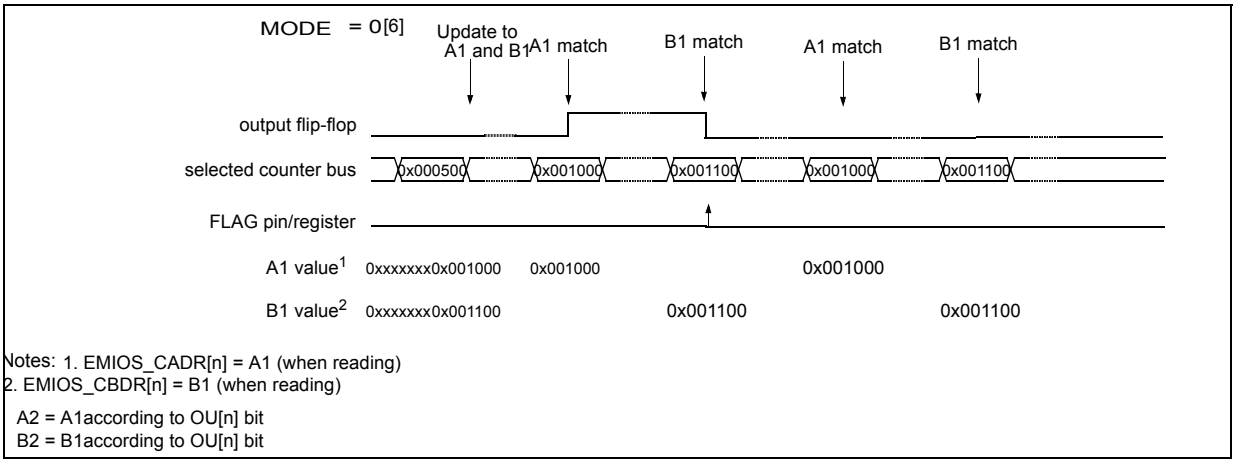
If subsequent enabled output compares occur on registers A1 and B1, pulses will continue to be generated, regardless of the state of the FLAG bit.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a comparison event in comparator A or B, respectively. Note that the FLAG bit is not affected by these forced operations.

Note: If both registers (A1 and B1) are loaded with the same value, the B match prevails concerning the output pin state (output flip-flop is set to the complement of EDPOL), the FLAG bit is set and both comparators are disabled.

Figure 318 and *Figure 319* show how the Unified Channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.

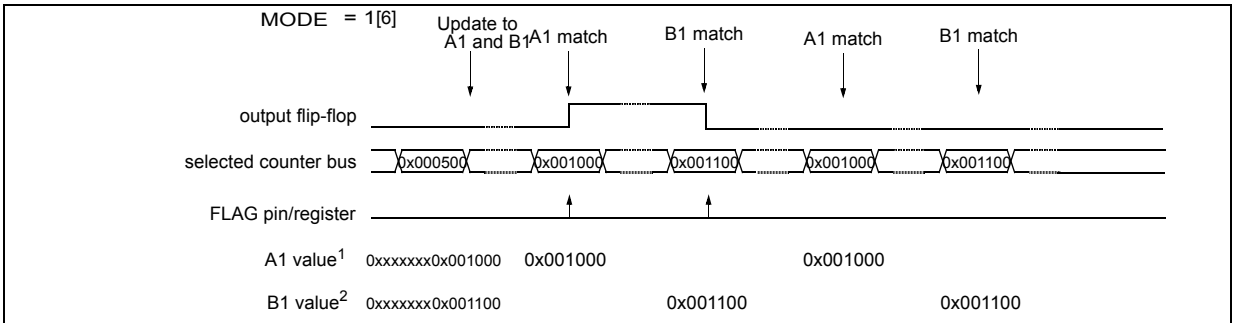
Figure 318. Double Action Output Compare with FLAG set on the second match



Notes: 1. EMIOS_CADR[n] = A1 (when reading)
2. EMIOS_CBDR[n] = B1 (when reading)

A2 = A1according to OU[n] bit
B2 = B1according to OU[n] bit

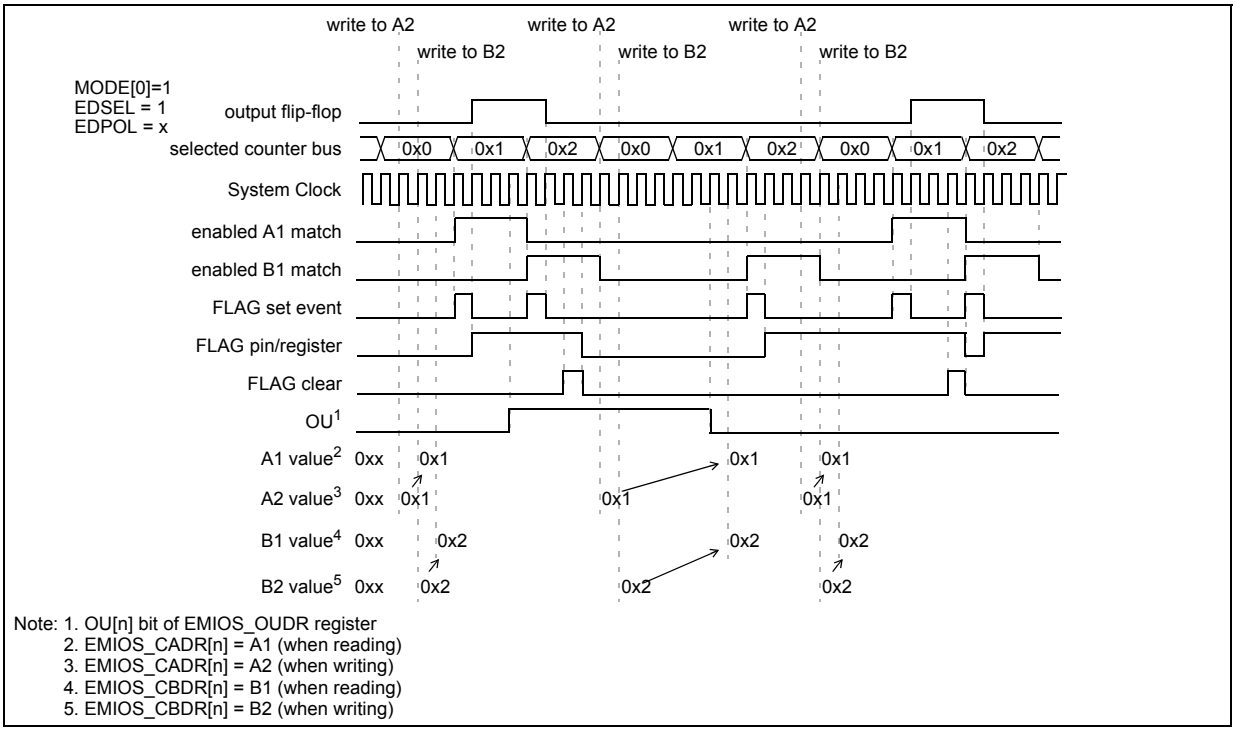
Figure 319. Double Action Output Compare with FLAG set on both matches



Notes: 1. EMIOS_CADR[n] = A1 (when reading)
2. EMIOS_CBDR[n] = B1 (when reading)

A2 = A1according to OU[n] bit
B2 = B1according to OU[n] bit

Figure 320. DAOC with transfer disabling example



21.4.1.1.7 Pulse/Edge Accumulation (PEA) Mode

The PEA mode returns the time taken to detect a desired number of input events. MODE[6] bit selects between continuous or single shot operation.

After writing to register A1, the internal counter is cleared on the first input event, ready to start counting input events and the selected timebase is latched into register B2. On the match between the internal counter and register A1, a counter bus capture is triggered to register A2 and B2. The data previously held in register B2 is transferred to register B1 and the FLAG bit is set to indicate that an event has occurred. The desired time interval can be determined by subtracting register B1 from A2. Registers EMIOS_CADR[n] and EMIOS_CBDR[n] return the values in register A2 and B1, respectively.

As part of the coherency mechanism, reading EMIOS_CADR[n] disables transfers from B2 to B1. These transfers are disabled until the next read of the EMIOS_CBDR[n] register. Reading the EMIOS_CBDR[n] register re-enables transfers from B2 to B1, to take effect at the next transfer event, as described above.(l)

In order to have coherent data in continuous mode of operation the following steps should be performed, assuming FLAG is initially cleared:

l. If B1 was not updated due to B2 to B1 transfer being disabled after reading register EMIOS_CADR[n], further EMIOS_CADR[n] and EMIOS_CBDR[n] reads will not return coherent data until a new bus capture is triggered to registers A2 and B2. This capture event is indicated by the channel FLAG being asserted. If enabled, the FLAG also generates an interrupt.

1. Wait for FLAG assertion;
2. Read EMIOS_CADR[n] register;
3. Read EMIOS_CBDR[n] register;
4. Clear FLAG bit;
5. Return to step 1.

Accumulation cycles may be lost if the read is not performed in a timely manner. Whenever Overrun bit is asserted it means that one or more cycles have been lost.

Triggering of the counter clock (input event) is done by a rising or falling edge or both edges on the input pin. The polarity of the triggering edge is selected by the EDSEL and EDPOL bits in EMIOS_CCR[n] register.

For continuos operation mode (MODE[6] cleared, MODE[0:6]=0001000), the counter is cleared on the next input event after a FLAG generation and continues to operate as described above.

For single shot operation (MODE[6] set, MODE[0:6]=0001001), the counter is not cleared or incremented after a FLAG generation, until a new writing operation to register A is performed.

Figure 321 and *Figure 322* show how the Unified Channel can be used for continuos and single shot pulse/edge accumulation mode.

Figure 321. Pulse/Edge Accumulation continuous mode example

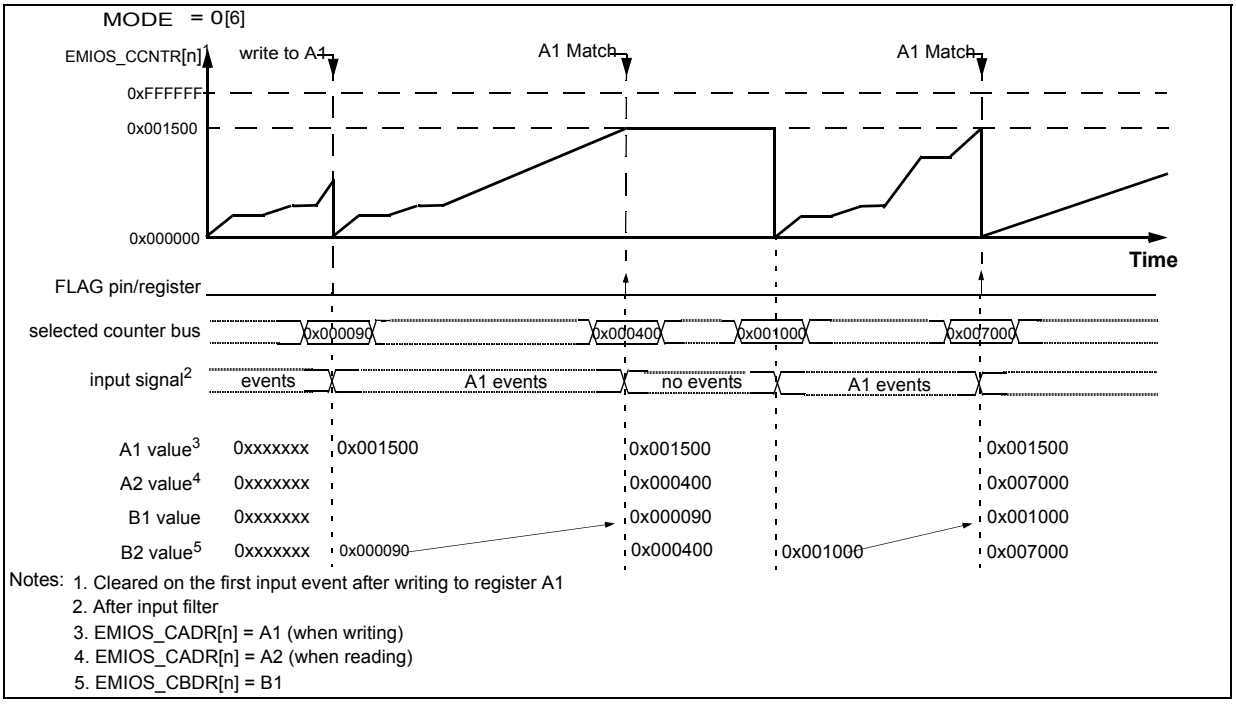
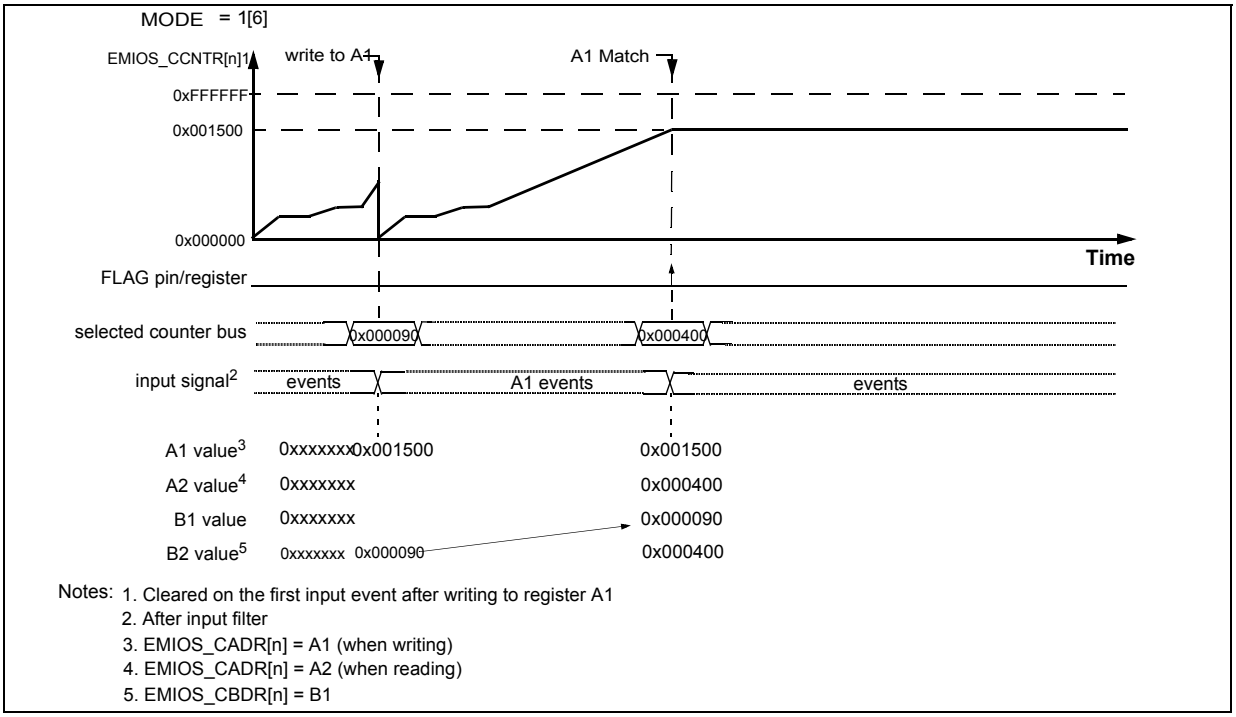


Figure 322. Pulse/Edge Accumulation single-shot mode example



21.4.1.1.8 Pulse/Edge Counting (PEC) Mode

The PEC mode returns the amount of pulses or edges detected on the input for a desired time window. MODE[6] bit selects between continuous or single shot operation.

Triggering of the internal counter is done by a rising or falling edge or both edges on the input signal. The polarity and the triggering edge is selected by EDSEL and EDPOL bits in EMIOS_CCR[n] register.

Register A1 holds the start time and register B1 holds the stop time for the time window. After writing to register A1, when a match occur between comparator A and the selected timebase, the internal counter is cleared and it is ready to start counting input events. When the time base matches comparator B, the internal counter is disabled and its content is transferred to register A2. At the same time the FLAG bit is set. Reading registers EMIOS_CCNTR[n] or A2 returns the amount of detected pulses.

For continuous operation (MODE[6] cleared, MODE[0:6]=0001010), the next match between comparator A and the selected time base clears the internal counter and counting is enabled again. In order to guarantee coherent measurements when reading EMIOS_CCNTR[n] after the FLAG is set, the software must check if the time base value is out of the time interval defined by registers A1 and B1. Alternatively register A2 always holds the latest available measurement providing coherent data at any time after the first FLAG had occurred. This register is addressed by the alternate address EMIOS_ALTA[n].

For single shot operation (MODE[6] set, MODE[0:6]=0001011), the next match between comparator A and the selected time base has no effect, until a new write to register A is performed. The EMIOS_CCNTR content is also transferred to register A2 when a match in the B comparator occurs.

Figure 323 and *Figure 324* show how the Unified Channel can be used for continuous or single shot pulse/edge counting mode.

Figure 323. Pulse/Edge Counting continuous mode example

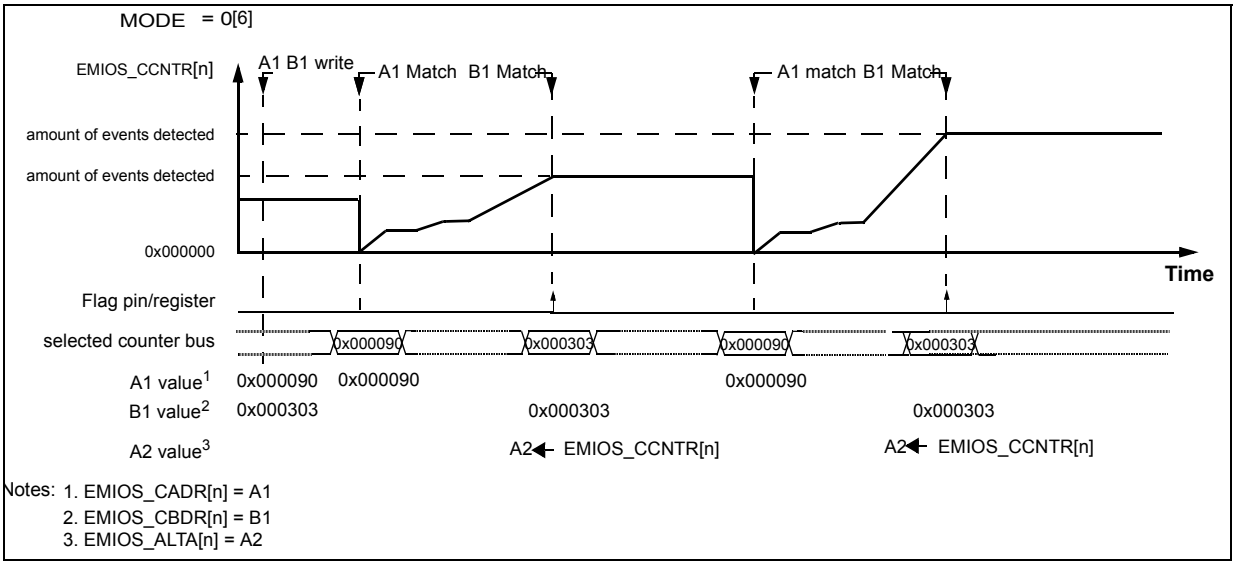
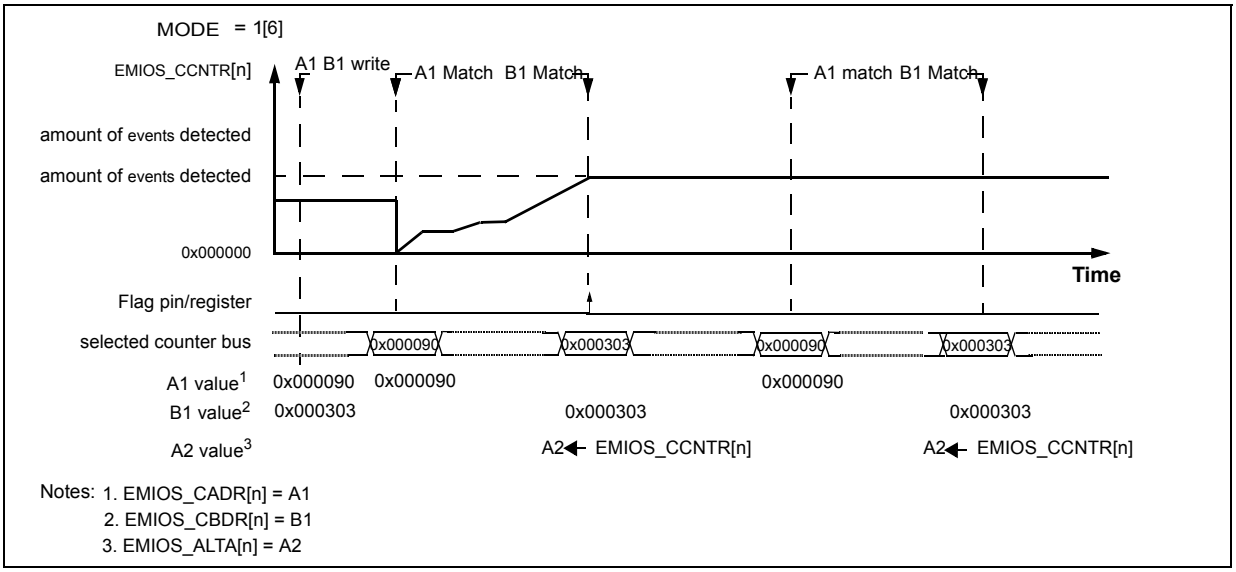


Figure 324. Pulse/Edge Counting single-shot mode example



21.4.1.1.9 Quadrature Decode (QDEC) Mode

Quadrature decode mode uses UC[n] operating in QDEC mode and the input programmable filter (IPF) from UC[n-1]. Note that UC[n-1] can be configured, at the same time, to an operation mode that does not use I/O pins, such as MC mode (modulus counter). The connection among the UCs is circular, i.e., when UC[0] is running in QDEC mode, the input programmable filter from UC[23] is being used.

This mode generates a FLAG every time the internal counter matches A1 register. The internal counter is automatically selected and is not cleared when entering this mode.

MODE[6] bit selects which type of encoder will be used: *count & direction* encoder or *phase_A & phase_B* encoders.

When operating with *count & direction* encoder (MODE[6] cleared), UC[n] input pin must be connected to the *direction* signal and UC[n-1] input pin must be connected to the *count* signal of the quadrature encoder. UC[n] EDPOL bit selects count direction according to *direction* signal and UC[n-1] EDPOL bit selects if the internal counter is clocked by the rising or falling edge of the *count* signal.

When operating with *phase_A & phase_B* encoder (MODE[6] set), UC[n] input pin must be connected to the *phase_A* signal and UC[n-1] input pin must be connected to the *phase_B* signal of the quadrature encoder. EDPOL bit selects the count direction according to the phase difference between *phase_A & phase_B* signals.

Figure 325 and *Figure 326* show two Unified Channels configured to quadrature decode mode for *count & direction* encoder and *phase_A & phase_B* encoders, respectively.

Figure 325. Quadrature Decode mode example with *count & direction* encoder

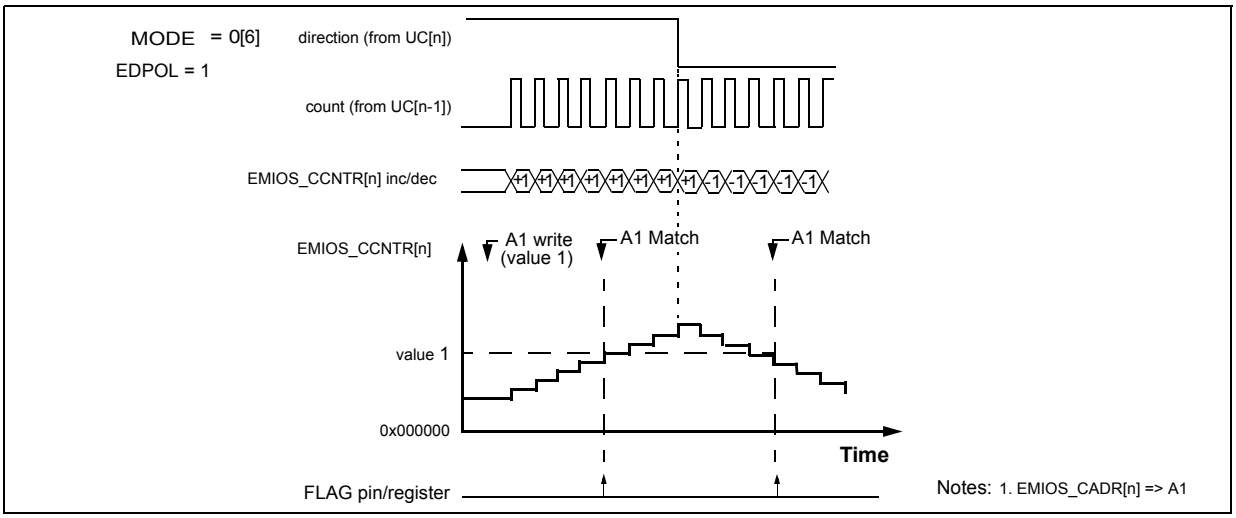
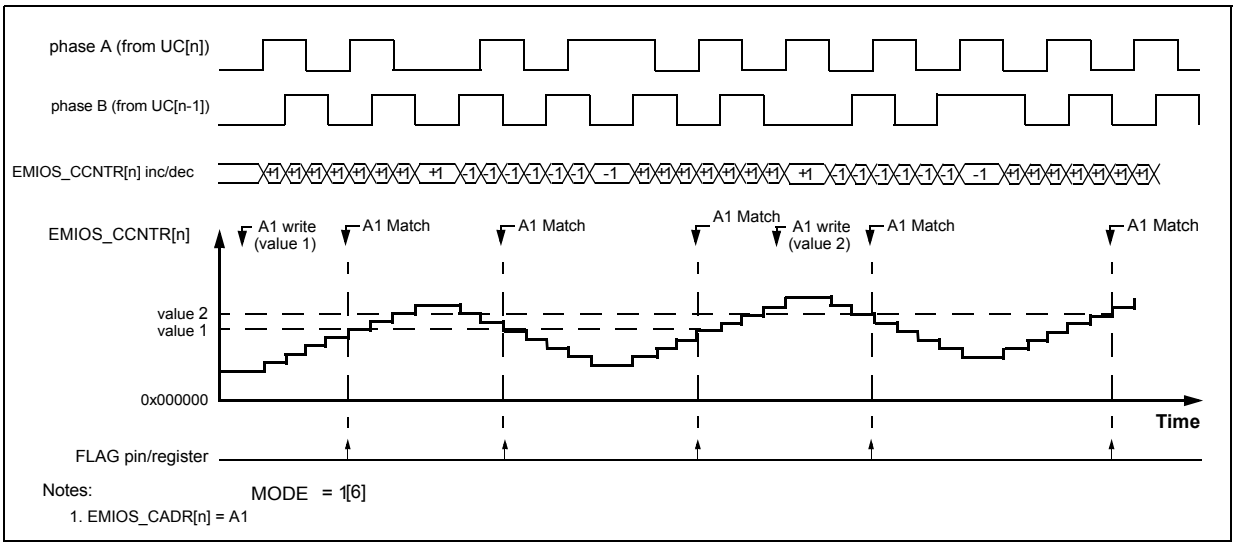


Figure 326. Quadrature Decode mode example with *phase_a & phase_B* encoder



21.4.1.1.10 Windowed Programmable Time Accumulation (WPTA) Mode

The WPTA mode (MODE[0:6]=0001110) accumulates the sum of the total high time or low time of an input signal over a programmable interval (time window).

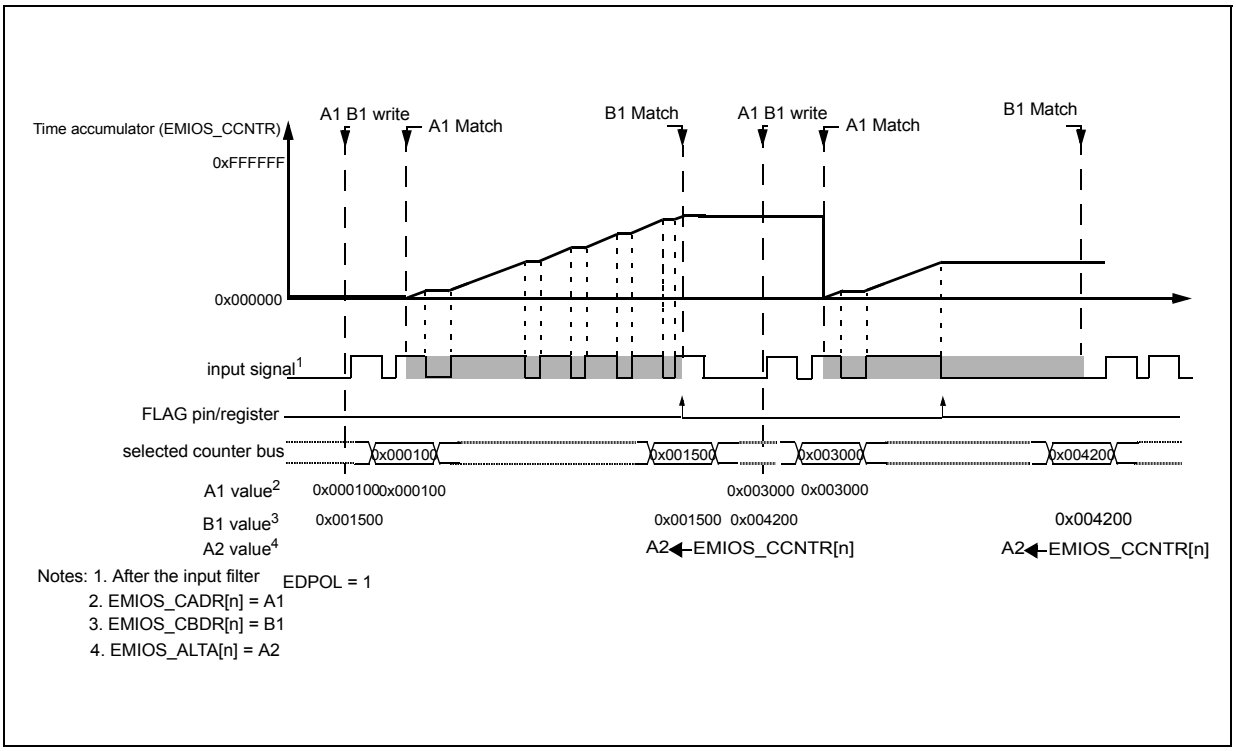
The prescaler bits UCPRE[0:1] in EMIOS_CCR[n] register define the increment rate of the internal counter.

Register A1 holds the start time and register B1 holds the stop time of the programmable time interval. When a match occurs between register A and the selected timebase, the internal counter is cleared and it is ready to start counting. The internal counter is used as a time accumulator, i.e., it counts up when the input signal has the same polarity of EDPOL bit in EMIOS_CCR[n] register and does not count otherwise. When a match occurs in comparator B, the internal counter is disabled regardless of the input signal polarity and the FLAG bit is set. At the same time the content of EMIOS_CCNTR[n] is transferred to register A2. Reading registers EMIOS_CCNTR[n] or A2 returns the high or low time of the input signal,

Note that EMIOS_CCNTR[n] is stable only outside the time window defined from A1 to B1 matches, otherwise its contents reflects a count in progress and not the final value. Alternatively to EMIOS_CCNTR register A2 returns the latest available measurement. Since this register is updated only at comparator B matches it always contains stable and up-to-date data. In this mode this register is accessible through the alternate register address EMIOS_ALTA[n].

Figure 327 shows how the Unified Channel can be used to accumulate high time.

Figure 327. Windowed Programmable Time Accumulation example



21.4.1.1.11 Modulus Counter (MC) Mode

The MC mode can be used to provide a time base for a counter bus or as a general purpose timer.

MODE[6] bit selects internal or external clock source when cleared or set, respectively. When external clock is selected, the input signal pin is used as the source and the triggering polarity edge is selected by the EDPOL and EDSEL in the EMIOS_CCR[n] register.

The internal counter counts up from the current value until it matches the value in register A1. Register B1 is cleared and is not accessible to the MCU. MODE[4] bit selects up mode or up/down mode, when cleared or set, respectively.

When in up count mode, a match between the internal counter and register A1 sets the FLAG and clears the internal counter. The timing of those events varies according to the MC mode setup as follows:

- Internal counter clearing on match start (MODE[0:6]=001000b)
  - External clock is selected if MODE[6] is set. In this case the internal counter clears as soon as the match signal occurs. The channel FLAG is set at the same time the match occurs. Note that by having the internal counter cleared as soon as the match occurs and incremented at the next input event a shorter zero count is generated. See *Figure 360* and *Figure 361*.
  - Internal clock source is selected if MODE[6] is cleared. In this case the counter clears as soon as the match signal occurs. The channel FLAG is set at the same time the match occurs. At the next prescaler tick after the match the internal counter remains at zero and only resumes counting on the following tick. See *Figure 360* and *Figure 362*.
- Internal counter clearing on match end (MODE[0:6]=001001b)
  - External clock is selected if MODE[6] is set. In this case the internal counter clears when the match signal is asserted and the input event occurs. The channel FLAG is set at the same time the counter is cleared. See *Figure 360* and *Figure 363*.
  - Internal clock source is selected if MODE[6] is cleared. In this case the internal counter clears when the match signal is asserted and the prescaler tick occurs. The channel FLAG is set at the same time the counter is cleared. See *Figure 360* and *Figure 363*.

Note: If internal clock source is selected and the prescaler of the internal counter is set to 1 the MC mode behaves the same way even in Clear on Match Start or Clear on Match End sub-modes.

When in up/down count mode (MODE[0:6]=00101bb), a match between the internal counter and register A1 sets the FLAG and changes the counter direction from increment to decrement. A match between register B1 and the internal counter changes the counter direction from decrement to increment and sets the FLAG only if MODE[5] bit is set.

Only values different than 0x0 must be written at A register. Loading 0x0 leads to unpredictable results.

Updates on A register or counter in MC mode may cause loss of match in the current cycle if the transfer occurs near the match. In this case, the counter may rollover and resume operation in the next cycle.

Register B2 has no effect in MC mode. Nevertheless, register B2 can be accessed for reads and writes by addressing EMIOS_CBDR.

Figure 328 and *Figure 329* show how the Unified Channel can be used as modulus counter in up mode and up/down mode, respectively.

Figure 328. Modulus Counter up mode example

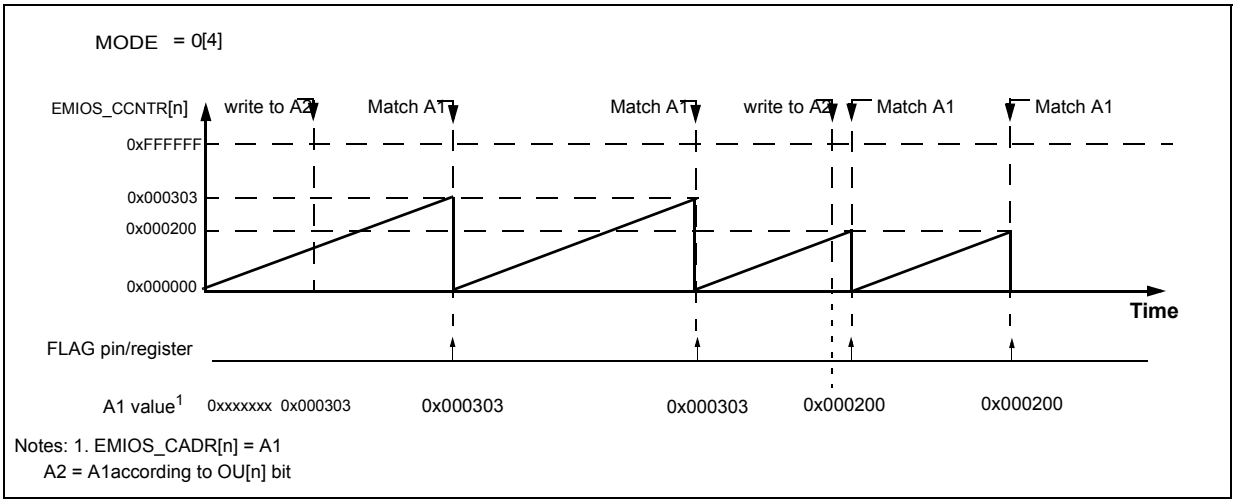
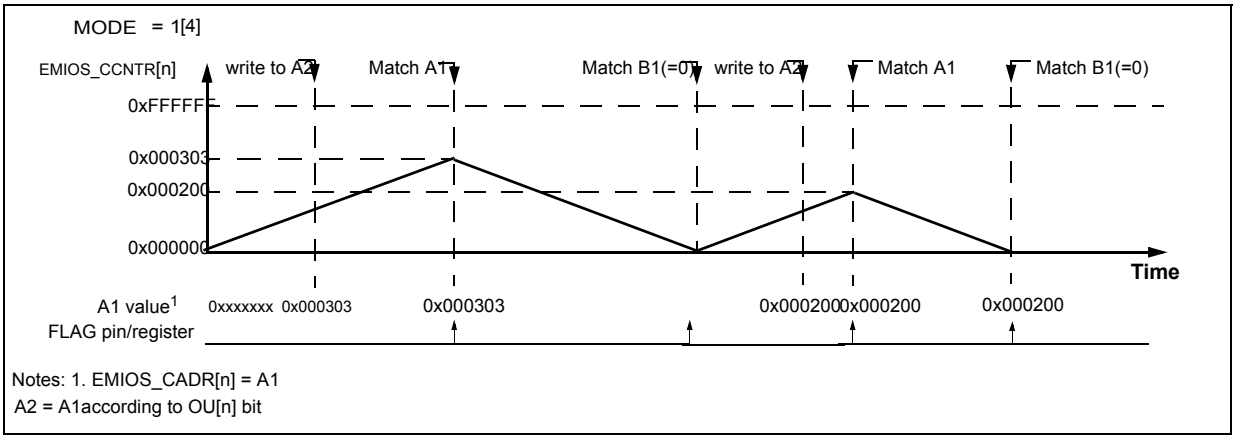


Figure 329. Modulus Counter up/down mode example



21.4.1.1.12 Modulus Counter Buffered (MCB) Mode

The MCB mode provides a time base which can be shared with other channels through the internal counter buses. Register A1 is double buffered thus allowing smooth transitions between cycles when changing A2 register value on the fly. A1 register is updated at the cycle boundary, which is defined as when the internal counter transitions to 0x1.

The internal counter values operates within a range from 0x1 up to register A1 value. If when entering MCB mode coming out from GPIO mode the internal counter value is not within that range then the A match will not occur causing the channel internal counter to wrap at the maximum counter value which is 0xff_ffff for a 24-bit counter. After the counter wrap occurs it returns to 0x1 and resume normal MCB mode operation. Thus in order to avoid the counter wrap condition make sure its value is within the 0x1 to A1 register value range when the MCB mode is entered.

MODE[6] bit selects internal clock source if cleared or external if set. When external clock is selected the input channel pin is used as the channel clock source. The active edge of this clock is defined by EDPOL and EDSEL bits in the EMIOS_CCR[n] channel register.

When entering in MCB mode, if up counter is selected by MODE[4]=0 (MODE[0:6]=101000b), the internal counter starts counting from its current value to up direction until A1 match occurs. The internal counter is set to 0x1 when its value matches A1 value and a clock tick occurs (either prescaled clock or input pin event).

If up/down counter is selected by setting MODE[4]=1, the counter changes direction at A1 match and counts down until it reaches the value 0x1. After it has reached 0x1 it is set to count in up direction again. B1 register is used to generate a match in order to set the internal counter in up-count direction if up/down mode is selected. Register B1 cannot be changed while this mode is selected.

Note that differently from the MC mode, the MCB mode counts between 0x1 and A1 register value. Only values greater than 0x1 must be written at A1 register. Loading values other than those leads to unpredictable results. The counter cycle period is equal to A1 value in up counter mode. If in up/down counter mode the period is defined by the expression: (2*A1)-2.

Figure 330 describes the counter cycle for several A1 values. Register A1 is loaded with A2 register value at the cycle boundary. Thus any value written to A2 register within cycle **n** will be updated to A1 at the next cycle boundary and therefore will be used on cycle **n+1**. The cycle boundary between cycle **n** and cycle **n+1** is defined as when the internal counter transitions from A1 value in cycle **n** to 0x1 in cycle **n+1**. Note that the FLAG is generated at the cycle boundary and has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.

Figure 330. Modulus Counter Buffered (MCB) Up Count mode

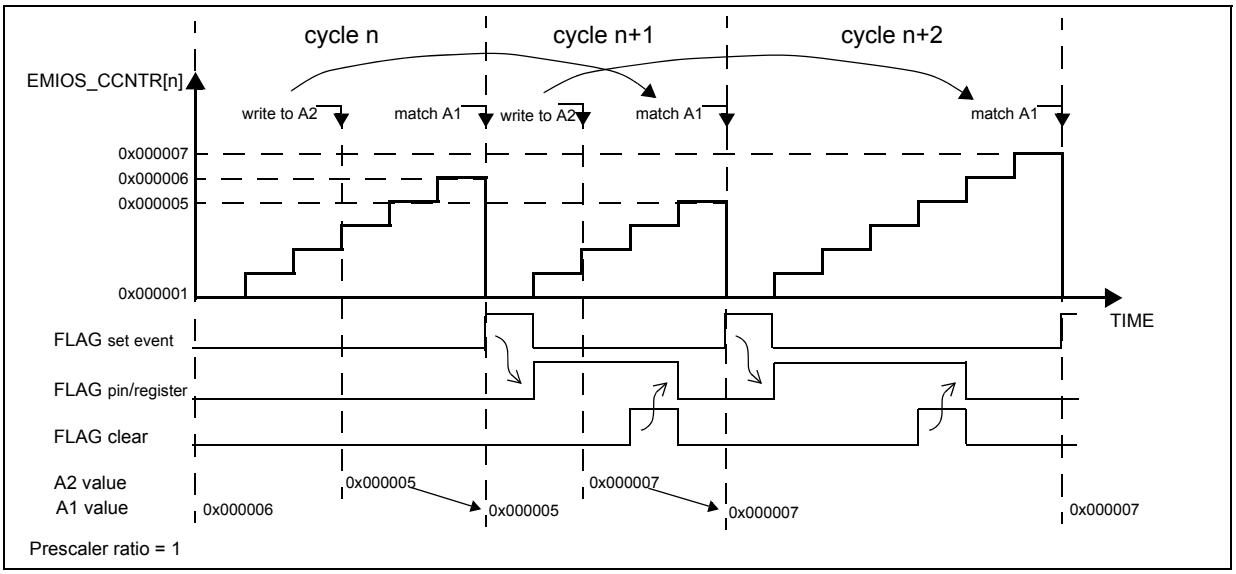


Figure 331 describes the MCB in up/down counter mode (MODE[0:6]=10101bb). A1 register is updated at the cycle boundary. If A2 is written in cycle **n**, this new value will be used in cycle **n+1** for A1 match. Flags are generated only at A1 match start if MODE[5] is 0. If MODE[5] is set to 1 flags are also generated at the cycle boundary.

Figure 331. Modulus Counter Buffered (MCB) Up/Down Mode

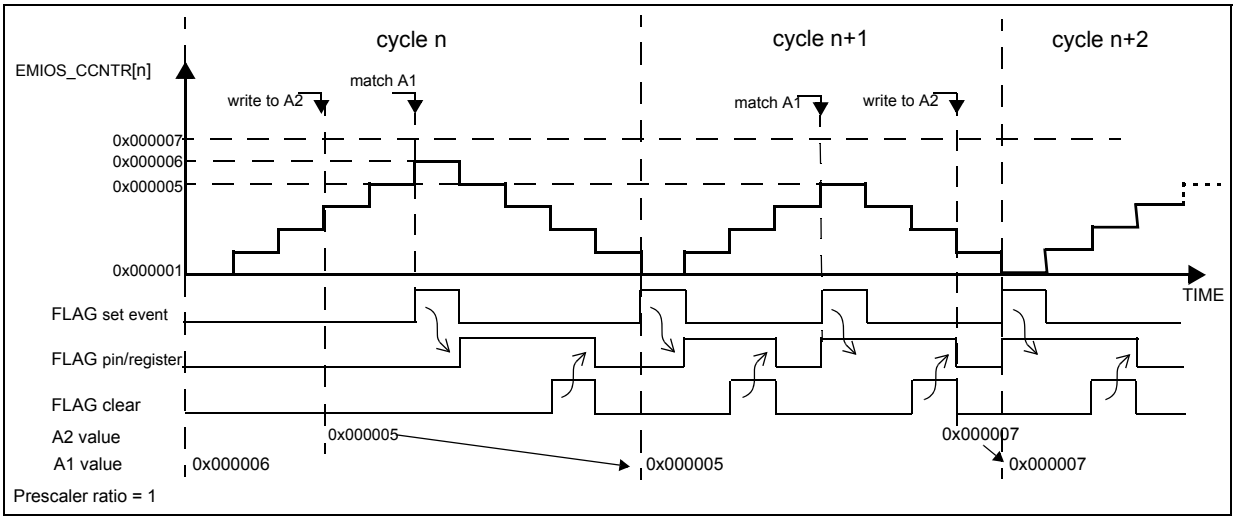


Figure 332 describes in more detail the A1 register update process in up counter mode. The A1 load signal is generated at the last system clock period of a counter cycle. Thus, A1 is updated with A2 value at the same time that the counter (EMIOS_CCNTR[n]) is loaded with 0x1. The load signal pulse has the duration of one system clock period. If A2 is written within cycle **n** its value is available at A1 at the first clock of cycle **n+1** and the new value is used for match at cycle **n+1**. The update disable bits OU[n] of EMIOS_OUDR register can be used to control the update of this register, thus allowing to delay the A1 register update for synchronization purposes.

Figure 332. MCB Mode A1 Register Update in Up Counter Mode

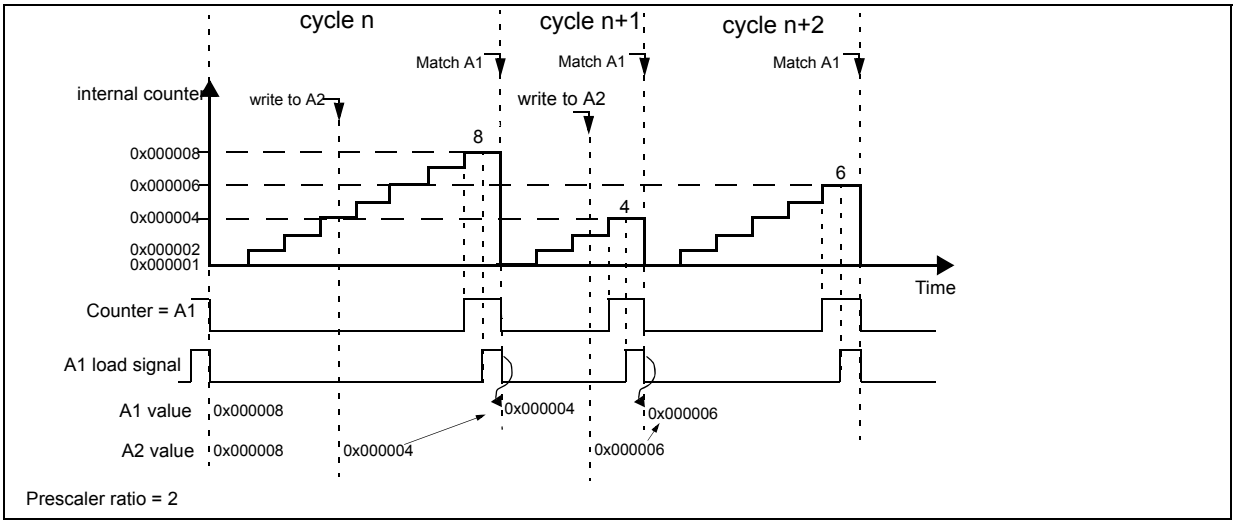
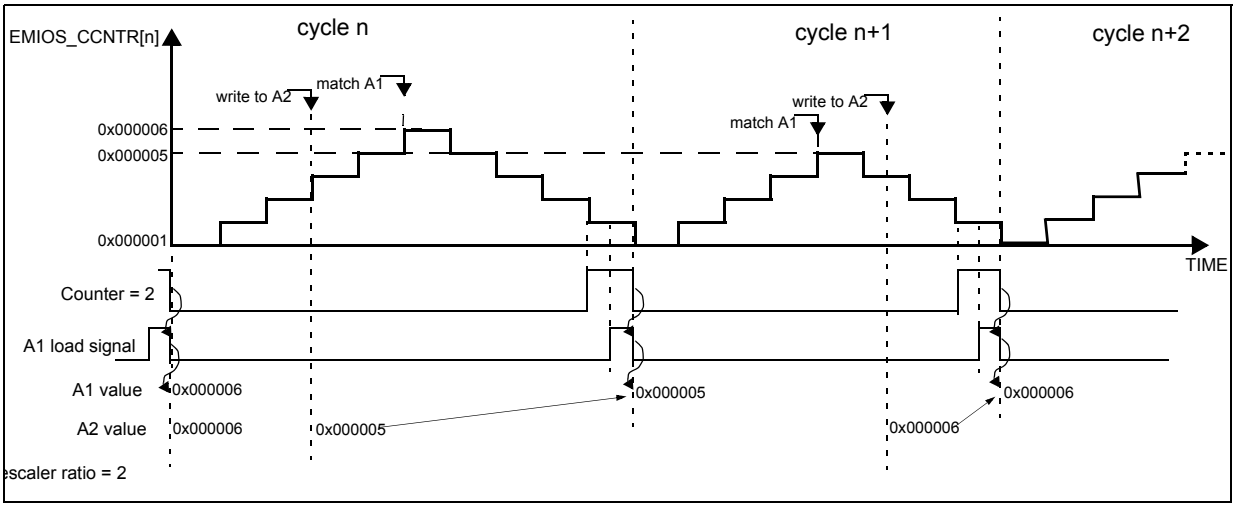


Figure 333 describes the A1 register update in up/down counter mode. Note that A2 can be written at any time within cycle **n** in order to be used in cycle **n+1**. Thus A1 receives this new value at the next cycle boundary. Note that the update disable bits OU[n] of EMIOS_OUDR register can be used to disable the update of A1 register.

Figure 333. MCB Mode A1 Register Update in Up/Down Counter Mode



21.4.1.1.13 Output Pulse Width and Frequency Modulation (OPWFM) Mode

In this mode, duty cycle of output signal is the value defined in register A1 plus one and the period is the value defined in register B1 plus one. MODE[6] bit controls the transfer from register B2 to B1, which can be done either immediately (MODE[6] cleared, MODE[0:6]=00110b0), providing the fastest change in the duty cycle, or at every match of register A1 (MODE[6] set, MODE[0:6]=00110b1).

When OPWFM mode is entered, coming out from GPIO mode, the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[n] register.

The internal counter is automatically selected as a time base, therefore the BSL[0:1] bits in register EMIOS_CCR[n] has no meaning. When a match on comparator A occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit and the internal counter is cleared.

FLAG can be generated at match B, when MODE[5] is cleared, or in both matches, when MODE[5] is set.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Also, FORCMB clears the internal counter. Note that the FLAG bit is not set by the FORCMA or FORCMB operations.

If subsequent comparisons occur on comparators A and B, the PWFM pulses continue to be output, regardless of the state of the FLAG bit.

In order to achieve 100% duty cycle, both registers A1 and B1 must be set to the same value. When a simultaneous match occurs on comparators A and B, the output flip-flop is set to the value of EDPOL bit. 0% duty cycle is possible by writing 0x0 to register A. When a match occurs, the output flip-flop is set at every period to the complement of EDPOL bit. The transfer from register B2 to B1 is still controlled by MODE[6] bit.

Note: Writing 0x0 to A1 and B1 produces a duty cycle of 0%.

Figure 334 shows the Unified Channel running in OPWFM mode with immediate register update and *Figure 335* shows the Unified Channel running in OPWFM mode with next period update.

Figure 334. OPWFM with immediate update

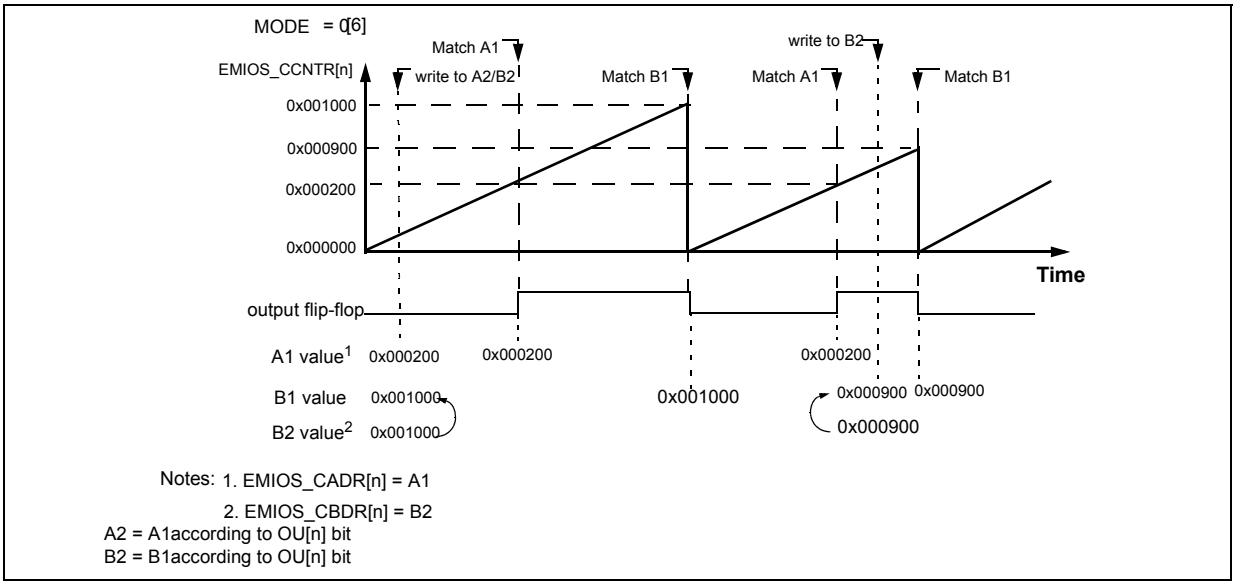
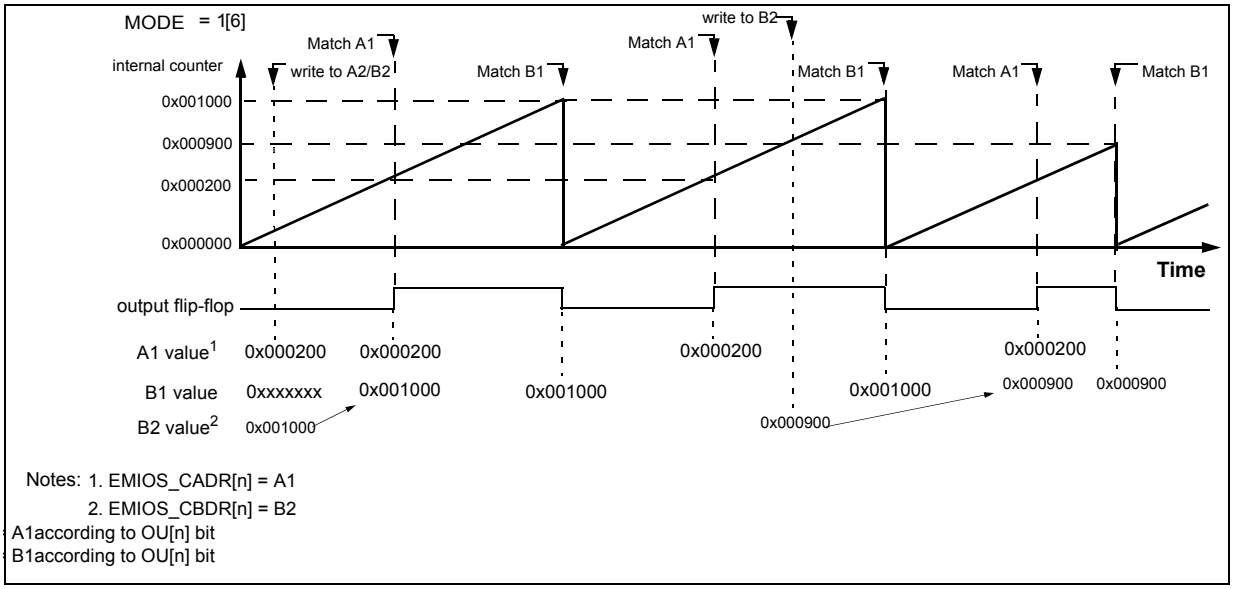


Figure 335. OPWFM with next period update



21.4.1.1.14 Output Pulse Width and Frequency Modulation Buffered (OPWFMB) Mode

This mode (MODE[0:6]=10110b0) provides waveforms with variable duty cycle and frequency. The internal channel counter is automatically selected as the time base when this mode is selected. A1 register indicates the duty cycle and B1 register the frequency. Both A1 and B1 registers are double buffered to allow smooth signal generation when changing the registers values on the fly. 0% and 100% duty cycles are supported.

At OPWFMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOS_CCR[n] register.

In order to provide smooth and consistent channel operation this mode differs substantially from the OPWFM mode. The main differences reside in the A1 and B1 registers update, on

the delay from the A1 match to the output pin transition and on the range of the internal counter values which starts from 0x1 up to B1 register value.

If when entering OPWFMB mode coming out from GPIO mode the internal counter value is not within that range then the B match will not occur causing the channel internal counter to wrap at the maximum counter value which is 0xff_ffff for a 24-bit counter. After the counter wrap occurs it returns to 0x1 and resume normal OPWFMB mode operation. Thus in order to avoid the counter wrap condition make sure its value is within the 0x1 to B1 register value range when the OPWFMB mode is entered.

When a match on comparator A occurs the output register is set to the value of EDPOL. When a match on comparator B occurs the output register is set to the complement of EDPOL. B1 match also causes the internal counter to transition to 0x1, thus restarting the counter cycle.

Only values greater than 0x1 are allowed to be written to B1 register. Loading values other than those leads to unpredictable results.

Figure 336 describes the operation of the OPWFMB mode regarding output pin transitions and A1/B1 registers match events. Note that the output pin transition occurs when the A1 or B1 match signal is deasserted which is indicated by the A1 match negedge detection signal. If register A1 is set to 0x4 the output pin transitions 4 counter periods after the cycle had started, plus one system clock cycle. Note that in the example shown in *Figure 336* the internal counter prescaler has a ratio of two.

Figure 336. OPWFMB A1 and B1 match to Output Register Delay

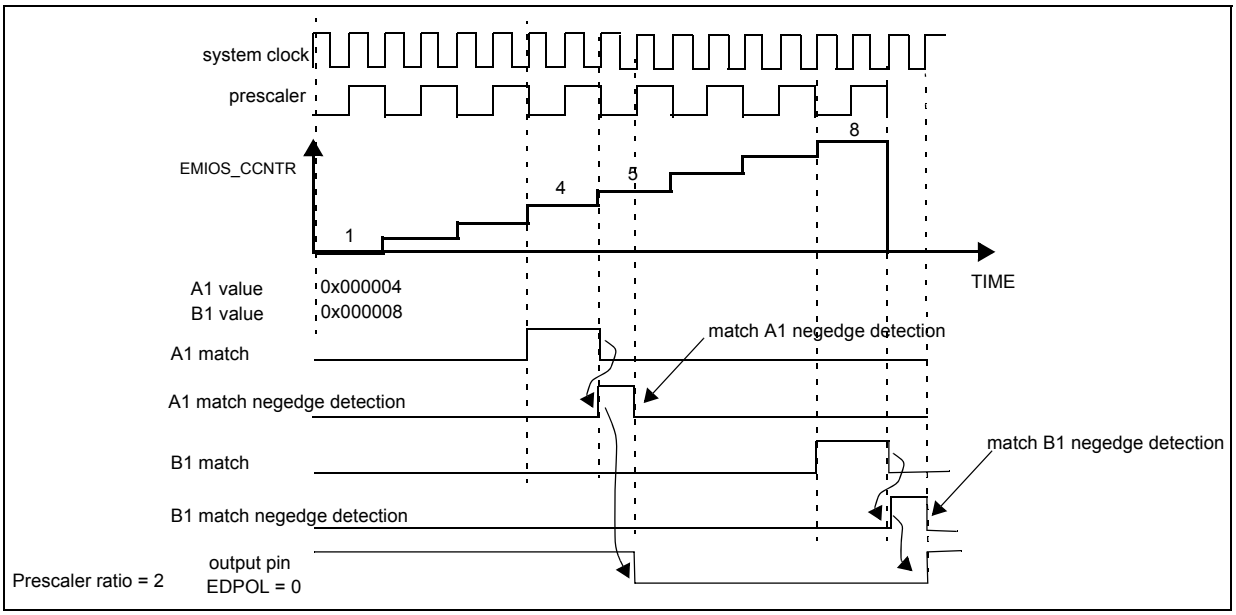


Figure 337 describes the generated output signal if A1 is set to 0x0. Since the counter does not reach zero in this mode, the channel internal logic infers a match as if A1=0x1 with the difference that in this case, the posedge of the match signal is used to trigger the output pin transition instead of the negedge used when A1=0x1. Note that A1 posedge match signal from cycle **n+1** occurs at the same time as B1 negedge match signal from cycle **n**. This allows to use the A1 posedge match to mask the B1 negedge match when they occur at the same time. The result is that no transition occurs on the output flip-flop and a 0% duty cycle is generated.

Figure 337. OPWFMB Mode with A1 = 0 (0% duty cycle)

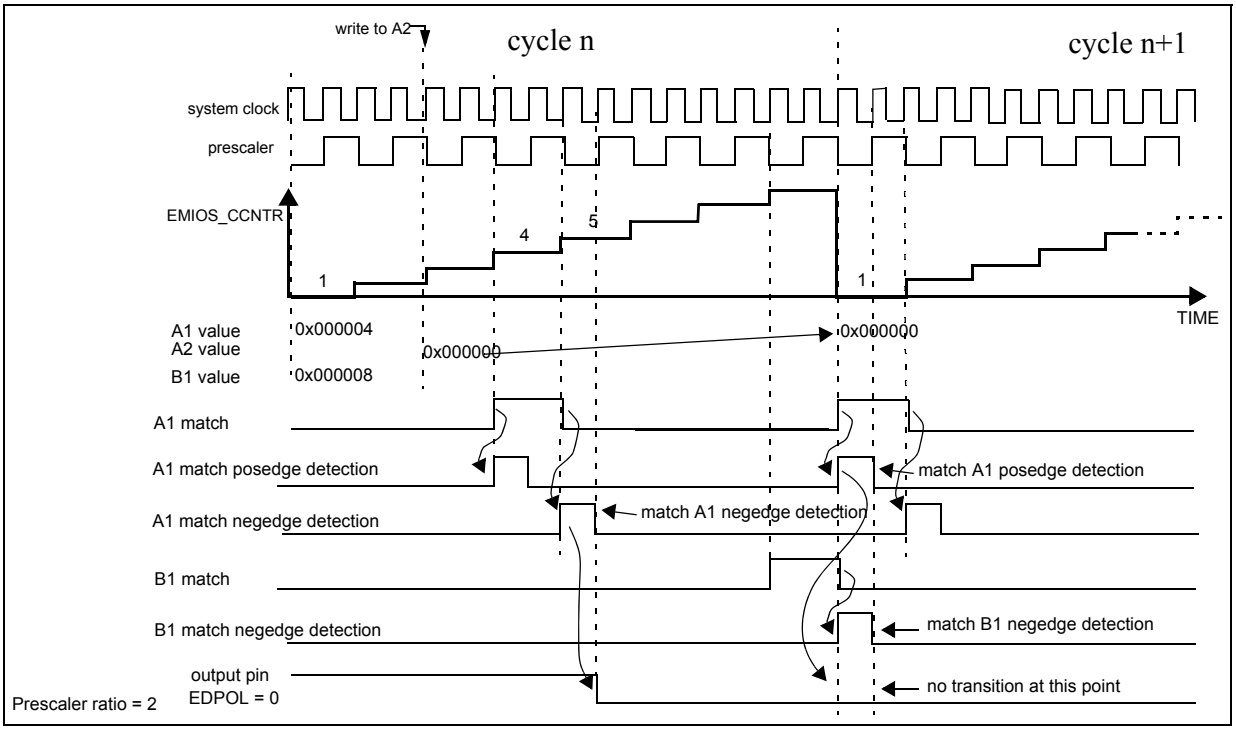


Figure 338 describes the timing for the A1 and B1 registers load. The A1 and B1 load use the same signal which is generated at the last system clock period of a counter cycle. Thus, A1 and B1 are updated respectively with A2 and B2 values at the same time that the counter (EMIOS_CCNTR[n]) is loaded with 0x1. This event is defined as the cycle boundary. The load signal pulse has the duration of one system clock period. If A2 and B2 are written within cycle **n** their values are available at A1 and B1, respectively, at the first clock of cycle **n+1** and the new values are used for matches at cycle **n+1**. The update disable bits OU[n] of EMIOS_OUDR register can be used to control the update of these registers, thus allowing to delay the A1 and B1 registers update for synchronization purposes.

In *Figure 338* it is assumed that both the channel and global prescalers are set to 0x1 (each divide ratio is two), meaning that the channel internal counter transitions at every four system clock cycles. FLAGs can be generated only on B1 matches when MODE[5] is cleared, or on both A1 and B1 matches when MODE[5] is set. Since B1 flag occurs at the cycle boundary, this flag can be used to indicate that A2 or B2 data written on cycle **n** were loaded to A1 or B1, respectively, thus generating matches in cycle **n+1**. Note that the FLAG has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.

Figure 338. OPWFMB A1 and B1 Registers Update and Flags

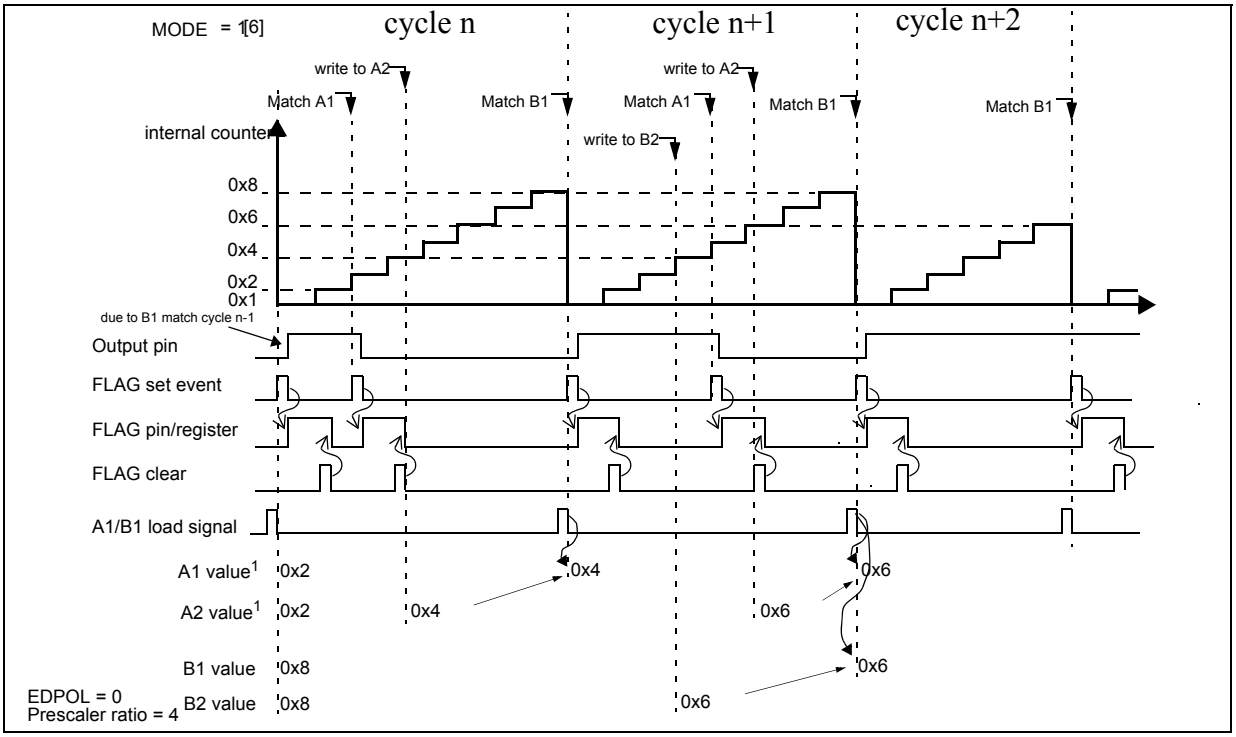
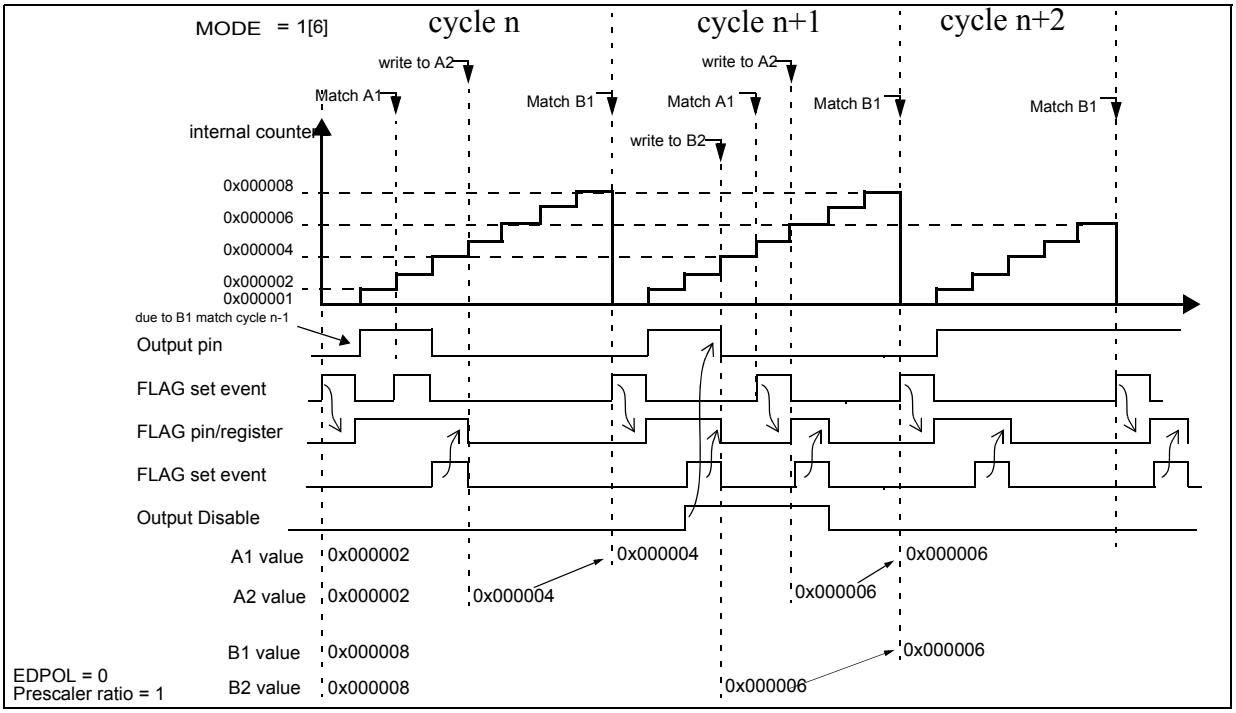


Figure 339 describes the operation of the Output Disable feature in OPWFMB mode. Differently from the OPWFM mode, the output disable forces the channel output flip-flop to EDPOL bit value. This functionality targets applications that use active high signals and a high to low transition at A1 match. In this case EDPOL should be set to 0. Note that both the channel and global prescalers are set to 0x0 (each divide ratio is one), meaning that the channel internal counter transitions at every system clock cycle.

Figure 339. OPWFMB Mode with Active Output Disable



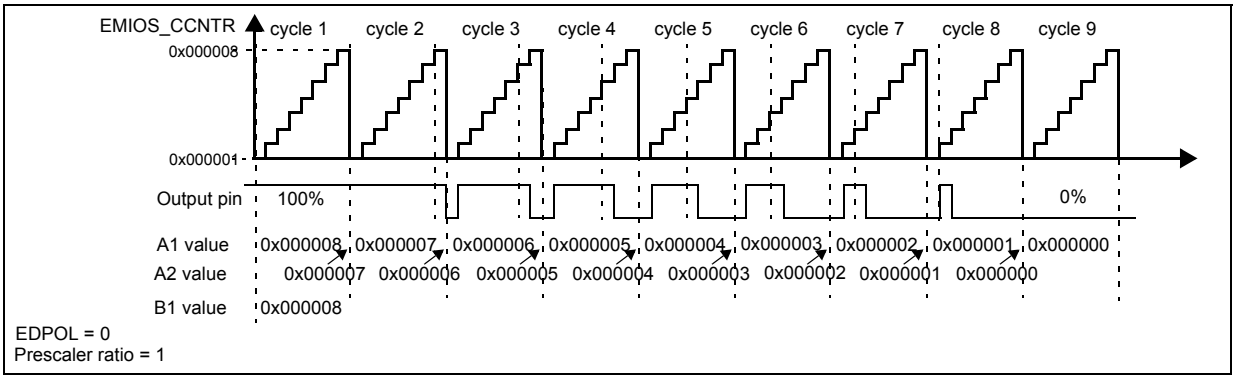
Note that the output disable has a synchronous operation, meaning that the assertion of the Output Disable input pin causes the channel output flip-flop to transition to EDPOL at the next system clock cycle. If the Output Disable input is deasserted the output pin transition at the following A1 or B1 match.

In *Figure 339* it is assumed that the Output Disable input is enabled and selected for the Channel. Please, refer to *Section 21.3.2.8: eMIOS200 UC Control Register (EMIOS_CCR[n])* for a detailed description about the ODIS and ODISSL bits, respectively enable and selection of the Output Disable inputs.

The FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on comparators A or B respectively. Similarly to a B1 match FORCMB sets the internal counter to 0x1. The FLAG bit is not set by the FORCMA or FORCMB bits being asserted.

Figure 340 describes the generation of 100% and 0% duty cycle signals. It is assumed EDPOL =0 and the resultant prescaler value is 1. Initially A1=0x8 and B1=0x8. In this case, B1 match has precedence over A1 match, thus the output flip-flop is set to the complement of EDPOL bit. This cycle corresponds to a 100% duty cycle signal. The same output signal can be generated for any A1 value greater or equal to B1.

Figure 340. OPWFMB Mode from 100% to 0% Duty Cycle



A 0% duty cycle signal is generated if A1=0x0 as shown in *Figure 340* cycle 9. In this case B1=0x8 match from cycle 8 occurs at the same time as the A1=0x0 match from cycle 9. Please, refer to *Figure 337* for a description of the A1 and B1 match generation. In this case A1 match has precedence over B1 match and the output signal transitions to EDPOL.

21.4.1.1.15 Center Aligned Output Pulse Width Modulation with Dead-Time (OPWMC) Mode

This operation mode generates a center aligned PWM with dead time insertion in the leading (MODE[0:6]=00111b1) or trailing edge (MODE[0:6]=00111b0).

The selected counter bus must be running an up/down time base, as shown in *Figure 329*. BSL[0:1] bits select the time base. Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base. Register B1 contains the dead time value and is compared with the internal counter. For a leading edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. Mode[6] bit selects between trailing and leading dead time insertion, respectively.

Note: The internal counter may be running in the internal prescaler ratio, while the selected time base may be running in a different prescaler ratio. The output signal may produce an unexpected output if the dead time interval is greater than the duty cycle of the PWM signal.

When OPWMC mode is entered, coming out from GPIO mode, the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[n] register.

When operating with leading edge dead time insertion, the first match between A1 and the selected time base clears the internal counter and switches the selected time base to the internal counter. When a match occurs between register B1 and the selected time base, the output flip-flop is set to the value of the EDPOL bit and the time base is switched to the selected counter bus. In the next match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

When operating with trailing edge dead time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit. In the next match between register A1 and the selected time base, the internal counter is cleared and the selected time base is switched to the internal counter. When a match occurs between register B1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit and the time base is switched to the selected counter bus. This sequence repeats continuously.

FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or in both edges, when MODE[5] is set.

At any time, the FORCMA or FORCMB bits are equivalent to a successful comparison on comparator A or B with the exception that the FLAG bit is not set.

Note: When in freeze state, the FORCMA or FORCMB bits only allow the software to force the output flip-flop to the level corresponding of a match on A or B respectively.

If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

In order to achieve a duty cycle of 100%, both registers A1 and B1 must be set to the same value. When a simultaneous match occurs between the selected time base and registers A1 and B1, the output flip-flop is set to the value of EDPOL bit and the selected time base switches to the selected counter bus, allowing a new cycle to begin at any time, as previously described. 0% duty cycle is possible by writing 0x0 to register A. When a match occurs, the output flip-flop is set to the complement of EDPOL bit and the selected time base switches to the selected counter bus, allowing a new cycle to begin at any time, as previously described. In both cases, FLAG is generated regardless of MODE[5] bit.

Note: If A1 and B1 are set to the 0x0, a 0% duty cycle waveform is produced.

Figure 341 and *Figure 342* show the Unified Channel running in OPWMC with leading and trailing dead time, respectively.

Figure 341. OPWMC with leading dead time insertion

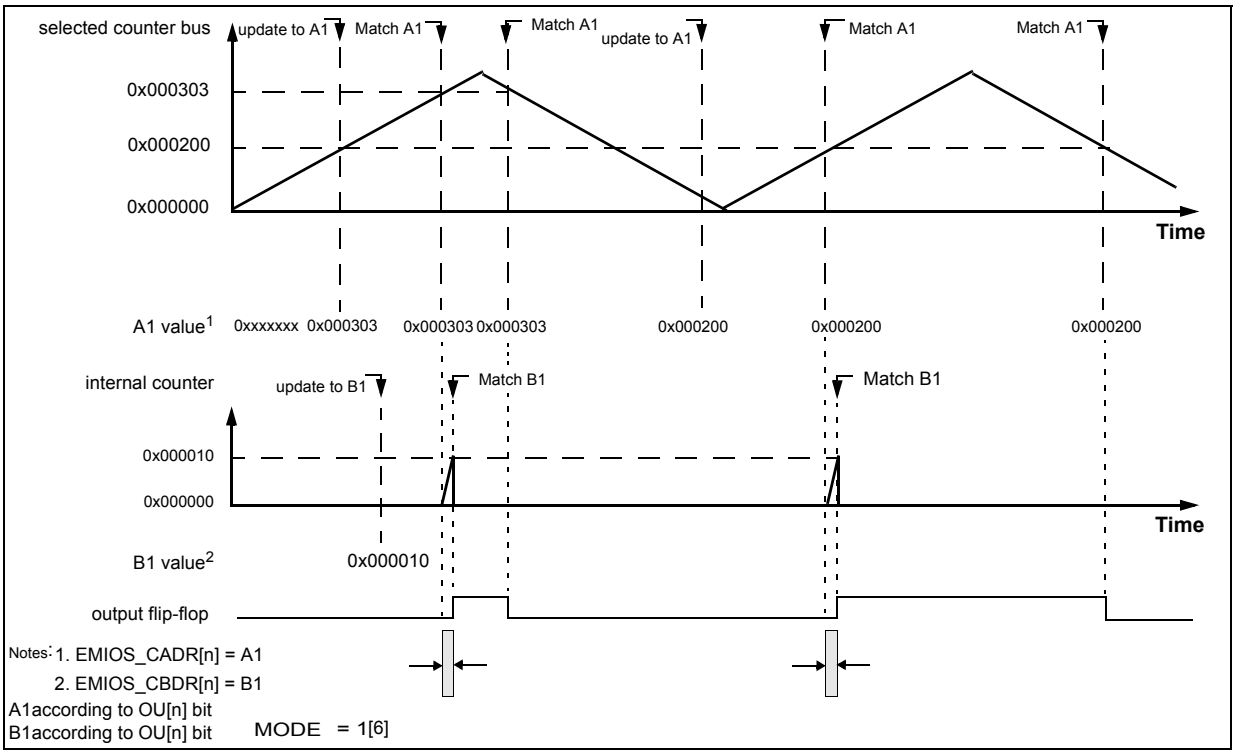
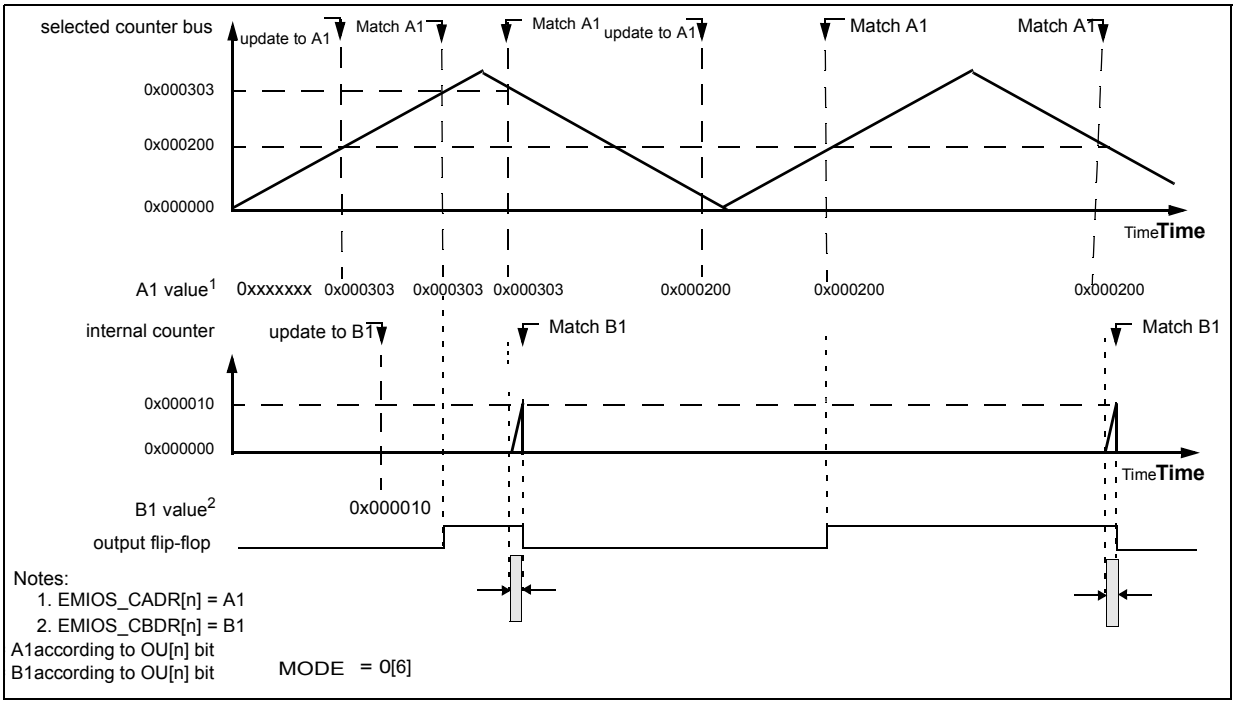


Figure 342. OPWMC with trailing dead time insertion



21.4.1.1.16 Center Aligned Output PWM Buffered with Dead-Time (OPWMCB) Mode

This operation mode generates a center aligned PWM with dead time insertion to the leading (MODE[0:6]=10111b1) or trailing edge (MODE[0:6]=10111b0). A1 and B1 registers are double buffered to allow smooth output signal generation when changing A2 or B2 registers values on the fly.

BSL[0:1] bits select the time base. The time base selected for a channel configured to OPWMCB mode should be a channel configured to MCB Up/Down mode, as shown in *Figure 331*. It is recommended to start the MCB channel time base after the OPWMCB mode is entered in order to avoid missing A matches at the very first duty cycle.

Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base.

Register B1 contains the dead time value and is compared against the internal counter. For a leading edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. Mode[6] bit selects between trailing and leading dead time insertion, respectively.

Note: The internal counter runs in the internal prescaler ratio, while the selected time base may be running in a different prescaler ratio.

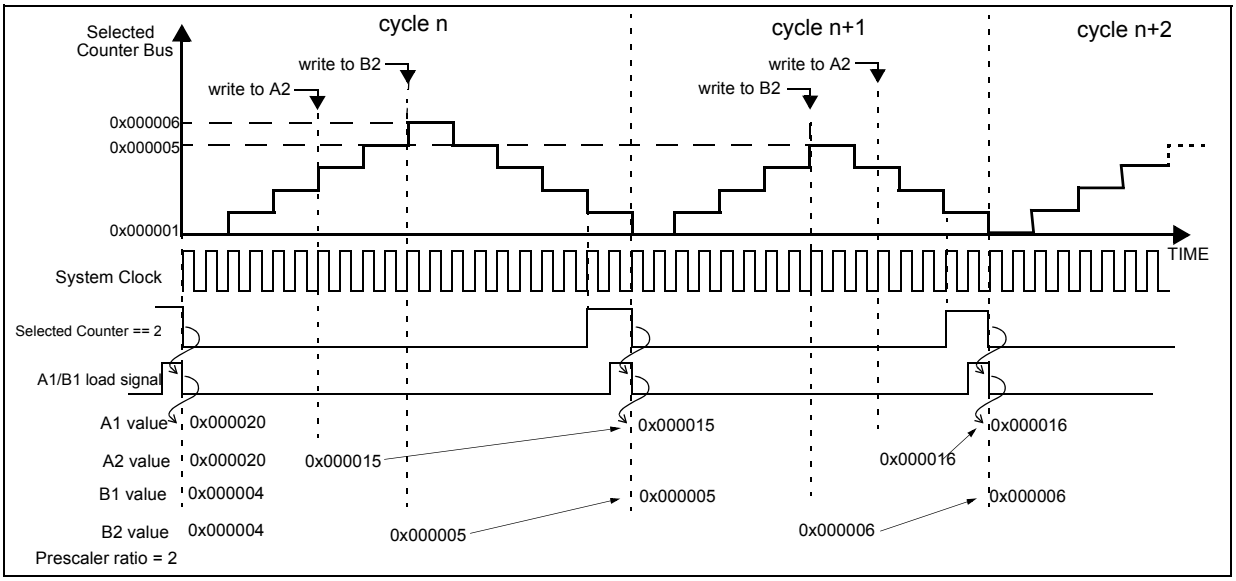
When OPWMCB mode is entered, coming out from GPIO mode, the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[n] register.

The following basic steps summarize proper OPWMCB startup, assuming the channels are initially in GPIO mode:

1. *[global]* Disable Global Prescaler.
2. *[MCB channel]* Disable Channel Prescaler.
3. *[MCB channel]* Write 0x1 at internal counter.
4. *[MCB channel]* Set A register.
5. *[MCB channel]* Set channel to MCB Up mode.
6. *[MCB channel]* Set prescaler ratio.
7. *[MCB channel]* Enable Channel Prescaler.
8. *[OPWMCB channel]* Disable Channel Prescaler.
9. *[OPWMCB channel]* Set A register.
10. *[OPWMCB channel]* Set B register.
11. *[OPWMCB channel]* Select time base input through BSL[1:0] bits.
12. *[OPWMCB channel]* Enter OPWMCB mode.
13. *[OPWMCB channel]* Set prescaler ratio.
14. *[OPWMCB channel]* Enable Channel Prescaler.
15. *[global]* Enable Global Prescaler.

Figure 343 describes the load of A1 and B1 registers which occurs when the selected counter bus transitions from 0x2 to 0x1. This event defines the cycle boundary. Note that values written to A2 or B2 within cycle **n** are loaded into A1 or B1 registers, respectively, and used to generate matches in cycle **n+1**.

Figure 343. OPWMCB A1 and B1 registers load



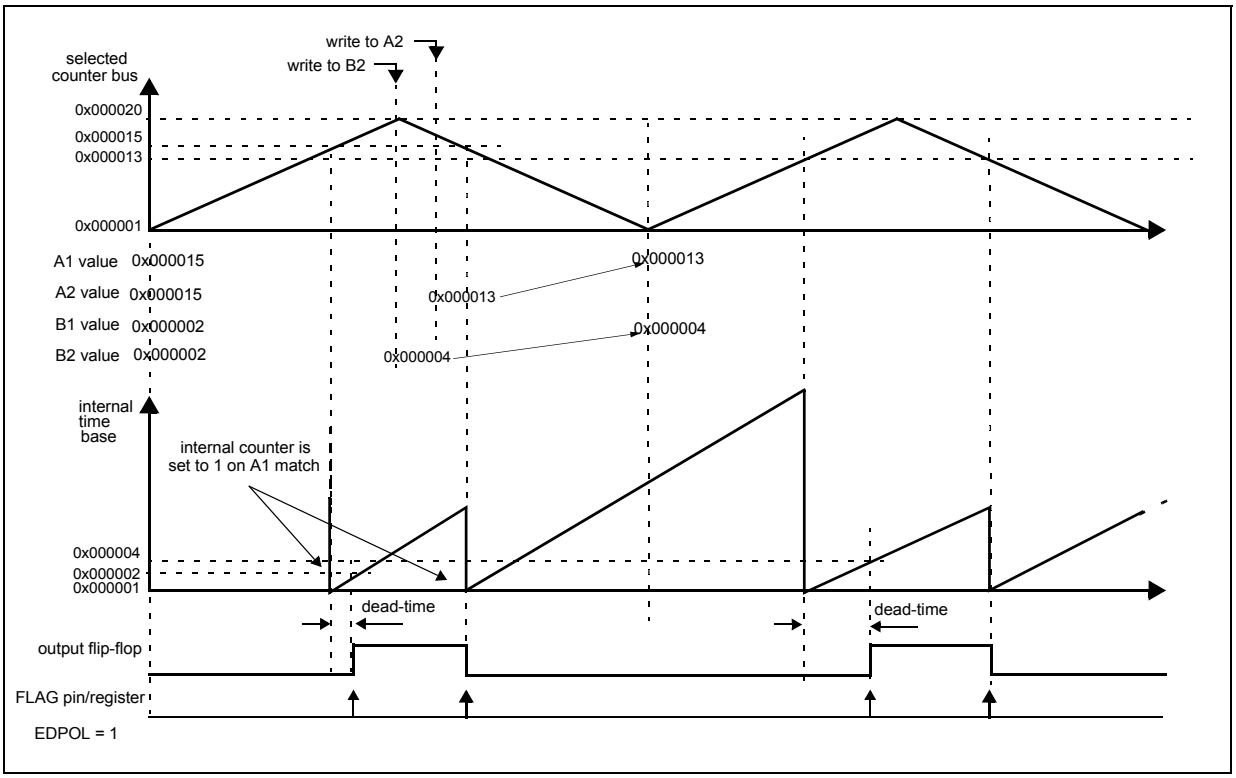
The OU[n] bit of EMIOS_OUDR register can be used to disable the A1 and B1 updates, thus allowing to synchronize the load on these registers with the load of A1 or B1 registers in others channels. Note that using the update disable bit A1 and B1 registers can be updated at the same counter cycle thus allowing to change both registers at the same time.

In this mode A1 matches always sets the internal counter to 0x1. When operating with leading edge dead time insertion the first A1 match sets the internal counter to 0x1. When a match occurs between register B1 and the internal time base, the output flip-flop is set to the value of the EDPOL bit. In the following match between register A1 and the selected time

base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously. The internal counter should not reach 0x0 as consequence of a rollover. In order to avoid it the user should not write to EMIOS_CBDR register a value greater than twice the difference between external count up limit and EMIOS_CADR value.

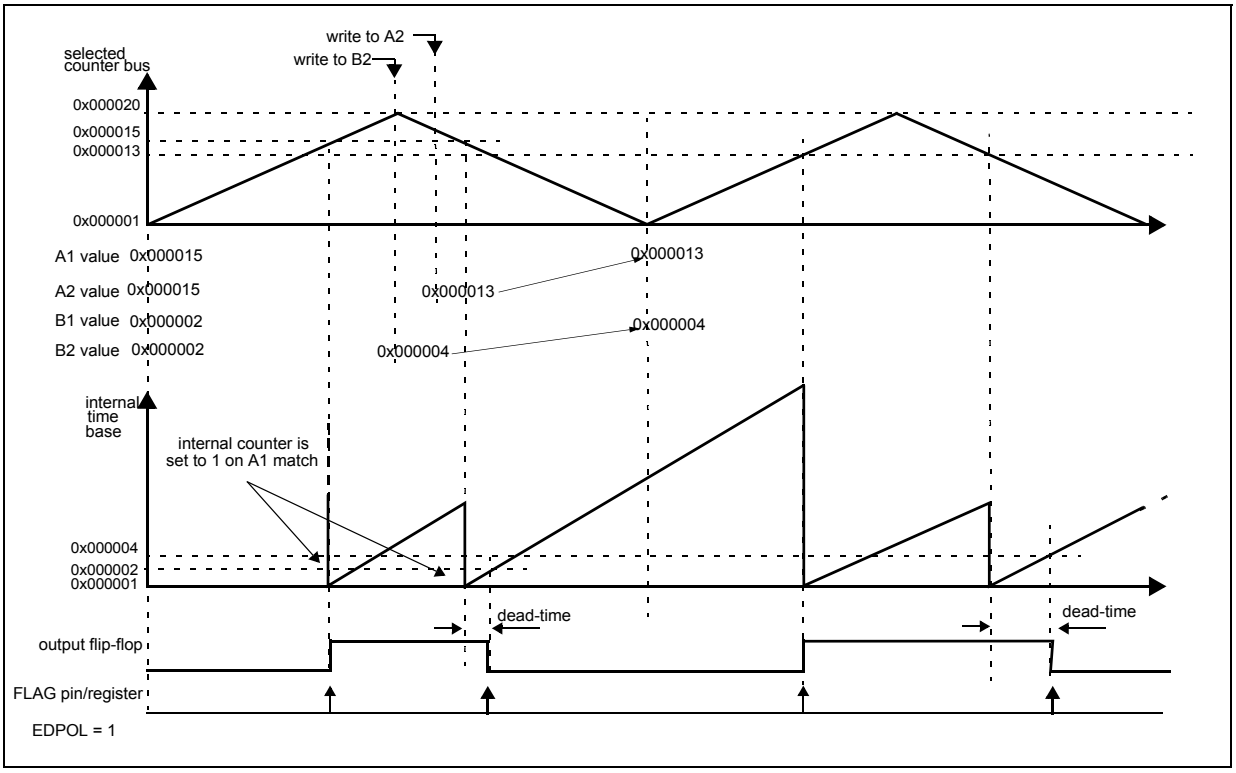
Figure 344 shows two cycles of a Center Aligned PWM signal. Note that both A1 and B1 register values are changing within the same cycle which allows to vary at the same time the duty cycle and dead time values.

Figure 344. OPWMCB with Lead Dead Time Insertion



When operating with trailing edge dead time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit and sets the internal counter to 0x1. In the second match between register A1 and the selected time base, the internal counter is set to 0x1 and B1 matches are enabled. When the match between register B1 and the selected time base occurs the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

Figure 345. OPWMCB with Trail Dead Time Insertion



FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or in both edges, when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

Note: In OPWMCB mode FORCMA and FORCMB do not have the same behavior as a regular match. Instead they force the output flip-flop to constant value which depends upon the selected dead time insertion mode, lead or trail and the value of the EDPOL bit.

FORCMA has different behaviors depending upon the selected dead time insertion mode, lead or trail. In lead dead time insertion FORCMA force a transition in the output flip-flop to the opposite of EDPOL. In trail dead time insertion the output flip-flop is forced to the value of EDPOL bit.

If FORCMB bit is set, the output flip-flop value depends upon the selected dead time insertion mode. In lead dead time insertion FORCMB forces the output flip-flop to transition to EDPOL bit value. In trail dead time insertion the output flip-flop is forced to the opposite of EDPOL bit value.

Note: FORCMA bit set does not set the internal time-base to 0x1 as a regular A1 match.

The FLAG bit is not set either in case of a FORCMA or FORCMB or even if both forces are issued at the same time.

Note: FORCMA and FORCMB have the same behavior even in Freeze or normal mode regarding the output pin transition.

When *FORCMA* is issued along with *FORCMB* the output flip-flop is set to the opposite of EDPOL bit value. This is equivalent of saying that.*FORCMA* has precedence over *FORCMB*

when lead dead time insertion is selected and *FORCMB* has precedence over *FORCMA* when trail dead time insertion is selected.

Duty cycle from 0% to 100% can be generated by setting appropriate values to A1 and B1 registers relatively to the period of the external time base. Setting A1=1 generates a 100% duty cycle waveform. Assuming EDPOL is set to one and OPWMCB mode with trail dead time insertion, 100% duty cycle signals can be generated if B1 occurs at or after the cycle boundary (external counter = 1). If A1 is greater than the maximum value of the selected counter bus period, then a 0% duty cycle is produced, only if the pin starts the current cycle in the opposite of EDPOL value. In case of 100% duty cycle, the transition from EDPOL to the opposite of EDPOL may be obtained by forcing pin, using FORCMA and/or FORCMB.

Note: If A1 is set to 0x1 at OPWMCB entry the 100% duty cycle may not be obtained in the very first PWM cycle due to the pin condition at mode entry.

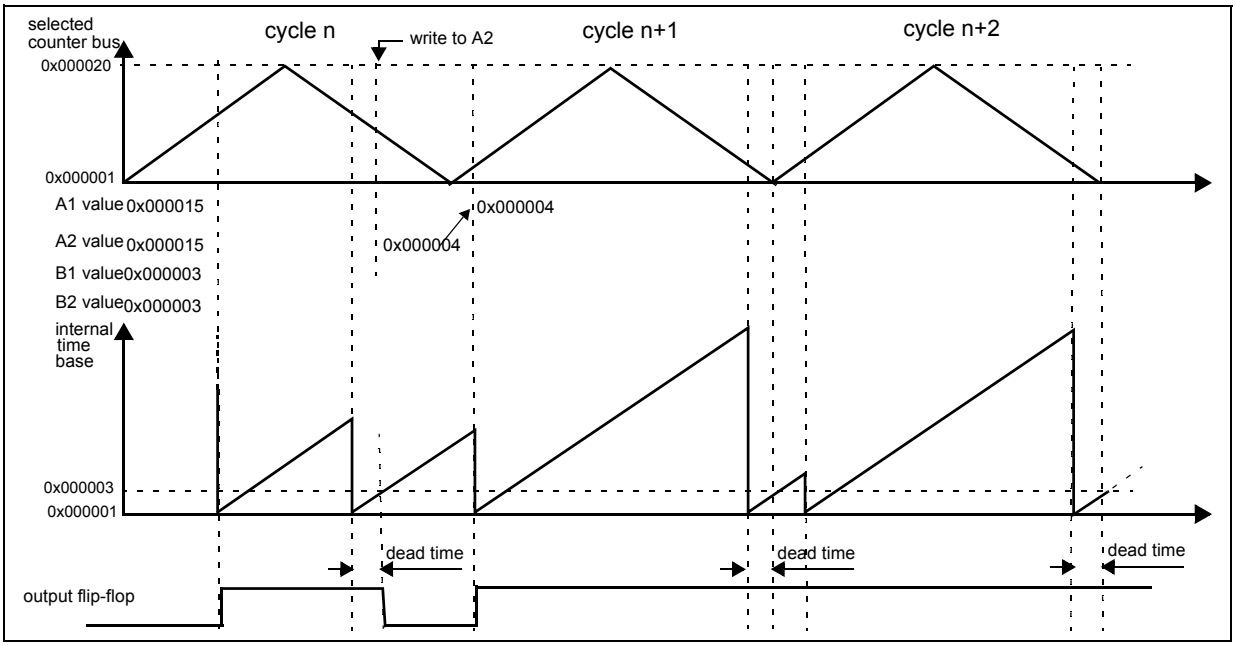
Only values different than 0x0 are allowed to be written to A1 register. If 0x0 is loaded to A1 the results are unpredictable.

Note: A special case occurs when A1 is set to (external counter bus period)/2, which is the maximum value of the external counter. In this case the output flip-flop is constantly set to the EDPOL bit value.

The internal channel logic prevents matches from one cycle to propagate to the next cycle. In trail dead time insertion B1 match from cycle **n** could eventually cross the cycle boundary and occur in cycle **n+1**. In this case B1 match is masked out and does not cause the output flip-flop to transition. Therefore matches in cycle **n+1** are not affected by the late B1 matches from cycle **n**.

Figure 346 shows a 100% duty cycle output signal generated by setting A1=4 and B1=3. In this case the trailing edge is positioned at the boundary of cycle **n+1**, which is actually considered to belong to cycle **n+2** and therefore does not cause the output flip-flip to transition.

Figure 346. OPWMCB with 100% Duty Cycle (A1=4 and B1=3)



The output disable feature, if enabled, causes the output flip-flop to transition to the EDPOL inverted state. This feature allows an application to force the channel output pin to a “safe” state. The internal channel matches continue to occur even in this case, thus generating Flags. As soon as the output disable is deasserted the channel output pin is again controlled by A1 and B1 matches. Note that this process is synchronous, meaning that the output channel pin transitions only on system clock edges.

It is important to notice that, such as in OPWMB and OPWFMB modes, the match signal used to set or clear the channel output flip-flop is generated on the deassertion of the channel combinational comparator output signal which compares the selected time base with A1 or B1 register values. Please refer to *Figure 336* which describes the delay from matches to output flip-flop transition in OPWFMB mode. The operation of OPWMCB mode is similar to OPWFMB regarding matches and output pin transition.

21.4.1.1.17 Output Pulse Width Modulation (OPWM) Mode

Registers A1 and B1 define the leading and trailing edges of the PWM output pulse, respectively. MODE[6] bit controls the transfer from register B2 to B1, which can be done either immediately (MODE[6] cleared, MODE[0:6]=01000b0), providing the fastest change in the duty cycle, or at every match of register A1 (MODE[6] set, MODE[0:6]=01000b1).

The value loaded in register A1 is compared with the value on the selected time base. When a match on comparator A occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit.

FLAG can be generated at match B, when MODE[5] is cleared, or in both matches, when MODE[5] is set.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Note that FLAG bit is not set by the FORCMA and FORCMB operations.

If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

At OPWM mode entry the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[n] register.

In order to achieve 100% duty cycle, both registers A1 and B1 must be set to the same value. When a simultaneous match on comparators A and B occur, the output flip-flop is set at every period to the value of EDPOL bit. 0% duty cycle is possible by writing 0x0 to register A. When a match occurs, the output flip-flop is set at every period to the complement of EDPOL bit. The transfer from register B2 to B1 is still controlled by MODE[6] bit.

Note: If A1 and B1 are set to the 0x0, a 0% duty cycle waveform is produced.

Figure 347 and *Figure 348* show the Unified Channel running in OPWM with immediate update and next period update, respectively.

Figure 347. PWM with immediate update

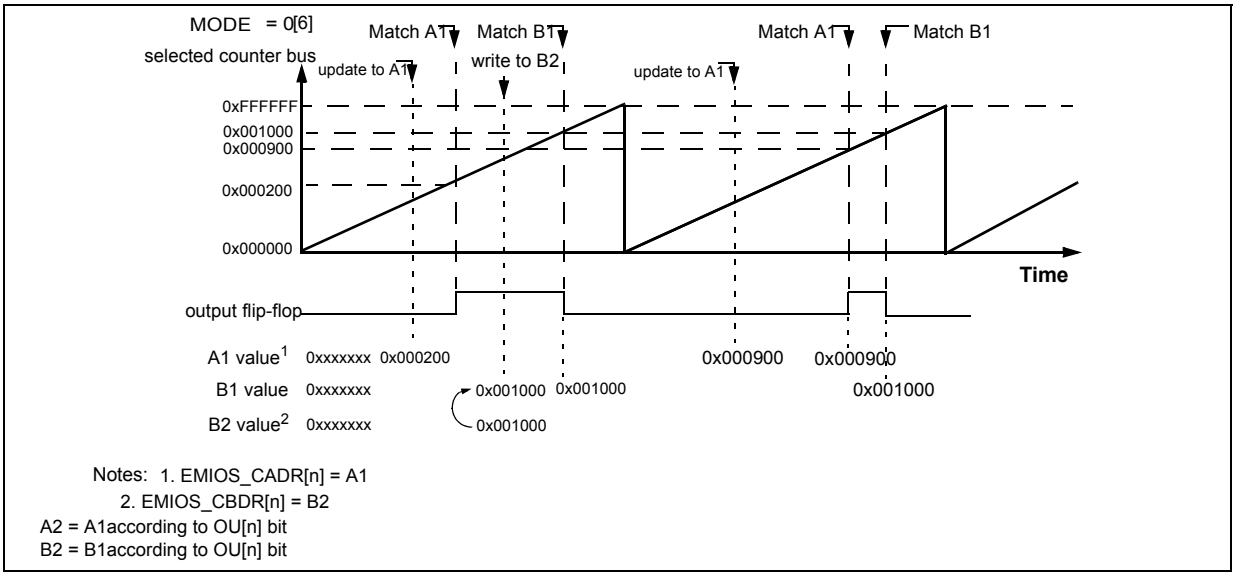
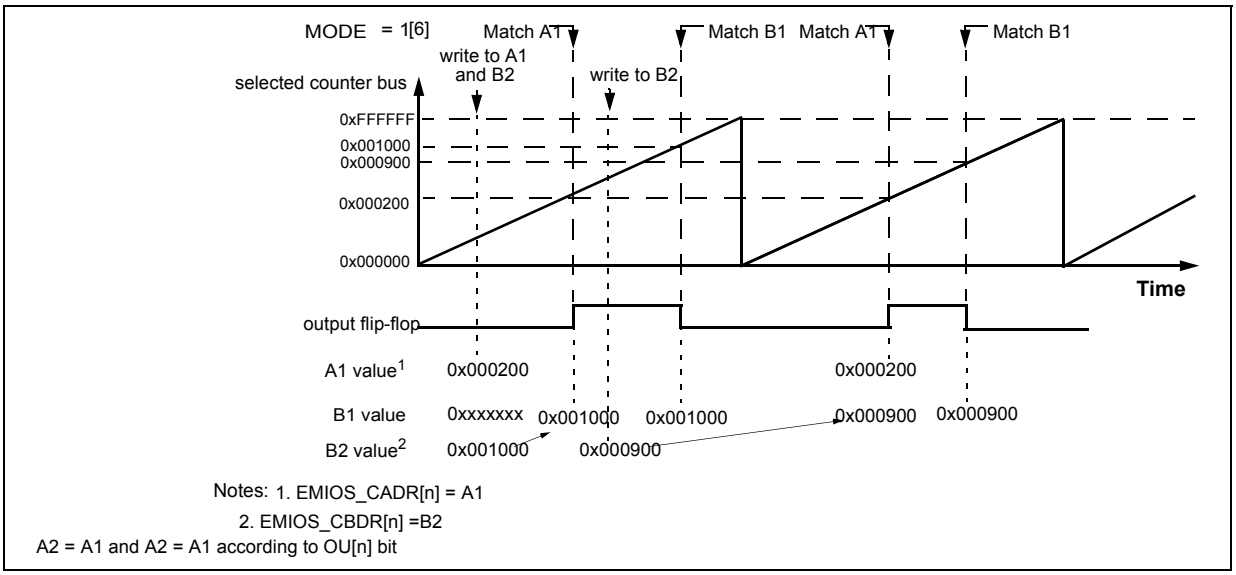


Figure 348. PWM with next period update



21.4.1.1.18 Output Pulse Width Modulation Buffered (OPWMB) Mode

OPWMB mode (MODE[0:6]=11000b0) is used to generate pulses with programmable leading and trailing edge placement. An external counter driven in MCB Up mode must be selected from one of the counter buses. A1 register value defines the first edge and B1 the second edge. The output signal polarity is defined by the EDPOL bit. If EDPOL is zero, a negative edge occurs when A1 matches the selected counter bus and a positive edge occurs when B1 matches the selected counter bus.

The A1 and B1 registers are double buffered and updated from A2 and B2, respectively, at the cycle boundary. The load operation is similar to the OPWFMB mode. Please refer to *Figure 338* for more information about A1 and B1 registers update.

FLAG can be generated at B1 matches, when MODE[5] is cleared, or in both A1 and B1 matches, when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A1 or B1 respectively. FLAG bit is not set by the FORCMA and FORCMB operations.

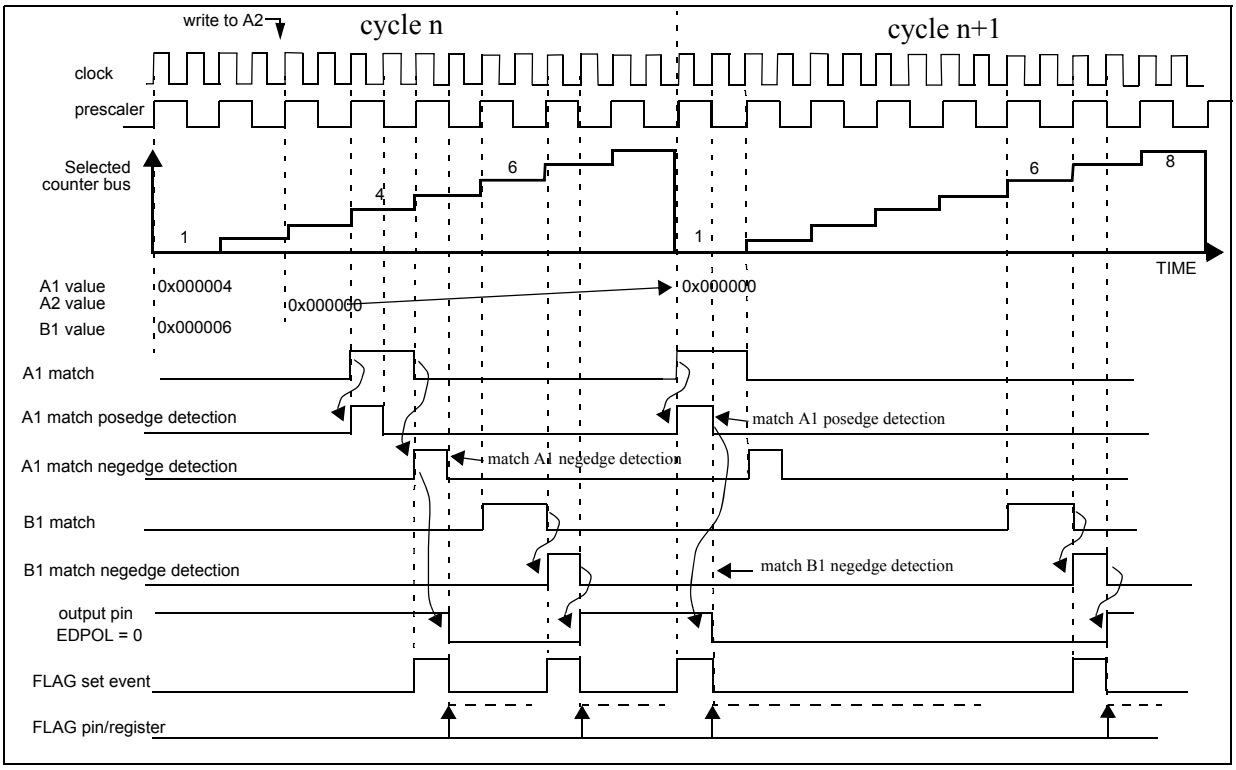
At OPWMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOS_CCR[n] register.

Following are described some rules applicable to the OPWMB mode:

- B1 matches have precedence over A1 matches if they occur at the same time within the same counter cycle
- A1=0 match from cycle **n** has precedence over B1 match from cycle **n-1**
- A1 matches are masked out if they occur after B1 match within the same cycle
- Any value written to A2 or B2 on cycle **n** is loaded to A1 and B1 registers at the following cycle boundary (assuming OU[n] bit of EMIOS_OUDR register is not asserted). Thus the new values will be used for A1 and B1 matches in cycle **n+1**

Figure 349 describes the operation of the OPWMB mode regarding A1 and B1 matches and the transition of the channel output pin. In this example EDPOL is set to zero.

Figure 349. OPWMB Mode Matches and Flags



Note that the output pin transitions are based on the negedges of the A1 and B1 match signals. *Figure 349* shows in cycle **n+1** the value of A1 register being set to zero. In this case the match posedge is used instead of the negedge to transition the output flip-flop.

Figure 350 describes the channel operation for 0% duty cycle. Note that the A1 match posedge signal occurs at the same time as the B1=0x8 negedge signal. In this case A1 match has precedence over B1 match, causing the output pin to remain at EDPOL bit value, thus generating a 0% duty cycle signal.

Figure 350. OPWMB Mode with 0% Duty Cycle

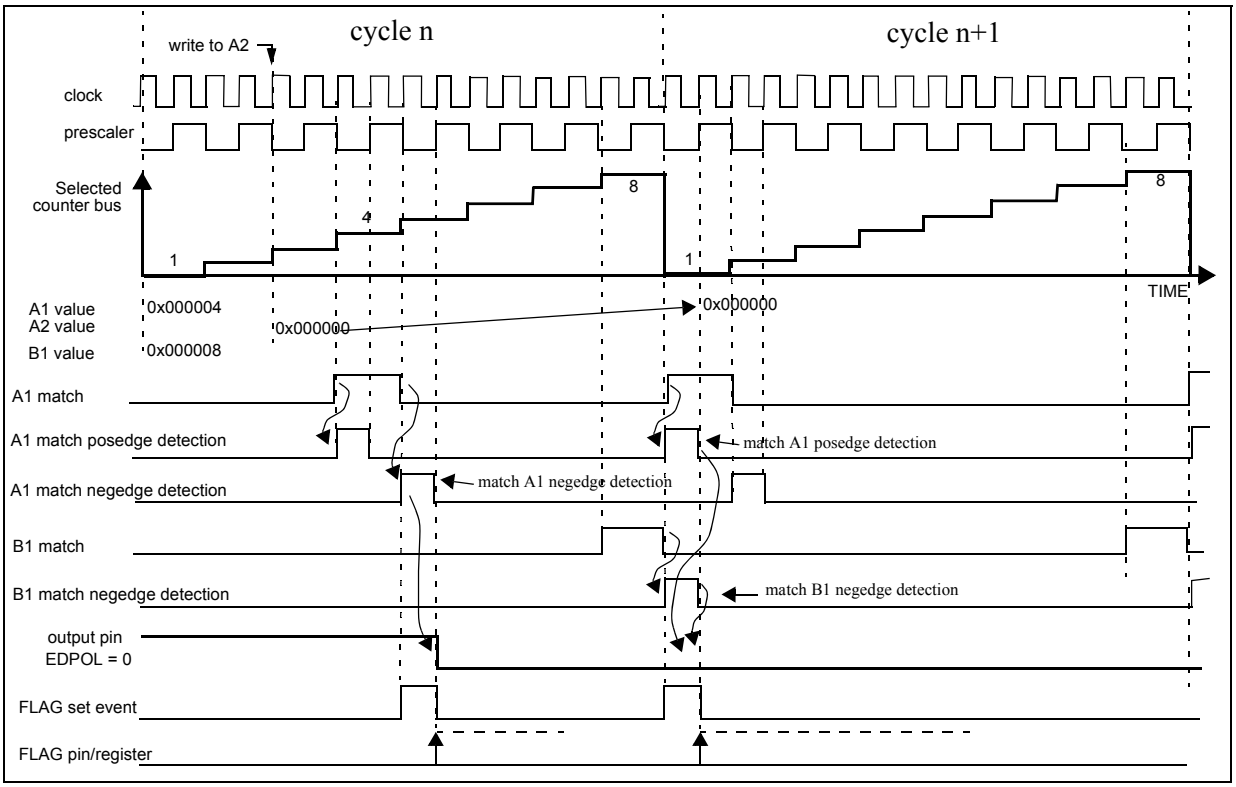


Figure 351 describes the operation of the OPWMB mode with the Output Disable signal being asserted. The Output Disable forces a transition in the output pin to the EDPOL bit value. After deasserted, the output disable allows the output pin to transition at the following A1 or B1 match. Note that the Output Disable does not modify the Flag bit behavior. Note that there is one system clock delay between the assertion of the output disable signal and the transition of the output pin to EDPOL.

Figure 351. OPWMB Mode with Active Output Disable

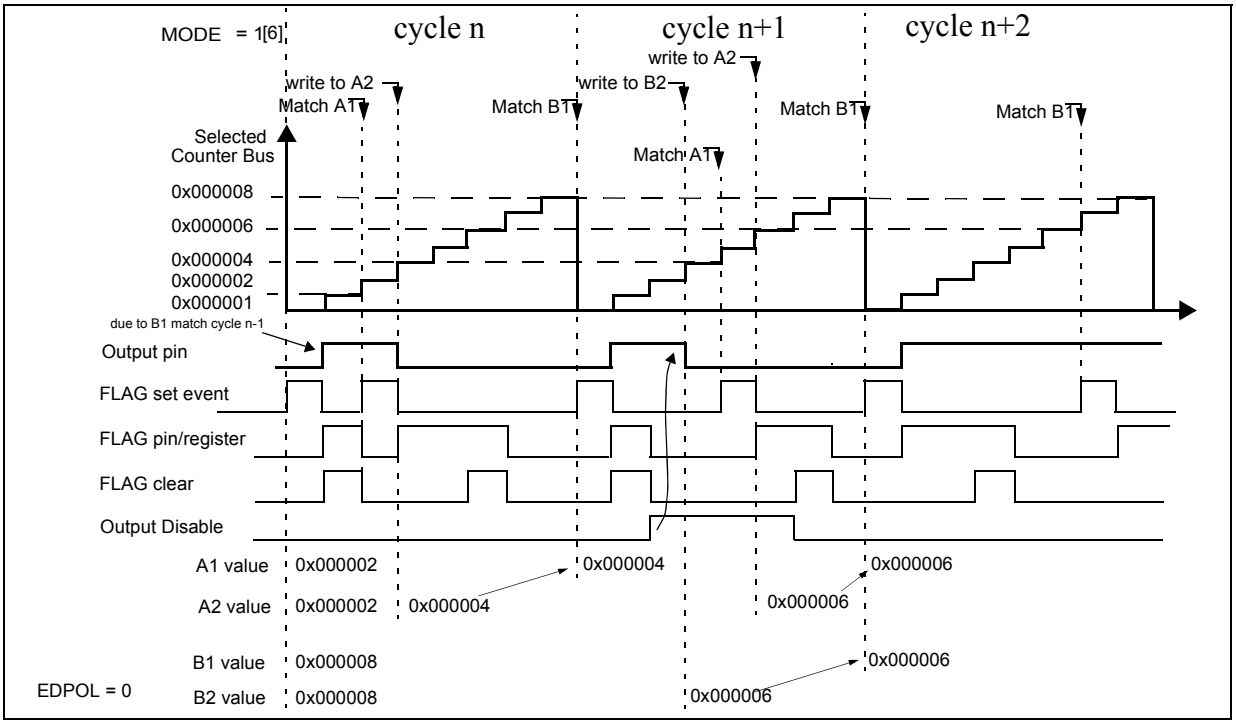
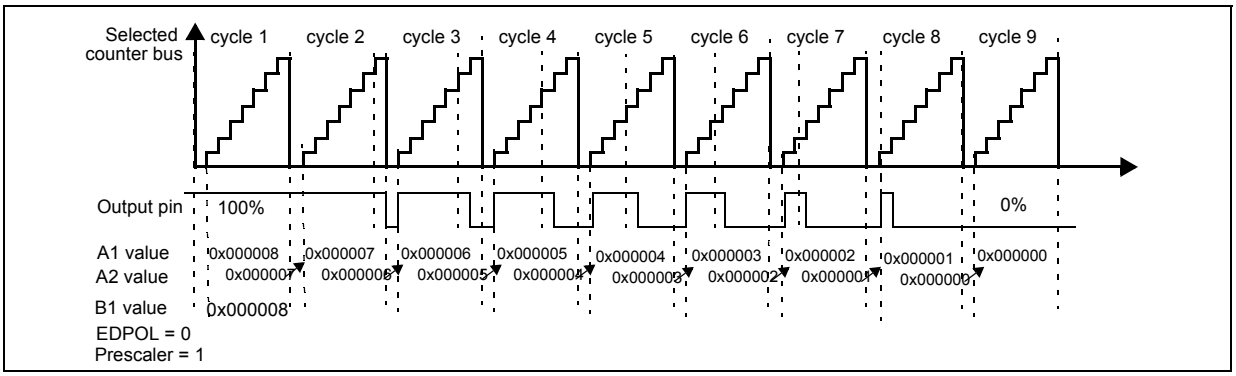


Figure 352 shows a waveform changing from 100% to 0% duty cycle. EDPOL in this case is zero. In this example B1 is programmed to the same value as the period of the external selected time base.

Figure 352. OPWMB Mode from 100% to 0% Duty Cycle



In *Figure 352* if B1 is set to a value lower than 0x8 it is not possible to achieve 0% duty cycle by only changing A1 register value. Since B1 matches have precedence over A1 matches the output pin transitions to the opposite of EDPOL bit at B1 match. Note also that if B1 is set to 0x9, for instance, B1 match does not occur, thus a 0% duty cycle signal is generated.

21.4.1.1.19 Output Pulse Width Modulation with Trigger (OPWMT) Mode

OPWMT mode (MODE[0:6]=0100110) is intended to support the generation of Pulse Width Modulation signals where the period is not modified while the signal is being output, but where the duty cycle will be varied and must not create glitches. The mode is intended to be used in conjunction with other channels executing in the same mode and sharing a common

timebase. It will support each channel with a fixed PWM leading edge position with respect to the other channels and the ability to generate a trigger signal at any point in the period that can be output from the module to initiate activity in other parts of the SoC such as starting ADC conversions.

An external counter driven in either MC Up or MCB Up mode must be selected from one of the counter buses.

Register A1 defines the leading edge of the PWM output pulse and as such the beginning of the PWM's period. This makes it possible to insure that the leading edge of multiple channels in OPWMT mode can occur at a specific time with respect to the other channels when using a shared timebase. This can allow the introduction of a fixed offset for each channel which can be particularly useful in the generation of lighting PWM control signals where it is desirable that edges are not coincident with each other to help eliminate noise generation. The value of register A1 represents the shift of the PWM channel with respect to the selected timebase. A1 can be configured with any value within the range of the selected time base. Note that registers loaded with 0x0 will not produce matches if the timebase is driven by a channel in MCB mode.

A1 is not buffered as the shift of a PWM channel must not be modified while the PWM signal is being generated. In case A1 is modified it is immediately updated and one PWM pulse could be lost.

EMIOS_CBDR[n] address gives access to B2 register for write and B1 register for read. Register B1 defines the trailing edge of the PWM output pulse and as such the duty cycle of the PWM signal. To synchronize B1 update with the PWM signal and so ensure a correct output pulse generation the transfer from B2 to B1 is done at every match of register A1. This behavior is the same as the OPWM mode with next period update.

EMIOS_OUDR register affects transfers between B2 and B1 only.

In order to account for the shift in the leading edge of the waveform defined by register A1 it will be necessary that the trailing edge, held in register B1, can roll over into the next period. This means that a match against the B1 register should not have to be qualified by a match in the A1 register. The impact of this would mean that incorrectly setting register B1 to a value less that register A1 will result in the output being held over a cycle boundary until the B1 value is encountered.

This mode provides a buffered update of the trailing edge by updating register B1 with register B2 contents only at a match of register A1.

The value loaded in register A1 is compared with the value on the selected time base. When a match on comparator A1 occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit.

Note that the output pin and flag transitions are based on the posedges of the A1, B1 and A2 match signals. Please refer to *Figure 349* in *Section 21.4.1.1.18: Output Pulse Width Modulation Buffered (OPWMB) Mode* for details on match posedge.

Register A2 defines the generation of a trigger event within the PWM period and A2 should be configured with any value within the range of the selected time base, otherwise no trigger will be generated. A match on the comparator will generate the FLAG signal but it has no effect on the PWM output signal generation. The typical setup to obtain a trigger with FLAG is enabling DMA and driving the channel's ipd_done input high.

A2 is not buffered and therefore its update is immediate. If the channel is running when a change is made this could cause either the loss of one trigger event or the generation of two

trigger events within the same period. Register A2 can be accessed by reading or writing the eMIOS200 UC Alternate A Register (EMIOS_ALTA) at UC[n] base address +0x14.

FLAG signal is set only at match on the comparator with A2. A match on the comparator with A1 or B1 or B2 has no effect on FLAG.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Any FORCMA and/or FORCMB has priority over any simultaneous match regarding to output pin transitions. Note that the load of B2 content on B1 register at an A match is not inhibited due to a simultaneous FORCMA/FORCMB assertion. If both FORCMA and FORCMB are asserted simultaneously the output pin goes to the opposite of EDPOL value such as if A1 and B1 registers had the same value. FORCMA assertion causes the transfer from register B2 to B1 such as a regular A match, regardless of FORCMB assertion.

If subsequent matches occur on comparators A1 and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

At OPWMT mode entry the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[n] register.

In order to achieve 0% duty cycle both registers A1 and B must be set to the same value. When a simultaneous match on comparators A and B occur, the output flip-flop is set at every period to the complement value of EDPOL.

In order to achieve 100% duty cycle the register B1 must be set to a value greater than maximum value of the selected time base. As a consequence if 100% duty cycle must be implemented the maximum counter value for the time base is 0xFFFFFE for a 24-bit counter and respectively 0xFFFE for a 16-bit counter. When a match on comparator A1 occurs the output flip-flop is set at every period to the value of EDPOL bit. The transfer from register B2 to B1 is still triggered by the match at comparator A.

As with other eMIOS200 mode, the OPWMT mode will implement the Output Disable function. Setting the ODIS bit in the eMIOS200 UC Control Register (EMIOS_CCR[n]) will enable the Output Disable function. If the selected Output Disable input signal is asserted for the channel, the output pin will go to the inverse of the EDPOL. The channel will continue to operate normally, although the output will be fixed. When the Output Disable input signal is negated, the output pin will return to operate as normal.

Figure 353 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and duty cycle update on next period update.

Figure 353. OPWMT example

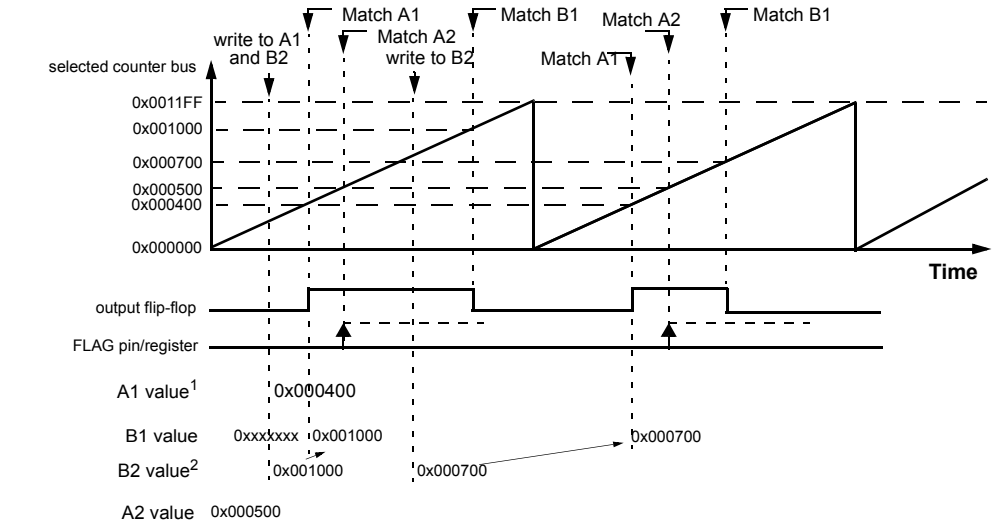


Figure 354 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 0% duty.

Figure 354. OPWMT with 0% Duty Cycle

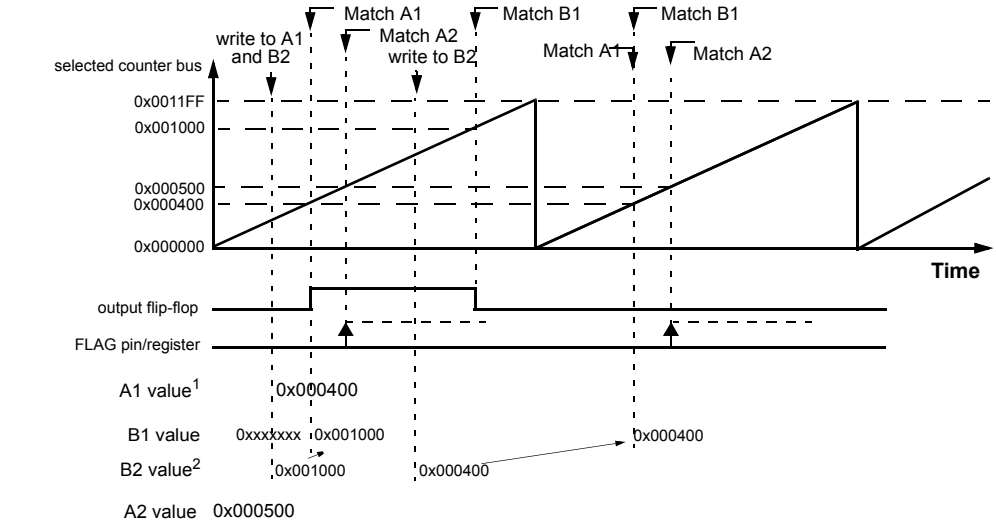
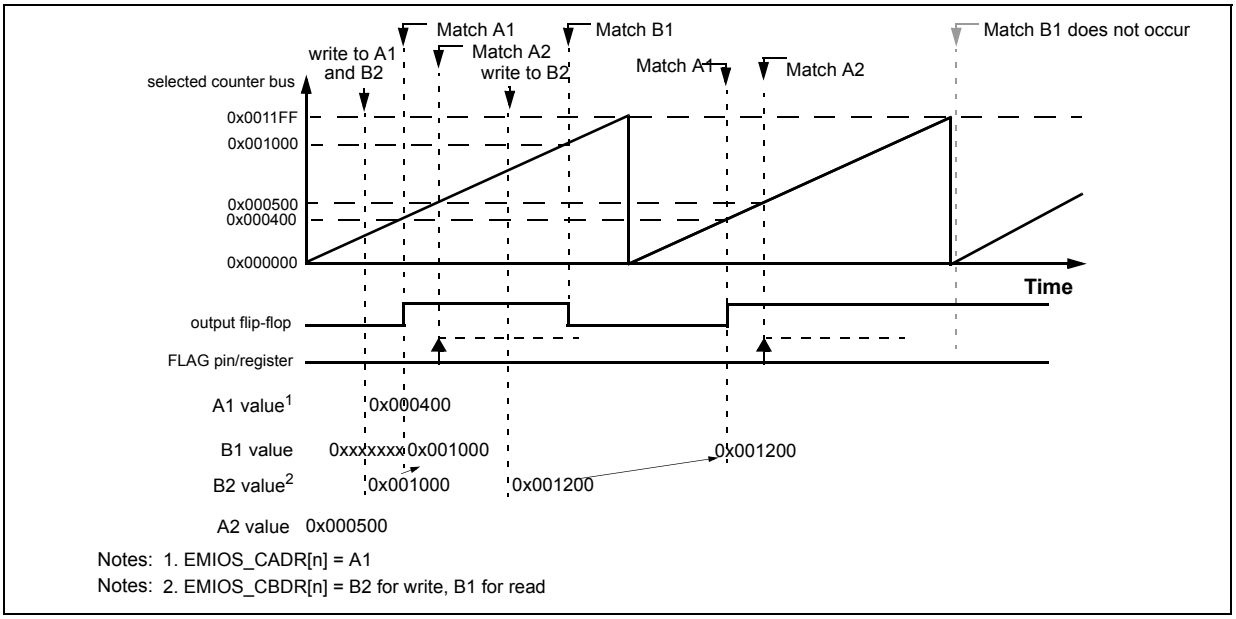


Figure 355 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 100% duty cycle.

Figure 355. OPWMT with 100% Duty Cycle

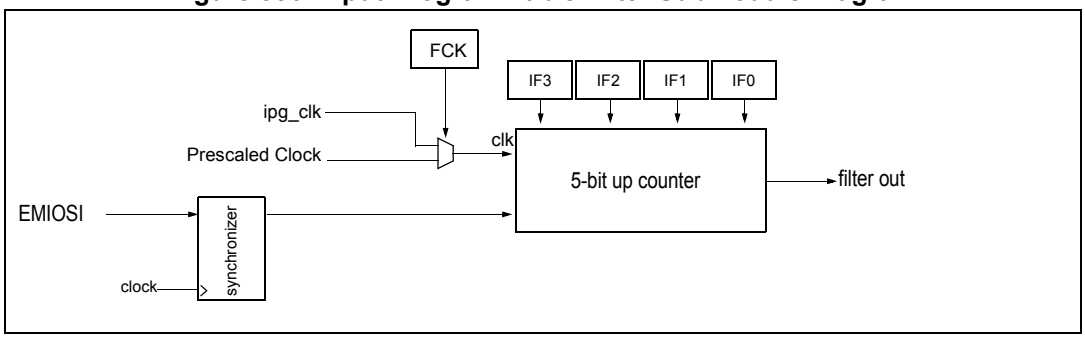


21.4.1.2 Input Programmable Filter (IPF)

The IPF ensures that only valid input pin transitions are received by the Unified Channel edge detector. A block diagram of the IPF is shown in *Figure 356*.

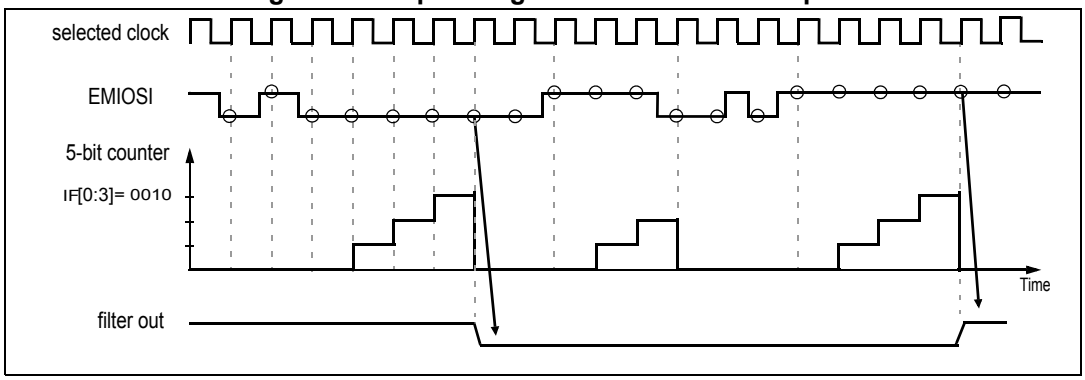
The IPF is a 5-bit programmable up counter that is incremented by the selected clock source, according to bits IF[0:3] in EMIOS_CCR[n] register.

Figure 356. Input Programmable Filter Submodule Diagram



The input signal is synchronized by system clock. When a state change occurs in this signal, the 5-bit counter starts counting up. As long as the new state is stable on the pin, the counter remains incrementing. If a counter overflows occurs, the new pin value is validated. In this case, it is transmitted as a pulse edge to the edge detector. If the opposite edge appears on the pin before validation (overflow), the counter is reset. At the next pin transition, the counter starts counting again. Any pulse that is shorter than a full range of the masked counter is regarded as a glitch and it is not passed on to the edge detector. A timing diagram of the input filter is shown in *Figure 357*.

Figure 357. Input Programmable Filter Example



The filter is not disabled during either freeze state or negated GTBE input.

21.4.1.3 Clock Prescaler (CP)

The CP divides the GCP output signal to generate a clock enable for the internal counter of the Unified Channels. The GCP output signal is prescaled by the value defined in *Table 219* according to the UCPRE[0:1] bits in EMIOS_CCR[n] register. The prescaler is enabled by setting the UCPREN bit in the EMIOS_CCR[n] and can be stopped at any time by clearing this bit, thereby stopping the internal counter in the Unified Channel.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write 0 at both GPREN bit in EMIOS_MCR register and UCPREN bit in EMIOS_CCR[n] register, thus disabling prescalers;
2. Write the desired value for prescaling rate at UCPRE[0:1] bits in EMIOS_CCR[n] register;
3. Enable channel prescaler by writing 1 at UCPREN bit in EMIOS_CCR[n] register;
4. Enable global prescaler by writing 1 at GPREN bit in EMIOS_MCR register.

The prescaler is not disabled during either freeze state or negated GTBE input.

21.4.1.4 Effect of Freeze on the Unified Channel

When in debug mode, FRZ bit in the EMIOS_MCR register and the FREN bit in the EMIOS_CCR[n] are both set, the internal counter and Unified Channel capture and compare functions are halted. The UC is frozen in its current state.

During freeze, all registers are accessible. When the Unified Channel is operating in an output mode, the force match functions remain available, allowing the software to force the output to the desired level.

Note that for input modes, any input events that may occur while the channel is frozen are ignored.

When exiting debug mode or freeze enable bit is cleared (FRZ in the EMIOS_MCR or FREN in the EMIOS_CCR[n] register) the channel actions resume, but may be inconsistent until channel enters GPIO mode again.

21.4.2 IP Bus Interface Unit (BIU)

The BIU provides the interface between the Internal Interface Bus (IIB) and the Peripheral Bus, allowing communication among all submodules and this IP interface.

The BIU allows 8, 16 and 32 bits access. They are performed over a 32-bit data bus in a single cycle clock.

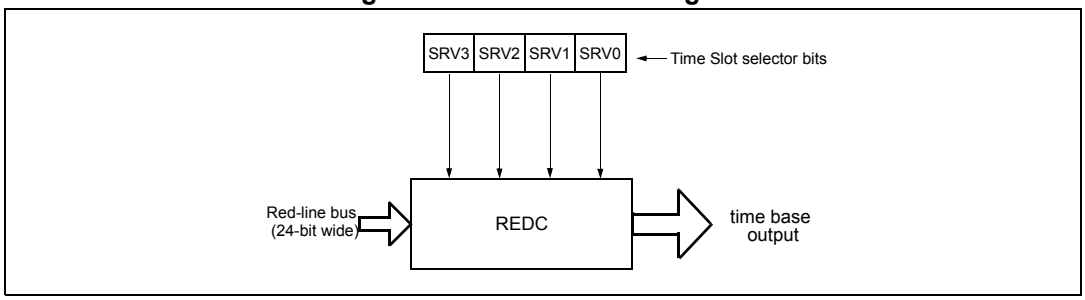
21.4.2.1 Effect of Freeze on the BIU

When the FRZ bit in the EMIOS_MCR register is set and the module is in debug mode, the operation of BIU is not affected.

21.4.3 STAC Bus Client submodule (REDC)

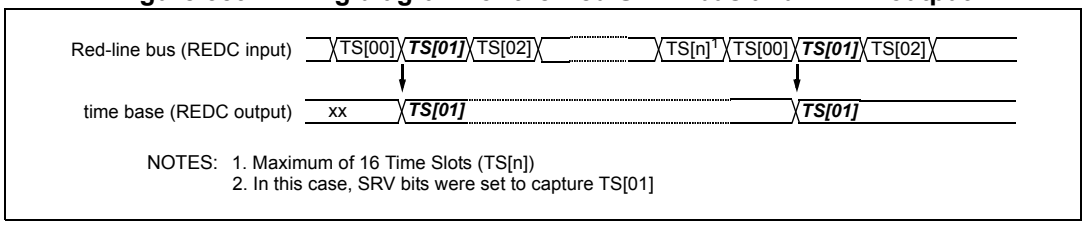
The REDC provides one external time base, imported from the STAC bus, to the Unified Channels. *Figure 358* provides a block diagram for the REDC module.

Figure 358. REDC block diagram



Bits SRV[0:3] in register EMIOS_MCR, selects the desired time slot of the Red-line bus to be output. *Figure 359* shows a timing diagram for the REDC.

Figure 359. Timing diagram for the Red STAC bus and REDC output



Every time the selected time slot change, the STAC bus output is updated.

21.4.3.1 Effect of Freeze on the STAC Bus

When the FRZ bit in the EMIOS_MCR register is set and the module is in debug mode, the operation of REDC submodule is not affected, i.e., there is no freeze function in this submodule.

21.4.4 Global Clock Prescaler Submodule (GCP)

The GCP divides the system clock to generate a clock for the CPs of the channels. The main clock signal is prescaled by the value defined in *Table 214* according to the GPRE[0:7] bits in EMIOS_MCR register. The global prescaler is enabled by setting the GPREN bit in

the EMIOS_MCR register and can be stopped at any time by clearing this bit, thereby stopping the internal counters in all the channels.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write 0 at GPREN bit in EMIOS_MCR register, thus disabling global prescaler;
2. Write the desired value for prescaling rate at GPRE[0:7] bits in EMIOS_MCR register;
3. Enable global prescaler by writing 1 at GPREN bit in EMIOS_MCR register.

The prescaler is not disabled during either freeze state or negated GTBE input.

21.4.4.1 Effect of Freeze on the GCP

When the FRZ bit in the EMIOS_MCR register is set and the module is in debug mode, the operation of GCP submodule is not affected, i.e., there is no freeze function in this submodule.

21.5 Initialization/Application Information

On resetting the eMIOS200 the Unified Channels enter GPIO input mode.

21.5.1 Considerations

Before changing an operating mode, the UC must be programmed to GPIO mode and EMIOS_CADR[n] and EMIOS_CBDR[n] registers must be updated with the correct values for the next operating mode. Then the EMIOS_CCR[n] register can be written with the new operating mode. If a UC is changed from one mode to another without performing this procedure, the first operation cycle of the selected time base can be random, i.e., matches can occur in random time if the contents of EMIOS_CADR[n] or EMIOS_CBDR[n] were not updated with the correct value before the time base matches the previous contents of EMIOS_CADR[n] or EMIOS_CBDR[n].

When interrupts are enabled, the software must clear the FLAG bits before exiting the interrupt service routine.

21.5.2 Application information

Correlated output signals can be generated by all output operation modes. Bits OU[n] of EMIOS_OUDR register can be used to control the update of these output signals.

In order to guarantee that the internal counters of correlated channels are incremented in the same clock cycle, the internal prescalers must be set up before enabling the global prescaler. If the internal prescalers are set after enabling the global prescaler, the internal counters may increment in the same ratio, but at a different clock cycle.

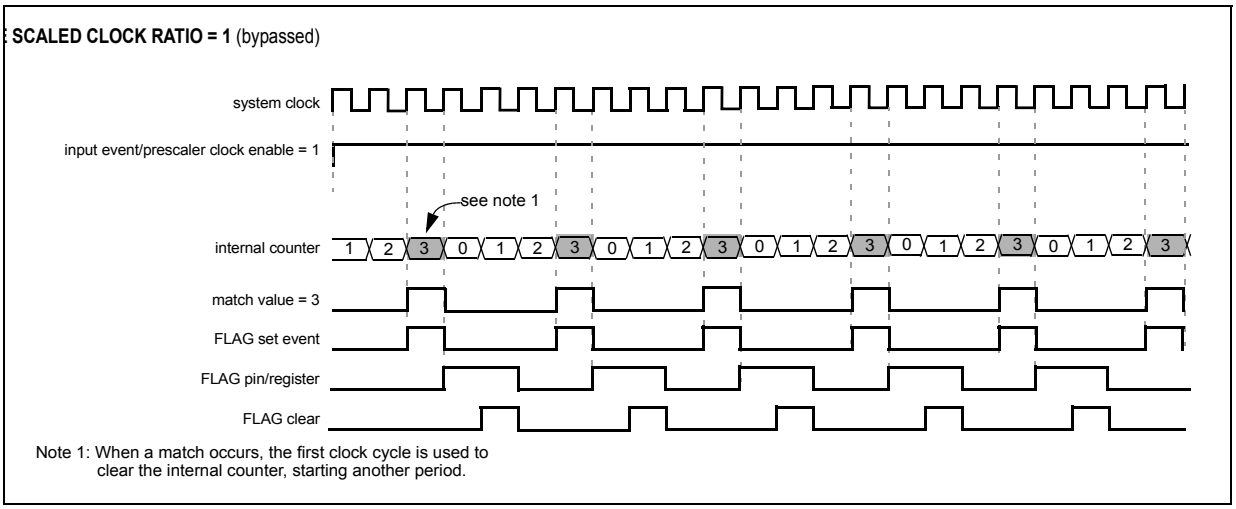
It is recommended to drive Output Disable Input signals with the emios_flag_out signals of some UCs running in SAIC mode. When an output disable condition happens, the software interrupt routine must service the output channels before servicing the channels running SAIC. This procedure avoid glitches in the output pins.

21.5.2.1 Time base generation

For MC and OPWFM with internal clock source operation modes, the internal counter rate can be modified by configuring the clock prescaler ratio. *Figure 360* shows an example of a time base with prescaler ratio equal to one.

Note: MCB and OPWFMB modes have a different behavior.

Figure 360. Time base period when running in the fastest prescaler ratio

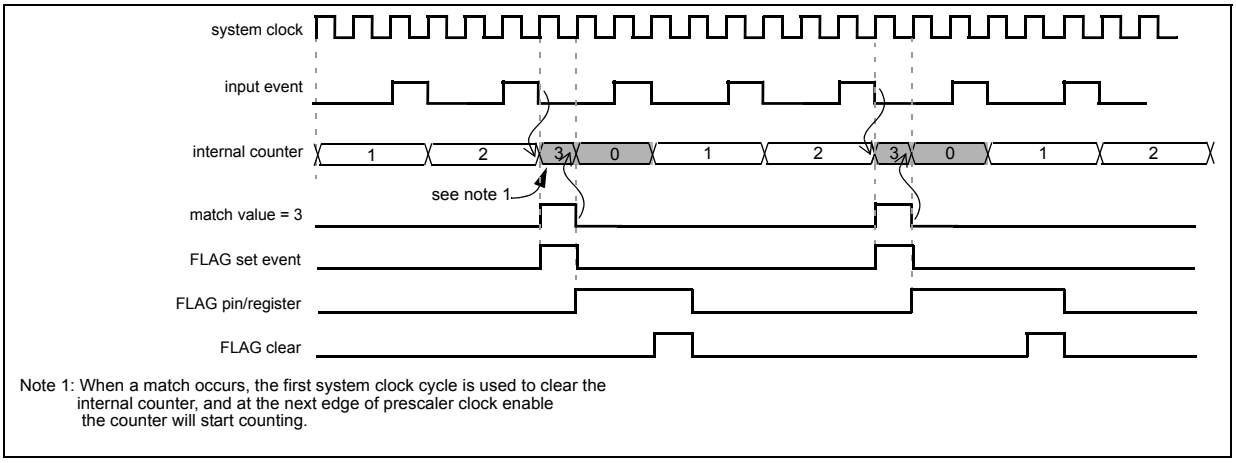


If the prescaler ratio is greater than one or external clock is selected, the counter may behave in three different ways depending on the channel mode:

- If MC mode and Clear on Match Start and External Clock source are selected the internal counter behaves as described in *Figure 361*.
- If MC mode and Clear on Match Start and Internal Clock source are selected the internal counter behaves as described in *Figure 362*.
- If MC mode and Clear on Match End are selected the internal counter behaves as described in *Figure 363*.
- If OPWFM mode is selected the internal counter behaves as described in *Figure 362*. The internal counter clears at the start of the match signal, skips the next prescaled clock edge and then increments in the subsequent prescaled clock edge.

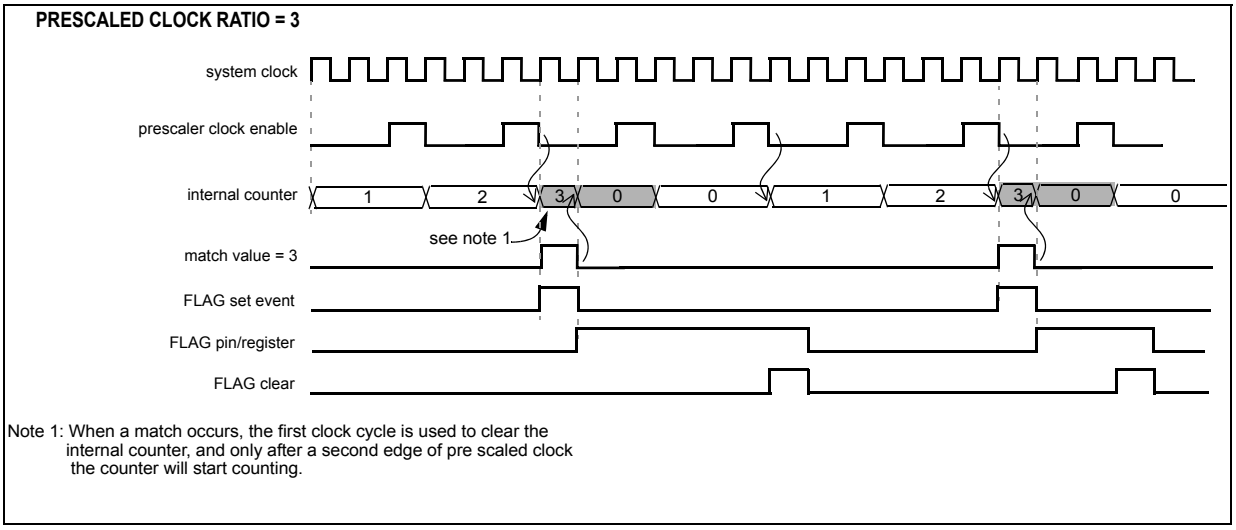
Note: MCB and OPWFMB modes have a different behavior.

Figure 361. Time base generation with external clock and clear on match start



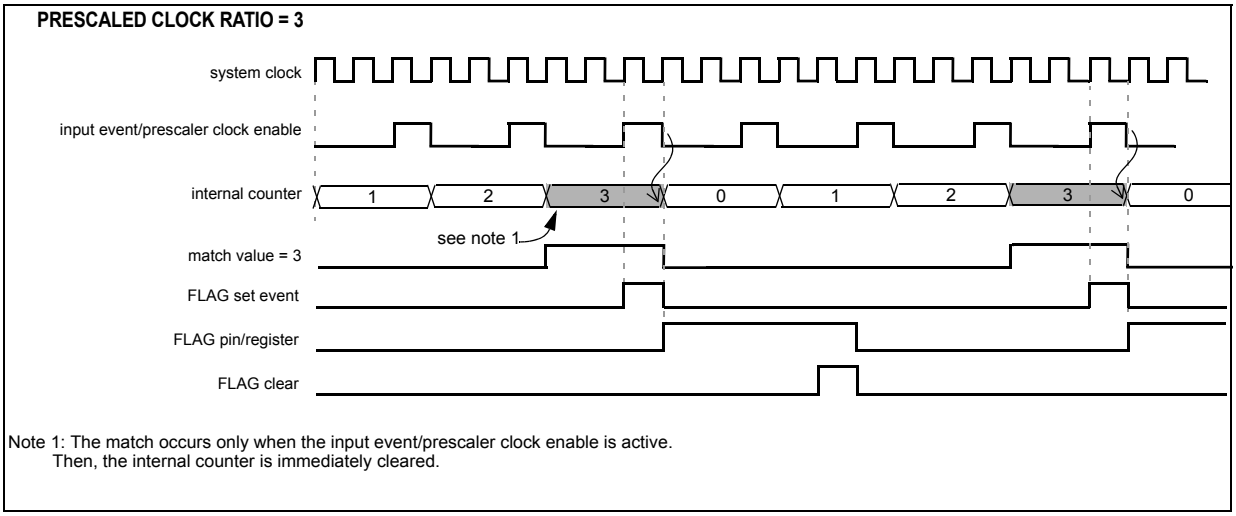
Note 1: When a match occurs, the first system clock cycle is used to clear the internal counter, and at the next edge of prescaler clock enable the counter will start counting.

Figure 362. Time base generation with internal clock and clear on match start



Note 1: When a match occurs, the first clock cycle is used to clear the internal counter, and only after a second edge of pre scaled clock the counter will start counting.

Figure 363. Time base generation with clear on match end



Note 1: The match occurs only when the input event/prescaler clock enable is active. Then, the internal counter is immediately cleared.

21.5.2.2 Coherent Accesses

For PEA mode, it is highly recommended that the software waits for a new FLAG set event before start reading EMIOS_CADR[n] and EMIOS_CBDR[n] registers to get a new measurement. The FLAG indicates that new data has been captured and it is the only way to assure data coherency.

The FLAG set event can be detected by polling the FLAG bit or by enabling the interrupt or DMA request generation.

Reading the EMIOS_CADR[n] register again in the same period of the last read of EMIOS_CBDR[n] register may lead to incoherent results. This will occur if the last read of EMIOS_CBDR[n] register occurred after a disabled B2 to B1 transfer.

21.5.2.3 Channel/Modes initialization

The following basic steps summarize basic output mode startup, assuming the channels are initially in GPIO mode:

1. *[global]* Disable Global Prescaler.
2. *[timebase channel]* Disable Channel Prescaler.
3. *[timebase channel]* Write initial value at internal counter.
4. *[timebase channel]* Set A/B register.
5. *[timebase channel]* Set channel to MC(B) Up mode.
6. *[timebase channel]* Set prescaler ratio.
7. *[timebase channel]* Enable Channel Prescaler.
8. *[output channel]* Disable Channel Prescaler.
9. *[output channel]* Set A/B register.
10. *[output channel]* Select timebase input through BSL[1:0] bits.
11. *[output channel]* Enter output mode.
12. *[output channel]* Set prescaler ratio (same ratio as timebase channel).
13. *[output channel]* Enable Channel Prescaler.
14. *[global]* Enable Global Prescaler.
15. *[global]* Enable Global Time Base.

The timebase channel and the output channel may be the same for some applications such as in OPWFM(B) mode or whenever the output channel is intended to run the timebase itself.

At any time the flags can be configured.

22 Enhanced Time Processing Unit (eTPU2)

This chapter details control and configuration information for the second-generation enhanced time processing unit, called eTPU2. Previous versions of the eTPU ancestry include:

- TPU3
- eTPU
- eTPU+

Many of the features found in the eTPU2 did not exist in earlier versions and eTPU2 code that uses the newer features is not backwards compatible with previous versions of the eTPU. Code written for earlier versions of the eTPU is forward compatible with the eTPU2 without change other than recompiling.

Note: For detailed information on programming the eTPU2, see the application notes available at www.st.com.

22.1 Introduction

The eTPU2 is an on-chip programmable I/O controller with its own core and memory system, enabling it to perform complex timing and I/O management independently of the device CPU. The eTPU2 is essentially an independent microcontroller designed for timing control, I/O handling, serial communications, motor control and engine control applications. Highlights include:

- Executes programs independently from the host core
- Detects and precisely records timing of input events
- Generates complex output waveforms
- Controlled by the core without requiring real-time host processing

The host core setup and service times for each input and output event are minimized. The eTPU2 improves the performance of the device by providing high resolution timing.

- eTPU2 dedicated channels include two match and two capture registers
- The eTPU2 includes up to two eTPU engines that are optimized with specific instructions to service channel hardware
- The eTPU2 creates no host overhead for servicing timing events

22.1.1 eTPU2 implementation

The configuration of the eTPU2 varies among devices. Devices in the SPC563Mxx family feature:

- Single eTPU engine
- 32 channels
- 3 KB of shared data memory (SDM). This memory is often referred to as shared parameter (data) RAM (SPRAM).
- 14 KB of shared code memory (SCM)

22.2 Block diagram

Figure 364 shows a top-level eTPU block diagram.

Figure 364. eTPU block diagram

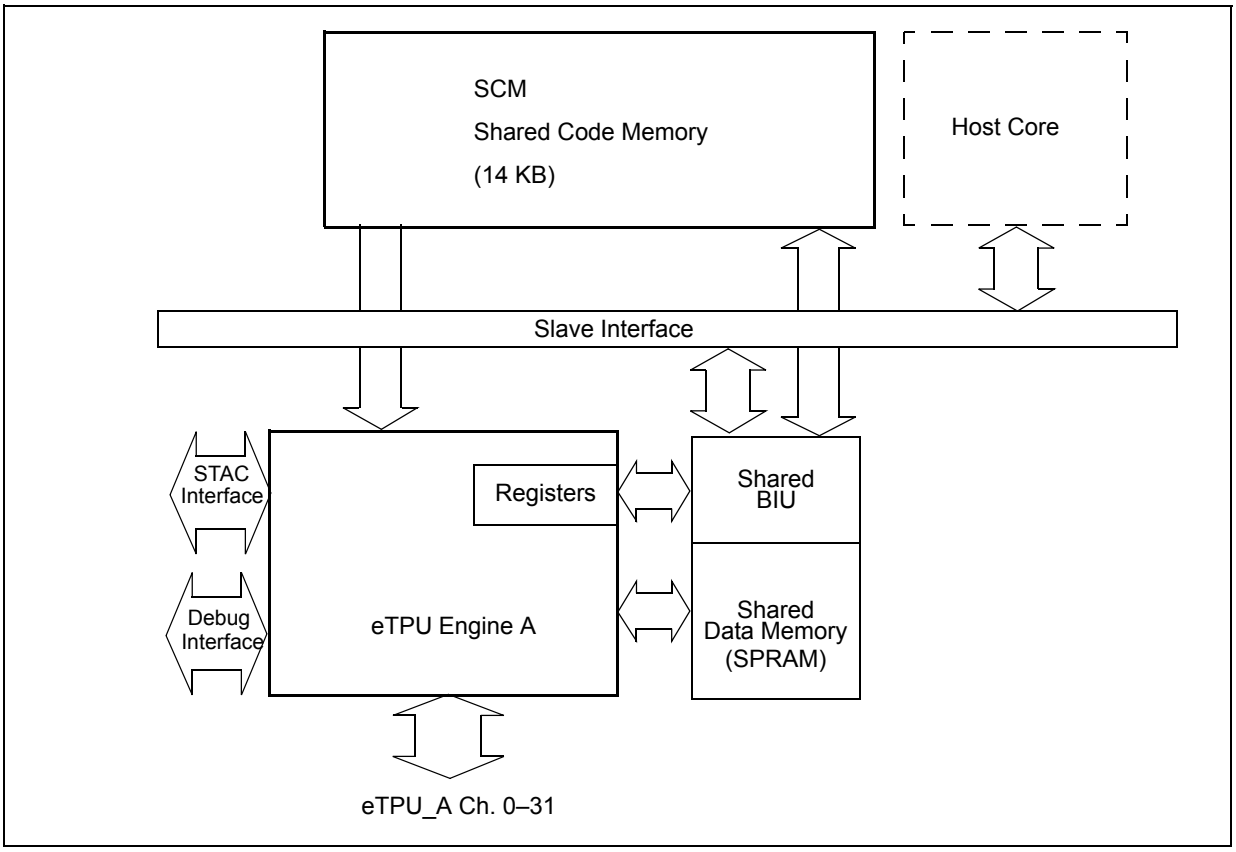
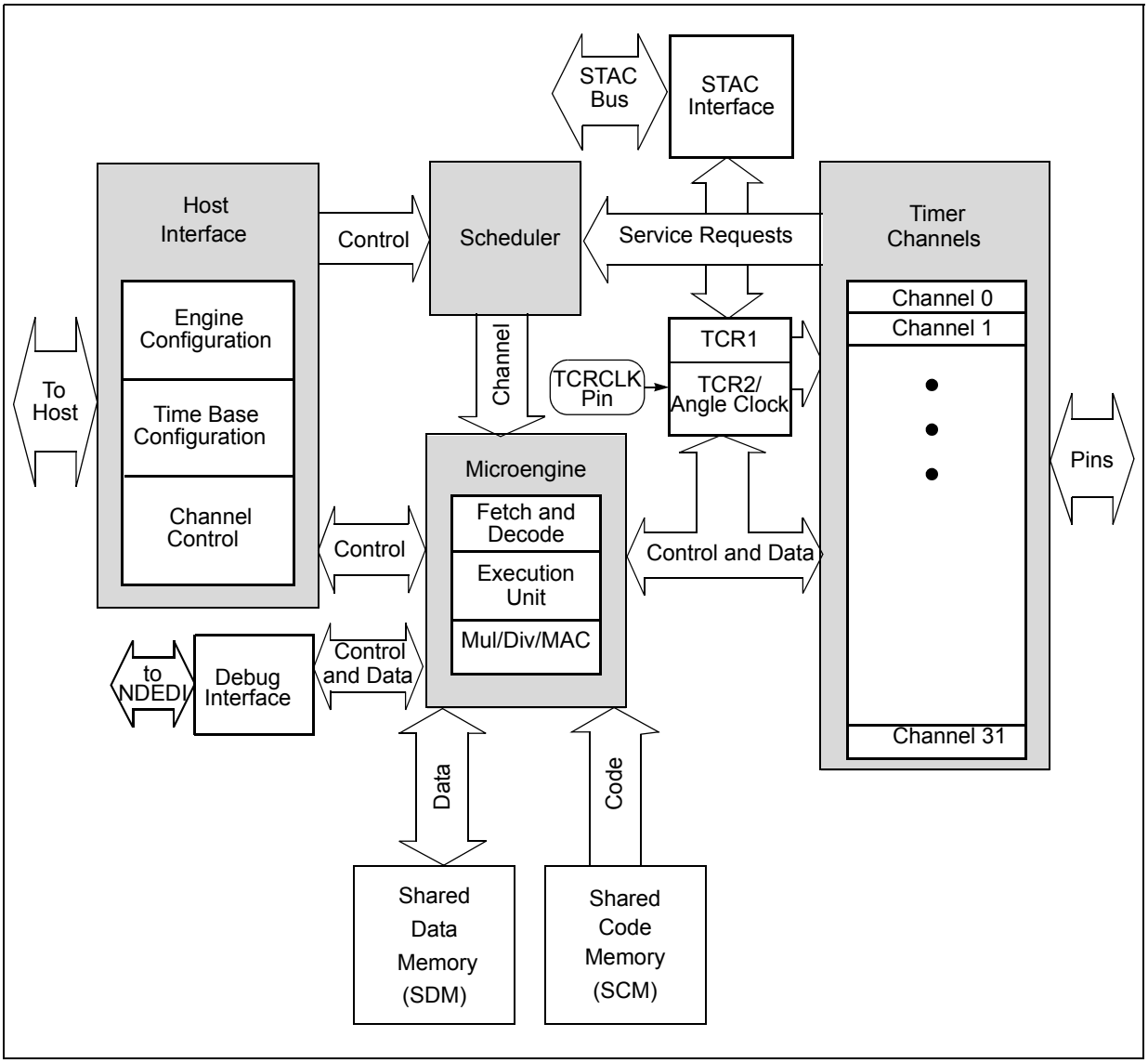


Figure 365 shows the block diagram for the eTPU engine.

Figure 365. eTPU engine block diagram



22.3 eTPU2 operation overview

The eTPU2 is a real-time microprocessor subsystem. Therefore it runs microengine code from instruction memory (SCM) to handle specific events and accesses data memory (SDM) for parameters, work data, and application state information. Events can originate from I/O channels (due to pin transitions and/or time base matches), device core requests, or inter-channel requests. Events that call for local eTPU processing activate the microengine by issuing a service request. The service request microcode can send an interrupt to the device core, but the core cannot be directly interrupted by I/O channel events.

Each channel is associated with a function that defines its behavior. A function is a software entity consisting of a set of microengine routines, called threads, that respond to eTPU2 service requests. Function routines, which reside in the SCM, are also responsible for

channel configuration. A function can be assigned to several channels, but a channel can only be associated with one function at a given moment. If the device core reconfigures the channel function, the eTPU2 can change the function assigned to that channel. The association between functions and channels is defined by the device core.

The eTPU2 hardware supplies resource sharing features that support concurrency:

- A hardware scheduler dispatches the service request microengine routines based on a set of priorities defined by the device's core. Each channel has its own unique priority assignment that primarily depends on CPU assignment. The channel's number is an inherent property also used to determine priority.
- A service request routine cannot be interrupted by another service request until it ends, that is, until an end instruction is executed. This sequence of uninterrupted instruction execution is called a thread. The core can terminate the thread by writing '1' to ETPU_ECR[FEND].
- Channel-specific contexts (registers and flags) are automatically switched between the end of a thread and the beginning of the next one.
- SDM arbitration, a dual-parameter coherency controller, and semaphores can be used to ensure coherent access to eTPU2 data and the device core.

22.3.1 eTPU2 engine

The eTPU2 engine processes input pin transitions and generates output pin waveforms. These events are triggered by eTPU2 timers (time bases) that are driven by a system clock to give absolute time control, or by an asynchronous counter such as an angle clock that can track the angle of a rotating shaft.

The eTPU2 engine consists of the following blocks:

- 32 independent timer channels
- Task scheduler
- Host interface
- Microengine that has dedicated hardware for input signal processing and output signal generation over the 32 I/O channels. Each channel can also choose between two 24-bit counter registers for a time base.

The microengine fetches microinstructions from shared code memory (SCM). eTPU2 application parameters and global and local variables, referred to as work data, are held in 32-bit shared data memory (SDM), which is also used for passing information between the device's core and the microengine. The bus interface unit (BIU) allows the core to access eTPU2 registers, SDM, and SCM.

22.3.1.1 Time bases

The eTPU engine has two 24-bit count registers (TCR1 and TCR2) that provide reference time bases for all match and input capture events. Prescalers for both time bases are controlled by the device core through bit fields in the eTPU engine configuration registers.

The values for each TCR1 and TCR2 counter register can be independently derived from the system clock or from an external input via the TCRCLK pin. In addition, the TCR2 time base can be derived from special angle-clock hardware that enables implementing angle-based functions. This feature is added to support advanced angle-based engine control applications.

The TCRs can also drive an eMIOS time base through the shared time and counter (STAC) bus, or they can be written by eTPU2 function software.

22.3.1.2 eTPU2 timer channels

The eTPU2 engine has 32 identical, independent channels. Each channel corresponds to an input/output signal pair. Every channel has access to both 24-bit counter registers, TCR1 and TCR2.

Each channel consists of event logic which supports a total of four events: two capture and two match events. The event logic contains two 24-bit capture registers and two 24-bit match registers. The match registers are compared to a selected TCR by greater-than-or-equal-to and equal-only comparators. The match and compare register pairs enable many combinations of single and double-action functions.

The channel configuration can be changed, with restrictions, by the microengine. Each channel can perform double capture, double match or a variety of other capture-match combinations. Service requests can be generated on one or both of the match events and/or on one of the capture events.

Digital filters that have different filtering modes are provided for the input signals.

Every channel can use any time base or angle counter for either match or capture operation. For example, a match on TCR1 can capture the value of TCR2. The channels can request service from the microengine due to recognized pin transitions (input events) or time base matches.

Every eTPU2 channel can be configured with the following combinations:

- Single input capture, no match
- Single input capture with single match timeout
- Single input capture with double match timeout with several double match submodes
- Double input capture with single or double match timeout with several double match submodes
- Single output match
- Double output match with several double match submodes
- Input-dependent output generation

The double match functionality has various combinations for generation of service request and determining pin actions.

22.3.2 Host interface

The engine’s host interface enables the device core to control the operation of the eTPU2. For the eTPU2 to start operation, the device core must initialize the eTPU2 by writing to the appropriate host interface registers to assign a function and priority to each channel. In addition, the device core writes to the host service request and channel configuration registers to further define operation for each initialized channel.

Note: The host transfers the code image for the eTPU2 microcode to the SCM, then the host enables eTPU2 access to the SCM (which also disables host access).

22.3.3 Shared data memory (SDM)

The SDM works as data RAM that can be accessed by the device core and the eTPU engine. This memory is used for either:

- Information transfer between the device core and the eTPU2
- Data storage for the eTPU2 microcode program

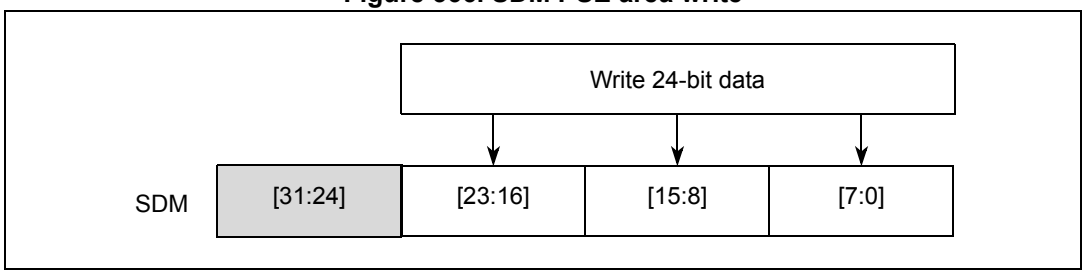
The SDM width is 32 bits, and is accessible by the host in any of the three formats: byte, 16-bit, or 32-bit. The eTPU2 can access the full 32 bits of the SDM, lower 24 bits or upper byte (8-bit).

The host can also access the SDM space mirrored in an alternate area with parameter sign extension (PSE). PSE allows for 24-bit data to be accessed as 32-bit sign-extended data without using the device's bandwidth to extend the data.

Parameter signal extension accesses differ from the usual host accesses to the original SDM area as follows:

- Writes are effective only to the lower three bytes of a word: the word's most significant byte (byte address) remains the same in SDM.

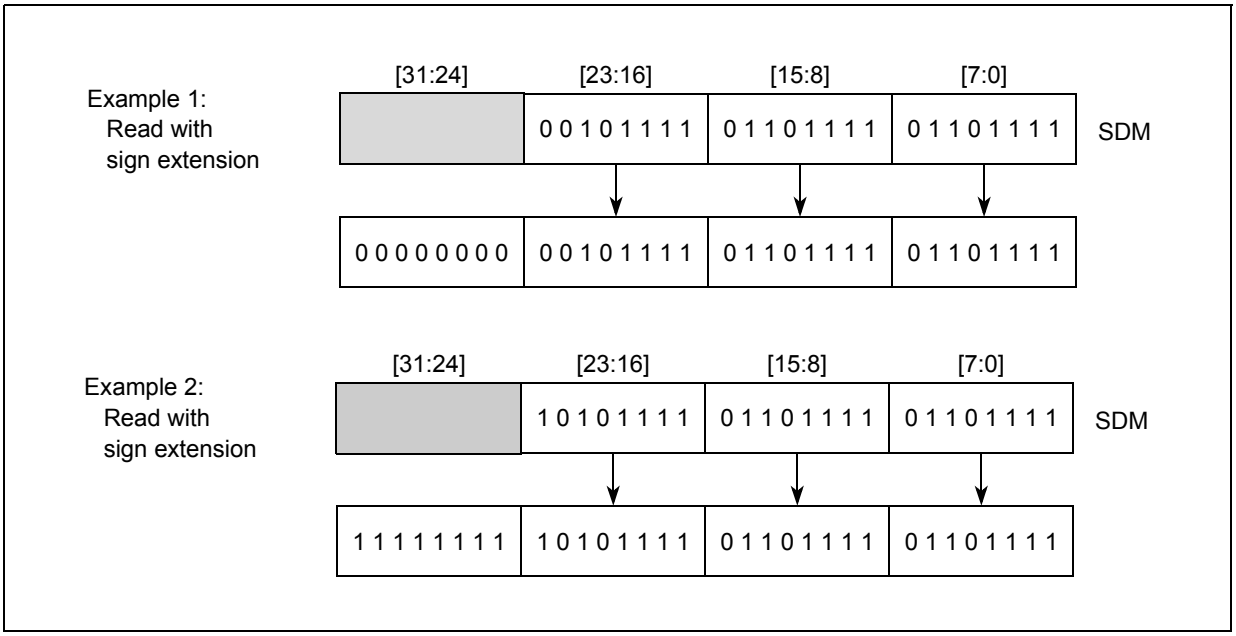
Figure 366. SDM PSE area write



Note: For the most significant byte, the word format is big endian, as in the default Power Architecture word format.

- Reads return the lower three bytes of a word sign-extended to 32 bits, that is: the most significant bit of the word's second most significant byte (byte addresses) is copied in all eight bits of the most significant read byte.

Figure 367. PSE accesses



Each eTPU2 channel can be associated with a variable number of parameters located in the SDM, according to its selected function. Each function can require a different number of parameters. During eTPU2 initialization the host has to program channel base addresses, allocating proper parameters for each channel according to its selected function.

In the host address space each parameter occupies four bytes (32 bits). eTPU2 usage of the upper byte is achieved by having a 32-bit Preload (P) register that can access the upper byte, the lower 24 bits, or all the 32 bits. The microcode can switch between access sizes at any time.

Each function can require a different number of parameters. During the eTPU2 initialization the host has to program channel base addresses, allocating proper parameters for each channel according to its selected function.

22.3.4 Task scheduler

As mentioned in *Section 22.3: eTPU2 operation overview*, every channel function is composed of one or more threads, and threads cannot be interrupted by host or channel events, such as channel servicing. The function of the task scheduler, therefore, is to recognize and prioritize the channels needing service and grant execution time to each channel. The time given to an individual thread for execution or service is called a time slot. The duration of a time slot is determined by the number of instructions executed in the thread plus SDM wait-states received, and varies in length. Although several channels can request service at the same time, the function threads must be executed serially.

At any time, an arbitrary number of channels can require service. The channel logic, eTPU2 microcode, or the host application notifies the scheduler by issuing a service request.

Out of reset, all channels are disabled. The device core makes a channel active by assigning it one of three priorities: high, middle, or low. The scheduler determines the order in which channels are serviced based on channel number and assigned priority. The priority mechanism, implemented in hardware, ensures that all requesting channels are serviced.

22.3.5 Microengine

The eTPU2 microengine is a simple RISC implementation that performs each instruction in a microcycle of two system clocks, while pre-fetching the next instruction through an instruction pipeline. Instruction execution time is constant for the arithmetic logic unit (ALU) unless it gets wait states from SDM arbitration.

Microcode is stored in shared code memory (SCM) that is 32 bits wide. The microengine instruction set provides basic arithmetic and logic operations, flow control (jumps and subroutine calls), SDM access, and channel configuration and control. The instruction formats are defined in a way that allow particular combinations of two or three of these operations with unconflicting resources to be executed in parallel in the same microcycle, improving performance.

The microengine also has an independent multiply/divide/MAC unit that performs these complex operations in parallel with other microengine instructions.

Channel functionality is integrated to the instruction set through channel control operations and conditional branch operations, which support jumps/calls on channel-specific conditions. This allows quick and terse channel configuration and control code, contributing to reduced service time.

22.3.6 Debug interface

Nexus level 3 debug support is available through the eTPU2 Nexus development interface (NDEDI).

22.4 Features

The eTPU2 includes these distinctive features:

- Up to 32 channels for the eTPU engine: each channel is associated with an I/O signal pair
  - Enhanced input digital filters on the input pins for improved noise immunity. The eTPU2 digital filter can use two samples, three samples, or work in continuous mode.
  - Orthogonal channels, except for channel 0: each channel can perform any time function. Each time function can be assigned to more than one channel at a given time, so each signal can have any functionality. Channel 0 has the same capabilities of the others, but can also work with special angle counter logic (see below).
  - A link service request allows activation of a channel thread by request of another channel, even between eTPU engines.
  - A host service request allows activation of a channel thread by a device core request.
  - Each channel has an event mechanism that supports single and double action functionality in various combinations. It includes two 24-bit capture registers, two 24-bit match registers, 24-bit greater-equal or equal-only comparator.

- Two independent 24-bit time bases for channel synchronization
  - The first time base can be clocked by the system clock with programmable prescaler division from 2–512 (in steps of 2).
  - The first time base can also be clocked by an external signal with programmable prescaler divisions of 1–256.
  - The second time base can be clocked by an external signal with programmable prescaler divisions from 1–64 or by the system clock divided by eight.
  - The second time base has a programmable prescaler that applies to all TCR2 clock inputs except the angle counter.
  - The second time base counter can work as an angle counter, enabling angle-based applications to match angle instead of time.
  - The second time base can alternatively be used as a pulse accumulator gated by an external signal.
  - Either time base can be written or read by the eTPU engine at any time.
  - Either time base can be read, but not written, by the host.
- Event-triggered RISC processor (microengine)
  - Two-stage pipeline implementation (fetch and execution), with separate instruction memory (SCM) and data memory (SDM)
  - Two-system-clock microcycle fixed-length instruction execution for the ALU
  - 14 KB of shared code memory (SCM)
  - 3 KB of shared data memory (SDM)
  - Instruction set with embedded channel support, including specialized channel control subinstructions and conditional branching on channel-specific flags.
  - Channel-oriented addressing: channel-bound address mode with host configured channel base address enables a function to operate independently on different channels.
  - Channel-bound data address space of up to 128 32-bit parameters (512 bytes)
  - Global parameter address mode allows access to common channel data of up to 256 32-bit parameters (1024 bytes).
  - Support for indirect and stacked data access schemes
  - Parallel execution of: data access, ALU, channel control and flow control subinstructions in selected combinations
  - 24-bit registers and ALU, plus one 32-bit register for full-width SDM access
  - Additional 24-bit multiply/MAC/divide unit which supports all signed/unsigned/multiply/MAC combinations, and unsigned 24-bit divide. The MAC/divide unit works in parallel with the regular microcode commands.
- Resource sharing features resolve channel contention for common use of channel registers, memory and microengine time
  - Hardware scheduler works as a ‘task management’ unit, dispatching event service routines by predefined, host-configured priority.
  - Hardware breakpoints on data access, qualified by address and/or data values.
  - Hardware breakpoints on instruction address.
  - Automatic channel context switch when a ‘task switch’ occurs; that is, one function thread ends and another begins to service a request from another channel.

Channel-specific registers, flags and parameter base address are automatically loaded for the next serviced channel.

  - Individual channel priority setting in three levels: high, middle, and low.
  - Scheduler priority scheme allows calculation of worst case latency for event servicing and ensures servicing of all channels by preventing permanent blockage.
  - SDM shared between host core and the eTPU2 engine, supporting channel-to-channel or host-to-channel communication.
  - Hardware implementation of four semaphores allows for resource arbitration between channels in the eTPU engine.
  - Hardware semaphores are directly supported by the microengine instruction set.
  - Coherent dual-parameter controller allows coherent (to microengines) accesses to two parameters by the host.
- Test and development support features
  - Nexus level 3 debug support through the eTPU2 Nexus block (NDEDI)
  - Software breakpoints
  - SCM (code memory) continuous signature-check built-in code integrity test multiple input signature calculator (MISC): runs concurrently with eTP2U normal operation
- eTPU2 enhancements over eTPU
  - TCR1, channel logic and digital filters (both channel and TCRCLK) now have an option to run at divisions of full system clock speed, besides system clock / 2.
  - Channels support unordered transitions: transition B can now be detected before transition A. Related to this enhancement, TDLA and TDLB can now be independently negated by microcode.
  - Added a new User Programmable Channel Mode: the blocking, enabling, service request and capture characteristics of this channel mode can be programmed via microcode.
  - Microinstructions now provide an option to issue Interrupt and Data Transfer requests selected by CHAN. They can also be requested simultaneously at the same instruction.
  - Channel Flags 0 and 1 can now be tested for branching, besides selecting the entry point.
  - Channel digital filters can be bypassed.
  - Scheduler priority-passing mechanism can now be disabled.
  - New Watchdog mechanism kills threads over a programmable timeout.
  - New counter allows microengine load information collection for performance analysis
  - Channels 1 and 2 (besides channel 0) can now be selected to control the EAC.
  - Timebase prescalers are now reset when the GTBE input is negated, guaranteeing synchronization with eMIOS in all cases.
  - New MISC flag indicates when an SCM signature calculation round is completed. This allows measuring of the average MISC scan period in a real application situation.
  - New channel TCCEA flag allows continuous capture even after TDLA is set, making it fully compatible with TPU behavior.

- New branch condition PRSS tells the pin state at the time when a channel (match or transition) service request occurred.
- MRLEA/B can now be negated independently by microcode.
- New engine Relative address mode allows a function to access SDM address space common to one engine, but distinct between engines.
- Error Correction support for Code (SCM) and Data (SDM) memories (available on selected MCUs).
- All changes above are upward compatible with the classic eTPU, so that legacy object code (both Host and microcode) runs on eTPU+ and eTPU2 without modification.

22.5 Modes of operation

The eTPU2 is capable of working in the following modes:

- User Configuration Mode
- User Mode
- Debug Mode
- Module Disable Mode

22.5.1 User Configuration Mode

By having access to the shared code memory (SCM), the core has the ability to program the eTPU2 core with time functions.

22.5.2 User Mode

In user mode, the core does not access the eTPU2 shared code memory and predefined eTPU functions are used.

22.5.3 Debug Mode

The core debugs eTPU2 code, accessing special trace and debug features via Nexus interface:

- Hardware breakpoint and watchpoint setting
- Access to internal registers
- Single-step execution
- Forced instruction execution
- Software breakpoint insertion and removal

22.5.4 Module Disable Mode

eTPU2 engine clocks are stopped through a register write to ETPU_ECR bit MDIS, saving power (input sampling is halted). eTPU2 engine can be placed in disable mode independently. Module disable mode stops only the eTPU engine clock, so that the shared BIU and global channel registers can be accessed, and interrupts and DMA requests can be cleared and enabled/disabled. An eTPU engine can enter module disable mode only after the currently-running thread is finished.

22.6 eTPU2 mode selection

User and user configuration are the production operating modes, and differ in the way these modes access SCM. Module disable mode is entered by setting ETPU_ECR[MDIS].

22.7 External signal description

There are 65 external signals for the eTPU engine:

- 32 channel input signals
- 32 channel output signals
- TCRCLK clock input

Additionally there are four internal output disable signals that implement the output disable feature needed for motor control.

22.8 eTPU2 external signal description

22.8.1 Output and input channel signals

The channel signal connections for the eTPU2 engine are described in *Section 3.5: DSPI Connections to eTPU_A, eMIOS and SIU*. Each eTPU2 channel is associated with an input and output.

To reduce the number of pins required by the device's eTPU2 while still maintaining the eTPU2's functionality, the eTPU2 is also internally wired to the DSPI (see *Section 3.5.1: DSPI_B Connectivity*).

The eTPU2 microcode can be programmed to set the output level of an eTPU channel in one of two manners:

- By forcing the logic level to a specific value
- By specifying the logic level output action when a match or transition event occurs

Every eTPU2 channel input has a digital filter to filter out noise pulses that have a width less than a specified value. This prevents small noise glitches from being recognized by the transition detect logic. Any pulses wider than the specified filter width are passed to the channel transition detect logic.

22.9 Memory map

The eTPU system simplified memory map is shown in *Table 221*. The base address for the eTPU module is listed as BASE. Each of the register areas shown can have their own reserved address areas.

Table 221 shows a high-level memory map.

Table 221. eTPU high-level memory map

Address	Register description
Base (0xC3FC_0000)–Base + 0x0000_001F	eTPU system module configuration registers
Base + 0x0000_0020–0x0000_002F	eTPU_A time base registers

Table 221. eTPU high-level memory map (continued)

Address	Register description
Base + 0x0000_0030–0x0000_01FF	Reserved
Base + 0x0000_0200–0x0000_02FF	eTPU_A global channel registers
Base + 0x0000_0300–0x0000_03FF	Reserved
Base + 0x0000_0400–0x0000_07FF	eTPU_A channel registers
Base + 0x0000_0800–0x0000_7FFF	Reserved
Base + 0x0000_8000–0x0000_8BFF	SDM (3 KB)
Base + 0x0000_8C00–0x0000_BFFF	Reserved
Base + 0x0000_C000–0x0000_CBFF	SDM PSE mirror(1) (3 KB)
Base + 0xCC00–0xFFFF	Reserved
Base + 0x0001_0000–0x0001_3FFF(2)	SCM (14 KB used)
Base + 0x0001_4000–0x0001_FFFF	Not writable. Reads the return value of ETPU_SCMOFFDATAR.

1. Parameter Sign Extension access area.
2. Do not access addresses in the SCM memory block that are unused. These addresses are reserved.

22.10 Register descriptions

Table 222 shows the eTPU registers and their locations.

Table 222. Detailed memory map

Address offset from base (Base: 0xC3FC_0000)	Register	Bits	Location
0x0000_0000	eTPU Module Configuration Register (ETPU_MCR)	32	*on page 639*
0x0000_0004	eTPU Coherent Dual-Parameter Controller Register (ETPU_CDCR)	32	*on page 642*
0x0000_0008	Reserved	—	—
0x0000_000C	eTPU MISC Compare Register (ETPU_MISCCMPR)	32	*on page 643*
0x0000_0010	eTPU SCM Off-Range Data Register (ETPU_SCMOFFDATAR)	32	*on page 643*
0x0000_0014	eTPU_A Engine Configuration Register (ETPU_ECR_A)	32	*on page 644*
0x0000_0018–0x0000_001C	Reserved	—	—
0x0000_0020	eTPU_A Time Base Configuration Register (ETPU_TBCR_A)	32	*on page 647*
0x0000_0024	eTPU_A Time Base 1 (TCR1) Visibility Register (ETPU_TB1R_A)	32	*on page 650*
0x0000_0028	eTPU_A Time Base 2 (TCR2) Visibility Register (ETPU_TB2R_A)	32	*on page 651*

Table 222. Detailed memory map (continued)

Address offset from base (Base: 0xC3FC_0000)	Register	Bits	Location
0x0000_002C	eTPU_A STAC Bus Configuration Register (ETPU_REDCR_A)	32	on page 652
0x0000_0030–0x0000_005C	Reserved	—	—
0x0000_0060	eTPU_A Watchdog Timer Register (ETPU_WDTR_A)	32	on page 653
0x0000_0064	Reserved	—	—
0x0000_0068	eTPU_A Idle Register (ETPU_IDLE_A)	32	on page 654
0x0000_006C–0x0000_01FF	Reserved	—	—
0x0000_0200	eTPU_A Channel Interrupt Status Register (ETPU_CISR_A)	32	on page 655
0x0000_0208–0x0000_020C	Reserved	—	—
0x0000_0210	eTPU_A Channel Data Transfer Request Status Register (ETPU_CDTRSR_A)	32	on page 655
0x0000_0218–0x0000_021C	Reserved	—	—
0x0000_0220	eTPU_A Channel Interrupt Overflow Status Register (ETPU_CIOSR_A)	32	on page 656
0x0000_0228–0x0000_022C	Reserved	—	—
0x0000_0230	eTPU_A Channel Data Transfer Request Overflow Status Register (ETPU_CDTROSR_A)	32	on page 657
0x0000_0238–0x0000_023C	Reserved	—	—
0x0000_0240	eTPU_A Channel Interrupt Enable Register (ETPU_CIER_A)	32	on page 657
0x0000_0248–0x0000_024C	Reserved	—	—
0x0000_0250	eTPU_A Channel Data Transfer Request Enable Register (ETPU_CDTRER_A)	32	on page 658
0x0000_0254–0x0000_025C	Reserved	—	—
0x0000_0260	eTPU_A Watchdog Status Register (ETPU_WDSR_A)	32	on page 659
0x0000_0258–0x0000_027F	Reserved	—	—
0x0000_0280	eTPU_A Channel Pending Service Status Register (ETPU_CPSSR_A)	32	on page 659
0x0000_0284–0x0000_028C	Reserved	—	—
0x0000_0290	eTPU_A Channel Service Status Register (ETPU_CSSR_A)	32	on page 660
0x0000_0294–0x0000_03FF	Reserved	—	—
0x0000_0400	eTPU_A Channel 0 Configuration Register (ETPU_C0CR_A)	32	on page 662

Table 222. Detailed memory map (continued)

Address offset from base (Base: 0xC3FC_0000)	Register	Bits	Location
0x0000_0404	eTPU_A Channel 0 Status Control Register (ETPU_C0SCR_A)	32	on page 663
0x0000_0408	eTPU_A Channel 0 Host Service Request Register (ETPU_C0HSRR_A)	32	on page 664
0x0000_040C	Reserved	—	—
0x0000_0410	eTPU_A Channel 1 Configuration Register (ETPU_C1CR_A)	32	on page 662
0x0000_0414	eTPU_A Channel 1 Status Control Register (ETPU_C1SCR_A)	32	on page 663
0x0000_0418	eTPU_A Channel 1 Host Service Request Register (ETPU_C1HSRR_A)	32	on page 664
0x0000_041C	Reserved	—	—
. . .	. . .	. . .	. . .
0x0000_05F0	eTPU_A Channel 31 Configuration Register (ETPU_C31CR_A)	32	on page 662
0x0000_05F4	eTPU_A Channel 31 Status Control Register (ETPU_C31SCR_A)	32	on page 663
0x0000_05F8	eTPU_A Channel 31 Host Service Request Register (ETPU_C31HSRR_A)	32	on page 664
0x0000_05FC–0x0000_7FFF	Reserved	—	—
0x0000_8000–0x0000_8BFF	Shared data memory (parameter RAM)	3 KB	—
0x0000_8C00–0x0000_BFFF	Reserved	—	—
0x0000_C000–0x0000_CBFF	SDM PSE mirror(1)	3 KB	on page 629
0x0000_CC00–0x0000_FFFF	Reserved	—	—
0x0001_0000–0x0001_37FF	SCM—Shared code memory(2)	14 KB	—
0x0001_3800–0x0001_FFFF	Reserved	—	—

1. Parameter sign extension access area.
2. SCM access is only available under certain conditions when ETPU_MCR[VIS] = 1. The SCM can only be written in 32-bit accesses.

22.10.1 System configuration registers

22.10.1.1 eTPU Module Configuration Register (ETPU_MCR)

This register is global to the eTPU engine, and resides in the shared BIU. ETPU_MCR gathers global configuration and status in the eTPU system, including global exception. It is also used for configuring the SCM (shared code memory) operation and test.

Address: Base + 0x0000_0000 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SDMERR	WDTOA	WDTOB	MGE1	MGE2	ILF1	ILF2	SCMERR	0	0	SCMSIZE				
W	GEC															
Reset	0	0	0	0	0	0	0	0	0	0	0	SCMSIZE				

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	SCMMISC	SCMMISF	SCMMISEN	0	0	VIS	0	0	0	0	0	GTBE
W					SCMMISCC											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 368. eTPU Module Configuration Register (ETPU_MCR)

Table 223. ETPU_MCR field descriptions

Field	Description
0 GEC	Global exception clear Negates global exception request and clears global exception status bits MGEA, MGEB, ILFA, ILFB and SCMMISF. A read always returns 0. Writes have the following effect: 0: Keep global exception request and status bits ILFA, ILFB, MGEA, MGEB, and SCMMISF as is. 1: Negate global exception, clear status bits ILFA, ILFB, MGEA, MGEB, and SCMMISF. GEC works the same way with the eTPU engine in stop mode.
1 SDMERR	SDMERR—SDM Read Error This flag indicates that an SDM read error occurred on a microengine read, generating a Global Exception. Errors from Host reads neither set this flag nor generate Global Exceptions. This bit is cleared by writing ‘1’ to GEC. 1: Global Exception requested by SDM read error is pending. 0: No Global Exception pending because of SDM read error.
2 WDTOA	Watchdog Timeout Flag WDTOA indicates that a Watchdog Timeout occurred in eTPU engine A, generating a Global Exception. This bit is cleared by writing ‘1’ to GEC. 1: Global Exception requested by Watchdog timeout 0: No Global Exception pending because of Watchdog timeout.

Table 223. ETPU_MCR field descriptions (continued)

Field	Description
3 WDTOB	Watchdog Timeout Flags WDTOB indicates that a Watchdog Timeout occurred in eTPU engine B, generating a Global Exception. This bit is cleared by writing ‘1’ to GEC. 1: Global Exception requested by Watchdog timeout 0: No Global Exception pending because of Watchdog timeout.
4 MGE1	Microcode global exception eTPU engine A Indicates that a global exception was asserted by microcode executed on the respective engine. The determination of the reason why the global exception was asserted is application dependent: it can be coded in an SDM status parameter, for instance. This bit is cleared by writing ‘1’ to GEC. 0: No microcode-requested global exception pending. 1: Global exception requested by microcode is pending.
5 MGE2	Microcode global exception eTPU engine B Indicates that a global exception was asserted by microcode executed on the respective engine. The determination of the reason why the global exception was asserted is application dependent: it can be coded in an SDM status parameter, for instance. This bit is cleared by writing ‘1’ to GEC. 0: No microcode requested global exception pending. 1: Global exception requested by microcode is pending.
6 ILF1	Invalid instruction flag eTPU_A Set by the microengine to indicate that an invalid instruction was decoded in eTPU engine A. This bit is cleared by host writing 1 to GEC. 0: Invalid Instruction not detected. 1: Invalid Instruction detected by eTPU_A.
7 ILF2	Invalid instruction flag eTPU_B Set by the microengine to indicate that an invalid instruction was decoded in eTPU engine B. This bit is cleared by host writing 1 to GEC. 0: Invalid Instruction not detected. 1: Invalid Instruction detected by eTPU_B.
8 SCMERR	SCM Read Error This flag indicates that an SCM read error occurred on a microengine read, generating a Global Exception. Errors from Host reads neither set this flag nor generate Global Exceptions. This bit is cleared by writing ‘1’ to GEC. 1: Global Exception requested by SCM read error is pending. 0: No Global Exception pending because of SCM read error.
9–10	Reserved
11–15 SCMSIZE [0:4]	SCM size Holds the number of 2 KB SCM Blocks minus 1. This value is MCU-dependent.
16–19	Reserved

Table 223. ETPU_MCR field descriptions (continued)

Field	Description
20 SCMMISC SCMMISCC	SCM MISC Complete, SCM MISC Complete Clear Flag SCMMISC indicates that MISC has completed the evaluation of the SCM signature since reset or the since the last time it was cleared. SCMMISC is cleared by writing '1' to SCMMISCC (at same bit position), and is not cleared when MISC is disabled (SCMMISEN=0). SCMMISC asserts at the end of the SCM memory scan, either if the signature matches or not. 1: MISC completed at least one SCM signature calculation and compare since the last time SCMMISC was cleared. 0: MISC has not yet completed an SCM signature calculation and compare since the last time SCMMISC was cleared.
21 SCMMISF	SCM MISC Flag Set by the SCM MISC (multiple input signature calculator) logic to indicate that the calculated signature does not match the expected value, at the end of a MISC iteration. 0: Signature mismatch not detected. 1: MISC has read entire SCM array and the expected signature in ETPU_MISCCMPR does not match the value calculated. This bit is cleared by writing '1' to GEC.
22 SCM MISEN	SCM MISC enable Used for enabling/disabling the operation of the MISC logic. SCMMISEN is readable and writable at any time. The MISC logic only operates when this bit is set to 1. When the bit is reset the MISC address counter is set to the initial SCM address. When enabled, the MISC continuously cycles through the SCM addresses, reading each and calculating a CRC. To save power, the MISC can be disabled by clearing the SCMMISEN bit. 0: MISC operation disabled. The MISC logic is reset to its initial state. 1: MISC operation enabled. (Toggling to 1 clears the SCMMISF bit) SCMMISEN is cleared automatically when MISC logic detects an error; that is, when SCMMISF transitions from 0 to 1, disabling the MISC operation.
23–24	Reserved
25 VIS	SCM visibility Determines SCM visibility to the slave bus interface and resets the MISC state (but SCMMISEN keeps its value). 0: SCM is not visible to the slave bus. Accessing SCM address space issues a bus error. 1: SCM is visible to the slave bus. The MISC state is reset. This bit is write protected when any of the engines are not in halt or stop states. When VIS=1, the ETPU_ECR MDIS bits are write protected, and only 32-bit aligned SCM writes are supported. The value written to SCM is unpredictable if other transfer sizes are used.
26–30	Reserved
31 GTBE	Global time base enable Enables time bases in both engines, allowing them to be started synchronously. An assertion of GTBE also starts the eMIOS time base(1). This enables the eTPU time bases and the eMIOS time base to all start synchronously. 1: Time bases in both eTPU engines and eMIOS are enabled to run. 0: Time bases in both engines are disabled to run. **Note:** When GTBE is turned off with Angle Mode enabled, the EAC must be reinitialized before GTBE is turned on again.

1. The eMIOS also has a GTBE bit. Assertion of either the eMIOS or eTPU GTBE bit starts time bases for the eMIOS and eTPU.

22.10.1.2 eTPU Coherent Dual-Parameter Controller Register (ETPU_CDCR)

ETPU_CDCR configures and controls dual-parameter coherent transfers.

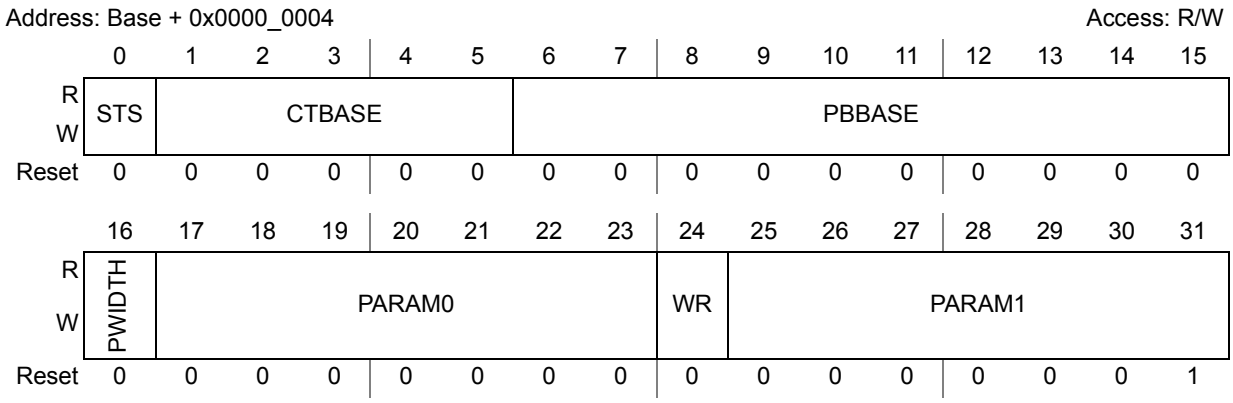


Figure 369. eTPU Coherent Dual-Parameter Controller Register (ETPU_CDCR)

Table 224. ETPU_CDCR field descriptions

Field	Description
0 STS	Start Set by the host to start the data transfer between the parameter buffer pointed by PBBASE and the target addresses selected by the concatenation of fields CTBASE and PARM0/1. The host receives wait-states until the data transfer is complete. Coherency logic resets STS once the data transfer is complete. 0: (Write) does not start a coherent transfer. 1: (Write) starts a coherent transfer.
1–5 CTBASE [0:4]	Channel transfer base This field concatenates with fields PARM0/PARM1 to determine the absolute offset (from the SDM base) of the parameters to be transferred: Parameter 0 address = {CTBASE, PARM0} × 4 + SDM base Parameter 1 address = {CTBASE, PARM1} × 4 + SDM base
6–15 PBBASE [0:9]	Parameter buffer base address Points to the base address of the parameter buffer location, with granularity of 2 parameters (8 bytes). The host (byte) address of the first parameter in the buffer is PBBASE × 8 + SDM Base Address.
16 PWIDTH	Parameter width selection Selects the width of the parameters to be transferred between the PB and the target address. 0: Transfer 24-bit parameters. The upper byte remains unchanged in the destination address. 1: Transfer 32-bit parameters. All 32 bits of the parameters are written in the destination address.
17–23 PARAM0 [0:6]	Channel parameter number 0 This field in concatenation with CTBASE[3:0] determine the address offset (from the SDM base address) of the parameter which is the destination or source (defined by WR) of the coherent transfer. The SDM address offset of the parameter is {CTBASE, PARM0} × 4. PARM0 allows non-contiguous parameters to be transferred coherently(1).
24 WR	Read/Write selection This bit selects the direction of the coherent data transfer. 0: Read operation. Data transfer is from the selected parameter RAM address to the PB. 1: Write operation. Data transfer is from the PB to the selected parameter RAM address.

Table 224. ETPU_CDCR field descriptions (continued)

Field	Description
25–31 PARAM1 [0:6]	Channel parameter number 1 This field in concatenation with CTBASE[3:0] determines the address offset (from the SDM base) of the parameter which is the destination or source (defined by WR) of the coherent transfer. The SDM address offset of the parameter is {CTBASE, PARM1} × 4. PARM1 allows non-contiguous parameters to be transferred coherently[1].

1. The parameter pointed by {CTBASE, PARM0} is the first transferred.

22.10.1.3 eTPU MISC Compare Register (ETPU_MISCCMPR)

The multiple input signature calculator compare register (ETPU_MISCCMPR) holds the 32-bit signature expected from the whole shared code memory (SCM) array. This register must be written by the host with the 32-bit word to be compared against the calculated signature at the end of the MISC cycle.

Address: Base + 0x0000_000C Access: R/W

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

Figure 370. eTPU MISC Compare Register (ETPU_MISCCMPR)

Table 225. ETPU_MISCCMPR field descriptions

Field	Description
0–31 ETPUMISCCMP[0:31]	Expected multiple input signature calculator compare register value.

22.10.1.4 eTPU SCM Off-Range Data Register (ETPU_SCMOFFDATAR)

ETPU_SCMOFFDATAR holds the 32-bit value returned when the SCM array is accessed at non implemented addresses, either by the host or by the microengine. This register can be written by the host with the 32-bit instruction to be executed by the microengine to recover from runaway code.

Note: The ETPU_SCMOFFDATAR reset value is the opcode of an instruction that disables matches, clears the TDLs and the MRLs; the opcode also issues an invalid instruction Global Exception, and ends the thread.

Address: Base + 0x0000_0010 Access: R/W

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

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

Figure 371. eTPU SCM Off-Range Data Register (ETPU_SCMOFFDATAR)

Table 226. ETPU_SCMOFFDATAR field descriptions

Field	Description
0–31 ETPUSCMOFFDATA	SCM Off-range read data value.

22.10.1.5 eTPU Engine Configuration Register (ETPU_ECR)

The ETPU_ECR holds configuration and status fields that are programmed in the eTPU engine.

Address: Base + 0x0000_0014 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEND	MDIS	0	STF	0	0	0	0	HLTF	0	0	0	FCSS	FPSCK		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 372. eTPU Engine Configuration Register (ETPU_ECR)

Table 227. ETPU_ECR field descriptions

Field	Description
0 FEND	Force end Assertion terminates any current running thread as if an END instruction have been executed. 0: Normal operation 1: Terminates current thread This bit is self-negating.

Table 227. ETPU_ECR field descriptions (continued)

Field	Description
1 MDIS	Module disable internal stop This is the low-power stop bit. When MDIS is set, the eTPU engine shuts down its internal clocks. TCR1 and TCR2 cease to increment, and input sampling stops. The eTPU engine asserts the stop flag (STF) bit to indicate that it has stopped. However, the BIU continues to run, and the host can access all registers except for the channel registers(1) and writes to time base registers. *Section 22.10.5: Channel configuration and control registers*. After MDIS is set, even before STF asserts, data read from the channel registers is not meaningful, a Bus Error is issued, and writes are unpredictable. When the MDIS bit is asserted while the microcode is executing, the eTPU stops when the thread is complete. 0: eTPU engine runs. 1: Commands eTPU engine to stop its clocks. Stop completes on the next system clock after the stop condition is valid. The MDIS bit is write-protected when ETPU_MCR[VIS] = 1. **Note:** After the MDIS has been switched from 1 to 0 or vice-versa, do not switch its value again until STF is switched to the same value.
2	Reserved
3 STF	Stop flag bit The eTPU engine asserts its stop flag (STF) to indicate that it has stopped. The host can then detect that the eTPU engine has actually stopped. The eTPU system is fully stopped when the STF bits of the eTPU engine is asserted. The eTPU engine only stops when any ongoing thread is complete in this case. 0: The eTPU engine is operating. 1: The eTPU engine has stopped (after the local MDIS bit has been asserted). Summarizing eTPU engine stop conditions, which STF reflects: STF A:= (after stop completed) MDIS A. STF A means the STF bit from eTPU engine A.
4–7	Reserved
8 HLTF	Halt mode flag If eTPU engine entered halt state, this flag is asserted. The flag remains asserted while the microengine is in halt state, even during a single-step or forced instruction execution. 0: eTPU engine is not halted. 1: eTPU engine is halted.
9–11	Reserved
12 FCSS	Filter Clock Source Selection Speeds up the filter clock source before the prescaler, allowing more input capture resolution at minimum prescaling. 1: Use system clock as EDF clock source before prescaler 0: Use system clock / 2 as EDF clock source before prescaler

Table 227. ETPU_ECR field descriptions (continued)

Field	Description
13–15 FPSCK [0:2]	Filter prescaler clock control Controls the prescaling of the clocks used in digital filters for the channel input signals and TCRCLK input. The following field values illustrate the filter prescaler clock control configuration: 000: Sample on system clock divided by 2 001: Sample on system clock divided by 4 010: Sample on system clock divided by 8 011: Sample on system clock divided by 16 100: Sample on system clock divided by 32 101: Sample on system clock divided by 64 110: Sample on system clock divided by 128 111: Sample on system clock divided by 256 Filtering can be controlled independently by the eTPU engine, but all input digital filters in the same eTPU engine have same clock prescaling.
16–17 CDFC [0:1]	Channel digital filter control Select a digital filtering mode for the channels when configured as inputs for improved noise immunity. The eTPU has three digital filtering modes for the channels which provide programmable trade-off between signal latency and noise immunity. Changing CDFC during eTPU normal input channel operation is not recommended since it changes the behavior of the transition detection logic while executing its operation. Channel digital filter control is illustrated in *Table 228*.
18	Reserved
19–23	Engine Relative Base Address This field value is concatenated with the AID instruction field in eTPU engine relative address mode to form the SPRAM address.
24 SPPDIS	Schedule Priority Passing Disable SPPDIS is used to disable the priority passing mechanism of the microengine scheduler. 1: Scheduler priority passing mechanism disabled. 0: Scheduler priority passing mechanism enabled. **Note:** SPPDIS bit must not be changed while any channel is enabled.
25–26	Reserved
27–31 ETB [0:4]	Entry table base Determines the location of the microcode entry table for the eTPU functions in SCM. *Table 229* shows the entry table base address options.

1. The time base registers can still be read in stop mode, but writes are ineffective and a bus error is issued. Global channel registers and SDM can be accessed normally.

Table 228. CDFC[0:1] configuration—Channel digital filter control

CDFC[0:1]	Selected digital filter
00	TPU2/3 two sample mode: Using the filter clock which is the system clock divided by (2, 4, 8,..., 256) as a sampling clock (selected by FPSCK field in ETPU_ECR), comparing two consecutive samples which agree with the other sets the input signal state. This is the default reset state.
01	Invalid value

Table 228. CDFC[0:1] configuration—Channel digital filter control (continued)

CDFC[0:1]	Selected digital filter
10	eTPU three sample mode: Similar to the TPU2/3 two sample mode, but comparing three consecutive samples which agree with the other sets the input signal state.
11	eTPU continuous mode: Signal needs to be stable for the whole filter clock period. This mode compares all the values at the rate of system clock divided by two, between two consecutive filter clock pulses. If all the values agree with the other, input signal state is updated.

Table 229. ETB[0:4] configuration—Entry table base address options

ETB[0:4]	Entry table base address for	
	CPU host address (byte format)	Microcode address (word format)
00000	0x0000_0000	0x0000_0000
00001	0x0000_0800	0x0000_0200
00010	0x0000_1000	0x0000_0400
. . . .	. . . .	. . . .
11110	0x0000_F000	0x0000_3C00
11111	0x0000_F800	0x0000_3E00

22.10.2 Time base registers

Time base registers allow the configuration and visibility of internally-generated time bases TCR1 and TCR2. The time base registers for the eTPU engine can only be accessed in supervisor mode.

Note: Writes to this register issue a bus error and are ineffective when MDIS = 1. Reads are always allowed.

22.10.2.1 eTPU Time Base Configuration Register (ETPU_TBCR)

This register configures several time base options.

Address: Base + 0x0000_0020 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCR2CTL			TCRCF		AM		0	0	0	TCR2P					
W																
Reset	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR1CTL		TCR1CS	0	0	0	0	0	TCR1P							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 373. eTPU Time Base Configuration Register (ETPU_TBCR)

Table 230. ETPU_TBCR field descriptions

Field	Description
0–2 TCR2CTL	TCR2 clock/gate control Part of the TCR2 clocking system. These bits determine the clock source for TCR2 before the prescaler. TCR2 can count on any detected edge of the TCRCLK signal or use it for gating system clock divided by 8. After reset, the TCRCLK signal rising edge is selected. TCR2 can also be clocked by the system clock divided by 8. TCR2CTL also determines the TCRCLK edge selected for angle tooth detection in angle mode. TCR2 clock sources are listed in *Table 232*.
3–4 TCRCF	TCRCLK signal filter control Controls the TCRCLK digital filter determining whether the TCRCLK signal input (after a synchronizer) is filtered with the same filter clock as the channel input signals or uses the system clock divided by 2, and also whether the TCRCLK digital filter works in integrator mode or two sample mode. *Table 233* describes TCRCLK filter clock/mode.
5–6 AM	Angle mode selection This field enables the Enhanced Angle Counter logic to generate angle information , and selects the tooth signal input and the channel used to process the same, as shown in *Table 231*. When EAC is not disabled by AM and neither TCR1 nor TCR2 are STAC Clients, the EAC (eTPU Angle Clock) hardware provides angle information to the channels using the TCR2 bus. When AM is reset (non-angle mode), the EAC operation is disabled, and its internal registers can be used as general purpose.

Table 231. AM - angle mode selection

Value	TCR2 value	Tooth signal	Tooth processing channel
0 0	Timebase (EAC operation disabled)	not applicable	
0 1	Angle Ticks	TCRCLK input	0
1 0		channel 1 input	1
1 1		channel 2 input	2

If TCR1 or TCR2 is a STAC Bus Client, the EAC operation is not allowed, and if AM is set the Angle Logic does not work properly.

Note: Changing AM might cause spurious transition detections on the channel selected by AM, depending on the channel mode and state .

Table 230. ETPU_TBCR field descriptions (continued)

Field	Description
7–9	Reserved
10–15 TCR2P	Timer count register 2 prescaler control Part of the TCR2 clocking system. TCR2 is clocked from the output of a prescaler. The prescaler divides its input by (TCR2P+1) allowing frequency divisions from 1 to 64. The prescaler input is the system clock divided by 8 (in gated or non-gated clock mode) or Internal Timebase input, or TCRCLK filtered input. This field has no effect on TCCR2 in Angle Mode.
16–17 TCR1CTL	TCR1 clock/gate control Part of the TCR1 clocking system. It determines the clock source for TCR1. TCR1 can count on detected rising edge of the TCRCLK signal or the system clock divided by 2. After reset TCRCLK signal is selected. The following values show the selection of the TCR1 clock source. 00: Selects TCRCLK as clock source for the TCR1 prescaler (must not be used in Angle Mode) 01: Invalid value 10: Selects system clock divided by 2 as clock source for the TCR1 prescaler 11: TCR1CTL shuts down TCR1 clock. TCR1 can still change if STAC client.
18 TCR1CS	TCR1 Clock Source TCR1CS provides the option to double the TCR1 incrementing speed, using system clock as its clock source instead of system clock / 2. 1: Use system clock as TCR1 clock source before the prescaler; can only be set in specific combinations with TCR1CTL (see *Table 234*). 0: Use system clock / 2 as TCR1 clock source before the prescaler, if that clock source is selected by TCR1CTL. The clock source of the EAC angle tick generator will still be an even division of system clock if TCR1CS = 1, obeying to the fields TCR1P as if TCR1CS = 0.
24–31 TCR1P	Timer count register 1 prescaler control Clocked from the output of a prescaler. The input to the prescaler is the internal eTPU system clock divided by 2 or the output of TCRCLK filter, or Peripheral Timebase input. The prescaler divides this input by (TCR1P+1) allowing frequency divisions from 1 up to 256.

Table 232. TCR2CTL configuration—TCR2 clock sources

TCR2CTL	AM = 0 (TCR2 clock)	AM = 1 (Angle Tooth Detection)
000	Gated DIV8 clock (system clock / 8). When the external TCRCLK signal is low, the DIV8 clock is blocked, preventing it from incrementing the TCR2 prescaler. When the external TCRCLK signal is high, TCR2 prescaler is incremented at the frequency of the system clock divided by 8.	Do not use 000 with AM = 1.
001	Rise transition on TCRCLK signal increments TCR2 prescaler.	Rising edge
010	Fall transition on TCRCLK signal increments TCR2 prescaler.	Falling edge
011	Rise or fall transition on TCRCLK signal increments TCR2 prescaler.	Rising or falling edge

Table 232. TCR2CTL configuration—TCR2 clock sources (continued)

TCR2CTL	AM = 0 (TCR2 clock)	AM = 1 (Angle Tooth Detection)
100	DIV8 clock (system clock / 8)	Do not use 1XX with AM = 1.
101	Invalid value	
110	Invalid value	
111	TCR2CTL shuts down TCR2 clocking, except on Angle Mode. TCR2 can also change as STAC client.	

Table 233. TCRCF configuration—TCRCLK filter clock/mode

TCRCF	Filter input	Filter mode
00	System clock divided by 2	Two sample
01	Filter clock of the channels	Two sample
10	System clock divided by 2	Integration
11	Filter clock of the channels	Integration

Table 234. TCR1 clock source

TCR1CTL	TCR1CS(1)	TCR1 clock before prescaler
00	0	Selects TCRCLK as clock source for the TCR1 prescaler(2)
01	0	Selects Peripheral Timebase clock as source for the TCR1 prescaler
10	0	Selects system clock divided by 2 as clock source for the TCR1 prescaler
10	1	Selects system clock as clock source for the TCR1 prescaler
11	0	TCR1 frozen, except as a STAC client

1. All other combinations of TCR1CTL and TCR1CS are reserved.
2. This selection must not be used in Angle Mode.

22.10.2.2 eTPU Time Base 1 (TCR1) Visibility Register (ETPU_TB1R)

This register provides visibility of the TCR1 time base for core host read access. This register is read-only. The value of the TCR1 time base shown can be driven by the TCR1 counter or imported, depending on the configuration set in ETPU_REDCR.

Address: Base + 0x0000_0024 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TCR1							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR1															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 374. eTPU Time Base 1 (TCR1) Visibility Register (ETPU_TB1R)

Table 235. ETPU_TB1R field descriptions

Field	Description
0–7	Reserved
8–31 TCR1[0:23]	TCR1 value Used on matches and captures.

22.10.2.3 eTPU Time Base 2 (TCR2) Visibility Register (ETPU_TB2R)

This register provides visibility of the TCR2 time base for core host read access. This register is read-only. The value of the TCR2 time base shown can be driven by the TCR2 counter, the angle mode logic, or imported from the STAC interface, depending on angle mode (an eTPU engine cannot import when in angle mode) and STAC interface configurations set in registers ETPU_TBCR and ETPU_REDCR.

Address: Base + 0x0000_0028 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TCR2							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR2															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 375. eTPU Time Base 2 (TCR2) Visibility Register (ETPU_TB2R)

Table 236. ETPU_TB2R field descriptions

Field	Description
0–7	Reserved
8–31 TCR2[0:23]	TCR2 value Used on matches and captures.

22.10.2.4 eTPU STAC Bus Configuration Register (ETPU_REDCR)

This register configures the eTPU STAC bus interface module and operation.

Address: Base + 0x0000_002C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REN1	RSC1	0	0	SERVER_ID1				0	0	0	0	SRV1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REN2	RSC2	0	0	SERVER_ID2				0	0	0	0	SRV2			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 376. STAC Bus Configuration Register (ETPU_REDCR)

Table 237. ETPU_REDCR field descriptions

Field	Description
0 REN1	TCR1 resource(1) client/server operation enable Enables or disables client/server operation for the eTPU STAC interface. REN1 enables TCR1. 0: Server/client operation for resource 1 is disabled. 1: Server/client operation for resource 1 is enabled.
1 RSC1	TCR1 resource server/client assignment Selects the eTPU data resource assignment to be used as a server or client. RSC1 selects the functionality of TCR1. For server mode, external plugging determines the unique server address assigned to each TCR. For a client mode, the SRV1 field determines the server address to which the client listens. 0: Resource client operation 1: Resource server operation
2–3	Reserved
4–7 SERVER_ID1	STAC bus address for TCR1 as a server
8–11	Reserved
12–15 SRV1[0:3]	TCR1 resource server Selects the address of the specific STAC Server the local TCR1 monitors when configured as a STAC client.
16 REN2	TCR2 resource1 client/server operation enable Enables or disables client/server operation for eTPU slave resources. REN2 enables TCR2 slave bus operations. 1: Server/client operation for resource 2 is enabled. 0: Server/client operation for resource 2 is disabled.

Table 237. ETPU_REDCR field descriptions (continued)

Field	Description
17 RSC2	TCR2[2] resource server/client assignment Selects the eTPU data resource assignment to be used as a server or client. RSC2 selects the functionality of TCR2. For server mode, external plugging determines the unique server address assigned to each TCR. For a client mode, the SRV2 field determines the Server address to which the client listens. 0: Resource client operation 1: Resource server operation
18–19	Reserved
20–23 SERVER_ID2	STAC bus address for TCR2 as a server.
24–27	Reserved
28–31 SRV2[0:3]	TCR2 resource server Selects the address of the specific STAC server the local TCR2 listens to when configured as a STAC client.

1. Resource identifies any parameter that changes in time and can be exported / imported from other device. For the eTPU, a resource can be TCR1 or TCR2 (either time or angle values).
2. When TCR2 is configured as a STAC bus client (REN2 = 1, RSC2 = 0) the angle clock hardware must be disabled (ETPU_TBCR[AM] = 0).

22.10.3 Engine related registers

This section gathers registers that are engine-related, other than ETPU_ECR (see *Section 22.10.1.5: eTPU Engine Configuration Register (ETPU_ECR)*).

22.10.3.1 eTPU Watchdog Timer Register (ETPU_WDTR)

This register configures the watchdog timer for the engine.

Offset: eTPU_A: eTPU_Base + 0x060 Access: User R/W

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

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

= Unimplemented or Reserved

Figure 377. eTPU Watchdog Timer Register (ETPU_WDTR)

Table 238. ETPU_WDTR field descriptions

Field	Description
0 WDM	Watchdog Mode WDM selects the Watchdog operation mode, as shown below. 0: Thread length 1: Busy length **Note:** The watchdog must be disabled first before a new mode is configured.
1-15	Reserved
16-31 WDCNT [15:0]	Watchdog Count This field indicates the maximum number of microcycles allowed for a thread (in thread length mode) or a sequence of threads (in busy length mode) before the current running thread is forced to end. **Note:** The TST microcycles are also counted by the watchdog.

22.10.3.2 eTPU Idle Register (ETPU_IDLE)

The Idle Counter Register (ETPU_IDLE) continuously counts microcycles in which the microengine is not busy with channel service. It can be used to measure the microengine utilization by rating the count measured during a period of time to the number of microcycles contained in the period. The Idle counter does not count microcycles when the eTPU engine is stopped, or is in TST or halt states.

Offset: eTPU_A: eTPU_Base + 0x068 Access: User R/W

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

= Unimplemented or Reserved

Figure 378. eTPU Idle Register (ETPU_IDLE)

Table 239. ETPU_IDLE field descriptions

Field	Description
0-31 IDLE_CNT[31:0]	Idle Count This is a free-running count of the number of idle microcycles in the microengine.
31 ICLR	Idle Clear This write-only bit is used to clear the idle count IDLE_CNT. 1: Clear the idle count IDLE_CNT 0: Do not clear idle count IDLE_CNT

22.10.4 Global channel registers

The registers in this section group, by type, the interrupt status and enable bits from all the channels. This organization eases management of all channels or groups of channels by a single interrupt handler routine. These bits are mirrored by the individual channel registers.

22.10.4.1 eTPU Channel Interrupt Status Register (ETPU_CISR)

Host interrupt status from all channels are grouped in ETPU_CISR. The bits are mirrored by the channels' status/control registers. For more information, see *Section 22.10.5.3: eTPU Channel n Status Control Register (ETPU_CnSCR)*.

Address: Base + 0x0000_0200 Access: R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIS31	CIS30	CIS29	CIS28	CIS27	CIS26	CIS25	CIS24	CIS23	CIS22	CIS21	CIS20	CIS19	CIS18	CIS17	CIS16
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	CIS15	CIS14	CIS13	CIS12	CIS11	CIS10	CIS9	CIS8	CIS7	CIS6	CIS5	CIS4	CIS3	CIS2	CIS1	CIS0
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 379. eTPU Channel Interrupt Status Register (ETPU_CISR)

Table 240. ETPU_CISR field descriptions

Field	Description
0–31 CIS*n*	Channel *n* interrupt status 0: indicates that channel *n* has no pending interrupt to the host core. 1: indicates that channel *n* has a pending interrupt to the host core. To clear a status bit, the host must write '1' to it.

22.10.4.2 eTPU Channel Data Transfer Request Status Register (ETPU_CDTRSR)

Data transfer request status from all channels are grouped in ETPU_CDTRSR. The bits are mirrored by the channels' status/control registers.

Address: Base + 0x0000_0210 Access: R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DTRS31	DTRS30	DTRS29	DTRS28	DTRS27	DTRS26	DTRS25	DTRS24	DTRS23	DTRS22	DTRS21	DTRS20	DTRS19	DTRS18	DTRS17	DTRS16
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	DTRS15	DTRS14	DTRS13	DTRS12	DTRS11	DTRS10	DTRS9	DTRS8	DTRS7	DTRS6	DTRS5	DTRS4	DTRS3	DTRS2	DTRS1	DTRS0
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 380. eTPU Channel Data Transfer Request Status Register (ETPU_CDTRSR)

Table 241. ETPU_CDTRSR field descriptions

Field	Description
0–31 DTRS*n*	Channel *n* data transfer request status 0: Indicates that channel *n* has no pending data transfer request. 1: Indicates that channel *n* has a pending data transfer request. To clear a status bit, the host must write ‘1’ to it.

22.10.4.3 eTPU Channel Interrupt Overflow Status Register (ETPU_CIOSR)

An interrupt overflow occurs when an interrupt is issued for a channel when the previous interrupt status bit for the same channel has not been cleared. Interrupt overflow status from all channels are grouped in ETPU_CIOSR. The bits are mirrored by the channels’ status/control registers. For information about channel status registers and overflow, see *Section 22.10.5.3: eTPU Channel n Status Control Register (ETPU_CnSCR)*.

Address: Base + 0x0000_0220 Access: R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIOS31	CIOS30	CIOS29	CIOS28	CIOS27	CIOS26	CIOS25	CIOS24	CIOS23	CIOS22	CIOS21	CIOS20	CIOS19	CIOS18	CIOS17	CIOS16
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	CIOS15	CIOS14	CIOS13	CIOS12	CIOS11	CIOS10	CIOS9	CIOS8	CIOS7	CIOS6	CIOS5	CIOS4	CIOS3	CIOS2	CIOS1	CIOS0
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 381. eTPU Channel Interrupt Overflow Status Register (ETPU_CIOSR)

Table 242. ETPU_CIOSR field descriptions

Field	Description
0–31 CIOS*n*	Channel *n* interrupt overflow status 0: Indicates that no interrupt overflow occurred in the channel 1: Indicates that an interrupt overflow occurred in the channel To clear a status bit, the host must write ‘1’ to it.

22.10.4.4 eTPU Channel Data Transfer Request Overflow Status Register (ETPU_CDTROSR)

Data transfer request overflow status from all channels are grouped in ETPU_CDTROSR. The bits are mirrored by the channels’ status/control registers. For more information on channel status registers and data transfer request overflow, see *Section 22.10.5.3: eTPU Channel n Status Control Register (ETPU_CnSCR)*.

Address: Base + 0x0000_0230 Access: R/W1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DTROS31	DTROS30	DTROS29	DTROS28	DTROS27	DTROS26	DTROS25	DTROS24	DTROS23	DTROS22	DTROS21	DTROS20	DTROS19	DTROS18	DTROS17	DTROS16
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	DTROS15	DTROS14	DTROS13	DTROS12	DTROS11	DTROS10	DTROS9	DTROS8	DTROS7	DTROS6	DTROS5	DTROS4	DTROS3	DTROS2	DTROS1	DTROS0
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 382. eTPU Channel Data Transfer Request Overflow Status Register (ETPU_CDTROSR)

Table 243. ETPU_CDTROSR field descriptions

Field	Description
0–31 DTROS*n*	Channel *n* data transfer request overflow status 0: indicates that no data transfer request overflow occurred in the channel. 1: indicates that a data transfer request overflow occurred in the channel. To clear a status bit, the host must write ‘1’ to it.

22.10.4.5 eTPU Channel Interrupt Enable Register (ETPU_CIER)

The host interrupt enable bits for all 32 channels are grouped in ETPU_CIER. The bits are mirrored by the channel configuration registers. For more information on channel configuration registers and interrupt enable, see *Section 22.10.5.2: eTPU Channel n Configuration Register (ETPU_CnCR)*.

Address: Base + 0x0000_0240 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CIE31	CIE30	CIE29	CIE28	CIE27	CIE26	CIE25	CIE24	CIE23	CIE22	CIE21	CIE20	CIE19	CIE18	CIE17	CIE16
Reset	0	0	0	0	0	0	0	0	0	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	CIE15	CIE14	CIE13	CIE12	CIE11	CIE10	CIE9	CIE8	CIE7	CIE6	CIE5	CIE4	CIE3	CIE2	CIE1	CIE0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 383. eTPU Channel Interrupt Enable Register (ETPU_CIER)

Table 244. ETPU_CIER field descriptions

Field	Description
0–31 CIE*n*	Channel *n* interrupt enable Enable the eTPU channels to interrupt the device core. 0: Interrupt disabled for channel *n*. 1: Interrupt enabled for channel *n*.

22.10.4.6 eTPU Channel Data Transfer Request Enable Register (ETPU_CDTRER)

Data transfer request enable status bits from all channels are grouped in ETPU_CDTRER. The bits are mirrored in the channels' configuration registers. For more on configuration registers and data transfer request enable, see *Section 22.10.5.2: eTPU Channel n Configuration Register (ETPU_CnCR)*. Access is restricted to supervisor only.

Address: Base + 0x0000_0250 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DTRE31	DTRE30	DTRE29	DTRE28	DTRE27	DTRE26	DTRE25	DTRE24	DTRE23	DTRE22	DTRE21	DTRE20	DTRE19	DTRE18	DTRE17	DTRE16
Reset	0	0	0	0	0	0	0	0	0	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	DTRE15	DTRE14	DTRE13	DTRE12	DTRE11	DTRE10	DTRE9	DTRE8	DTRE7	DTRE6	DTRE5	DTRE4	DTRE3	DTRE2	DTRE1	DTRE0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 384. eTPU Channel Data Transfer Request Enable Register (ETPU_CDTRER)

Table 245. ETPU_CDTRER field descriptions

Field	Description
0–31 DTRE*n*	Channel *n* data transfer request enable Enable data transfer requests for their respective channels. 0: Data transfer request disabled for channel *n*. 1: Data transfer request enabled for channel *n*.

22.10.4.7 eTPU Watchdog Status Register (ETPU_WDSR)

ETPU_WDSR indicates the watchdog influence in each of the eTPU engine channels.

Offset: eTPU_A: eTPU_Base + 0x260 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WDS31	WDS30	WDS29	WDS28	WDS27	WDS26	WDS25	WDS24	WDS23	WDS22	WDS21	WDS20	WDS19	WDS18	WDS17	WDS16
W	WDSC31	WDSC30	WDSC29	WDSC28	WDSC27	WDSC26	WDSC25	WDSC24	WDSC23	WDSC22	WDSC21	WDSC20	WDSC19	WDSC18	WDSC17	WDSC16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WDS15	WDS14	WDS13	WDS12	WDS11	WDS10	WDS9	WDS8	WDS7	WDS6	WDS5	WDS4	WDS3	WDS2	WDS1	WDS0
W	WDSC15	WDSC14	WDSC13	WDSC12	WDSC11	WDSC10	WDSC9	WDSC8	WDSC7	WDSC6	WDSC5	WDSC4	WDSC3	WDSC2	WDSC1	WDSC0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 385. ETPU_WDSR

Table 246. ETPU_WDSR field descriptions

Field	Description
0-31 WDSx	Channel x Watchdog Status 1: Watchdog forced end of the channel thread and disabled it. 0: No watchdog force on the channel.
0-31 WDSCx	Channel x Watchdog Status Clear 1: Clear watchdog status bit. 0: Keep watchdog status bit unaltered.

22.10.4.8 eTPU Channel Pending Service Status Register (ETPU_CPSSR)

ETPU_CPSSR is a read-only register that holds the status of the pending channel service requests.

Note: More than one source can request service when a channel's service request bit is set.

Address: Base + 0x0000_0280 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SR31	SR30	SR29	SR28	SR27	SR26	SR25	SR24	SR23	SR22	SR21	SR20	SR19	SR18	SR17	SR16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SR15	SR14	SR13	SR12	SR11	SR10	SR9	SR8	SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 386. eTPU Channel Pending Service Status Register (ETPU_CPSSR)

Table 247. ETPU_CPSSR field descriptions

Field	Description
0–31 SR*n*	Pending service request *n* Indicates a pending service request for channel *n*. The SR status for the pending request is negated at the time slot transition for the respective service thread. 0: No service request pending for channel *n* 1: Pending service request for channel *n*

Note: The pending service status bit for a channel is set when a service request is pending, even if the Channel is disabled (CPRn = 0).

22.10.4.9 eTPU Channel Service Status Register (ETPU_CSSR)

ETPU_CSSR holds the current channel service status on whether it is being serviced or not. Only one bit can be asserted in this register at a given time. When no channel is being serviced the register read value is 0x0000_0000. ETPU_CSSR is a read-only register. The register can be read during normal eTPU operation for monitoring the scheduler activity.

Note: The ETPU_CSSR is not an absolute indication of channel status. If more than one source is requesting service, the asserted status bit only indicates that one of the requests has been granted.

Note: Channel service status does not always reflect decoding of the CHAN register, since the CHAN register can be changed by the service thread microcode.

Address: Base + 0x0000_0290 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SS31	SS30	SS29	SS28	SS27	SS26	SS25	SS24	SS23	SS22	SS21	SS20	SS19	SS18	SS17	SS16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SS15	SS14	SS13	SS12	SS11	SS10	SS9	SS8	SS7	SS6	SS5	SS4	SS3	SS2	SS1	SS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 387. eTPU Channel Service Status Register (ETPU_CSSR)

Table 248. ETPU_CSSR field descriptions

Field	Description
0–31 SS*n*	Service status *n* Indicates that channel *n* is currently being serviced. It is updated at the 1st microcycle of a time slot transition. 0: Channel *n* is not currently being serviced 1: Channel *n* is currently being serviced

22.10.5 Channel configuration and control registers

Each channel of the eTPU engine has a group of three registers used to control, configure and check status of that channel as shown in *Table 249*.

Table 249. Channel registers structure

Channel offset	Register name
0x0000	eTPU channel configuration register (C*n*CR)
0x0004	eTPU channel status/control register (C*n*SCR)
0x0008	eTPU channel host service request register (C*n*HSRR)
0x000C	Reserved

22.10.5.1 Channel registers layout

One contiguous area is used to map all channel registers of the eTPU engine as shown in *Table 250*.

Table 250. eTPU channel register map

Address	Registers structure
Base + 0x0000_0400	eTPU_A channel 0 register structure
Base + 0x0000_0410	eTPU_A channel 1 register structure
Base + 0x0000_0420	eTPU_A channel 2 register structure
Base + 0x0000_0430–0x0000_05D0	. . .
Base + 0x0000_05E0	eTPU_A channel 30 register structure
Base + 0x0000_05F0	eTPU_A channel 31 register structure
Base + 0x0000_0600–0x0000_07FF	Reserved

There are 32 structures defined, one for each available channel in the eTPU. The base address for the structure presented can be calculated by using the following equation:

Channel_Register_Structure_Base_Address =
ETPU_Engine_Channel_Base + (channel_number × 0x0000_0010)

where:

ETPU_Engine_Channel_Base = ETPU_Base + (0x0000_0400).

22.10.5.2 eTPU Channel *n* Configuration Register (ETPU_C*n*CR)

The C*n*CR is a collection of the configuration bits related to an individual channel. Some of these bits are mirrored from the global channel registers. ETPU_C*n*CR accesses are supervisor only.

Address: Channel_Register_Base + 0x0000 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIE	DTRE	CPR		0	0	ETPD	ETCS	0	0	0	CFS				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 388. eTPU Channel n Configuration Register (ETPU_CnCR)

Table 251. C*n*CR field descriptions

Field	Description
0 CIE	Channel interrupt enable This bit is mirrored from the ETPU_CIER. 0: Disable interrupt for this channel. 1: Enable interrupt for this channel.
1 DTRE	Channel data transfer request enable This bit is mirrored from the ETPU_CDTRER. 0: Disable data transfer request for this channel. 1: Enable data transfer request for this channel.
2–3 CPR [0:1]	Channel priority Defines the priority level for the channel. The priority level is used by the hardware scheduler. The values for CPR[1:0] and priority levels are: 00: Disabled 01: Low 10: Middle 11: High
4–5	Reserved
6 ETPD	Entry point pin direction This bit selects which channel signal, input or output, is used in the entry point selection. The ETPD value has to be compatible with the function chosen for the channel, selected in the field CFS. 0: Use PSTI for entry point selection. 1: Use PSTO for entry point selection.
7 ETCS	Entry table condition select Determines the channel condition encoding scheme that selects the entry point to be taken in an entry table. The ETCS value has to be compatible with the function chosen for the channel, selected in C*n*CR[CFS]. Two condition encoding schemes are available. 1: Select alternate entry table condition encoding scheme. 0: Select standard entry table condition encoding scheme.

Table 251. C*n*CR field descriptions (continued)

Field	Description
8–10	Reserved
11–15 CFS [0:4]	Channel function select Defines the function to be performed by the channel. The function assigned to the channel has to be compatible with the channel condition encoding scheme, selected by C*n*CR[ETCS].
16 ODIS	Output disable Enables the channel to have its output forced to the value opposite to OPOL when the output disable input signal corresponding to the channel group that it belongs is active. 0: Turns off the output disable feature for the channel. 1: Turns on the output disable feature for the channel.
17 OPOL	Output polarity Determines the output signal polarity. The activation of the output disable signal forces, when enabled by C*n*CR[ODIS], the channel output signal to the opposite of this polarity. 0: Output active low (output disable drives output to high) 1: Output active high (output disable drives output to low)
18–20	Reserved
21–31 CPBA [0:10]	Channel *n* parameter base address The value of this field multiplied by 8 specifies the SDM parameter base host (byte) address for channel *n* (2-parameter granularity). The formula for calculating the absolute channel parameter base (byte) address, as seen by the host, is eTPU_Base + 0x8000 + CPBA × 8. The SDM is mirrored in the parameter sign extension (PSE) area. The formula to calculate the absolute channel parameter base (byte) address in the PSE area is eTPU_Base + 0xC000 + CPBA × 8.

22.10.5.3 eTPU Channel *n* Status Control Register (ETPU_C*n*SCR)

C*n*SCR is a collection of the interrupt status bits of the channel, and also the function mode definition (read-write). Bits CIS, CIOS, DTRS, and DTROS for each channel can also be accessed from ETPU_CISR, ETPU_CIOSR, ETPU_CDTRSR, and ETPU_CDTROSR respectively.

Note: The device core must write '1' to clear a status bit.

Address: Channel_Register_Base + 0x0004 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIS	CIOS	0	0	0	0	0	0	DTRS	DTROS	0	0	0	0	0	0
W	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	IPS	OPS	OBE	0	0	0	0	0	0	0	0	0	0	0	FM	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 389. eTPU Channel n Status Control Register (ETPU_CnSCR)

Table 252. C*n*SCR field descriptions

Field	Description
0 CIS	Channel interrupt status 0: Channel has no pending interrupt to the device core. 1: Channel has a pending interrupt to the device core. CIS is mirrored in the ETPU_CISR. The core must write '1' to clear CIS. For more information on ETPU_CISR and interrupts, see *Section 22.10.4.1: eTPU Channel Interrupt Status Register (ETPU_CISR)*.
1 CIOS	Channel interrupt overflow status 0: Interrupt overflow negated for this channel 1: Interrupt overflow asserted for this channel CIOS is mirrored in the ETPU_CIOSR. The core must write '1' to clear CIOS. For more information on the ETPU_CIOSR and interrupt overflow, see *Section 22.10.4.3: eTPU Channel Interrupt Overflow Status Register (ETPU_CIOSR)*.
2–7	Reserved
8 DTRS	Data transfer request status 0: Channel has no pending data transfer request. 1: Channel has a pending data transfer request. DTRS is mirrored in the ETPU_CDTRSR. The core must write '1' to clear DTRS. For more information on the ETPU_CDTRSR and data transfer, see *Section 22.10.4.2: eTPU Channel Data Transfer Request Status Register (ETPU_CDTRSR)*.
9 DTROS	Data transfer request overflow status 0: Data transfer request overflow negated for this channel. 1: Data transfer request overflow asserted for this channel. DTROS is mirrored in the ETPU_CDTROSR. The core must write '1' to clear DTROS. See *Section 22.10.4.4: eTPU Channel Data Transfer Request Overflow Status Register (ETPU_CDTROSR)* for more information on ETPU_CDTROSR and data transfer overflows.
10–15	Reserved
16 IPS	Channel input pin state Shows the current value of the filtered channel input signal state.
17 OPS	Channel output pin state Shows the current value driven in the channel output signal, including the effect of the external output disable feature. If the channel input and output signals are connected to the same pad, OPS reflects the value driven to the pad. This is not necessarily the actual pad value, which drives the value in the IPS bit.
18 OBE	Output Buffer Enable This bit shows the state of the channel output buffer enable signal, controlled by microcode.
19–29	Reserved
30–31 FM[0:1]	Channel function mode(1) Each function can use this field for specific configuration. These bits can be tested by microengine code.

1. These bits are equivalent to the TPU/TPU2/TPU3 host sequence (HSQ) bits.

22.10.5.4 eTPU Channel x Host Service Request Register (ETPU_CxHSRR)

CxHSRR is used by the Host to issue service requests to the channel.

Offset: Channel_Register_Base + 0x8 Access: User R/W

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

	16	17	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	HSR		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 390. eTPU Channel x Host Service Request Register (ETPU_CxHSRR)

Table 253. CxHSRR field descriptions

Field	Description
0–28	Reserved
29–31 HSR[2:0]	Host Service Request This field is used by the Host CPU to request service to the channel. HSR = 000: No Host Service Request pending HSR > 000: Function-dependent Host Service Request pending. HSR value turns to 000 automatically at the end of microengine service for that channel, but only if the thread started due to an HSR. Host should write HSR > 0 only when HSR = 0. Writing HSR = 000 withdraws a pending request if scheduler did not begin to resolve the Entry Point yet, but it does not abort the service thread from that point on.

22.11 Initialization and application information

After initial power-on reset, the eTPU remains in an idle state (except when debug is asserted on power-on reset—in this case, the microengine awakens in halt state). In addition, the SCM must be initialized with the eTPU application prior to configuring the eTPU.

23 Enhanced Queued Analog-to-Digital Converter (eQADC)

23.1 Introduction

23.1.1 Module overview

The Enhanced Queued Analog-to-Digital Converter (eQADC) block provides accurate and fast conversions for a wide range of applications. The eQADC provides a parallel interface to two on-chip analog-to-digital converters (ADCs), a single master to single slave serial interface to an off-chip external device, and a parallel side interface to one or more on-chip digital signal processing (DSP) modules (for example, a decimation filter). The two on-chip ADCs are architected to allow access to all the analog channels.

The eQADC transfers commands from multiple Command FIFOs (CFIFOs) to the on-chip ADCs or to the external device. The multiple Result FIFOs (RFIFOs) can receive data from the on-chip ADCs, from an off-chip external device or from an on-chip DSP module. Data from the on-chip ADCs can be routed to the side interface, processed by the on-chip DSP and then routed back through the side interface to the RFIFOs. The eQADC supports software and external hardware triggers from other blocks to initiate transfers of commands from the CFIFOs to the on-chip ADCs or to the external device. It also monitors the fullness of CFIFOs and RFIFOs, and accordingly generates DMA or interrupt requests to control data movement between the FIFOs and the system memory, which is external to the eQADC.

23.1.2 Block diagram

Figure 391 is the block diagram for the eQADC.

Figure 391. eQADC block diagram

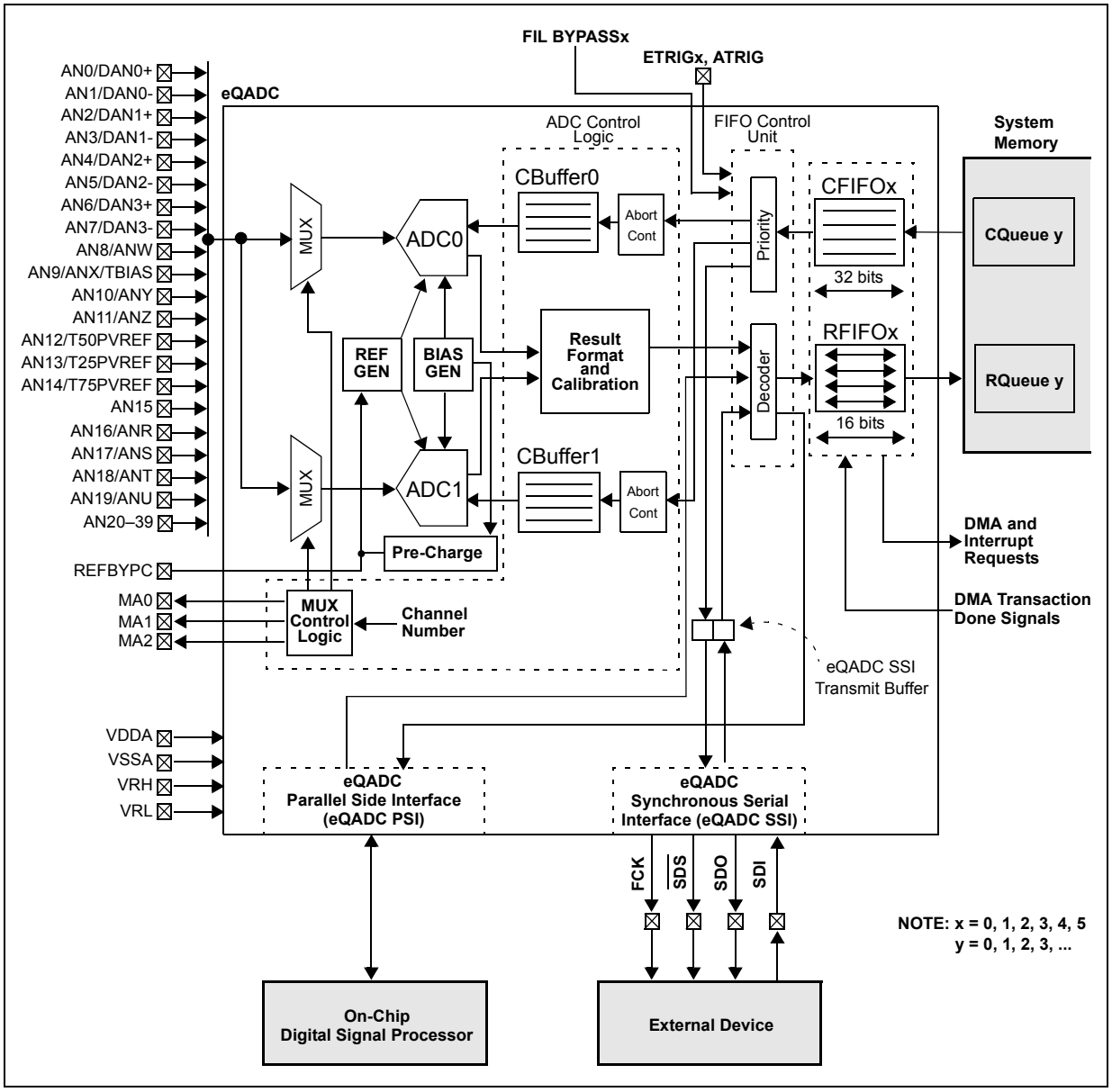


Figure 391 shows the primary components inside the eQADC. The eQADC consists of the *FIFO Control Unit* which controls the CFIFOs and the RFIFOs, the *ADC Control Logic* which controls the two on-chip ADCs, the *eQADC Synchronous Serial Interface* (eQADC SSI) which allows communication with an external device, and the *eQADC Parallel Side Interface* (eQADC PSI) which allows communication with on-chip eQADC companion[(m)] modules. There are six CFIFOs and six RFIFOs, each with four entries except CFIFO0, which can have eight entries for streaming mode.

m. Depending on the device, companion modules may be DSP modules, decimation filters, or any other processing block.

The *FIFO Control Unit* performs the following functions:

- Prioritizes the CFIFOs to determine which CFIFOs will have their commands transferred
- Supports software and hardware triggers to start command transfers from a particular CFIFO
- Decodes command data from the CFIFOs, and accordingly, sends these commands to one of the two on-chip ADCs or to the external device
- Decodes result data from on-chip ADCs or from the external device, and transfers data to the appropriate RFIFO or to the parallel side interface

The *ADC Control Logic* manages the execution of commands bound for on-chip ADCs. It interfaces with the CFIFOs via two 2-entry command buffers (CBuffers) with abort control and with the RFIFOs and side interface via the *Result Format and Calibration Sub-Block*. The *ADC Control Logic* performs the following functions:

- Buffers command data for execution
- Decodes command data and accordingly generates control signals for the two on-chip ADCs
- Detects abort request, stores aborted commands and buffers immediate conversion commands
- Formats and calibrates conversion result data coming from the on-chip ADCs
- Generates the internal multiplexer control signals and the select signals used by the external multiplexers

The eQADC SSI allows for a full duplex, synchronous, serial communication between the eQADC and an external device.

The eQADC PSI allows for a full duplex, synchronous, parallel communication between the eQADC and several on-chip companion modules.

Figure 391 also depicts data flow through the eQADC. Commands are contained in system memory in a user defined data structure. The most likely data structure to be used is a queue as depicted in the *Figure 391*[(n)]. Command data is moved from the command queue (CQueue) to the CFIFOs by either the host CPU or by the DMAC. Once a CFIFO is triggered and becomes the highest priority CFIFO using a certain CBuffer, command data is transferred from the CFIFO to the on-chip ADCs, or to the external device. The ADC executes the command, and the result is moved through the *Result Format and Calibration Sub-Block* to either the side interface or to the RFIFO. Data from the external device or on-chip companion module bypasses the *Result Format and Calibration Sub-Block* and is moved directly to its specified RFIFO. When data is stored in an RFIFO, data is moved from the RFIFO by the host CPU or by the DMAC to a data structure in system memory depicted in the *Figure 391* as a result queue (RQueue).

For users familiar with the QADC, the eQADC system upgrades the functionality provided by that block. Refer to *Section 23.6.7: eQADC versus QADC* for a comparison between the eQADC and QADC.

n. Command and result data can be stored in system memory in any user defined data structure. However, in this document it will be assumed that the data structure of choice is a queue, since it is the most likely data structure to be used and because queues are the only type of data structure supported by the DMAC.

23.1.3 Features

The eQADC block includes these distinctive features:

- 2 independent on-chip redundant signed digit (RSD) cyclic ADCs
  - 8, 10, and 12-bit AD resolution
  - Targets up to 10-bit accuracy at 500K Sample/s (ADC_CLK = 7.5 MHz) and 8-bit accuracy at 1M Sample/s (ADC_CLK = 15 MHz) for differential conversions
  - Selectable common mode conversion range (0–5V; 0–2.5V; 0–1.25V)
  - Differential conversions
  - Differential channels include variable gain amplifier for improved dynamic range (×1, ×2, ×4)
  - Differential channels include programmable pull-up and pull-down resistors for biasing and sensor diagnostics (200k ohms; 100k ohms; 5k ohms)
  - Sample times of 2 (default), 8, 64 or 128 ADC clock cycles
  - Provides time stamp information when requested
  - Parallel interface to eQADC CFIFOs and RFIFOs
  - Supports both right-justified unsigned and signed formats for conversion results
  - The REFBYPC pin stabilizes one of internal generated reference
  - Temperature sensor
  - Ability to measure directly V_{DD}
- Automatic application of ADC calibration constants
  - Provision of reference voltages (25% VREF and 75% VREF) for ADC calibration purposes
- 40 input channels available to the 2 on-chip ADCs
- 4 pairs of differential analog input channels
- Full duplex synchronous serial interface to an external device
  - Has a free-running clock for use by the external device
  - Supports a 26-bit message length
  - Transmits a null message when there are no triggered CFIFOs with commands bound for external CBuffers, or when there are triggered CFIFOs with commands bound for external CBuffers but the external CBuffers are full
- Parallel Side Interface to communicate with several on-chip companion modules
- STAC Client Interface to import an alternative timebase to the internal time stamp
- Priority Based CFIFOs
  - Supports six CFIFOs with fixed priority. The lower the CFIFO number, the higher its priority. When commands of distinct CFIFOs are bound for the same CBuffer, the higher priority CFIFO is always served first.
  - Immediate conversion command feature with conversion abort control
  - Streaming mode operation of CFIFO0 to execute some commands several times
  - Supports software and several hardware trigger modes to arm a particular CFIFO
  - Generates interrupt when command coherency is not achieved
- External Hardware Triggers
  - Supports rising edge, falling edge, high level and low level triggers
  - Supports configurable digital filter

  - Supports controls to bypass the trigger digital filters
- Two Triggers operation mode for queue0
  - Additional internal trigger (not filtered) called Advance trigger that is used to enable the external trigger of queue0 and to control the loop behavior of CFIFO0
- Supports 4 to 8 external 8-to-1 muxes which can expand the input channel number from 40 to 96
- Upgrades the functionality provided by the QADC

23.2 Modes of operation

This section describes the operation modes of the eQADC.

23.2.1 Normal Mode

This is the default operational mode when the eQADC is not in streaming mode or background debug or stop mode.

23.2.2 Debug Mode

Upon a debug mode entry request, eQADC behavior will vary according to the status of the DBG field in the eQADC Module Configuration Register (EQADC_MCR). If DBG is programmed to 0b00, the debug mode entry request is ignored. If DBG is programmed to 0b10 or to 0b11, the eQADC will enter debug mode. In case the eQADC SSI is enabled, the free running clock (FCK) output to external device will not stop when DBG is programmed to 0b11, but FCK will stop in low phase, when DBG is programmed to 0b10.

During debug mode, the eQADC will not transfer commands from any CFIFOs, no null messages will be transmitted to the external device, no data will be returned to any RFIFO, no hardware trigger event will be captured, and all eQADC registers can be accessed as in Normal mode. The latter implies that CFIFOs can still be triggered using software triggers, since no scheme is implemented to write-protect registers during debug mode. DMA and interrupt requests continue to be generated as in Normal Mode.

If at the time the debug mode entry request is detected, there are commands in the on-chip CBuffers that were already under execution, these commands will be completed but the generated results, if any, will not be sent to the RFIFOs until debug mode is exited. Commands whose execution has not started will not be executed until debug mode is exited. The clock associated with an on-chip ADC stops, during its low phase, after the ADC ceases executing commands. The time base counter will only stop after all on-chip ADCs cease executing commands.

When exiting debug mode, the eQADC relies on the CFIFO operation modes and on the CFIFO status to determine the next command entry to transfer.

The eQADC internal behavior after the debug mode entry request is detected differs depending on the status of command transfers.

- No command transfer is in progress.

  The eQADC immediately halts future command transfers from any CFIFO.

  If a null message is being transmitted, eQADC will complete the serial transmission before halting future command transfers. If valid data (conversion result or data read

from an ADC register) is received at the end of transmission, it will not be sent to an RFIFO until debug mode is exited.

If the null message transmission is aborted, the eQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after debug mode is exited.

- Command transfer is in progress.

  eQADC will complete the transfer and update CFIFO status before halting future command transfers from any CFIFO. Command transfers to the internal CBuffers are considered completed when a command is written to the buffers.

  Command transfers to the external device are considered completed when the serial transmission of the command is completed. If valid data (conversion result or data read from an ADC register) is received at the end of a serial transmission, it will not be sent to an RFIFO until debug mode is exited. The CFIFO status bits will still be updated after the completion of the serial transmission, therefore, after debug mode entry request is detected, the eQADC status bits will only stop changing several system clock cycles after the on-going serial transmission completes.

  If the command message transmission is aborted, the eQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after debug mode is exited.

- Command/Null message transfer through serial interface was aborted but next serial transmission did not start.

  If the debug mode entry request is detected between the time a previous serial transmission was aborted and the start of the next transmission, the eQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after debug mode is exited.

23.2.3 Stop Mode

Upon a stop mode entry request detection, the eQADC progressively halts its operations until it reaches a static, stable state from which it can recover when returning to Normal mode. eQADC then asserts an acknowledge signal, indicating that it is static and that the clock input can be stopped. In stop mode, the free running clock (FCK) output to external device will stop during its low phase if the eQADC SSI is enabled, and no hardware trigger events will be captured. The latter implies that, as long as the system clock is running, CFIFOs can still be triggered using software triggers, since no scheme is implemented to write-protect registers during stop mode.

If at the time the stop mode entry request is detected, there are commands in the on-chip CBuffers that were already under execution, these commands will be completed but the generated results, if any, will not be sent to the RFIFOs until stop mode is exited. Commands whose execution has not started will not be executed until stop mode is exited.

After these remaining commands are executed, the clock input to the ADCs is stopped. The ADC clock stops during its low phase. The time base counter will only stop after all on-chip ADCs cease executing commands. Only then, the stop acknowledge signal is asserted. When exiting stop mode, the eQADC relies on the CFIFO operation modes and on the CFIFO status to determine the next command entry to transfer.

The eQADC internal behavior after the stop mode entry request is detected differs depending on the status of the command transfer.

- No command transfer is in progress

  The eQADC immediately halts future command transfers from any CFIFO.

  If a null message is being transmitted, eQADC will complete the transmission before halting future command transfers. If valid data (conversion result or data read from an ADC register) is received at the end of the transmission, it will not be sent to an RFIFO until stop mode is exited.

  If the null message transmission is aborted, the eQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after stop mode is exited.

- Command transfer is in progress

  eQADC will complete the transfer and update CFIFO status before halting future command transfers from any CFIFO. Command transfers to the internal CBuffers are considered completed when a command is written to the buffers.

  Command transfers to the external device are considered completed when the serial transmission of the command is completed. If valid data (conversion result or data read from an ADC register) is received at the end of a serial transmission, it will not be sent to an RFIFO until stop mode is exited. The CFIFO status bits will still be updated after the completion of the serial transmission, therefore, after stop mode entry request is detected, the eQADC status bits will only stop changing several system clock cycles after the on-going serial transmission completes.

  If the command message transmission is aborted, the eQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after stop mode is exited.

- Command/Null message transfer through serial interface was aborted but next serial transmission did not start.

  If the stop mode entry request is detected between the time a previous serial transmission was aborted and the start of the next transmission, the eQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transferred after stop mode is exited.

23.3 External signal description

23.3.1 Overview

Table 254 provides a list of external pins.

Note: At chip integration level, some of the digital and analog signals listed here might share pins or not be available external to the chip. Refer to the Signals chapter for details.

Table 254. External signals

Name	Port	Function	Reset state	Type
AN0/DAN0+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog

Table 254. External signals (continued)

Name	Port	Function	Reset state	Type
AN1/DAN0–	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN2/DAN1+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog
AN3/DAN1–	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN4/DAN2+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog
AN5/DAN2–	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN6/DAN3+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog
AN7/DAN3–	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN8/ANW	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN9/ANX	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN10/ANY	Input	Single-ended analog input/Single-ended analog input from external multiplexers	—	Analog
AN11/ANZ	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN12	Input/Output	Single-ended analog input	—	Analog
AN13	Input/Output	Single-ended analog input	—	Analog
AN14	Input/Output	Single-ended analog input	—	Analog
AN15	Input	Single-ended analog input	—	Analog
AN16/ANR (1)	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN17/ANS(1)	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN18/ANT(1)	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN19/ANU(1)	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN20	Input	Single-ended analog input	—	Analog
AN21	Input	Single-ended analog input	—	Analog
AN22	Input	Single-ended analog input	—	Analog
AN23	Input	Single-ended analog input	—	Analog

Table 254. External signals (continued)

Name	Port	Function	Reset state	Type
AN24	Input	Single-ended analog input	—	Analog
AN25	Input	Single-ended analog input	—	Analog
AN26	Input	Single-ended analog input	—	Analog
AN27	Input	Single-ended analog input	—	Analog
AN28	Input	Single-ended analog input	—	Analog
AN29	Input	Single-ended analog input	—	Analog
AN30	Input	Single-ended analog input	—	Analog
AN31	Input	Single-ended analog input	—	Analog
AN32	Input	Single-ended analog input	—	Analog
AN33	Input	Single-ended analog input	—	Analog
AN34	Input	Single-ended analog input	—	Analog
AN35	Input	Single-ended analog input	—	Analog
AN36	Input	Single-ended analog input	—	Analog
AN37	Input	Single-ended analog input	—	Analog
AN38	Input	Single-ended analog input	—	Analog
AN39	Input	Single-ended analog input	—	Analog
MA0	Output	External multiplexer control signal	0	Digital
MA1	Output	External multiplexer control signal	0	Digital
MA2	Output	External multiplexer control signal	0	Digital
FCK	Output	eQADC SSI free running clock	0	Digital
SDS	Output	eQADC SSI serial data select	1	Digital
SDI	Input	eQADC SSI serial data in	—	Digital
SDO	Output	eQADC SSI serial data out	0	Digital
VDDA	Input	Analog Positive Power Supply	—	Power
VSSA	Input	Analog Negative Power Supply	—	Power
VRH	Input	Voltage Reference High	—	Power
VRL	Input	Voltage Reference Low	—	Power
REFBYPC	Input	External Bypass capacitor Pin	—	Power
ETRIG0	Input	External trigger for CFIFO0	—	Digital
ETRIG1	Input	External trigger for CFIFO1	—	Digital
ETRIG2	Input	External trigger for CFIFO2	—	Digital
ETRIG3	Input	External trigger for CFIFO3	—	Digital
ETRIG4	Input	External trigger for CFIFO4	—	Digital
ETRIG5	Input	External trigger for CFIFO5	—	Digital

1. Can be disabled or not using configuration parameters.

23.3.2 Detailed signal descriptions

23.3.2.1 AN0/DAN0+ — Single-ended analog input/Differential analog input positive terminal

AN0 is a single-ended analog input to the two on-chip ADCs. DAN0+ is the positive terminal of the differential analog input DAN0 (DAN0+ - DAN0–).

23.3.2.2 AN1/DAN0– — Single-ended analog input/Differential analog input negative terminal

AN1 is a single-ended analog input to the two on-chip ADCs. DAN0– is the negative terminal of the differential analog input DAN0 (DAN0+ - DAN0–).

23.3.2.3 AN2/DAN1+ — Single-ended analog input/Differential analog input positive terminal

AN2 is a single-ended analog input to the two on-chip ADCs. DAN1+ is the positive terminal of the differential analog input DAN1 (DAN1+ - DAN1–).

23.3.2.4 AN3/DAN1– — Single-ended analog input/Differential analog input negative terminal

AN3 is a single-ended analog input to the two on-chip ADCs. DAN1– is the negative terminal of the differential analog input DAN1 (DAN1+ - DAN1–).

23.3.2.5 AN4/DAN2+ — Single-ended analog input/Differential analog input positive terminal

AN4 is a single-ended analog input to the two on-chip ADCs. DAN2+ is the positive terminal of the differential analog input DAN2 (DAN2+ - DAN2–).

23.3.2.6 AN5/DAN2– — Single-ended analog input/Differential analog input negative terminal

AN5 is a single-ended analog input to the two on-chip ADCs. DAN2– is the negative terminal of the differential analog input DAN2 (DAN2+ - DAN2–).

23.3.2.7 AN6/DAN3+ — Single-ended analog input/Differential analog input positive terminal

AN6 is a single-ended analog input to the two on-chip ADCs. DAN3+ is the positive terminal of the differential analog input DAN3 (DAN3+ - DAN3–).

23.3.2.8 AN7/DAN3– — Single-ended analog input/Differential analog input negative terminal

AN7 is a single-ended analog input to the two on-chip ADCs. DAN3– is the negative terminal of the differential analog input DAN3 (DAN3+ - DAN3–).

23.3.2.9 AN8/ANW — Single-ended analog input/Single-ended analog input from external multiplexers

AN8 is a single-ended analog input to the two on-chip ADCs. ANW is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.

23.3.2.10 AN9/ANX — Single-ended analog input/Single-ended analog input from external multiplexers

AN9 is a single-ended analog input to the two on-chip ADCs. ANX is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.

23.3.2.11 AN10/ANY — Single-ended analog input/Single-ended analog input from external multiplexers

AN10 is a single-ended analog input to the two on-chip ADCs. ANY is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.

23.3.2.12 AN11/ANZ — Single-ended analog input/Single-ended analog input from external multiplexers

AN11 is a single-ended analog input to the two on-chip ADCs. ANZ is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.

23.3.2.13 AN12 — Single-ended analog input

AN12 is a single-ended analog input to the two on-chip ADCs.

23.3.2.14 AN13F — Single-ended analog input

AN13 is a single-ended analog input to the two on-chip ADCs.

23.3.2.15 AN14 — Single-ended analog input

AN14 is a single-ended analog input to the two on-chip ADCs.

23.3.2.16 AN15 — Single-ended analog input

AN15 is a single-ended analog inputs to the two on-chip ADCs.

23.3.2.17 AN16/ANR — Single-ended analog input/Single-ended analog input from external multiplexers

AN16 is a single-ended analog input to the two on-chip ADCs. ANR is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode. The external multiplexing capability is a device option.

23.3.2.18 AN17/ANS — Single-ended analog input/Single-ended analog input from external multiplexers

AN17 is a single-ended analog input to the two on-chip ADCs. ANS is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode. The external multiplexing capability is a device option.

23.3.2.19 AN18/ANT — Single-ended analog input/Single-ended analog input from external multiplexers

AN18 is a single-ended analog input to the two on-chip ADCs. ANT is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode. The external multiplexing capability is a device option.

23.3.2.20 AN19/ANU — Single-ended analog input/Single-ended analog input from external multiplexers

AN19 is a single-ended analog input to the two on-chip ADCs. ANU is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode. The external multiplexing capability is a device option.

23.3.2.21 AN20–AN39 — Single-ended analog input

AN20 through AN39 are single-ended analog inputs to the two on-chip ADCs.

23.3.2.22 INA_ADC0_0–INA_ADC0_9 — Single-ended analog input

INA_ADC0_0 through INA_ADC0_9 are single-ended analog inputs to the on-chip ADC0.

23.3.2.23 INA_ADC1_0–INA_ADC1_9 — Single-ended analog input

INA_ADC1_0 through INA_ADC1_9 are single-ended analog inputs to the on-chip ADC1.

23.3.2.24 MA0–MA2 — External multiplexer control signals

MA0, MA1, and MA2 combined form a select signal associated with external multiplexers.

23.3.2.25 FCK — eQADC SSI free-running clock

FCK is a free-running clock signal for synchronizing transmissions between the eQADC (master) and the external (slave) device.

23.3.2.26 $\overline{\text{SDS}}$ — eQADC SSI serial data select

$\overline{\text{SDS}}$ is the serial data select output. It is used to indicate to the external (slave) device when it can latch incoming serial data, when it can output its own serial data, and when it must abort a data transmission. $\overline{\text{SDS}}$ corresponds to the chip select signal in a conventional SPI interface.

23.3.2.27 SDI — eQADC SSI serial data in

SDI is the serial data input signal from the external (slave) device.

23.3.2.28 SDO — eQADC SSI serial data out

SDO is the serial data output signal to the external (slave) device.

23.3.2.29 VRH, VRL — Voltage reference high and voltage reference low

VRH and VRL are voltage references for the ADCs. VRH is the highest voltage reference, while VRL is the lowest voltage reference.

23.3.2.30 VDDA, VSSA — 5V VDD and VSS for the 5V analog components

VDDA is the positive power supply pin for the ADCs and VSSA is the negative power supply pin for the ADCs. Refer to electrical specifications.

23.3.2.31 REFBYPC — Reference bypass capacitor

The REFBYPC pin is used to connect an external bypass capacitor between REFBYPC and VRL. The value of this capacitor should be 100 nf. This bypass capacitor is used to provide a stable reference voltage for the ADC.

23.3.2.32 ETRIG0–ETRIG5 — External triggers

The external trigger signals are for hardware triggering. The eQADC can detect rising edge, falling edge, high level and low level on each of the external trigger signals. ETRIGx triggers CFIFOx. The eQADC also supports configurable digital filters for these external trigger signals. These digital filters can be bypassed by individual input control signals called eqadc_intern_trig_selx.

23.4 Memory map/register definition

This section provides memory maps and detailed descriptions of all registers. Data written to or read from reserved areas of the memory map is undefined.

23.4.1 eQADC memory map

This section provides memory maps for the eQADC block.

Table 255. eQADC memory map

Address	Use	Access	Location
EQADC_BASE+0x000	eQADC Module Configuration Register (EQADC_MCR)	Unrestricted	*on page 682*
EQADC_BASE+0x004	eQADC Test Register (EQADC_TST)	Test	
EQADC_BASE+0x008	eQADC Null Message Send Format Register (EQADC_NMSFR)	Unrestricted	*on page 683*
EQADC_BASE+0x00C	eQADC External Trigger Digital Filter Register (EQADC_ETDFR)	Unrestricted	*on page 684*
EQADC_BASE+0x010	eQADC CFIFO Push Register 0 (EQADC_CFPR0)	Write only	*on page 685*
EQADC_BASE+0x014	eQADC CFIFO Push Register 1 (EQADC_CFPR1)	Write only	*on page 685*
EQADC_BASE+0x018	eQADC CFIFO Push Register 2 (EQADC_CFPR2)	Write only	*on page 685*
EQADC_BASE+0x01C	eQADC CFIFO Push Register 3 (EQADC_CFPR3)	Write only	*on page 685*
EQADC_BASE+0x020	eQADC CFIFO Push Register 4 (EQADC_CFPR4)	Write only	*on page 685*

Table 255. eQADC memory map (continued)

Address	Use	Access	Location
EQADC_BASE+0x024	eQADC CFIFO Push Register 5 (EQADC_CFPR5)	Write only	*on page 685*
EQADC_BASE+0x028	Reserved	—	—
EQADC_BASE+0x02C	Reserved	—	—
EQADC_BASE+0x030	eQADC Result FIFO Pop Register 0 (EQADC_RFPR0)	Read only	*on page 686*
EQADC_BASE+0x034	eQADC Result FIFO Pop Register 1 (EQADC_RFPR1)	Read only	*on page 686*
EQADC_BASE+0x038	eQADC Result FIFO Pop Register 2 (EQADC_RFPR2)	Read only	*on page 686*
EQADC_BASE+0x03C	eQADC Result FIFO Pop Register 3 (EQADC_RFPR3)	Read only	*on page 686*
EQADC_BASE+0x040	eQADC Result FIFO Pop Register 4 (EQADC_RFPR4)	Read only	*on page 686*
EQADC_BASE+0x044	eQADC Result FIFO Pop Register 5 (EQADC_RFPR5)	Read only	*on page 686*
EQADC_BASE+0x048	Reserved	—	—
EQADC_BASE+0x04C	Reserved	—	—
EQADC_BASE+0x050	eQADC CFIFO Control Register 0 (EQADC_CFCR0)	Unrestricted	*on page 687*
EQADC_BASE+0x054	eQADC CFIFO Control Register 1 (EQADC_CFCR1)	Unrestricted	*on page 687*
EQADC_BASE+0x058	eQADC CFIFO Control Register 2 (EQADC_CFCR2)	Unrestricted	*on page 687*
EQADC_BASE+0x05C	Reserved	—	—
EQADC_BASE+0x060	eQADC Interrupt and DMA Control Register 0 (EQADC_IDCR0)	Unrestricted	*on page 690*
EQADC_BASE+0x064	eQADC Interrupt and DMA Control Register 1 (EQADC_IDCR1)	Unrestricted	*on page 690*
EQADC_BASE+0x068	eQADC Interrupt and DMA Control Register 2 (EQADC_IDCR2)	Unrestricted	*on page 690*
EQADC_BASE+0x06C	Reserved	—	—
EQADC_BASE+0x070	eQADC FIFO and Interrupt Status Register 0 (EQADC_FISR0)	Unrestricted	*on page 693*
EQADC_BASE+0x074	eQADC FIFO and Interrupt Status Register 1 (EQADC_FISR1)	Unrestricted	*on page 693*
EQADC_BASE+0x078	eQADC FIFO and Interrupt Status Register 2 (EQADC_FISR2)	Unrestricted	*on page 693*
EQADC_BASE+0x07C	eQADC FIFO and Interrupt Status Register 3 (EQADC_FISR3)	Unrestricted	*on page 693*

Table 255. eQADC memory map (continued)

Address	Use	Access	Location
EQADC_BASE+0x080	eQADC FIFO and Interrupt Status Register 4 (EQADC_FISR4)	Unrestricted	on page 693
EQADC_BASE+0x084	eQADC FIFO and Interrupt Status Register 5 (EQADC_FISR5)	Unrestricted	on page 693
EQADC_BASE+0x088	Reserved	—	—
EQADC_BASE+0x08C	Reserved	—	—
EQADC_BASE+0x090	eQADC CFIFO Transfer Counter Register 0 (EQADC_CFTCR0)	Unrestricted	on page 698
EQADC_BASE+0x094	eQADC CFIFO Transfer Counter Register 1 (EQADC_CFTCR1)	Unrestricted	on page 698
EQADC_BASE+0x098	eQADC CFIFO Transfer Counter Register 2 (EQADC_CFTCR2)	Unrestricted	on page 698
EQADC_BASE+0x09C	Reserved	—	—
EQADC_BASE+0x0A0	eQADC CFIFO Status Snapshot Register 0 (EQADC_CFSSR0)	Read only	on page 699
EQADC_BASE+0x0A4	eQADC CFIFO Status Snapshot Register 1 (EQADC_CFSSR1)	Read only	on page 699
EQADC_BASE+0x0A8	eQADC CFIFO Status Snapshot Register 2 (EQADC_CFSSR1)	Read only	on page 699
EQADC_BASE+0x0AC	eQADC CFIFO Status Register (EQADC_CFSR)	Read only	on page 702
EQADC_BASE+0x0B0	Reserved	—	—
EQADC_BASE+0x0B4	eQADC Synchronous Serial Interface Control Register (EQADC_SSICR)	Unrestricted	on page 703
EQADC_BASE+0x0B8	eQADC Synchronous Serial Interface Receive Data Register (EQADC_SSIRDR)	Read Only	on page 704
EQADC_BASE+0x0BC – EQADC_BASE+0x0CC	Reserved	—	—
EQADC_BASE+0x0D0	eQADC STAC Client Configuration Register (EQADC_REDLCCR)	Unrestricted	on page 705
EQADC_BASE+0x0D4 – EQADC_BASE+0x0FC	Reserved	—	—
EQADC_BASE+0x100 – EQADC_BASE+0x10C	eQADC CFIFO0 Registers (EQADC_CF0Rw) (w = 0, .., 3)	Read only	on page 706
EQADC_BASE+0x110 – EQADC_BASE+0x11C	eQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w = 0, .., 3)	Read only	on page 708
EQADC_BASE+0x120 – EQADC_BASE+0x13C	Reserved	—	—
EQADC_BASE+0x140 – EQADC_BASE+0x14C	eQADC CFIFO1 Registers (EQADC_CF1Rw) (w = 0, .., 3)	Read only	on page 706

Table 255. eQADC memory map (continued)

Address	Use	Access	Location
EQADC_BASE+0x150 – EQADC_BASE+0x17C	Reserved	—	—
EQADC_BASE+0x180 – EQADC_BASE+0x18C	eQADC CFIFO2 Registers (EQADC_CF2Rw) (w = 0, .., 3)	Read only	*on page 706*
EQADC_BASE+0x190 – EQADC_BASE+0x1BC	Reserved	—	—
EQADC_BASE+0x1C0 – EQADC_BASE+0x1CC	eQADC CFIFO3 Registers (EQADC_CF3Rw) (w = 0, .., 3)	Read only	*on page 706*
EQADC_BASE+0x1D0 – EQADC_BASE+0x1FC	Reserved	—	—
EQADC_BASE+0x200 – EQADC_BASE+0x20C	eQADC CFIFO4 Registers (EQADC_CF4Rw) (w = 0, .., 3)	Read only	*on page 706*
EQADC_BASE+0x210 – EQADC_BASE+0x23C	Reserved	—	—
EQADC_BASE+0x240 – EQADC_BASE+0x24C	eQADC CFIFO5 Registers (EQADC_CF5Rw) (w = 0, .., 3)	Read only	*on page 706*
EQADC_BASE+0x250 – EQADC_BASE+0x2FC	Reserved	—	—
EQADC_BASE+0x300 – EQADC_BASE+0x30C	eQADC RFIFO0 Registers (EQADC_RF0Rw) (w = 0, .., 3)	Read only	*on page 709*
EQADC_BASE+0x310 – EQADC_BASE+0x33C	Reserved	—	—
EQADC_BASE+0x340 – EQADC_BASE+0x34C	eQADC RFIFO1 Registers (EQADC_RF1Rw) (w = 0, .., 3)	Read only	*on page 709*
EQADC_BASE+0x350 – EQADC_BASE+0x37C	Reserved	—	—
EQADC_BASE+0x380 – EQADC_BASE+0x38C	eQADC RFIFO2 Registers (EQADC_RF2Rw) (w = 0, .., 3)	Read only	*on page 709*
EQADC_BASE+0x390 – EQADC_BASE+0x3BC	Reserved	—	—
EQADC_BASE+0x3C0 – EQADC_BASE+0x3CC	eQADC RFIFO3 Registers (EQADC_RF3Rw) (w = 0, .., 3)	Read only	*on page 709*
EQADC_BASE+0x3D0 – EQADC_BASE+0x3FC	Reserved	—	—
EQADC_BASE+0x400 – EQADC_BASE+0x40C	eQADC RFIFO4 Registers (EQADC_RF4Rw) (w = 0, .., 3)	Read only	*on page 709*
EQADC_BASE+0x410 – EQADC_BASE+0x43C	Reserved	—	—

Table 255. eQADC memory map (continued)

Address	Use	Access	Location
EQADC_BASE+0x440 – EQADC_BASE+0x44C	eQADC RFIFO5 Registers (EQADC_RF5Rw) (w = 0, .., 3)	Read only	*on page 709*
EQADC_BASE+0x450 – EQADC_BASE+0x7FC	Reserved	—	—

23.4.2 eQADC register descriptions

23.4.2.1 eQADC Module Configuration Register (EQADC_MCR)

The eQADC Module Configuration Register (EQADC_MCR) contains bits used to control how the eQADC responds to a debug mode entry request, and to enable the eQADC SSI interface.

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	ICEA0	ICEA1	0	ESSIE		0	DBG	
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 392. eQADC Module Configuration Register (EQADC_MCR)

Table 256. EQADC_MCR field descriptions

Field	Description
0–23	Reserved
24–25 ICEA*n*	Immediate Conversion Command Enable ADC*n* (*n* = 0, 1) ICEA*n* enables the eQADC to abort on-chip ADC*n* current conversion and to start the immediate conversion command from CFIFO0 in the requested ADC*n*. 1: Enable immediate conversion command request. 0: Disable immediate conversion command request.
26	Reserved

Table 256. EQADC_MCR field descriptions (continued)

Field	Description
27–28 ESSIE[0:1]	eQADC Synchronous Serial Interface Enable Field The ESSIE field defines the eQADC synchronous serial interface operation according to the following values: 0b00: eQADC SSI is disabled 0b01: Reserved 0b10: eQADC SSI is enabled, FCK is free running, and serial transmissions are disabled. 0b11: eQADC SSI is enabled, FCK is free running, and serial transmissions are enabled. **Note:** Disabling the eQADC SSI (0b00 write to ESSIE) or serial transmissions from the eQADC SSI (0b10 write to ESSIE) while a serial transmission is in progress results in the abort of that transmission. **Note:** When disabling the eQADC SSI, the FCK will not stop until it reaches its low phase.
29	Reserved
30–31 DBG[0:1]	Debug enable The DBG field defines the eQADC response to a debug mode entry request according to the following values: 0b00: Do not enter debug mode. 0b01: Reserved 0b10: Enter debug mode. If the eQADC SSI is enabled, FCK stops while the eQADC is in debug mode. 0b11: Enter debug mode. If the eQADC SSI is enabled, FCK is free running while the eQADC is in debug mode.

23.4.2.2 eQADC Null Message Send Format Register (EQADC_NMSFR)

The eQADC Null Message Send Format Register (EQADC_NMSFR) defines the format of the null message sent to the external device.

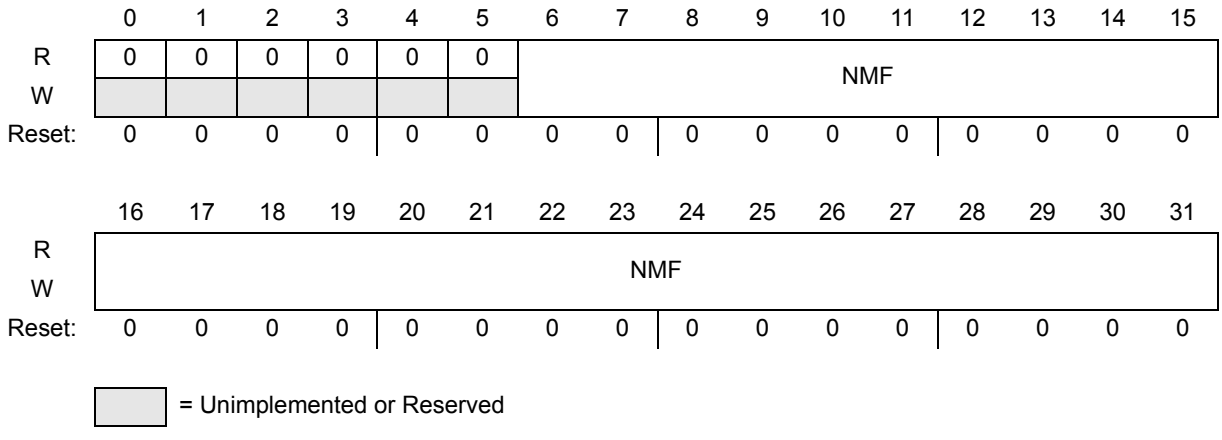


Figure 393. eQADC Null Message Send Format Register (EQADC_NMSFR)

Table 257. EQADC_NMSFR field descriptions

Field	Description
0–5	Reserved
6–31 NMF[0:25]	Null Message Format The NMF field contains the programmable null message send value for the eQADC. The value written to this register will be sent as a null message when serial transmissions from the eQADC SSI are enabled (ESSIE field configured to 0b11 in the eQADC Module Configuration Register (EQADC_MCR)) and either: – There are no triggered CFIFOs with commands bound for external CBuffers, or; – There are triggered CFIFOs with commands bound for external CBuffers but the external CBuffers are full. Refer to *Section 23.5.2.3.2.3: Null message format for external device operation* for more information on the format of a null message.

Note: The eQADC Null Message Send Format Register only affects how the eQADC sends a null message, but it has no control on how the eQADC detects a null message on receiving data. The eQADC detects a null message by decoding the MESSAGE_TAG field on the receive data. Refer to Table 288 for more information on the MESSAGE_TAG field.

Note: Writing to the eQADC Null Message Send Format Register while the serial transmissions are enabled (ESSIE field configured to 0b11 in the eQADC Module Configuration Register (EQADC_MCR)) is not recommended.

23.4.2.3 eQADC External Trigger Digital Filter Register (EQADC_ETDFR)

The eQADC External Trigger Digital Filter Register (EQADC_ETDFR) is used to set the minimum time a signal must be held in a logic state on the CFIFO triggers inputs to be recognized as an edge or level gated trigger. The Digital Filter Length field specifies the minimum number of system clocks that must be counted by the digital filter counter to recognize a logic state change. However, there is a control signal that can be used to bypass the digital filter when this is not needed.

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

	16	17	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	DFL			
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 394. eQADC External Trigger Digital Filter Register (EQADC_ETDFR)

Table 258. EQADC_ETDFR field descriptions

Field	Description
0–27	Reserved

Table 258. EQADC_ETDFR field descriptions (continued)

Field	Description
28–31 DFL[0:3]	Digital Filter Length The DFL field specifies the minimum number of system clocks that must be counted by the digital filter counter to recognize a logic state change. The count specifies the sample period of the digital filter which is calculated according to the following equation: $FilterPeriod = (SystemClockPeriod \times 2^{DFL}) + 1(SystemClockPeriod)$ Minimum clock counts for which an ETRIG signal needs to be stable to be passed through the filter are shown in *Table 259*. Refer to *Section 23.5.4.5: External trigger event detection* for more information on the digital filter. **Note:** The DFL field must only be written when the MODEx of all CFIFOs are configured to disabled. When the digital filter is bypassed by using the input control, the DFL is not considered and the trigger input signal is not filtered.

Table 259. Minimum required time to valid ETRIG

DFL[0:3]	Minimum clock count	Minimum time (ns) (system clock = 80 MHz)
0b0000	2	25.0
0b0001	3	37.5
0b0010	5	62.5
0b0011	9	112.5
0b0100	17	212.5
0b0101	33	412.5
0b0110	65	812.5
0b0111	129	1612.5
0b1000	257	3212.5
0b1001	513	6412.5
0b1010	1025	12812.5
0b1011	2049	25612.5
0b1100	4097	51212.5
0b1101	8193	102412.5
0b1110	16385	204812.5
0b1111	32769	409612.5

23.4.2.4 eQADC CFIFO Push Registers (EQADC_CFPR)

The eQADC CFIFO Push Registers (EQADC_CFPR) provide a mechanism to fill the CFIFOs with command messages from the CQueues. Refer to *Section 23.5.4: eQADC command FIFOs* for more information on the CFIFOs and to *Section 23.5.2.3: Message format in eQADC* for a description on command message formats.

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

	16	17	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	CF_PUSHx															
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 395. eQADC CFIFO Push Register x (EQADC_CFPRx)

Table 260. EQADC_CFPRx field descriptions

Field	Description
0–31 CF_PUSHx[0:31]	CFIFO Push Data x When CFIFOx is not full, writing to the whole word or any bytes of EQADC_CFPRx will push the 32-bit CF_PUSHx value into CFIFOx. Writing to the CF_PUSHx field also increments the corresponding CFCTRx value by one in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR). When the CFIFOx is full, the eQADC ignores any write to the CF_PUSHx. Reading the EQADC_CFPRx always returns zero.

Note: Only whole words must be written to EQADC_CFPR. Writing half-words or bytes to EQADC_CFPR will still push the whole 32-bit CF_PUSH field into the corresponding CFIFO, but undefined data will fill the areas of CF_PUSH that were not specifically designated as target locations for the write.

23.4.2.5 eQADC Result FIFO Pop Registers (EQADC_RFPR)

The eQADC Result FIFO Pop Registers (EQADC_RFPR) provide a mechanism to retrieve data from RFIFOs.

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

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

= Unimplemented or Reserved

Figure 396. eQADC RFIFO Pop Register x (EQADC_RFPRx)

Table 261. EQADC_RFPRx field descriptions

Field	Description
0–15	Reserved
16–31 RF_POPx[0:15]	Result FIFO Pop Data x When RFIFOx is not empty, the RF_POPx contains the next unread entry value of RFIFOx. Reading a word, a half-word, or any bytes from EQADC_RFPRx will pop one entry from RFIFOx and cause the RFCTRx field to be decremented by one in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR). When the RFIFOx is empty, any read on EQADC_RFPRx returns undefined data value and does not decrement the RFCTRx value. Writing to EQADC_RFPRx has no effect.

23.4.2.6 eQADC CFIFO Control Registers (EQADC_CFCR)

The eQADC CFIFO Control Registers (EQADC_CFCR) contain bits that affect CFIFOs. These bits specify the CFIFO operation mode and can invalidate all of the CFIFO contents.

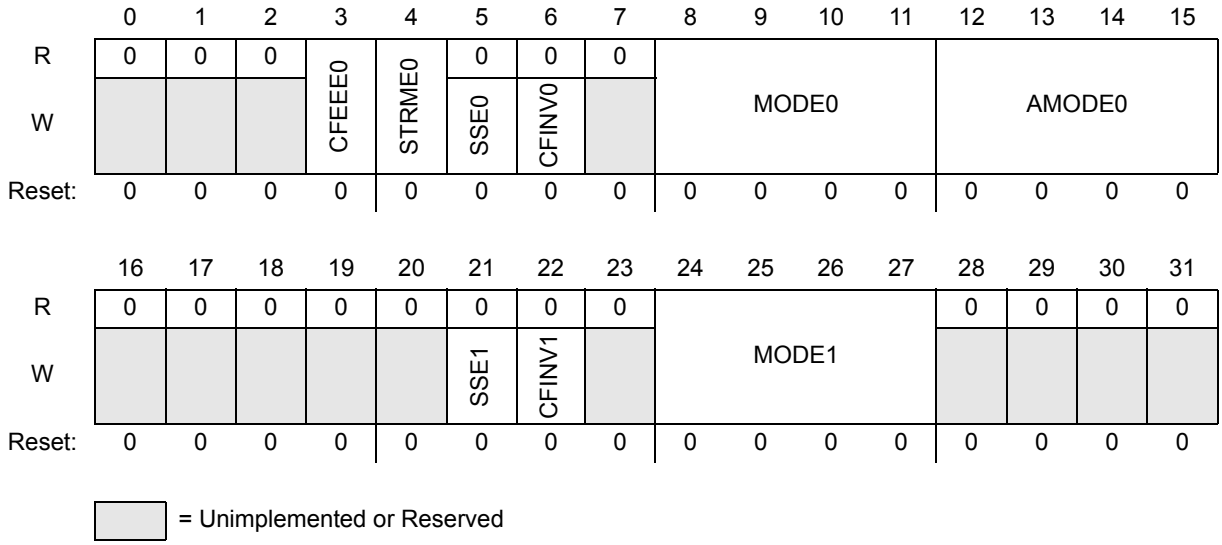


Figure 397. eQADC CFIFO Control Register 0 (EQADC_CFCR0)

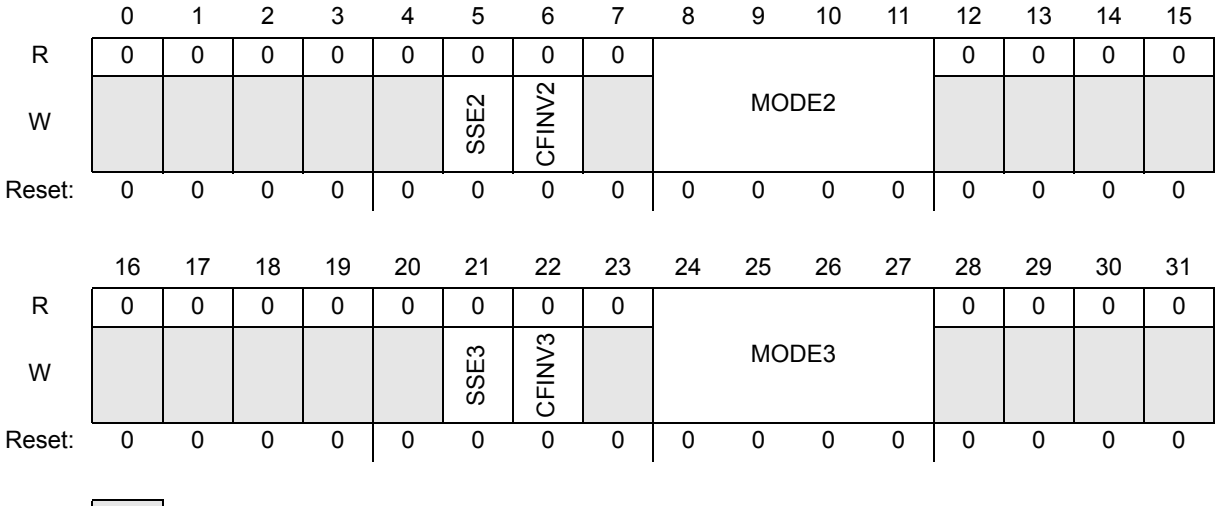


Figure 398. eQADC CFIFO Control Register 1 (EQADC_CFCR1)

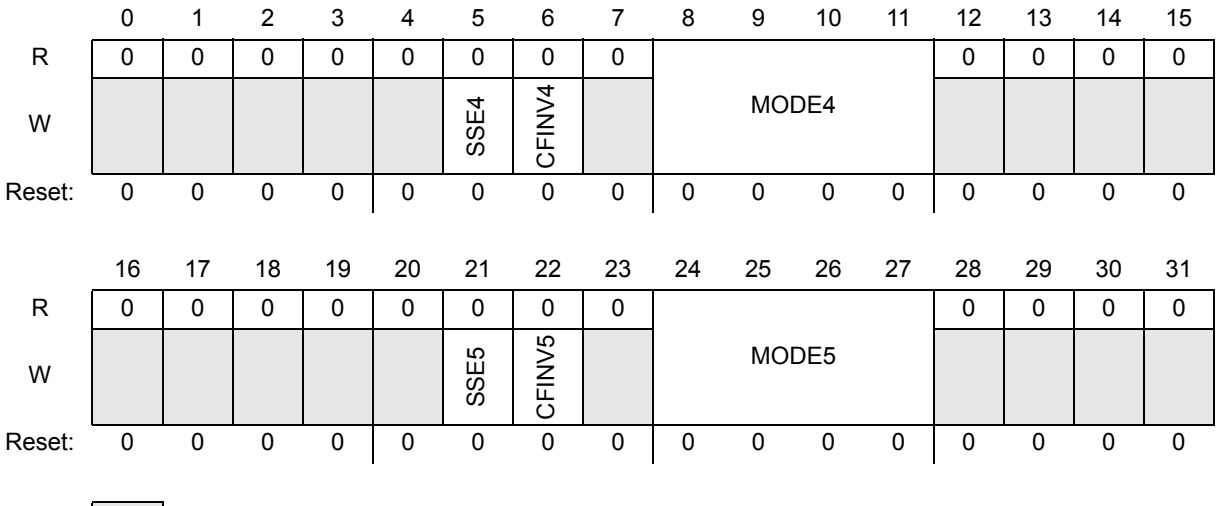


Figure 399. eQADC CFIFO Control Register 2 (EQADC_CFCR2)

Table 262. EQADC_CFCR*x* field descriptions

Field	Description
CFCEE0	CFIFO0 Entry Number Extension Enable The CFEEE0 bit is used to enable the extension of the CFIFO0 entries. When in extended mode, the CFIFO0 total entries is the sum of normal entries plus the defined value for extension. For more details, refer to *Section 23.5.4.2: CFIFO0 streaming mode description*. 1: Enable the extension of CFIFO0 entries. 0: CFIFO0 has a normal value of entries.

Table 262. EQADC_CFCR*x* field descriptions (continued)

Field	Description
STRME0	CFIFO0 Streaming Mode Operation Enable The STRME0 bit is used to enable the streaming mode of operation of CFIFO0. In this case, it is possible to repeat some sequence of commands of this FIFO. For more details, refer to *Section 23.5.4.2: CFIFO0 streaming mode description*. 1: Enable the streaming mode of CFIFO0. 0: Streaming mode of CFIFO0 is disabled.
SSEx	CFIFO Single-Scan Enable Bit x The SSEx bit is used to set the SSSx bit in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR). Writing a ‘1’ to SSEx will set the SSSx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) if the CFIFO is in single-scan mode. When SSSx is already asserted, writing a ‘1’ to SSEx also has no effect. If the CFIFO is in continuous-scan mode or is disabled, writing a ‘1’ to SSEx will not set SSSx. Writing a ‘0’ to SSEx has no effect. SSEx always is read as ‘0’. 1: Set the SSSx bit. 0: No effect.
CFINVx	CFIFO Invalidate Bit x The CFINVx bit causes the eQADC to invalidate all entries of CFIFOx. Writing a ‘1’ to CFINVx will reset the value of CFCTRx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR). Writing a ‘1’ to CFINVx also resets the Push Next Data Pointer, Transfer Next Data Pointer to the first entry of CFIFOx in *Figure 448*. CFINVx always is read as ‘0’. Writing a ‘0’ has no effect. 1: Invalidate all of the entries in the corresponding CFIFO. 0: No effect. **Note:** Writing CFINVx only invalidates commands stored in CFIFOx; previously transferred commands that are waiting for execution, that is commands stored in the CBuffers, will still be executed, and results generated by them will be stored in the appropriate RFIFO. **Note:** CFINVx must not be written unless the MODEx is configured to disabled, and CFIFO status is IDLE.

Table 262. EQADC_CFCR*x* field descriptions (continued)

Field	Description
MODEx[0:3]	CFIFO Operation Mode x The MODEx field selects the CFIFO operation mode for CFIFOx. Refer to *Section 23.5.4.6: CFIFO scan trigger modes* for more information on CFIFO trigger mode. 0b0000: Disabled 0b0001: Software Trigger, Single Scan 0b0010: Low Level Gated External Trigger, Single Scan 0b0011: High Level Gated External Trigger, Single Scan 0b0100: Falling Edge External Trigger, Single Scan 0b0101: Rising Edge External Trigger, Single Scan 0b0110: Falling or Rising Edge External Trigger, Single Scan 0b0111–0b1000: Reserved 0b1001: Software Trigger, Continuous Scan 0b1010: Low Level Gated External Trigger, Continuous Scan 0b1011: High Level Gated External Trigger, Continuous Scan 0b1100: Falling Edge External Trigger, Continuous Scan 0b1101: Rising Edge External Trigger, Continuous Scan 0b1110: Falling or Rising Edge External Trigger, Continuous Scan 0b1111: Reserved **Note:** If MODEx is not disabled, it must not be changed to any other mode besides disabled. If MODEx is disabled and the CFIFO status is IDLE, MODEx can be changed to any other mode.
AMODE0[0:3]	CFIFO0 Advance Trigger Operation Mode 0 The AMODE0 field selects the trigger mode for the ATRIG0 trigger signal in streaming mode. The use of reserved values drives to unknown behavior of the block. 0b0000: Disabled 0b0001: Reserved 0b0010: Low Level Gated External Trigger 0b0011: High Level Gated External Trigger 0b0100: Falling Edge External Trigger 0b0101: Rising Edge External Trigger 0b0110: Falling or Rising Edge External Trigger 0b0111–0b1111: Reserved **Note:** It is necessary to set to “Disabled” before changing to a different trigger type. If AMODE0 is not disabled, it must not be changed to any mode other than disabled. If AMODE0 is disabled and the CFIFO0 status is IDLE, AMODE0 can be changed to any other mode.

23.4.2.7 eQADC Interrupt and DMA Control Registers (EQADC_IDCR)

The eQADC Interrupt Control Registers (EQADC_IDCR) contain bits to enable the generation of interrupt or DMA requests when the corresponding flag bits are set in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCIE0	TORIE0	PIE0	EOQIE0	CFUIE0	0	CFFE0	CFFS0	0	0	0	0	RFOIE0	0	RFDE0	RFDS0
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NCIE1	TORIE1	PIE1	EOQIE1	CFUIE1	0	CFFE1	CFFS1	0	0	0	0	RFOIE1	0	RFDE1	RFDS1
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 400. eQADC Interrupt and DMA Control Register 0 (EQADC_IDCR0)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCIE2	TORIE2	PIE2	EOQIE2	CFUIE2	0	CFFE2	CFFS2	0	0	0	0	RFOIE2	0	RFDE2	RFDS2
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NCIE3	TORIE3	PIE3	EOQIE3	CFUIE3	0	CFFE3	CFFS3	0	0	0	0	RFOIE3	0	RFDE3	RFDS3
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 401. eQADC Interrupt and DMA Control Register 1 (EQADC_IDCR1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCIE4	TORIE4	PIE4	EOQIE4	CFUIE4	0	CFFE4	CFFS4	0	0	0	0	RFOIE4	0	RFDE4	RFDS4
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NCIE5	TORIE5	PIE5	EOQIE5	CFUIE5	0	CFFE5	CFFS5	0	0	0	0	RFOIE5	0	RFDE5	RFDS5
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 402. eQADC Interrupt and DMA Control Register 2 (EQADC_IDCR2)

Table 263. EQADC_IDCR*x* field descriptions

Field	Description
NCIEx	Non-Coherency Interrupt Enable x NCIEx enables the eQADC to generate an interrupt request when the corresponding NCFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted. 1: Enable non-coherency interrupt request. 0: Disable non-coherency interrupt request.
TORIEx	Trigger Overrun Interrupt Enable x TORIEx enables the eQADC to generate an interrupt request when the corresponding TORFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted. Apart from generating an independent interrupt request for a CFIFOx Trigger Overrun event, the eQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.5.8: eQADC DMA/Interrupt request* for details. 1: Enable trigger overrun interrupt request. 0: Disable trigger overrun interrupt request.
PIEx	Pause Interrupt Enable x PIEx enables the eQADC to generate an interrupt request when the corresponding PFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted. 1: Enable pause interrupt request. 0: Disable pause interrupt request.
EOQIEx	End of Queue Interrupt Enable x EOQIEx enables the eQADC to generate an interrupt request when the corresponding EOQFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted. 1: Enable End of Queue interrupt request. 0: Disable End of Queue interrupt request.
CFUIEx	CFIFO Underflow Interrupt Enable x CFUIEx enables the eQADC to generate an interrupt request when the corresponding CFUFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted. Apart from generating an independent interrupt request for a CFIFOx underflow event, the eQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.5.8: eQADC DMA/Interrupt request* for details. 1: Enable Underflow Interrupt request. 0: Disable Underflow Interrupt request.
CFFEx	CFIFO Fill Enable x CFFEx enables the eQADC to generate an interrupt request (CFFSx is asserted) or DMA request (CFFSx is negated) when CFFFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted. 1: Enable CFIFO Fill DMA or Interrupt request. 0: Disable CFIFO Fill DMA or Interrupt request. **Note:** CFFEx must not be negated while a DMA transaction is in progress.

Table 263. EQADC_IDCR*x* field descriptions (continued)

Field	Description
CFFSx	CFIFO Fill Select x CFFSx selects if a DMA or interrupt request is generated when CFFFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted. If CFFEx is asserted, the eQADC generates an interrupt request when CFFSx is negated, or it generates a DMA request if CFFSx is asserted. 1: Generate DMA request to move data from the system memory to CFIFOx. 0: Generate interrupt request to move data from the system memory to CFIFOx. **Note:** CFFSx must not be negated while a DMA transaction is in progress.
RFOIEx	RFIFO Overflow Interrupt Enable x RFOIEx enables the eQADC to generate an interrupt request when the corresponding RFOFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted. Apart from generating an independent interrupt request for an RFIFOx overflow event, the eQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.5.8: eQADC DMA/Interrupt request* for details. 1: Enable Overflow Interrupt request. 0: Disable Overflow Interrupt request.
RFDEx	RFIFO Drain Enable x RFDEx enables the eQADC to generate an interrupt request (RFDSx is asserted) or DMA request (RFDSx is negated) when RFDFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted. 1: Enable RFIFO Drain DMA or Interrupt request. 0: Disable RFIFO Drain DMA or Interrupt request. **Note:** RFDEx must not be negated while a DMA transaction is in progress.
RFDSx	RFIFO Drain Select x RFDSx selects if a DMA or interrupt request is generated when RFDFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted. If RFDEx is asserted, the eQADC generates an interrupt request when RFDSx is negated, or it generates a DMA request when RFDSx is asserted. 1: Generate DMA request to move data from RFIFOx to the system memory. 0: Generate interrupt request to move data from RFIFOx to the system memory. **Note:** RFDSx must not be negated while a DMA transaction is in progress.

23.4.2.8 eQADC FIFO and Interrupt Status Registers (EQADC_FISR)

The eQADC FIFO and Interrupt Status Registers (EQADC_FISR) contain flag and status bits for each CFIFO and RFIFO pair. Write ‘1’ to a flag bit to clear it. Writing ‘0’ has no effect. Status bits are read only. These bits indicate the status of the FIFO itself.

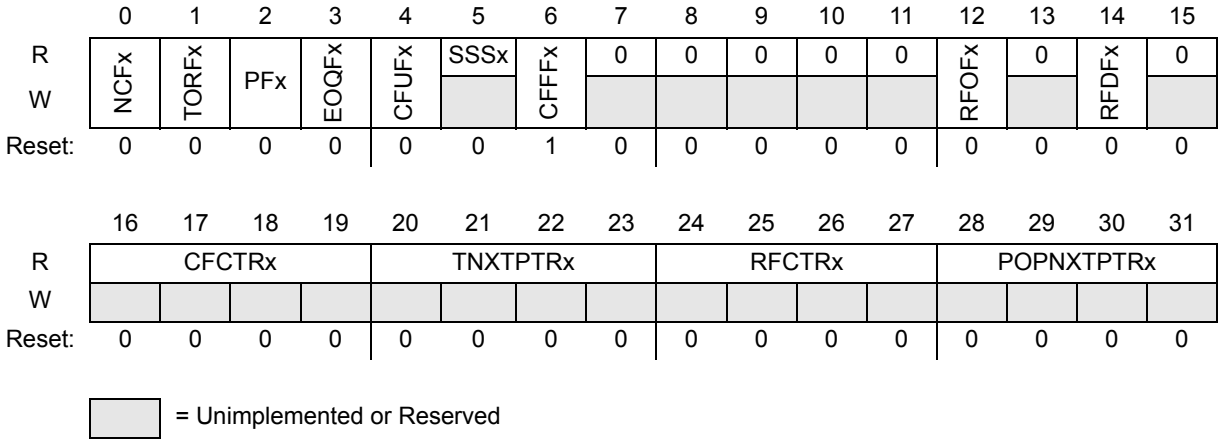


Figure 403. eQADC FIFO and Interrupt Status Register x (EQADC_FISRx)

Table 264. EQADC_FISRx field descriptions

Field	Description
NCFx	Non-Coherency Flag NCFx is set whenever a command sequence being transferred through CFIFOx becomes non coherent. If NCIEx in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR), and NCF*x* are asserted, an interrupt request will be generated. Write '1' to clear NCFx. Writing a '0' has no effect. For more information refer to *Section 23.5.4.7.5: Command sequence non-coherency detection*. 1: Command sequence being transferred by CFIFOx became non-coherent. 0: Command sequence being transferred by CFIFOx is coherent.
TORFx	Trigger Overrun Flag for CFIFOx TORFx is set when trigger overrun occurs for the specified CFIFO in edge or level trigger mode. Trigger overrun occurs when an already triggered CFIFO receives an additional trigger. When TORIE*x* in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR), and TORF*x* are asserted, an interrupt request will be generated. Apart from generating an independent interrupt request for a CFIFOx Trigger Overrun event, the eQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.5.8: eQADC DMA/Interrupt request* for details. Write '1' to clear the TORF*x* bit. Writing a '0' has no effect. 1: Trigger overrun occurred. 0: No trigger overrun occurred. **Note:** The trigger overrun flag will not set for CFIFOs configured for software trigger mode.

Table 264. EQADC_FISRx field descriptions (continued)

Field	Description
PFx	Pause Flag x PF behavior changes according to the CFIFO trigger mode. In edge trigger mode, PFx is set when the eQADC completes the transfer of an entry with an asserted Pause bit from CFIFOx. In level trigger mode, when CFIFOx is in TRIGGERED status, PFx is set when CFIFO status changes from TRIGGERED due to the detection of a closed gate. An interrupt routine, generated due to the asserted PF, can be used to verify if a complete scan of the CQueue was performed. If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status. If a closed gate is detected while a command transfer to an on-chip CBuffer is taking place, it will only affect the CFIFO status when the transfer completes. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed. In software trigger mode, PFx will never become asserted. If PIE*x* in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR), and PFx are asserted, an interrupt will be generated. Write ‘1’ to clear the PF*x*. Writing a ‘0’ has no effect. Refer to *Section 23.5.4.7.3: Pause status* for more information on the Pause Flag. 1: Entry with asserted PAUSE bit was transferred from CFIFOx (CFIFO in edge trigger mode), or CFIFO status changes from TRIGGERED due to detection of a closed gate (CFIFO in level trigger mode). 0: Entry with asserted PAUSE bit was not transferred from CFIFOx (CFIFO in edge trigger mode), or CFIFO status did not change from TRIGGERED due to detection of a closed gate (CFIFO in level trigger mode). **Note:** In edge trigger mode, an asserted PFx only implies that the eQADC has finished transferring a command with an asserted PAUSE bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO. **Note:** In software or level trigger mode, when the eQADC completes the transfer of an entry from CFIFOx with an asserted PAUSE bit, PFx will not be set and transfer of commands will continue without pausing.
EOQFx	End of Queue Flag x EOQFx indicates that an entry with an asserted EOQ bit was transferred from CFIFOx to the on-chip ADCs or to the external device—see *Section 23.5.2.3: Message format in eQADC* for details about command message formats. When the eQADC completes the transfer of an entry with an asserted EOQ bit from CFIFOx, EOQFx will be set. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed. If the EOQIEx bit in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR), and EOQFx are asserted, an interrupt will be generated. Write ‘1’ to clear the EOQFx bit. Writing a ‘0’ has no effect. Refer to *Section 23.5.4.7.2: CQueue completion status* for more information on the End of Queue Flag. 1: Entry with asserted EOQ bit was transferred from CFIFOx. 0: Entry with asserted EOQ bit was not transferred from CFIFOx. **Note:** An asserted EOQFx only implies that the eQADC has finished transferring a command with an asserted EOQ bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

Table 264. EQADC_FISRx field descriptions (continued)

Field	Description
CFUFx	CFIFO Underflow Flag x CFUFx indicates an underflow event on CFIFOx. CFUFx is set when CFIFOx is in TRIGGERED state and it becomes empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. When CFUIEx in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR), and CFUFx are both asserted, the eQADC generates an interrupt request. Apart from generating an independent interrupt request for a CFIFOx underflow event, the eQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.5.8: eQADC DMA/Interrupt request* for details. Write ‘1’ to clear CFUFx. Writing a ‘0’ has no effect. 1: A CFIFO underflow event occurred. 0: No CFIFO underflow event occurred.
SSSx	CFIFO Single-Scan Status Bit x An asserted SSSx bit enables the detection of trigger events for CFIFOs programmed into single-scan level- or edge-trigger mode, and works as trigger for CFIFOs programmed into single-scan software-trigger mode. Refer to *Section 23.5.4.6.2: Single-scan Mode* for further details. The SSSx bit is set by writing a ‘1’ to the SSEx bit in the eQADC CFIFO Control Registers (EQADC_CFCR). The eQADC clears the SSSx bit when a command with an asserted EOQ bit is transferred from a CFIFO in single-scan mode, when a CFIFO is in single-scan level-trigger mode and its status changes from TRIGGERED due to the detection of a closed gate, or when the value of the CFIFO operation mode (MODEx) in the eQADC CFIFO Control Registers (EQADC_CFCR) is changed to disabled. Writing to SSSx has no effect. SSSx has no effect in continuous-scan or in disabled mode. 1: CFIFO in single-scan level- or edge-trigger mode will detect a trigger event, or CFIFO in single-scan software-trigger mode is triggered. 0: CFIFO in single-scan level- or edge-trigger mode will ignore trigger events, or CFIFO in single-scan software-trigger mode is not triggered.
CFFFx	CFIFO Fill Flag x CFFFx is set when the CFIFOx is not full. When CFFE*x* in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR), and CFFFx are both asserted, an interrupt or a DMA request will be generated depending on the status of the CFFSx bit. When CFFSx is negated (interrupt requests selected), software clears CFFFx by writing a ‘1’ to it. Writing a ‘0’ has no effect. When CFFSx is asserted (DMA requests selected), CFFFx is automatically cleared by the eQADC when the CFIFO becomes full. 1: CFIFOx is not full. 0: CFIFOx is full. **Note:** Writing ‘1’ to CFFFx when CFFSx is asserted (DMA requests selected) is not allowed. **Note:** When generation of interrupt requests is selected (CFFSx = 0), CFFFx must only be cleared in the ISR after the CFIFOx push register is accessed.

Table 264. EQADC_FISRx field descriptions (continued)

Field	Description
RFOFx	RFIFO Overflow Flag x RFOFx indicates an overflow event on RFIFOx. RFOF*x* is set when RFIFOx is already full, and a new data is received from the on-chip ADCs or from the external device. The RFIFOx will not overwrite older data in the RFIFO, and the new data will be ignored. When RFOIEx in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR), and RFOFx are both asserted, the eQADC generates an interrupt request. Apart from generating an independent interrupt request for an RFIFOx overflow event, the eQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See *Section 23.5.8: eQADC DMA/Interrupt request* for details. Write ‘1’ to clear RFOFx. Writing a ‘0’ has no effect. 1: An RFIFO overflow event occurred. 0: No RFIFO overflow event occurred.
RFDF*x*	RFIFO Drain Flag x RFDFx indicates if RFIFOx has valid entries or not. RFDFx is set when the RFIFOx has at least one valid entry in it. When RFDEx in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR), and RFDFx are both asserted, an interrupt or a DMA request will be generated depending on the status of the RFDSx bit. When RFDSx is negated (interrupt requests selected), software clears RFDFx by writing a ‘1’ to it. Writing a ‘0’ has no effect. When RFDSx is asserted (DMA requests selected), RFDFx is automatically cleared by the eQADC when the RFIFO becomes empty. 1: RFIFOx has at least one valid entry. 0: RFIFOx is empty. **Note:** Writing ‘1’ to RFDFx when RFDSx is asserted (DMA requests selected) is not allowed. **Note:** When the generation of interrupt requests is selected (RFDSx = 0), RFDFx must only be cleared in the ISR after the RFIFOx pop register is accessed.
CFCTRx[0:3]	CFIFOx Entry Counter CFCTRx indicates the number of commands stored in the CFIFOx. When the eQADC completes transferring a piece of new data from the CFIFOx, it decrements CFCTRx by one. Writing a word or any bytes to the corresponding eQADC CFIFO Push Registers (EQADC_CFPR) increments CFCTRx by one. Writing any value to CFCTRx has no effect.
TNXTPTRx[0:3]	CFIFOx Transfer Next Pointer TNXTPTRx indicates the index of the next entry to be removed from CFIFOx when it completes a transfer. When TNXTPTRx is zero, it points to the entry with the smallest memory-mapped address inside CFIFOx. TNXTPTRx is only updated when a command transfer is completed. If the maximum index number (CFIFO depth minus one) is reached, TNXTPTRx is wrapped to zero, else, it is incremented by one. For details refer to *Section 23.5.4.1: CFIFO basic functionality*. Writing any value to TNXTPTRx has no effect.
RFCTRx[0:3]	RFIFOx Entry Counter RFCTRx indicates the number of data items stored in the RFIFOx. When the eQADC stores a piece of new data into RFIFOx, it increments RFCTRx by one. Reading the whole word, half-word or any bytes of the corresponding eQADC Result FIFO Pop Registers (EQADC_RFPR) decrements RFCTRx by one. Writing any value to RFCTRx itself has no effect.

Table 264. EQADC_FISRx field descriptions (continued)

Field	Description
POPNXTPTRx[0:3]	RFIFOx Pop Next Pointer POPNXTPTRx indicates the index of the entry that will be returned when EQADC_RFPRx is read. When POPNXTPTRx is zero, it points to the entry with the smallest memory-mapped address inside RFIFOx. POPNXTPTRx is updated when EQADC_RFPRx is read. If the maximum index number (RFIFO depth minus one) is reached, POPNXTPTRx is wrapped to zero, else, it is incremented by one. For details refer to *Section 23.5.5.1: RFIFO basic functionality*. Writing any value to POPNXTPTRx has no effect.

23.4.2.9 eQADC CFIFO Transfer Counter Registers (EQADC_CFTCR)

The eQADC CFIFO Transfer Counter Registers (EQADC_CFTCR) record the number of commands transferred from a CFIFO. The EQADC_CFTCR supports the monitoring of command transfers from a CFIFO.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	TC_CF0										
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	TC_CF1										
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 404. eQADC CFIFO Transfer Counter Register 0 (EQADC_CFTCR0)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	TC_CF2										
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	TC_CF3										
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 405. eQADC CFIFO Transfer Counter Register 1 (EQADC_CFTCR1)

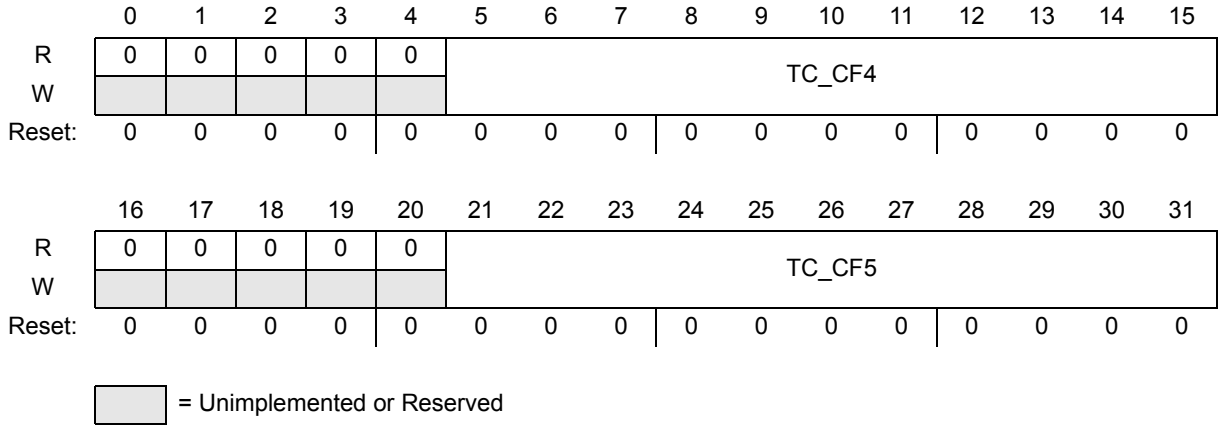


Figure 406. eQADC CFIFO Transfer Counter Register 2 (EQADC_CFTCR2)

Table 265. EQADC_CFTCRx field descriptions

Field	Description
TC_CFx[0:10]	Transfer Counter for CFIFOx TC_CFx counts the number of commands which have been completely transferred from CFIFOx. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The transfer of entries bound for an external device is considered completed when the serial transmission of the entry is completed. The eQADC increments the TC_CFx value by one after a command is transferred. TC_CFx resets to zero after the eQADC completes transferring a command with an asserted EOQ bit. Writing any value to TC_CFx sets the counter to that written value. **Note:** If CFIFOx is in TRIGGERED state when its MODEx field is programmed to disabled, the exact number of entries transferred from the CFIFO until that point—TC_CFx—is only known after the CFIFO status changes to IDLE, as indicated by CFSx. For details refer to *Section 23.5.4.6.1: Disabled Mode*.

23.4.2.10 eQADC CFIFO Status Snapshot Registers (EQADC_CFSSR)

The eQADC CFIFO Status Snapshot Registers (EQADC_CFSSR) contain status fields to track the operation status of each CFIFO and the transfer counter of the last CFIFO to initiate a command transfer to the internal/external CBuffers. EQADC_CFSSR0–1 are related to the on-chip CBuffers (CBuffer0–1) while EQADC_CFSSR2 is related to the external CBuffers. All fields of a particular EQADC_CFSSR are captured at the beginning of a command transfer to the CBuffer associated with that register. Note that captured status register values are associated with previous command transfer. This means that the EQADC_CFSSRs capture the status registers before the status registers change because of the transfer of the current command that is about to be popped from the CFIFO. The EQADC_CFSSRs are read only. Writing to the EQADC_CFSSRs has no effect.

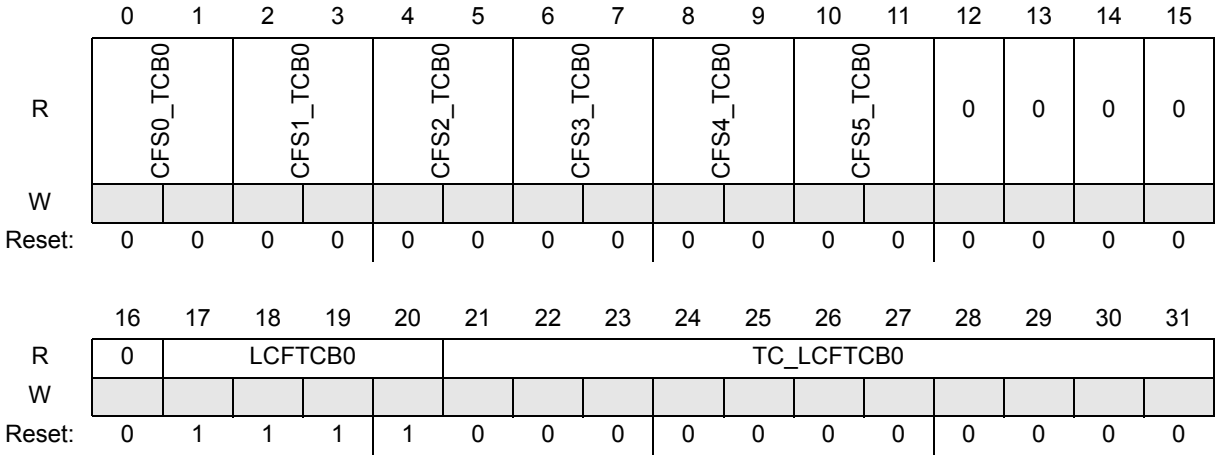


Figure 407. eQADC CFIFO Status Snapshot Register 0 (EQADC_CFSSR0)

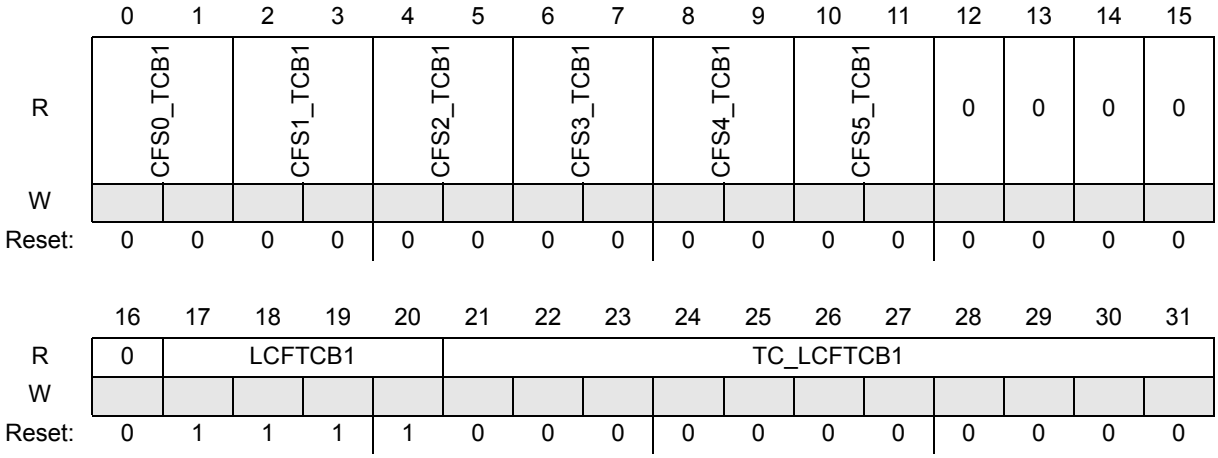


Figure 408. eQADC CFIFO Status Snapshot Register 1 (EQADC_CFSSR1)

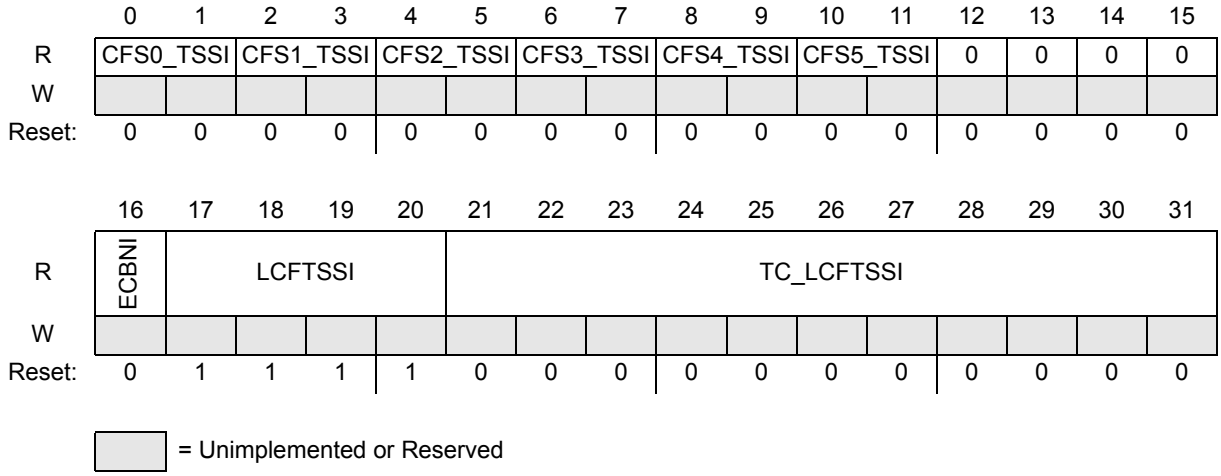


Figure 409. eQADC CFIFO Status Snapshot Register 2 (EQADC_CFSSR2)

Table 266. EQADC_CFSSRx field descriptions

Field	Description
CFSx_TCB*n*[0:1]	CFIFO Status at Transfer to CBuffer*n* *(n = 0, 1)* CFSx_TCB*n* indicates the CFIFOx status of previously completed command transfer. CFSx_TCB*n* is a copy of the corresponding CFSx in the eQADC CFIFO Status Register (EQADC_CFSR), captured at the time a current command transfer to CBuffer*n* is initiated.
LCFTCB*n*[0:3]	Last CFIFO to Transfer to CBuffer*n* *(n = 0, 1)* LCFTCB*n* holds the CFIFO number to have completed a previous command transfer to CBuffer*n*. 0b0000: Last command was transferred from CFIFO0 0b0001: Last command was transferred from CFIFO1 0b0010: Last command was transferred from CFIFO2 0b0011: Last command was transferred from CFIFO3 0b0100: Last command was transferred from CFIFO4 0b0101: Last command was transferred from CFIFO5 0b0110–0b1110: Reserved 0b1111: No command was transferred to CBuffer*n*
TC_LCFTCB*n*[0:10]	Transfer Counter for Last CFIFO to transfer commands to CBuffer*n* TC_LCFTCB*n* indicates the number of commands which have been completely transferred from CFIFOx when a current command transfer from CFIFOx to CBuffer*n* is initiated. TC_LCFTCB*n* is a copy of the corresponding TC_CFx in the eQADC CFIFO Transfer Counter Registers (EQADC_CFTCR), captured at the time a current command transfer from CFIFOx to CBuffer*n* is initiated. This field has no meaning when LCFTCB*n* is 0b1111.
CFSx_TSSI[0:1]	CFIFO Status at Transfer through the eQADC SSI CFSx_TSSI indicates the CFIFOx status of previously completed serial transmission through the eQADC SSI. CFSx_TSSI is a copy of the corresponding CFSx in the eQADC CFIFO Status Register (EQADC_CFSR), capture at the time a current serial transmission through the eQADC SSI is initiated.
ECBNI	External CBuffer Number Indicator ECBNI indicates to which external CBuffer the previous command was transmitted. 1: Last command was transferred to CBuffer3. 0: Last command was transferred to CBuffer2.

Table 266. EQADC_CFSSRx field descriptions (continued)

Field	Description
LCFTSSI[0:3]	Last CFIFO to Transfer Commands through the eQADC SSI LCFTSSI holds the CFIFO number to have completed a previous command transfer to an external CBuffer through the eQADC SSI. LCFTSSI does not indicate the transmission of null messages. 0b0000: Last command was transferred from CFIFO0 0b0001: Last command was transferred from CFIFO1 0b0010: Last command was transferred from CFIFO2 0b0011: Last command was transferred from CFIFO3 0b0100: Last command was transferred from CFIFO4 0b0101: Last command was transferred from CFIFO5 0b0110–0b1110: Reserved 0b1111: No command was transferred to an external CBuffer
TC_LCFSSI[0:10]	Transfer Counter for Last CFIFO to Transfer commands through eQADC SSI TC_LCFTSSI indicates the number of commands which have been completely transferred from a particular CFIFO when a command transfer from that CFIFO to an external CBuffer is initiated. TC_LCFTSSI is a copy of the corresponding TC_CFx in the eQADC CFIFO Transfer Counter Registers (EQADC_CFTCR), captured at the time a current command transfer to an external CBuffer is initiated. This field has no meaning when LCFTSSI is 0b1111.

23.4.2.11 eQADC CFIFO Status Register (EQADC_CFSR)

The eQADC CFIFO Status Register (EQADC_CFSR) contains the current CFIFO status. The EQADC_CFSR is read only. Writing to the EQADC_CFSR has no effect.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS0		CFS1		CFS2		CFS3		CFS4		CFS5		0	0	0	0
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	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

Figure 410. eQADC CFIFO Status Register (EQADC_CFSR)

Table 267. EQADC_CFSR field descriptions

Field	Description
CFSx[0:1]	CFIFO Status CFSx indicates the current status of CFIFO*x*. Refer to *Table 268* for more information on CFIFO status.

Table 268. Current CFIFO status

CFIFO status	CFSx[0:1] value	Explanation
IDLE	0b00	CFIFO is disabled. CFIFO is in single-scan edge or level trigger mode and does not have SSS asserted. eQADC completed the transfer of the last entry of the CQueue in single-scan mode.
Reserved	0b01	—
WAITING FOR TRIGGER	0b10	CFIFO Mode is modified to continuous-scan edge or level trigger mode. CFIFO Mode is modified to single-scan edge or level trigger mode and SSS is asserted. CFIFO Mode is modified to single-scan software trigger mode and SSS is negated. CFIFO is paused. eQADC transferred the last entry of the queue in continuous-scan edge trigger mode.
TRIGGERED	0b11	CFIFO is triggered.

23.4.2.12 eQADC SSI Control Register (EQADC_SSICR)

The eQADC SSI Control Register (EQADC_SSICR) configures the SSI sub-block.

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

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

= Unimplemented or Reserved

Figure 411. eQADC SSI Control Register (EQADC_SSICR)

Table 269. EQADC_SSICR field descriptions

Field	Description
MDT[0:2]	Minimum Delay after Transmission MDT field defines the minimum delay after transmission time (t_{MDT}) expressed in serial clock (FCK) periods. t_{MDT} is minimum time $\overline{SDS}$ should be kept negated between two consecutive serial transmissions. The values below show the minimum delay after transfer time according to how MDT is set. 0b000: 1 0b001: 2 0b010: 3 0b011: 4 0b100: 5 0b101: 6 0b110: 7 0b111: 8 **Note:** The MDT field must only be written when the serial transmissions from the eQADC SSI are disabled—see ESSIE field in the eQADC Module Configuration Register (EQADC_MCR).
BR[0:3]	Baud Rate Field The BR field selects system clock divide factor as shown below. The baud clock is calculated by dividing the system clock by the clock divide factor specified with the BR field. 0b0000: 2 0b0001: 3 0b0010: 4 0b0011: 5 0b0100: 6 0b0101: 7 0b0110: 8 0b0111: 9 0b1000: 10 0b1001: 11 0b1010: 12 0b1011: 13 0b1100: 14 0b1101: 15 0b1110: 16 0b1111: 17 **Note:** The BR field must only be written when the eQADC SSI is disabled—see ESSIE field in the eQADC Module Configuration Register (EQADC_MCR). **Note:** If the system clock is divided by an odd number then the serial clock will have a duty cycle different from 50%.

23.4.2.13 eQADC SSI Receive Data Register (EQADC_SSIRDR)

The eQADC SSI Receive Data Register (EQADC_SSIRDR) records the last message received from the external device.

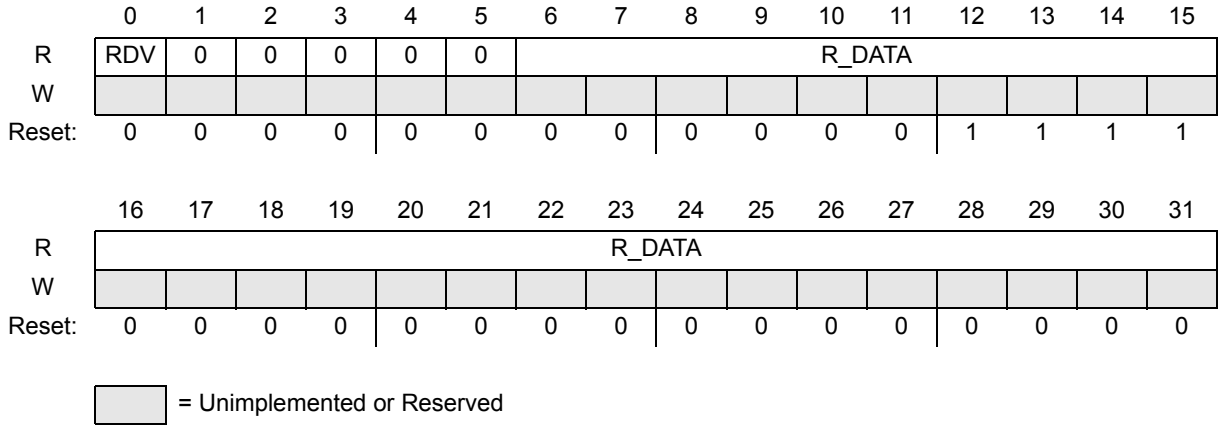


Figure 412. eQADC SSI Receive Data Register (EQADC_SSIRDR)

Table 270. EQADC_SSIRDR field descriptions

Field	Description
RDV	Receive Data Valid Bit The RDV bit indicates if the last received data is valid. This bit is cleared automatically whenever the EQADC_SSIRDR is read. Writes have no effect. 1: Receive data is valid. 0: Receive data is not valid.
R_DATA[0:25]	eQADC Receive DATA Field The R_DATA contains the last result message that was shifted in. Writes to the R_DATA have no effect. Messages that were not completely received due to a transmission abort will not be copied into EQADC_SSIRDR.

23.4.2.14 eQADC STAC Client Configuration Register (EQADC_REDLCCR)

The eQADC STAC Client Configuration Register (EQADC_REDLCCR) contains bits used to control which time slots the eQADC selects to obtain predefined external time bases.

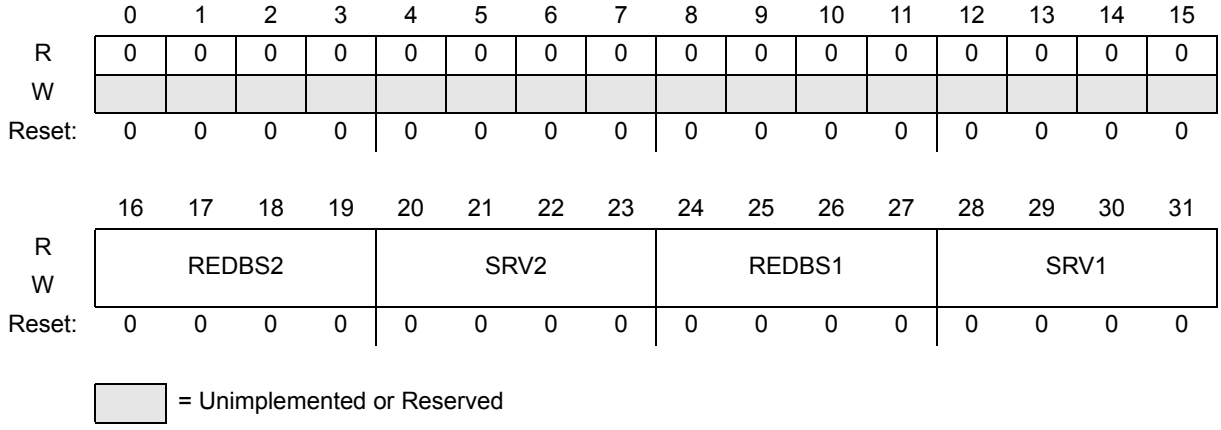


Figure 413. eQADC STAC Client Configuration Register (EQADC_REDLCCR)

Table 271. EQADC_REDLCCR field descriptions

Field	Description
REDBS*m*[0:3]	STAC Timebase Bits Selection *m* (*m* = 1, 2) The REDBS*m* field selects 16 bits from the total of 24 bits that are received from the STAC interface as described in below. Consider TBASE*m*[0:23] the selected time base from slot SRV*m* 0b0000: TBASEm[15:00:15] 0b0001: TBASEm[16:11:16] 0b0010: TBASEm[17:22:17] 0b0011: TBASEm[18:33:18] 0b0100: TBASEm[19:44:19] 0b0101: TBASEm[20:55:20] 0b0110: TBASEm[21:66:21] 0b0111: TBASEm[22:77:22] 0b1000: TBASEm[23:88:23] Others: Reserved
SRV*m*[0:3]	STAC Server Data Slot Selector *m* (*m* = 1, 2) The field SRV*m* indicates the slot number that contains the desired time base value sent by the STAC server. It is possible to have up to 16 different sources to be selected.

23.4.2.15 eQADC CFIFO Registers (EQADC_CFxRw) (x = 0, ..,5; w = 0, .., 3)

The eQADC CFIFO Registers (EQADC_CFxRw) (x = 0, .., 5; w = 0, .., 3) provide visibility of the contents of a CFIFO for debugging purposes. Each CFIFO has four registers which are uniquely mapped to its four 32-bit entries. Refer to *Section 23.5.4: eQADC command FIFOs* for more information on CFIFOs. These registers are read only. Data written to these registers is ignored.

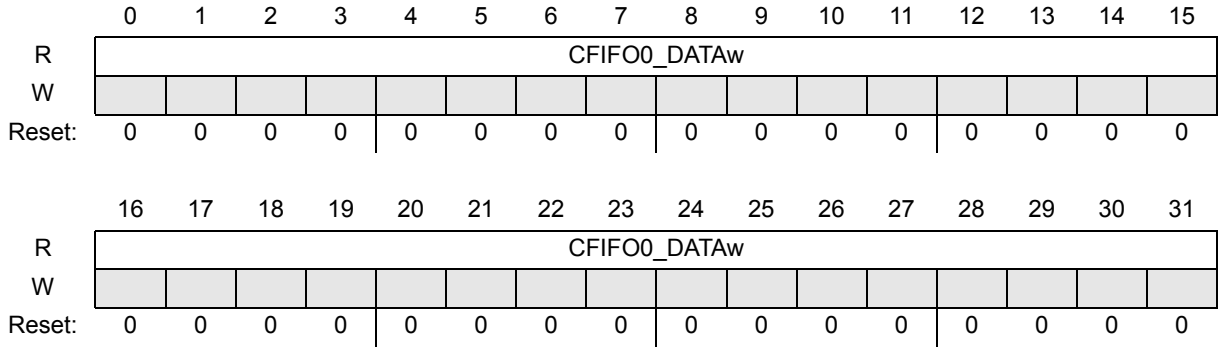


Figure 414. eQADC CFIFO0 Registers (EQADC_CF0Rw) (w = 0,..., 3)

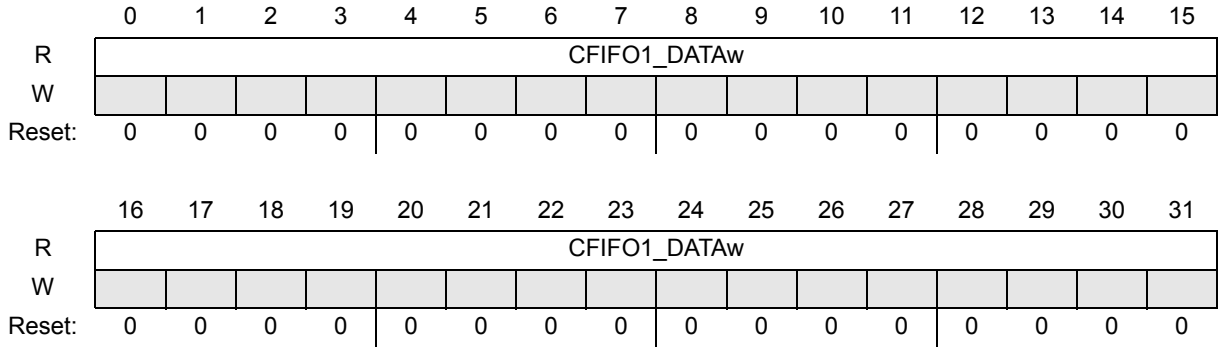


Figure 415. eQADC CFIFO1 Registers (EQADC_CF1Rw) (w = 0,..., 3)

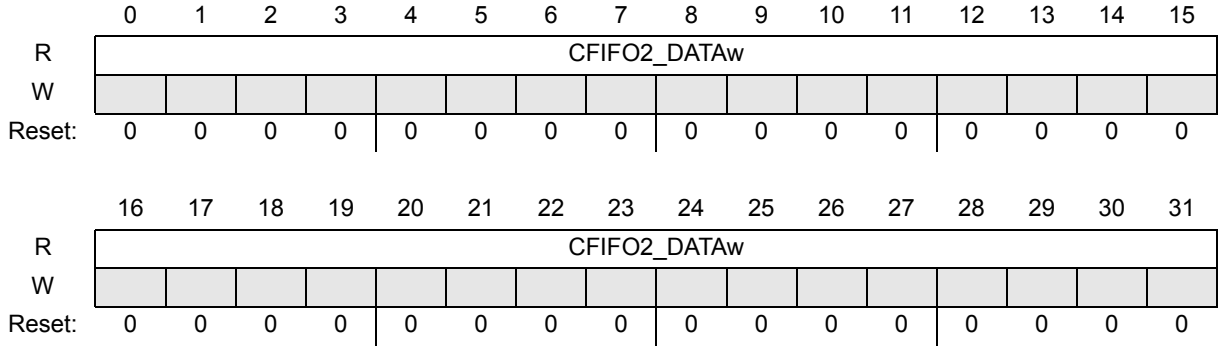


Figure 416. eQADC CFIFO2 Registers (EQADC_CF2Rw) (w = 0,..., 3)

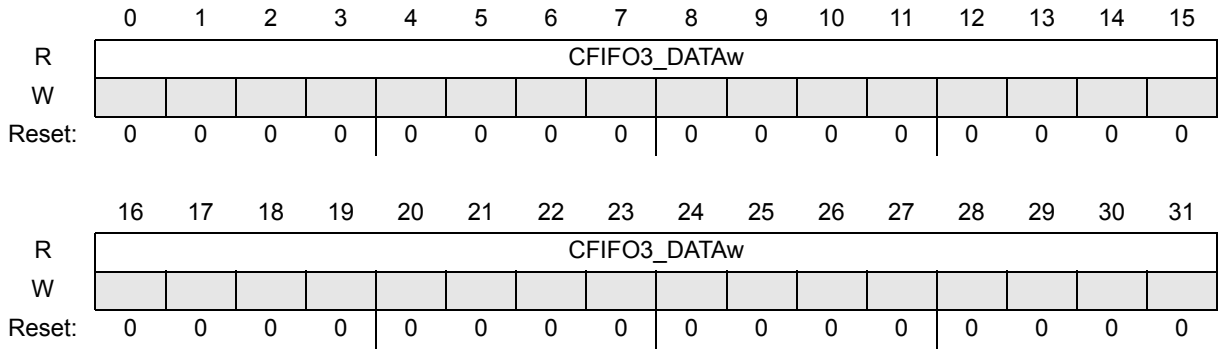


Figure 417. eQADC CFIFO3 Registers (EQADC_CF3Rw) (w = 0,..., 3)

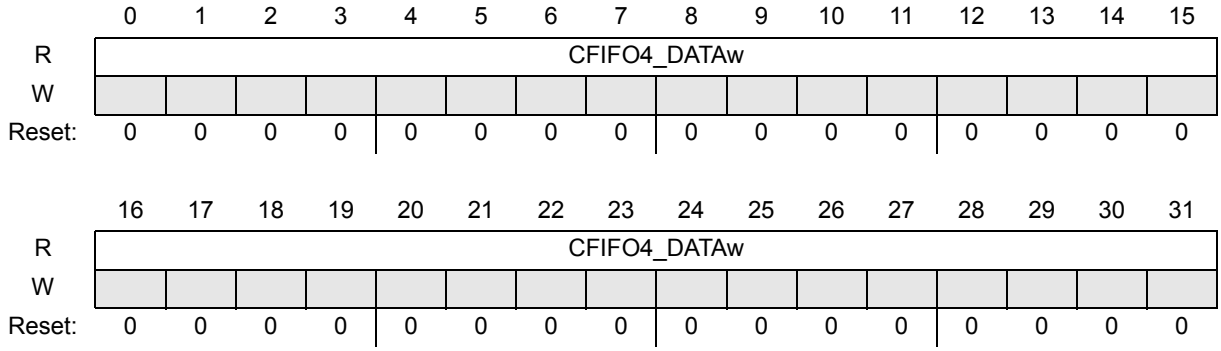


Figure 418. eQADC CFIFO4 Registers (EQADC_CF4Rw) (w = 0,...,3)

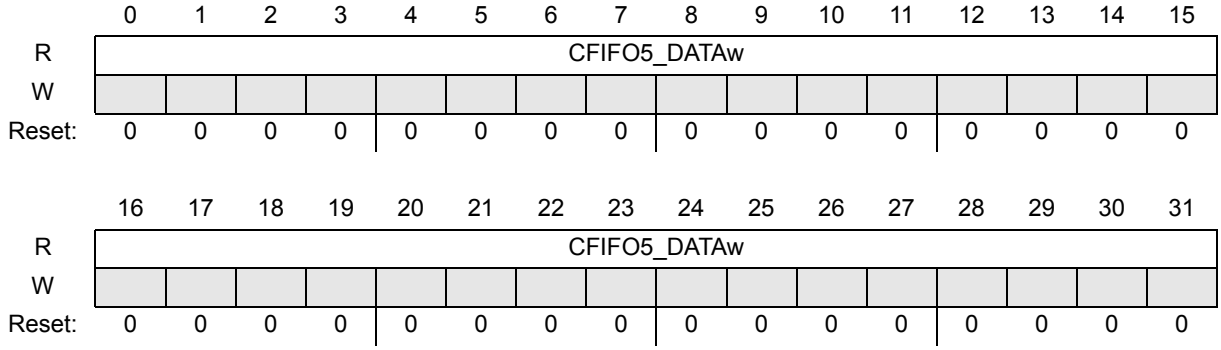


Figure 419. eQADC CFIFO5 Registers (EQADC_CF5Rw) (w = 0,..., 3)

Table 272. EQADC_CFxRw field descriptions

Field	Description
CFIFOx_DATAw[0:31]	CFIFOx Data w (w = 0, .., 3) Reading CFIFOx_DATAw returns the value stored on the wth entry of CFIFOx. Each CFIFO is composed of four 32-bit entries, with register 0 being mapped to the one with the smallest memory mapped address.

23.4.2.16 eQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w = 0, .., 3)

The eQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w = 0, .., 3) provide visibility of the contents of the extended portion of CFIFO0 for debugging purposes. There are four registers which are uniquely mapped to the CFIFO's four 32-bit entries. Refer to *Section 23.5.4: eQADC command FIFOs* for more information on CFIFOs. These registers are read only. Data written to these registers is ignored.

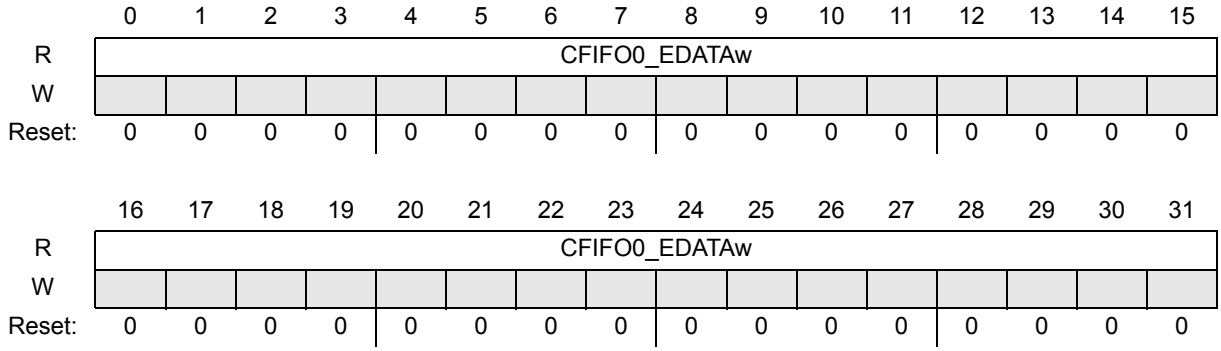


Figure 420. eQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w = 0,..., 3)

Table 273. EQADC_CF0ERw field descriptions

Field	Description
CFIFOx_EDATAw[0:31]	CFIFOx_EDATAw (w = 0, .., 3) Reading CFIFOx_EDATAw returns the value stored on the wth entry of CFIFOx. Each CFIFO is composed of four 32-bit entries, with register 0 being mapped to the one with the smallest memory mapped address.

23.4.2.17 eQADC RFIFO Registers (EQADC_RFxRw) (x = 0, .., 5; w = 0, .., 3)

The eQADC RFIFO Registers (EQADC_RFxRw) (x = 0, .., 5; w = 0, .., 3) provide visibility of the contents of a RFIFO for debugging purposes. Each RFIFO has four registers which are uniquely mapped to its four 16-bit entries. Refer to *Section 23.5.5: eQADC result FIFOs* for more information on RFIFOs. These registers are read only. Data written to these registers is ignored.

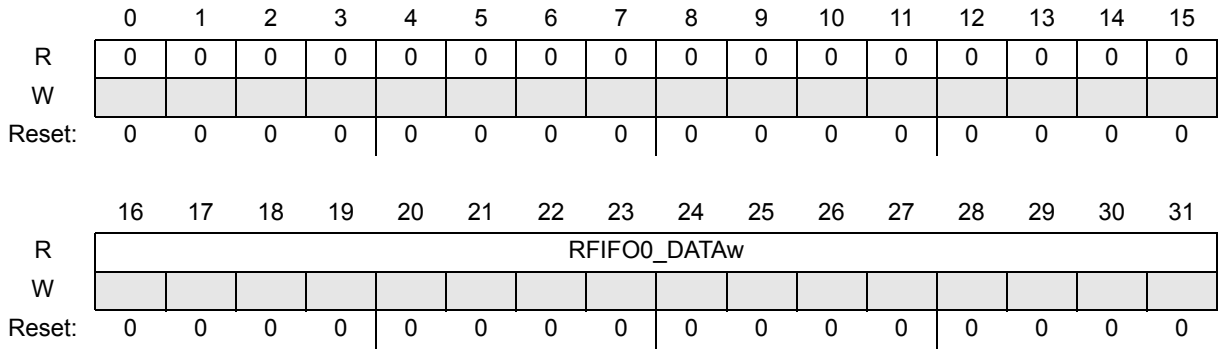


Figure 421. eQADC RFIFO0 Registers (EQADC_RF0Rw) (w = 0,..., 3)

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO1_DATAw															
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 422. eQADC RFIFO1 Registers (EQADC_RF1Rw) (w = 0,..., 3)

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO2_DATAw															
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 423. eQADC RFIFO2 Registers (EQADC_RF2Rw) (w = 0,..., 3)

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO3_DATAw															
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 424. eQADC RFIFO3 Registers (EQADC_RF3Rw) (w = 0,..., 3)

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO4_DATAw															
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 425. eQADC RFIFO4 Registers (EQADC_RF4Rw) (w = 0,..., 3)

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO5_DATAw															
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 426. eQADC RFIFO5 Registers (EQADC_RF5Rw) (w = 0,..., 3)

Table 274. EQADC_RFxRw field descriptions

Field	Description
RFIFOx_DATAw[0:15]	RFIFOx Data w (w = 0, .., 3) Reading RFIFOx_DATAw returns the value stored on the wth entry of RFIFOx. Each RFIFO is composed of four 16-bit entries, with register 0 being mapped to the one with the smallest memory mapped address.

23.4.3 On-chip ADC registers

This section describes a list of registers that control on-chip ADC operation. The ADC registers are not part of the CPU accessible memory map. These registers can only be accessed indirectly through configuration commands. There are four non-memory-mapped registers per ADC, plus 12 registers shared by both ADCs. The address, usage, and access privilege of each register is shown in *Table 275*. Data written to or read from reserved areas of the memory map is undefined.

Their assigned addresses are the values used to set the ADC_REG_ADDRESS field of the read/write configurations commands bound for the on-chip ADCs. These are half-word addresses. Further, the following restrictions apply when accessing these registers:

- Registers ADC0_CR, ADC0_GCCR, ADC0_OCCR, ADC0_AGR and ADC0_AOR can only be accessed by configuration commands sent to CBuffer0.
- Registers ADC1_CR, ADC1_GCCR, ADC1_OCCR, ADC1_AGR and ADC1_AOR can only be accessed by configuration commands sent to CBuffer1.
- Registers ADC_TSCR, ADC_TBCR, ADC_ACR1–8 and ADC_PUDCR0–7 can be accessed by configuration commands sent to CBuffer0 or to CBuffer1. A data write to any of these registers through a configuration command sent to CBuffer0 will write the same memory location as when writing to it through a configuration command sent to CBuffer1.

Note: Simultaneous write accesses from CBuffer0 and CBuffer1 to any of the shared registers are not allowed.

Table 275. On-chip ADC memory map

ADC address	Use	Access	Location
0x00	ADC0/ADC1[(1)] Conversion Command for Standard Configuration	Write	*on page 730*
0x01	ADC0/ADC1 Configuration Control Register (ADC0_CR, ADC1_CR)	Write/Read	*on page 714*
0x02	Time Stamp Control Register (ADC_TSCR)	Write/Read	*on page 718*
0x03	Time Base Counter Register (ADC_TBCR)	Write/Read	*on page 719*
0x04	ADC0/ADC1 Gain Calibration Constant Register (ADC0_GCCR, ADC1_GCCR)	Write/Read	*on page 719*
0x05	ADC0/ADC1 Offset Calibration Constant Register (ADC0_OCCR, ADC1_OCCR)	Write/Read	*on page 720*
0x06–0x07	Reserved	—	—
0x08	ADC0/ADC1 Conversion Command for Alternate Configuration 1	Write	*on page 733*
0x09	ADC0/ADC1 Conversion Command for Alternate Configuration 2	Write	*on page 733*
0x0A	ADC0/ADC1 Conversion Command for Alternate Configuration 3	Write	*on page 733*
0x0B	ADC0/ADC1 Conversion Command for Alternate Configuration 4	Write	*on page 733*
0x0C	ADC0/ADC1 Conversion Command for Alternate Configuration 5	Write	*on page 733*
0x0D	ADC0/ADC1 Conversion Command for Alternate Configuration 6	Write	*on page 733*
0x0E	ADC0/ADC1 Conversion Command for Alternate Configuration 7	Write	*on page 733*

Table 275. On-chip ADC memory map (continued)

ADC address	Use	Access	Location
0x0F	ADC0/ADC1 Conversion Command for Alternate Configuration 8	Write	on page 733
0x10–0x2F	Reserved	—	—
0x30	Alternate Configuration 1 Control Register (ADC_ACR1)	Write/Read	on page 721
0x31	ADC0/ADC1 Alternate Gain 1 Register (ADC0_AGR1, ADC1_AGR1)	Write/Read	on page 723
0x32	ADC0/ADC1 Alternate Offset 1 Register (ADC0_AOR1, ADC1_AOR1)	Write/Read	on page 724
0x33	Reserved	—	—
0x34	Alternate Configuration 2 Control Register (ADC_ACR2)	Write/Read	on page 721
0x35	ADC0/ADC1 Alternate Gain 2 Register (ADC0_AGR2, ADC1_AGR2)	Write/Read	on page 723
0x36	ADC0/ADC1 Alternate Offset 2 Register (ADC0_AOR2, ADC1_AOR2)	Write/Read	on page 724
0x37	Reserved	—	—
0x38	Alternate Configuration 3 Control Register (ADC_ACR3)	Write/Read	on page 721
0x39	Reserved	—	—
0x3A	Reserved	—	—
0x3B	Reserved	—	—
0x3C	Alternate Configuration 4 Control Register (ADC_ACR4)	Write/Read	on page 721
0x3D	Reserved	—	—
0x3E	Reserved	—	—
0x3F	Reserved	—	—
0x40	Alternate Configuration 5 Control Register (ADC_ACR5)	Write/Read	on page 721
0x41	Reserved	—	—
0x42	Reserved	—	—
0x43	Reserved	—	—
0x44	Alternate Configuration 6 Control Register (ADC_ACR6)	Write/Read	on page 721
0x45	Reserved	—	—
0x46	Reserved	—	—
0x47	Reserved	—	—

Table 275. On-chip ADC memory map (continued)

ADC address	Use	Access	Location
0x48	Alternate Configuration 7 Control Register (ADC_ACR7)	Write/Read	*on page 721*
0x49	Reserved	—	—
0x4A	Reserved	—	—
0x4B	Reserved	—	—
0x4C	Alternate Configuration 8 Control Register (ADC_ACR8)	Write/Read	*on page 721*
0x4D–0x6F	Reserved	—	—
0x70	Pull Up/Down Control Register0 (ADC_PUDCR0)	Write/Read	*on page 724*
0x71	Pull Up/Down Control Register1 (ADC_PUDCR1)	Write/Read	*on page 724*
0x72	Pull Up/Down Control Register2 (ADC_PUDCR2)	Write/Read	*on page 724*
0x73	Pull Up/Down Control Register3 (ADC_PUDCR3)	Write/Read	*on page 724*
0x74	Pull Up/Down Control Register4 (ADC_PUDCR4)	Write/Read	*on page 724*
0x75	Pull Up/Down Control Register5 (ADC_PUDCR5)	Write/Read	*on page 724*
0x76	Pull Up/Down Control Register6 (ADC_PUDCR6)	Write/Read	*on page 724*
0x77	Pull Up/Down Control Register7 (ADC_PUDCR7)	Write/Read	*on page 724*
0x78–0x97	Reserved for ADC_PUDCR8 to ADC_PUDCR39	—	—
0x98–0xFF	Reserved	—	—

1. Throughout the table, ADC0/ADC1 indicates that if the command is stored in CBuffer0 it will be applied to ADC0 and if in CBuffer1 it applies to ADC1. If this indication is omitted the register applies for both ADC0 and ADC1, independent of the CBuffer used.

23.4.3.1 ADC0/1 Control Registers (ADC0_CR and ADC1_CR)

The ADC0/1 Control Registers (ADC0/1_CR) is used to define the standard configuration of the ADC. In the standard configuration, the parameters contained in the Alternate Configuration Control Registers (ADC_ACR1–8) are fixed at their reset value. A conversion uses the standard configuration when the conversion command (with the standard format) is written to address 0x00 of the on-chip ADC memory map. Refer to *Section 23.5.2.3.1.1: Conversion command format for the standard configuration*.

ADC0 register address: 0x01

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ADC0_EN	0	0	0	ADC0_EMUX	0	ADC0_TBSEL		ADC0_ODD_PS	ADC0_CLK_DTY	ADC0_CLK_SEL	ADC0_CLK_PS				
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

ADC1 register address: 0x01

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ADC1_EN	0	0	0	ADC1_EMUX	0	ADC1_TBSEL		ADC1_ODD_PS	ADC1_CLK_DTY	ADC1_CLK_SEL	ADC1_CLK_PS				
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

= Unimplemented or Reserved

Figure 427. ADC0/1 Control Registers (ADC0/1_CR)

Table 276. ADC0/1_CR field descriptions

Field	Description
ADC0/1_EN	Enable bit for ADC0/1 ADC0/1_EN enables ADC0/1 to perform A/D conversions. Refer to *Section 23.5.6.1: Enabling and disabling the on-chip ADCs* for details. 1: ADC is enabled and ready to perform A/D conversions. 0: ADC is disabled. Clock supply to ADC0/1 is stopped. **Note:** Conversion commands sent to the CBuffer of a disabled ADC are ignored by the ADC control hardware. **Note:** When the ADC0/1_EN status is changed from asserted to negated, the ADC Clock will not stop until it reaches its low phase.
ADC0/1_EMUX	External Multiplexer enable for ADC0/1 When ADC0/1_EMUX is asserted, the MA pins will output digital values according to the number of the external channel being converted for selecting external multiplexer inputs. Refer to *Section 23.5.7: Internal/External multiplexing* for a detailed description about how ADC0/1_EMUX affects channel number decoding. 1: External multiplexer enabled; external multiplexer channels can be selected. 0: External multiplexer disabled; no external multiplexer channels can be selected. **Note:** Both ADC0 and ADC1 of an eQADC module pair must be enabled before calibrating or using either ADC0 or ADC1 of the pair. Failure to enable both ADC0 and ADC1 of the pair can result in inaccurate conversions. **Note:** Both ADC0/1_EMUX bits must not be asserted at the same time. **Note:** The ADC0/1_EMUX bit must only be written when the ADC0/1_EN bit is negated. ADC0/1_EMUX can be set during the same write cycle used to set ADC0/1_EN.

Table 276. ADC0/1_CR field descriptions (continued)

Field	Description
ADC0/1_TBSEL[0:1]	Timebase Selection for ADC0/1 The ADC0/1_TBSEL[0:1] field selects the time information to be used as timestamp as shown below. 00: Selects internally generated time base as time stamp 01: Selects imported time base 1 indicated by SRV1 bit field of EQADC_REDLCCR 10: Selects imported time base 2 indicated by SRV2 bit field of EQADC_REDLCCR 11: Reserved **Note:** This selection is overridden by the corresponding field ATBSEL in the ADC_ACR1–8 registers when the alternate conversion command is used.
ADC0/1_ODD_PS	Clock Prescaler Odd Rates Selector for ADC0/1 The ADC0/1_CLK_DTY field controls the duty rate of the ADC0/1 clock when the ADC0/1_CLK_PS field is asserted. The generated clock has an odd number of system clock cycles, therefore this field is used to select a clock duty higher or lower than 50%. 1: Odd divide factor is selected. The final divide factor is dependent of ADC0/1_CLK_PS field. 0: Even divide factor is selected. The final divide factor is dependent of ADC0/1_CLK_PS field.
ADC0/1_CLK_DTY	Clock Duty Rate Selector for ADC0/1 (for odd divide factors) The ADC0/1_ODD_PS field is used together with the ADC0/1_CLK_PS field to define even/odd divide factors in the generation of the ADC0/1 clocks. Refer to *Table 277* for available divide factors. 1: clock high pulse is longer 1 clock cycle than low portion. 0: clock low interval is longer 1 clock cycle than high pulse.
ADC0/1_CLK_SEL	Clock Selector for ADC0/1 The ADC0/1_CLK_SEL is used to select between the system clock signal or the prescaler output signal. The prescaler provides the system clock signal divided by a even factor from 2 to 64. This is required to permit the ADC to run as fast as possible when the device is in Low Power Active mode and system clock is around 1 MHz. 1: System clock is selected - maximum frequency. 0: Prescaler output clock is selected. **Note:** The ADC0/1_CLK_SEL bits must only be written when the ADC0/1_EN bit is negated. ADC0/1_CLK_SEL can be set during the same write cycle used to set ADC0/1_EN.
ADC0/1_CLK_PS[0:4]	Clock Prescaler Field for ADC0/1 The ADC0/1_CLK_PS field controls the system clock divide factor for the ADC0/1 clock as in *Table 277*. See *Section 23.5.6.2: ADC clock and conversion speed* for details about how to set ADC0/1_CLK_PS. **Note:** The ADC0/1_CLK_PS field must only be written when the ADC0/1_EN bit is negated. This field can be configured during the same write cycle used to set ADC0/1_EN.

Table 277. System clock divide factor for ADC clock

ADC0/1_CLK_PS[0:4]	System clock divide factor	
	ADC0/1_ODD_PS = 0	ADC0/1_ODD_PS = 1
0b00000	2	3
0b00001	4	5
0b00010	6	7
0b00011	8	9
0b00100	10	11
0b00101	12	13
0b00110	14	15
0b00111	16	17
0b01000	18	19
0b01001	20	21
0b01010	22	23
0b01011	24	25
0b01100	26	27
0b01101	28	29
0b01110	30	31
0b01111	32	33
0b10000	34	35
0b10001	36	37
0b10010	38	39
0b10011	40	41
0b10100	42	43
0b10101	44	45
0b10110	46	47
0b10111	48	49
0b11000	50	51
0b11001	52	53
0b11010	54	55
0b11011	56	57
0b11100	58	59
0b11101	60	61
0b11110	62	63
0b11111	64	65

23.4.3.2 ADC Time Stamp Control Register (ADC_TSCR)

The ADC Time Stamp Control Register (ADC_TSCR) contains a system clock divide factor used in the making of the time base counter clock. It determines the frequency the time base counter will run at. ADC_TSCR can be accessed by configuration commands sent to CBuffer0 or to CBuffer1. A data write to ADC_TSCR through a configuration command sent to CBuffer0 will write the same memory location as when writing to it through a configuration command sent to CBuffer1.

Note: Simultaneous write accesses from CBuffer0 and CBuffer1 to ADC_TSCR are not allowed.

ADC0/1 register address: 0x02

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

= Unimplemented or Reserved

Figure 428. ADC Time Stamp Control Register (ADC_TSCR)

Table 278. ADC_TSCR field descriptions

Field	Description
TBC_CLK_PS[0:3]	Time Base Counter Clock Prescaler The TBC_CLK_PS field contains the system clock divide factor for the time base counter. It controls the accuracy of the time stamp. The prescaler is disabled when TBC_CLK_PS is set to 0b0000. See *Table 279* for the time stamp clock divider factors. **Note:** If TBC_CLK_PS is not set to disabled, it must not be changed to any other value besides disabled. If TBC_CLK_PS is set to disabled it can be changed to any other value.

Table 279. Clock divide factor for time stamp

TBC_CLK_PS[0:3] value	System clock divide factor	Clock to time stamp counter for a 120 MHz system clock (MHz)
0b0000	Disabled	Disabled
0b0001	1	120
0b0010	2	60
0b0011	4	30
0b0100	6	20
0b0101	8	15
0b0110	10	12
0b0111	12	10
0b1000	16	7.5
0b1001	32	3.75
0b1010	64	1.88

Table 279. Clock divide factor for time stamp (continued)

TBC_CLK_PS[0:3] value	System clock divide factor	Clock to time stamp counter for a 120 MHz system clock (MHz)
0b1011	128	0.94
0b1100	256	0.47
0b1101	512	0.23
0b1110–0b1111	Reserved	—

23.4.3.3 ADC Time Base Counter Registers (ADC_TBCR)

The ADC Time Base Counter Register (ADC_TBCR) contains the current value of the time base counter. ADC_TBCR can be accessed by configuration commands sent to CBuffer0 or to CBuffer1. A data write to ADC_TBCR through a configuration command sent to CBuffer0 will write the same memory location as when writing to it through a configuration command sent to CBuffer1.

Note: Simultaneous write accesses from CBuffer0 and CBuffer1 to ADC_TBCR are not allowed.

ADC0/1 register address: 0x03

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

= Unimplemented or Reserved

Figure 429. ADC Time Base Counter Register (ADC_TBCR)

Table 280. ADC_TBCR field descriptions

Field	Description
TBC_VALUE[0:15]	Time Base Counter VALUE Field The TBC_VALUE field contains the current value of the time base counter. Reading TBC_VALUE returns the current value of time base counter. Writes to TBC_VALUE register load the written data to the counter. The time base counter counts from 0x0000 to 0xFFFF and wraps when reaching 0xFFFF.

23.4.3.4 ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)

The ADC0/1 Gain Calibration Constant Register (ADC0/1_GCCR) contains the gain calibration constant used to fine-tune the ADC0/1 conversion results. Refer to *Section 23.5.6.6: ADC calibration feature* for details about the calibration scheme used in the eQADC.

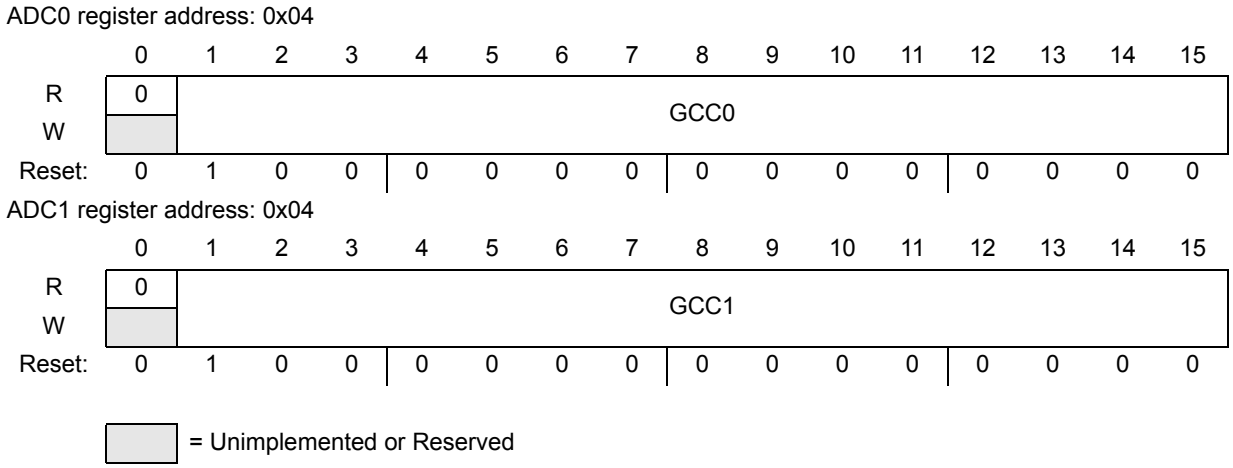


Figure 430. ADC0/1 Gain Calibration Constant Registers (ADC0/1_GCCR)

Table 281. ADC0/1_GCCR field descriptions

Field	Description
GCC0/1[0:14]	Gain calibration constant for ADC0/1 GCC0/1 contains the gain calibration constant used to fine-tune ADC0/1 conversion results. It is an unsigned 15-bit fixed pointed value. The gain calibration constant is an unsigned fixed point number expressed in the *GCC_INT.GCC_FRAC* binary format. The integer part of the gain constant (GCC_INT) contains a single binary digit while its fractional part (GCC_FRAC) contains fourteen digits. For details about the GCC data format refer to *Section 23.5.6.6.2: MAC unit and operand data format*.

23.4.3.5 ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)

The ADC0/1 Offset Calibration Constant Register (ADC0/1_OCCR) contains the offset calibration constant used to fine-tune ADC0/1 conversion results. The offset constant is a signed 14-bit integer value. Refer to *Section 23.5.6.6: ADC calibration feature* for details about the calibration scheme used in the eQADC.

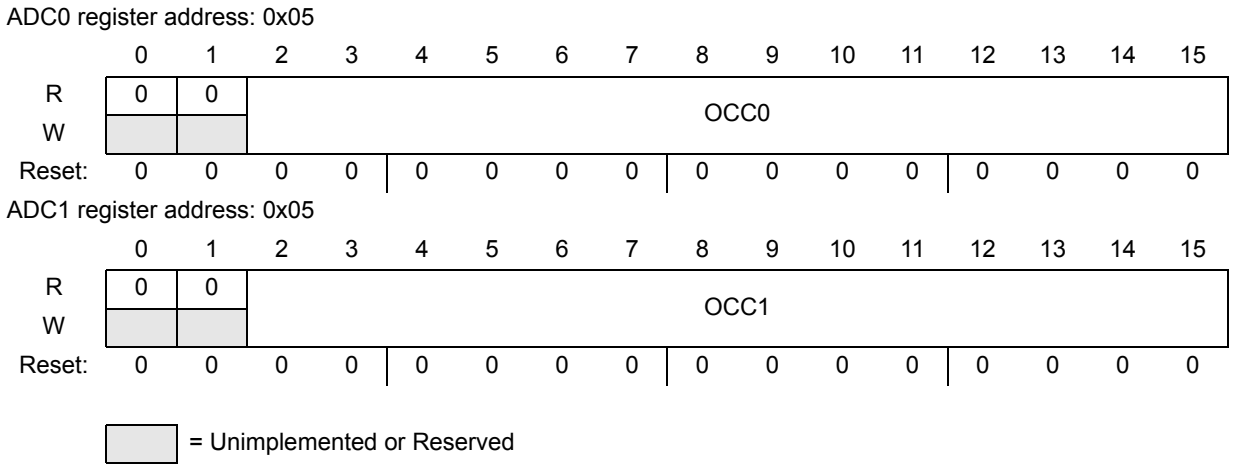


Figure 431. ADC0/1 Offset Calibration Constant Registers (ADC0/1_OCCR)

Table 282. ADC0/1_OCCR field descriptions

Field	Description
OCC0/1[0:13]	Offset Calibration Constant of ADC0/1 OCC0/1 contains the offset calibration constant used to fine-tune ADC0/1 conversion results. Negative values should be expressed using the two's complement representation.

23.4.3.6 Alternate Configuration 1–8 Control Registers (ADC_ACR1–8)

The Alternate Configuration Control Registers (ADC_ACR1–8) are used to configure the alternate configurations of the ADC. There are eight possible alternate configurations, each one associated with one of the ADC_ACR1–8 registers. All alternate configurations share the same standard configuration parameters from the ADC0/1_CRs, plus additional configuration parameters contained in the ADC_ACR1–8. A conversion uses one of the alternate configurations when the conversion command (with the alternate configuration format) is written to an address in the range 0x08–0x0F of the on-chip ADC memory map. Refer to *Section 23.5.2.3.1.2: Conversion command format for alternate configurations*.

ADC0/1 register address: 0x30
ADC0/1 register address: 0x34
ADC0/1 register address: 0x38
ADC0/1 register address: 0x3C
ADC0/1 register address: 0x40
ADC0/1 register address: 0x44
ADC0/1 register address: 0x48
ADC0/1 register address: 0x4C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RET_INH	0	DEST				FMTA	0	RESSEL		0	0	ATBSEL		PRE_GAIN	
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 432. Alternate Configuration 1–8 Control Registers (ADC_ACR1–8)

Table 283. ADC_ACR1–8 field descriptions

Field	Description
RET_INH	Result Transfer Inhibit / Decimation Filter Pre-Fill This bit is used to inhibit the transfer of the result data from the peripheral module to the result queue. When the module is a Decimation Filter, this bit sets the filter in a special mode (PRE-FILL) in which it does not generate decimated samples out from the conversion results received from the eQADC block, but the conversion samples are used by the filter algorithm. This feature allows a proper initialization of the Decimation Filter without generating any decimated result. Or this bit is useful for sending the result of the ADC to the STAC bus master but not putting the result in the result queue. 1: No result transfer to result queue / Decimation Filter PRE-FILL mode 0: Result transfer to result queue / Decimation Filter in filtering mode
DEST[0:3]	Conversion Result Destination Selection The DEST[0:3] field selects the destination of the conversion result generated by the Alternate Conversion Command according to the values shown below. 0000: The conversion result is sent to the RFIFOs. The data format is specified by the FFMT bit in the conversion command. 0001–1111: The conversion result is sent to the Parallel Side Interface. Up to 15 devices can be connected to the parallel interface, using the DEST field to select which one receives the conversion result. The data format is specified by the FMTA bit in the Alternate Configuration Control Register. This field also affects the behavior of the FMTA bit and the FFMT bit of the conversion command for alternate configurations (see *Section 23.5.2.3.1.2: Conversion command format for alternate configurations*).
FMTA	Conversion Data Format for Alternate Configuration If the DEST field is not 0b000, the FMTA bit specifies how the 12-bit conversion data returned by the ADCs is formatted into the 16-bit data which is sent to the parallel side interface. 1: Right justified signed 0: Right justified unsigned
RESSEL[0:1]	ADC Resolution Selection The RESSEL[0:1] field selects the resolution of the ADC according to the values shown below. 00: ADC set to 12-bit resolution 01: ADC set to 10-bit resolution 10: ADC set to 8-bit resolution 11: Reserved

Table 283. ADC_ACR1–8 field descriptions (continued)

Field	Description
ATBSEL[0:1]	Alternate Command Timebase Selector The ATBSEL[0:1] field selects the time information to be used as timestamp according to the values shown below. 00: Selects internally generated time base as time stamp 01: Selects imported time base 1 indicated by SRV1 bit field of EQADC_REDLCCR 10: Selects imported time base 2 indicated by SRV2 bit field of EQADC_REDLCCR 11: Reserved **Note:** This selection overrides the corresponding fields ADC0/1_TBSEL in the ADC0/1_CRs when the alternate conversion command is used.
PRE_GAIN[0:1]	ADC Pre-gain control The PRE_GAIN[0:1] controls the gain of the ADC input stage by changing the internal ADC iterations in the gain stage. The gain is selected according to the values shown below. 00: ×1 gain 01: ×2 gain 10: ×4 gain 11: Reserved

23.4.3.7 ADC0/1 Alternate Gain Registers (ADC0_AGR1–2 and ADC1_AGR1–2)

The Alternate Gain Registers (ADC0_AGRx and ADC1_AGRx, x = 1–2) contain the gain calibration constants used to fine-tune ADC conversion results for alternate configurations 1 or 2. A conversion from an ADC uses the corresponding ADC0_AGRx or ADC1_AGRx register when the conversion command (with the alternate configuration format) is written to an address in the range 0x08–0x09 of the on-chip ADC memory map. Refer to *Section 23.5.6.6: ADC calibration feature* for details about the calibration scheme used in the eQADC.

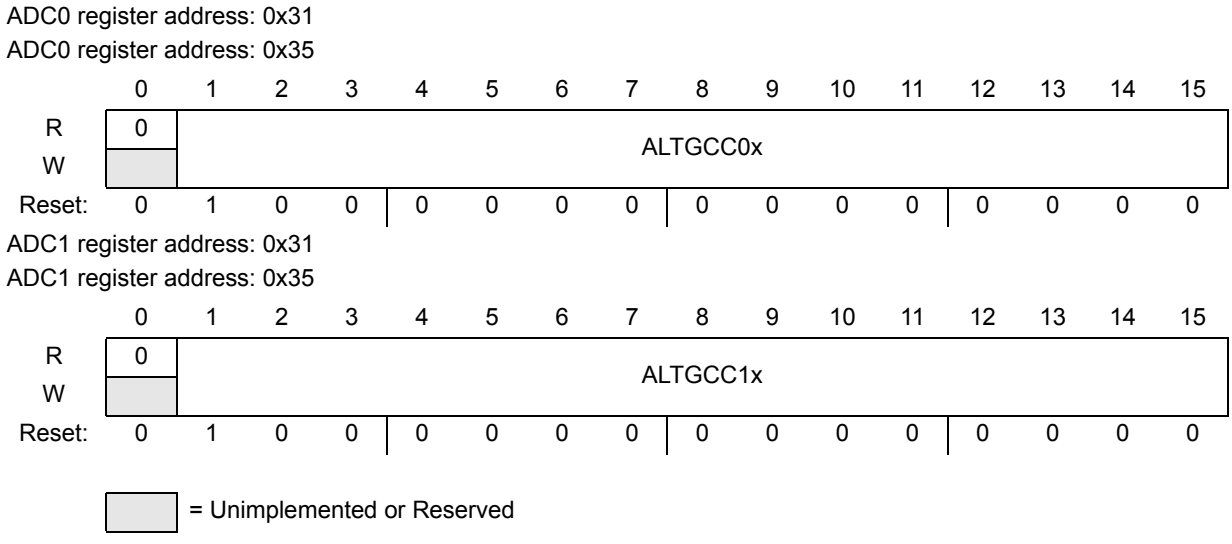


Figure 433. ADC0/1 Alternate x Gain Register (ADC0/1_AGRx, x = 1–2)

Table 284. ADC0/1_AGRx field descriptions

Field	Description
ALTGCC0/1x[0:14]	Alternate Gain Calibration Constant ALTGCC0/1x[0:14] contain the gain calibration constants used to fine-tune ADC0/1 conversion results for alternate configurations 1 and 2. The gain calibration constants are 15-bit unsigned fixed point numbers expressed in the *GCC_INT.GCC_FRAC* binary format. The integer part of the gain constants (GCC_INT) contain a single binary digit while their fractional part (GCC_FRAC) contain fourteen digits. For details about the GCC data format refer to *Section 23.5.6.6.2: MAC unit and operand data format*.

23.4.3.8 ADC0/1 Alternate Offset Register (ADC0_AOR1–2 and ADC1_AOR1–2)

The Alternate Offset Registers (ADC0_AORx and ADC1_AORx, x = 1–2) contain the offset calibration constants used to fine-tune ADC conversion results for alternate configurations 1 and 2. The offset constants are signed 14-bit integer values. Refer to *Section 23.5.6.6: ADC calibration feature* for details about the calibration scheme used in the eQADC.

ADC0 register address: 0x32
ADC0 register address: 0x36

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	ALTOCC0x													
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC1 register address: 0x32
ADC1 register address: 0x36

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	ALTOCC1x													
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 434. ADC0/1 Alternate x Offset Registers (ADC0/1_AORx, x = 1–2)

Table 285. ADC0/1_AORx field descriptions

Field	Description
ALTOCC0/1x[0:13]	Alternate Offset Calibration Constant ALTOCC0/1x[0:13] contain the offset calibration constants used to fine-tune ADC conversion results for alternate configurations 1 or 2. Negative values should be expressed using the two's complement representation.

23.4.3.9 ADC Pull Up/Down Control Register *x* (ADC_PUDCRx, *x* = 0–7)

The ADC Pull Up/Down Control Register *x* (ADC_PUDCR*x*) contains configuration bits for pull up and pull down resistors present at ADC input channels *x*, *x* = 0 to 7.

ADC0/1 register address: 0x70–0x77

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	CH_PULL*x*		°	0	PULL_STR*x*		0	0	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

Figure 435. ADC Pull Up/Down Control Register x (ADC_PUDCRx, x = 0–7)

Table 286. ADC_PUDCRx field descriptions

Field	Description
CH_PULL*x*[0:1]	Channel *x* Pull Up/Down Control bits The CH_PULL*x*[0:1] field controls the pull up/down configuration of the channel *x* as shown below. 00: No Pull resistors connected to the channel 01: Pull Up resistor connected to the channel 10: Pull Down resistor connected to the channel 11: Pull Up and Pull Down resistors connected to the channel
PULL_STR*x*[0:1]	Pull Up/Down Strength Control bits of channel *x* The PULL_STR*x*[0:1] bit field defines the strength of the channel *x* pull up or down resistors, as shown below. 00: Reserved 01: 200 Kohms pull resistor 10: 100 Kohms pull resistor 11: 5 Kohms (approx.) pull resistor(1)

1. This set is not available for CH_PULL_x = 11.

23.5 Functional description

23.5.1 Overview

The eQADC provides a parallel interface to two on-chip ADCs, a single master to single slave serial interface to an off-chip external device and a parallel side interface to an on-chip companion module, like a decimation filter. The two on-chip ADCs are architected to allow access to all the analog channels.

Initially, command data is contained in system memory in a user defined data structure which is likely to be a queue as depicted in *Figure 391*[o]. Command data is moved between the CQueues and CFIFOs by the host CPU or by the DMAC which respond to interrupt and DMA requests generated by the eQADC. The eQADC supports software and hardware triggers from other blocks or external pins to initiate transfers of commands from the multiple CFIFOs to the on-chip ADCs or to the external device.

o. Command and result data can be stored in the system memory in any user defined data structure. However, in this document it will be assumed that the data structure of choice is a queue, since it is the most likely data structure to be used and because queues are the only type of data structure supported by the DMAC.

CFIFOs can be configured in single-scan or continuous-scan mode. When a CFIFO is configured in single-scan mode, the eQADC scans the CQueue one time. The eQADC stops transferring commands from the triggered CFIFO after detecting the EOQ bit set in the last transfer. After an EOQ bit is detected, software involvement is required to rearm the CFIFO so that it can detect new trigger events.

When a CFIFO is configured for continuous-scan mode, the whole CQueue is scanned multiple times. After the detection of an asserted EOQ bit in the last command transfer, command transfers can continue or not depending on the mode of operation of the CFIFO.

CFIFO0 has a special configuration option to allow a repetitive sequence of conversion commands (streaming mode) with high priority characteristics (abort operation) or not. This feature is useful with the immediate conversion command feature that allows the immediate execution of a conversion command or a sequence of commands with critical timing even with the possibility of abortion of some current ADC conversion in progress. The aborted command is stored and executed again as soon as the critical timing commands have been finished.

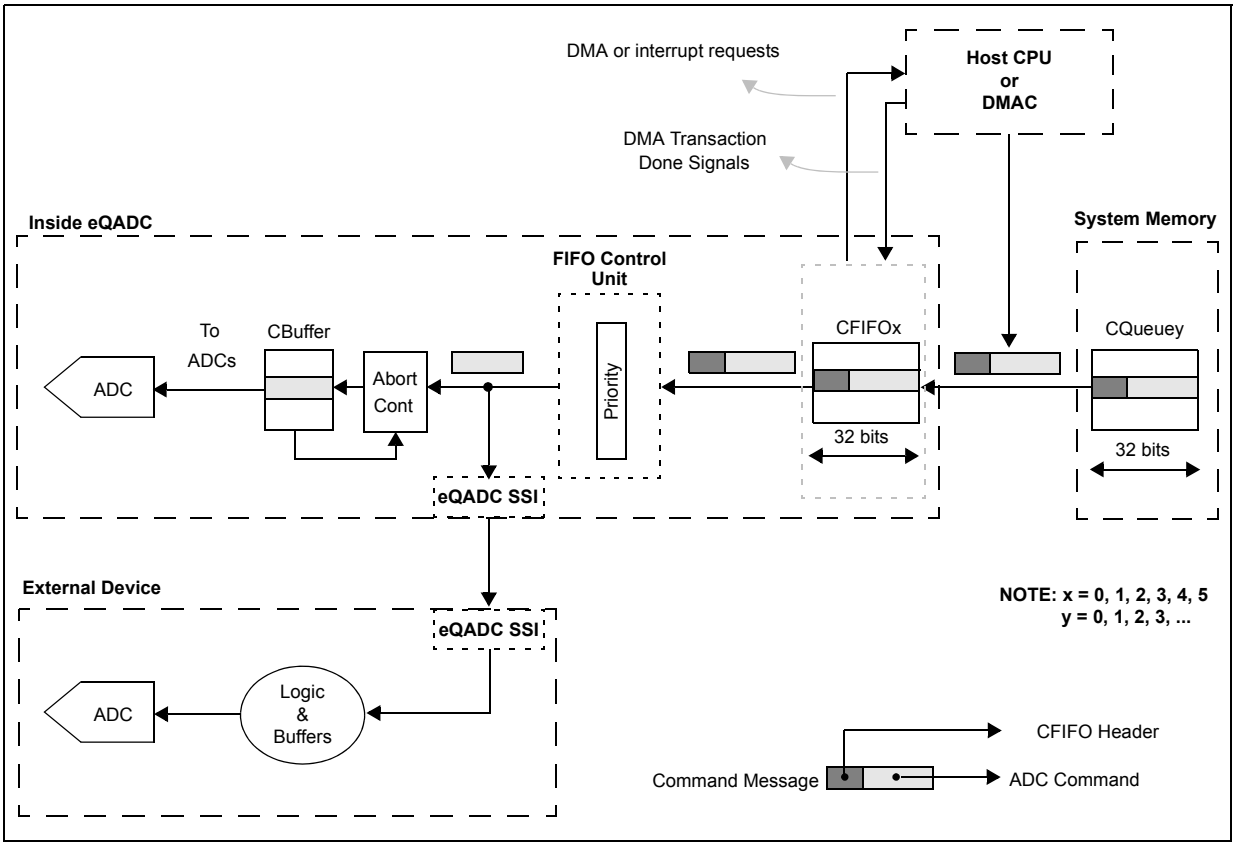
The multiple Result FIFOs (RFIFOs) can receive data from the on-chip ADCs, from an off-chip external device or from an on-chip companion module. Data from the on-chip ADCs can be routed to the side interface, processed by the on-chip companion module and then routed back through the side interface to the RFIFOs.

23.5.2 Data flow in eQADC

23.5.2.1 Overview and basic terminology

Figure 436 shows how command data flows inside the eQADC system. A Command Message is the predefined format at which command data is stored on the CQueues. A Command message has 32 bits and is composed of two parts: a CFIFO header and an ADC Command. Command messages are moved from the CQueues to the CFIFOs by the host CPU or by the DMAC as they respond to interrupt and DMA requests generated by the eQADC. The eQADC generates these requests whenever a CFIFO is not full. The *FIFO Control Unit* will only transfer to a CBuffer the ADC command part of the Command Message. Information in the CFIFO header together with the upper bit of the ADC command is used by the *FIFO Control Unit* to arbitrate which triggered CFIFO will be transferring the next command. Since command transfer through the serial interface can take significantly more time than a parallel transfer to the on-chip ADCs, command transfers for on-chip ADCs occur concurrently with the ones through the serial interface.

Figure 436. Command flow during eQADC operation



ADC commands sent to the on-chip CBuffers are executed in a first-in-first-out basis with exception when the immediate conversion command function is enabled. Three types of results can be expected: data read from an ADC register, a conversion result, or a time stamp. The order at which ADC commands sent to the external device are executed, and the type of results that can be expected depends on the architecture of that device with the exception of unsolicited data like null messages for example.

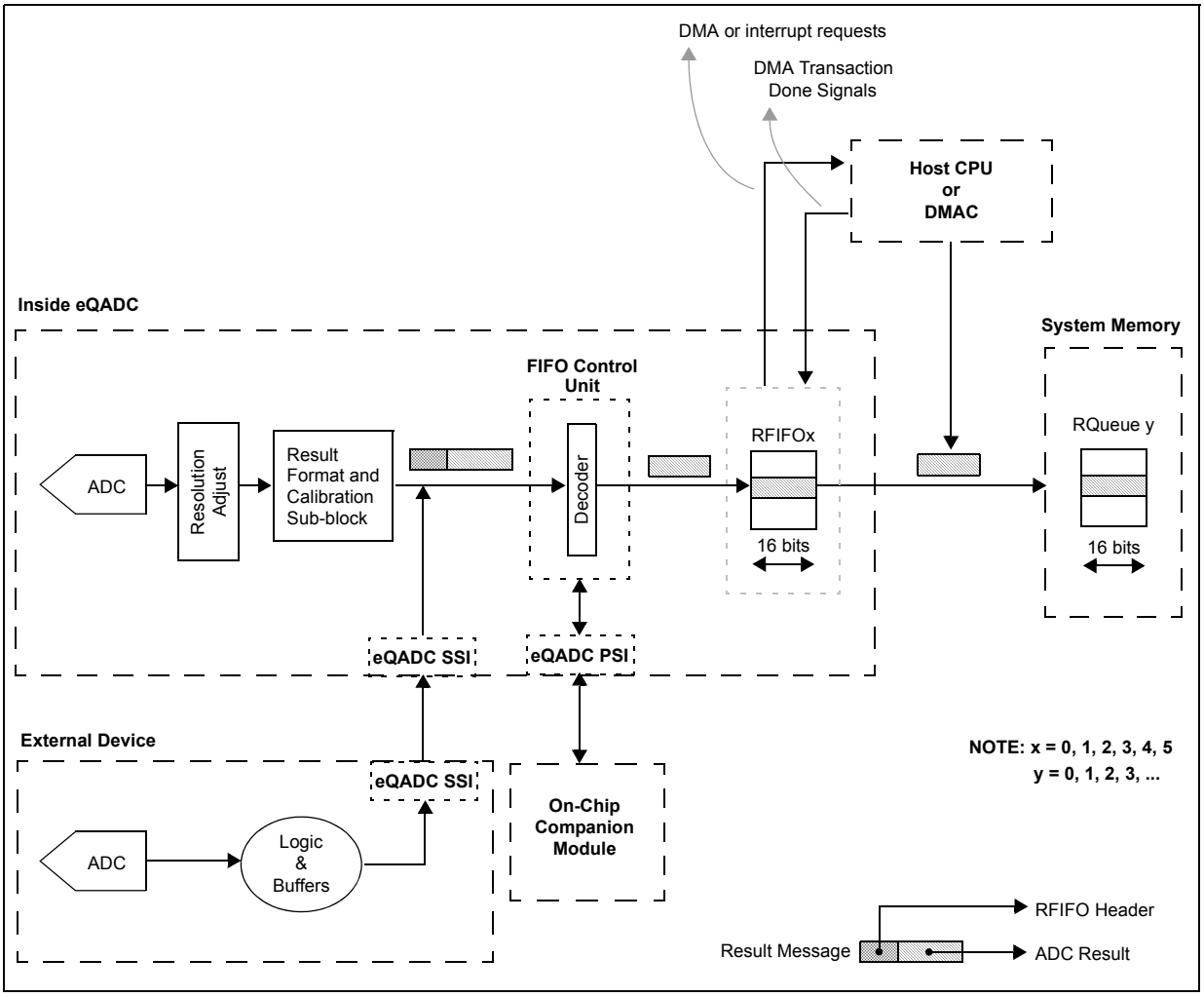
Note: While the eQADC pops commands out from a CFIFO, it also is checking the number of entries in the CFIFO and generating requests to fill it. The process of pushing and popping commands to and from a CFIFO can occur simultaneously. However, this is not true for CFIFO0 when configured to operate in streaming mode for popping.

The *FIFO Control Unit* expects all incoming results to be shaped in a predefined Result Message format. *Figure 437* shows how result data flows inside the eQADC system. Results generated on the on-chip ADCs are adjusted considering the selected resolution of the ADC and are formatted into result messages inside the *Result Format and Calibration Sub-Block*. This result message can be routed directly to one of the RFIFOs or to an on-chip companion module via the parallel side interface. After the data is processed by the companion module, it can be routed back to one of the RFIFOs via the side interface with the correct format. Results returning from the external device are already formatted into result messages and therefore bypass the *Result Format and Calibration Sub-Block*. A result message is composed of an RFIFO header and an ADC Result. The *FIFO Control Unit* decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result should be sent. Once in an RFIFO, the ADC result is moved to the corresponding RQueue by the host CPU or by the DMAC as they respond to interrupt and

DMA requests generated by the eQADC. The eQADC generates these requests whenever an RFIFO has at least one entry.

Note: While conversion results are returned, the eQADC is checking the number of entries in the RFIFO and generating requests to empty it. The process of pushing and popping ADC results to and from an RFIFO can occur simultaneously.

Figure 437. Result flow during eQADC operation



23.5.2.2 Assumptions/Requirements regarding the external device

The external device exchanges command and result data with the eQADC through the eQADC SSI interface. This section explains the minimum requirements an external device has to meet to properly interface with the eQADC. Some assumptions about the architecture of the external device are also described.

23.5.2.2.1 eQADC SSI protocol support

The external device must fully support the eQADC SSI protocol as specified in *Section 23.5.9: eQADC synchronous serial interface (SSI) sub-block*. Support for the abort feature is optional. When aborts are not supported, all command messages bound for an external CBuffer must have the ABORT_ST bit negated—see *Section 23.5.2.3.2.1:*

Command message format for external device operation.

23.5.2.2.2 Number of command buffers and result buffers

The external device should have a minimum of one and a maximum of two Command Buffers (CBuffer) to store command data sent from the eQADC. Even if more than two CBuffers are implemented in the external device, they are not recognized by the eQADC as valid destinations for commands. In this document, these two CBuffers will be referred as CBuffer2 and CBuffer3. The external device decides to which external CBuffer a command should go by decoding the upper bit (BN bit) of the ADC command—see *Section 23.5.2.3.2.1: Command message format for external device operation*. An external device that only implements one CBuffer can ignore the BN bit.

The limit of two CBuffers does not limit the number of RBuffers in the slave device.

23.5.2.2.3 Command execution and result return

Commands sent to an specific CBuffer should be executed in that order they were received.

Results generated by the execution of commands of a CBuffer should be returned in the order the CBuffer received these commands.

23.5.2.2.4 Null and result messages

The external device must be capable of correctly processing null messages as specified in *Section 23.4.2.2: eQADC Null Message Send Format Register (EQADC_NMSFR)*.

In case no valid result data is available to be sent to the eQADC, the external device must send data in the format specified in *Section 23.5.2.3.2.3: Null message format for external device operation*.

In case valid result data is available to sent to the eQADC, the external device must send data in the format specified in *Section 23.5.2.3.2.2: Result message format for external device operation*.

The BUSY0/1 fields of all messages sent from the external device to the eQADC must be correctly encoded according to the latest information on the fullness state of the CBuffers. For example, if the CBuffer2 is empty before the end of the current serial transmission and if at the end of this transmission the external device receives a command to CBuffer2, then the BUSY0 field, that is to be sent to the eQADC on the next serial transmission, should be encoded assuming that CBuffer2 has one entry.

23.5.2.3 Message format in eQADC

This section explains the command and result message formats used for on-chip ADC operation and for external device operation.

A Command Message is the predefined format at which command data is stored on the CQueues. A Command message has 32 bits and is composed of two parts: a CFIFO header and an ADC Command. The size of the CFIFO header is fixed to 6 bits, and it works as inputs to the *FIFO Control Unit*. It controls when a CQueue ends, when it pauses, if commands are sent to internal or external buffers, and if it can abort a serial data transmission. Information contained in the CFIFO header, together with the upper bit of the ADC Command is used by the *FIFO Control Unit* to arbitrate which triggered CFIFO will transfer the next command. ADC commands are encoded inside the least significant 26 bits of the command message.

A Result message is composed of an RFIFO header and an ADC Result. The *FIFO Control Unit* decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result should be sent. An ADC result is always 16 bits long.

23.5.2.3.1 Message formats for on-chip ADC operation

This section describes the Command/Result message formats used for on-chip ADC operation.

Note: Although this subsection describes how the command and result messages are formatted to communicate with the on-chip ADCs, nothing prevents the programmer from using a different format when communicating with an external device through the serial interface. Refer to Section 23.5.2.3.2: Message formats for external device operation. Apart from the BN bit, the ADC Command of a command message can be formatted to communicate to an arbitrary external device provided that the device returns an RFIFO header in the format expected by the eQADC. When the FIFO Control Unit receives return data message, it decodes the message tag field and stores the 16-bit data into the corresponding RFIFO.

23.5.2.3.1.1 Conversion command format for the standard configuration

Figure 438 describes the format for conversion commands when interfacing with the on-chip ADCs in the standard configuration. The standard configuration is selected when the lowest byte (bits 24–31) of the conversion command is set to zero. In the standard configuration, the conversion result is always routed to one of the RFIFOs. A time stamp information can be optionally requested.

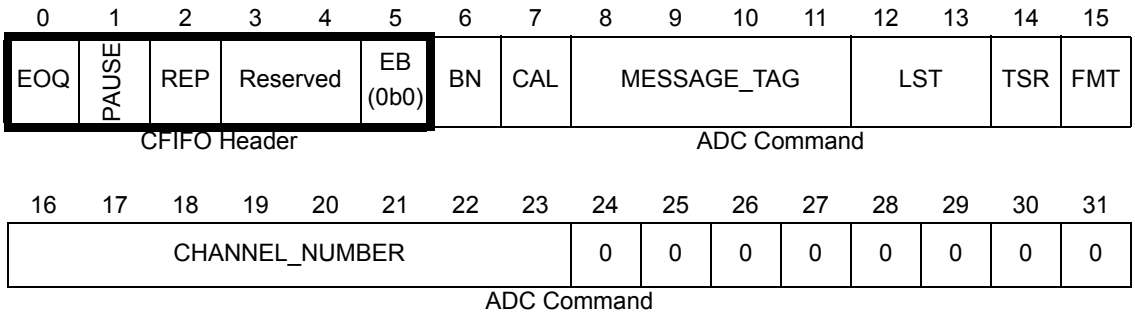


Figure 438. Conversion command format for standard configuration

Table 287. Conversion command format for standard configuration – field descriptions

Field	Description
0 EOQ	End Of Queue Bit The EOQ bit is asserted in the last command of a CQueue to indicate to the eQADC that a scan of the CQueue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to zero. Depending on the CFIFO mode of operation, the CFIFO status will also change upon the detection of an asserted EOQ bit on the last transferred command—see *Section 23.5.4.6: CFIFO scan trigger modes* for details. 1: Last entry of the CQueue. 0: Not the last entry of the CQueue. **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
1 PAUSE	Pause Bit The Pause bit allows software to create sub-queues within a CQueue. When the eQADC completes the transfer of a command with an asserted Pause bit, the CFIFO enters the WAITING FOR TRIGGER state. Refer to *Section 23.5.4.7.1: CFIFO operation status* for a description of the state transitions. The Pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 1: Enter WAITING FOR TRIGGER state after transfer of the current Command Message. 0: Do not enter WAITING FOR TRIGGER state after transfer of the current Command Message. **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
2 REP	Repeat/loop Start Point Indication Bit The REP bit is asserted in the command to indicate where is the start point of the sub-queue to be repeated when the streaming mode is enabled. The PAUSE bit indicates the end point of the sub-queue. Therefore, both can occur in the same command or in separated ones. If two or more REP bits are read before a PAUSE bit, this is an error case and the intermediary REP bits are ignored. 1: Indicates the start point of the sub-queue to be repeated. 0: It is not the start point of a loop.
3–4	Reserved
5 EB	External Buffer Bit A negated EB bit indicates that the command is sent to an internal CBuffer. 1: Not applicable to this configuration 0: Command is sent to an internal buffer
6 BN	Buffer Number Bit BN indicates which buffer the message will be stored in. Buffers 1 and 0 can be either internal or external depending on the EB bit setting. 1: Message stored in buffer 1 0: Message stored in buffer 0
7 CAL	Calibration Bit CAL indicates if the returning conversion result must be calibrated. 1: Calibrate conversion result. 0: Do not calibrate conversion result.

Table 287. Conversion command format for standard configuration – field descriptions

Field	Description
8–11 MESSAGE_TAG[0:3]	MESSAGE_TAG Field The MESSAGE_TAG allows the eQADC to separate returning results into different RFIFOs. The field values and the corresponding MESSAGE_TAG meanings are as follows: 0b0000: Result is sent to RFIFO 0 0b0001: Result is sent to RFIFO 1 0b0010: Result is sent to RFIFO 2 0b0011: Result is sent to RFIFO 3 0b0100: Result is sent to RFIFO 4 0b0101: Result is sent to RFIFO 5 0b0110–0b0111: Reserved 0b1000: Null message received 0b1001: Reserved for customer use (*) 0b1010: Reserved for customer use (*) 0b1011–0b1111: Reserved When the eQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the external device/on-chip ADC returns the result with the same MESSAGE_TAG. The eQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data. (*) These messages are treated as null messages. Therefore, they must obey the format for incoming null messages and return valid BUSY0/1 fields. Refer to *Section 23.5.2.3.2.3: Null message format for external device operation*.
12–13 LST[0:1]	Long Sampling Time These two bits determine the duration of the sampling time in ADC clock cycles. 0b00: 2 sampling cycles (ADC clock cycles) 0b01: 8 sampling cycles (ADC clock cycles) 0b10: 64 sampling cycles (ADC clock cycles) 0b11: 128 sampling cycles (ADC clock cycles)
14 TSR	Time Stamp Request TSR indicates the request for a time stamp. When TSR is asserted, the on-chip *ADC Control Logic* returns a time stamp for the current conversion command after the conversion result is sent to the RFIFOs. See *Section 23.5.6.3: Time stamp feature* for details. 1: Return conversion time stamp after the conversion result 0: Return conversion result only
15 FMT	Conversion Data Format FMT specifies to the eQADC how to format the 12-bit conversion data returned by the ADCs into the 16-bit format which is sent to the RFIFOs. See *Section 23.5.2.3.1.5: ADC result format for on-chip ADC operation* for details. 1: Right justified signed 0: Right justified unsigned
16–23 CHANNEL_NUMBER [0:7]	Channel Number Field The CHANNEL_NUMBER field selects the analog input channel. The software programs this field with the channel number corresponding to the analog input pin to be sampled and converted. See *Section 23.5.7.1: Channel assignment* for details.

23.5.2.3.1.2 Conversion command format for alternate configurations

Figure 439 describes the format for conversion commands when interfacing with the on-chip ADCs in one of the eight alternate configurations. An alternate configuration is selected when the lowest byte (bits 24–31) of the conversion command is set to a value in the range 0x08–0x0F. Each value in this range selects one of the eight alternate configurations (0x08 selects Alternate Configuration 1, 0x0F selects Alternate Configuration 8). In the alternate configurations, the conversion result can be routed to one of the RFIFOs or to the parallel side interface to communicate with an on-chip companion module. A bit field in the corresponding Alternate Configuration Control Register selects the Internal RFIFO or Parallel Side Interface as the destination for the conversion result. A time stamp information can be optionally requested.

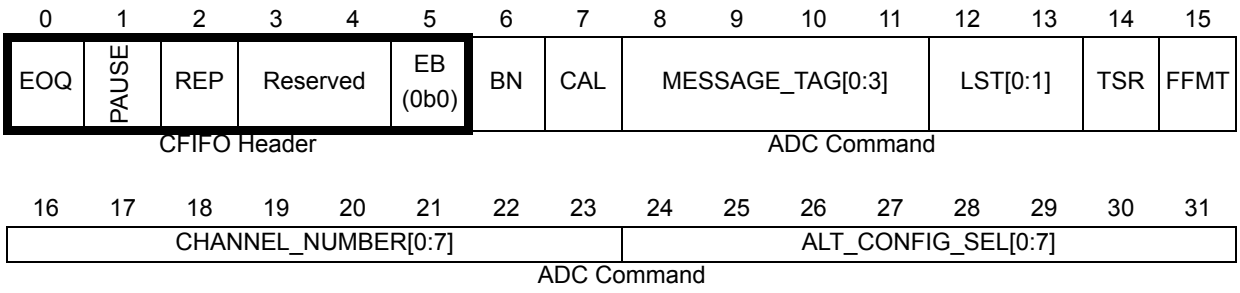


Figure 439. Conversion command format for alternate configurations

Table 288. Conversion command format for alternate configurations – field descriptions

Field	Description
0 EOQ	End Of Queue Bit The EOQ bit is asserted in the last command of a CQueue to indicate to the eQADC that a scan of the CQueue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to zero. Depending on the CFIFO mode of operation, the CFIFO status will also change upon the detection of an asserted EOQ bit on the last transferred command—see *Section 23.5.4.6: CFIFO scan trigger modes* for details. 1: Last entry of the CQueue 0: Not the last entry of the CQueue **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.

Table 288. Conversion command format for alternate configurations – field descriptions

Field	Description
1 PAUSE	Pause Bit The Pause bit allows software to create sub-queues within a CQueue. When the eQADC completes the transfer of a command with an asserted Pause bit, the CFIFO enters the WAITING FOR TRIGGER state. Refer to *Section 23.5.4.7.1: CFIFO operation status* for a description of the state transitions. The Pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 1: Enter WAITING FOR TRIGGER state after transfer of the current Command Message. 0: Do not enter WAITING FOR TRIGGER state after transfer of the current Command Message. **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
2 REP	Repeat/loop Start Point Indication Bit The REP bit is asserted in the command to indicate where is the start point of the sub-queue to be repeated when the streaming mode is enabled. The PAUSE bit indicates the end point of the sub-queue. Therefore, both can occur in the same command or in separated ones. If two or more REP bits are read before a PAUSE bit, this is an error case and the intermediary REP bits are ignored. 1: Indicates the start point of the sub-queue to be repeated. 0: It is not the start point of a loop.
3–4	Reserved
5 EB	External Buffer Bit A negated EB bit indicates that the command is sent to an internal CBuffer. 1: Not applicable to this configuration 0: Command is sent to an internal buffer
6 BN	Buffer Number Bit BN indicates which buffer the message will be stored in. Buffers 1 and 0 can be either internal or external depending on the EB bit setting. 1: Message stored in buffer 1 0: Message stored in buffer 0
7 CAL	Calibration Bit CAL indicates if the returning conversion result must be calibrated. 1: Calibrate conversion result. 0: Do not calibrate conversion result.

Table 288. Conversion command format for alternate configurations – field descriptions

Field	Description
8–11 MESSAGE_TAG[0:3]	MESSAGE_TAG Field The MESSAGE_TAG allows the eQADC to separate returning results into different RFIFOs. The field values and the corresponding MESSAGE_TAG meanings are as follows: 0b0000: Result is sent to RFIFO 0 0b0001: Result is sent to RFIFO 1 0b0010: Result is sent to RFIFO 2 0b0011: Result is sent to RFIFO 3 0b0100: Result is sent to RFIFO 4 0b0101: Result is sent to RFIFO 5 0b0110–0b0111: Reserved 0b1000: Null message received 0b1001: Reserved for customer use (*) 0b1010: Reserved for customer use (*) 0b1011–0b1111: Reserved When the eQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the external device/on-chip ADC returns the result with the same MESSAGE_TAG. The eQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data. (*) These messages are treated as null messages. Therefore, they must obey the format for incoming null messages and return valid BUSY0/1 fields. Refer to *Section 23.5.2.3.2.3: Null message format for external device operation*.
12–13 LST[0:1]	Long Sampling Time These two bits determine the duration of the sampling time in ADC clock cycles. 0b00: 2 sampling cycles (ADC clock cycles) 0b01: 8 sampling cycles (ADC clock cycles) 0b10: 64 sampling cycles (ADC clock cycles) 0b11: 128 sampling cycles (ADC clock cycles)
14 TSR	Time Stamp Request TSR indicates the request for a time stamp. When TSR is asserted, the on-chip *ADC Control Logic* returns a time stamp for the current conversion command after the conversion result is sent to the RFIFOs. See *Section 23.5.6.3: Time stamp feature* for details. 1: Return conversion time stamp after the conversion result. 0: Return conversion result only.

Table 288. Conversion command format for alternate configurations – field descriptions

Field	Description
15 FFMT	Flush or Format The function of this bit depends on the DEST field of the Alternate Configuration Control Register. If DEST is equal to 0b000, then FFMT defines the format in which the 12-bit conversion result are stored in the RFIFOs. If DEST is not equal to 0b000, then the FFMT bit is used to send a flush (soft-reset) signal through the parallel side interface to the companion module addressed by the DEST field. In case DEST is not equal to 0b000, the FMTA bit in the Alternate Configuration Control register is used to define the conversion result format. 1: Conversion Result Format set to right justified signed if DEST is equal to 0b000. A flush signal is sent through the side interface if DEST is not equal to 0b000. 0: Conversion Result Format set to right justified unsigned if DEST is equal to 0b000. No flush signal is sent through the side interface if DEST is not equal to 0b000. **Note:** The flush signal can be asserted along with a valid conversion result. In this case the companion module should execute the software-reset first and then consider the conversion result as a valid data for the filtering algorithm.
16–23 CHANNEL_NUMBER [0:7]	Channel Number Field The CHANNEL_NUMBER field selects the analog input channel. The software programs this field with the channel number corresponding to the analog input pin to be sampled and converted. See *Section 23.5.7.1: Channel assignment* for details.
24–31 ALT_CONFIG_SEL [0:7]	Alternate Configuration Selection This field selects one of the alternate configurations according to the following values: 0x08: Alternate configuration 1 0x09: Alternate configuration 2 0x0A: Alternate configuration 3 0x0B: Alternate configuration 4 0x0C: Alternate configuration 5 0x0D: Alternate configuration 6 0x0E: Alternate configuration 7 0x0F: Alternate configuration 8

23.5.2.3.1.3 Write configuration command format for on-chip ADC operation

Figure 440 describes the command message format for a write configuration command when interfacing with the on-chip ADCs. A write configuration command is used to set the control registers of the on-chip ADCs. No conversion data will be returned for a write configuration command. Write configuration commands are differentiated from read configuration commands by a negated R/W bit.

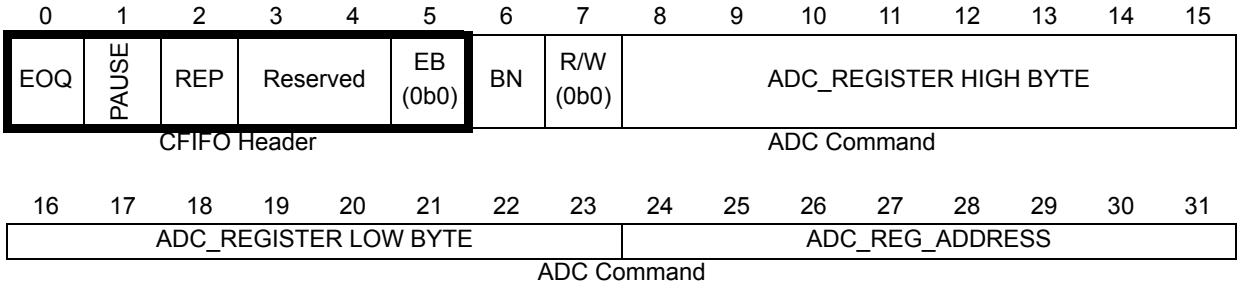


Figure 440. Write configuration command format for on-chip ADC operation

Table 289. Write configuration command format for on-chip ADC operation – field descriptions

Field	Description
0 EOQ	End Of Queue Bit The EOQ bit is asserted in the last command of a CQueue to indicate to the eQADC that a scan of the CQueue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to zero. Depending on the CFIFO mode of operation, the CFIFO status will also change upon the detection of an asserted EOQ bit on the last transferred command—see *Section 23.5.4.6: CFIFO scan trigger modes* for details. 1: Last entry of the CQueue. 0: Not the last entry of the CQueue. **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
1 PAUSE	Pause Bit The Pause bit allows software to create sub-queues within a CQueue. When the eQADC completes the transfer of a command with an asserted Pause bit, the CFIFO enters the WAITING FOR TRIGGER state. Refer to *Section 23.5.4.7.1: CFIFO operation status* for a description of the state transitions. The Pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 1: Enter WAITING FOR TRIGGER state after transfer of the current Command Message. 0: Do not enter WAITING FOR TRIGGER state after transfer of the current Command Message. **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
2 REP	Repeat/loop Start Point Indication Bit The REP bit is asserted in the command to indicate where is the start point of the sub-queue to be repeated when the streaming mode is enabled. The PAUSE bit indicates the end point of the sub-queue. Therefore, both can occur in the same command or in separated ones. If two or more REP bits are read before a PAUSE bit, this is an error case and the intermediary REP bits are ignored. 1: Indicates the start point of the sub-queue to be repeated. 0: It is not the start point of a loop.

Table 289. Write configuration command format for on-chip ADC operation – field descriptions

Field	Description
3–4	Reserved
5 EB	External Buffer Bit A negated EB bit indicates that the command is sent to an internal CBuffer. 1: Not applicable to this configuration 0: Command is sent to an internal buffer
6 BN	Buffer Number Bit BN indicates which buffer the message will be stored in. Buffers 1 and 0 can be either internal or external depending on the EB bit setting. 1: Message stored in buffer 1. 0: Message stored in buffer 0.
7 R/W	Read/Write bit A negated R/W indicates a write configuration command. 0: Write
8–15 ADC_REGISTER_HIGH_BYTE [0:7]	ADC Register High Byte Field REGISTER_HIGH_BYTE is the value to be written into the most significant 8 bits of control/configuration register when the R/W bit is negated.
16–23 ADC_REGISTER_LOW_BYTE [0:7]	ADC Register Low Byte Field REGISTER_LOW_BYTE is the value to be written into the least significant 8 bits of a control/configuration register when the R/W bit is negated.
24–31 ADC_REG_ADDRESS [0:7]	ADC Register Address The ADC_REG_ADDRESS field selects a register on the ADC register set to be written or read. Only half-word addresses can be used.

23.5.2.3.1.4 Read configuration command format for on-chip ADC operation

Figure 441 describes the command message format for a read configuration command when interfacing with the on-chip ADCs. A read configuration command is used to read the contents of the on-chip ADC registers which are only accessible via command messages. Read configuration commands are differentiated from write configuration commands by an asserted R/W bit.

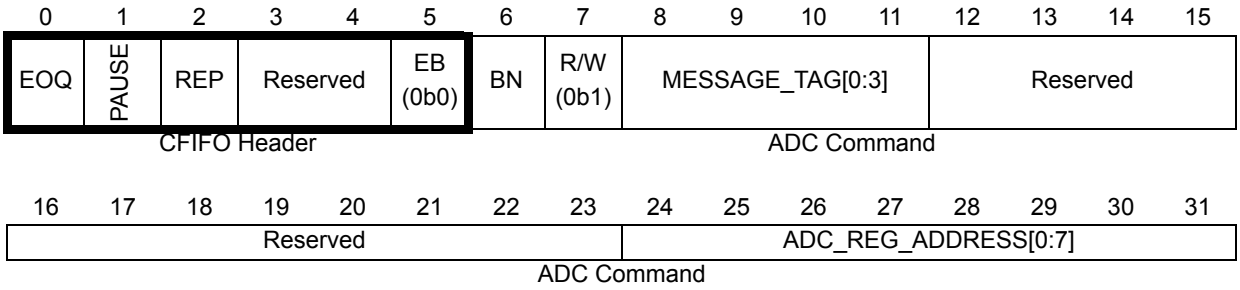


Figure 441. Read configuration command format for on-chip ADC operation

Table 290. Read configuration command format for on-chip ADC operation – field descriptions

Field	Description
0 EOQ	End Of Queue Bit The EOQ bit is asserted in the last command of a CQueue to indicate to the eQADC that a scan of the CQueue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to zero. Depending on the CFIFO mode of operation, the CFIFO status will also change upon the detection of an asserted EOQ bit on the last transferred command—see *Section 23.5.4.6: CFIFO scan trigger modes* for details. 1: Last entry of the CQueue 0: Not the last entry of the CQueue **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
1 PAUSE	Pause Bit The Pause bit allows software to create sub-queues within a CQueue. When the eQADC completes the transfer of a command with an asserted Pause bit, the CFIFO enters the WAITING FOR TRIGGER state. Refer to *Section 23.5.4.7.1: CFIFO operation status* for a description of the state transitions. The Pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 1: Enter WAITING FOR TRIGGER state after transfer of the current Command Message. 0: Do not enter WAITING FOR TRIGGER state after transfer of the current Command Message. **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
2 REP	Repeat/loop Start Point Indication Bit The REP bit is asserted in the command to indicate where is the start point of the sub-queue to be repeated when the streaming mode is enabled. The PAUSE bit indicates the end point of the sub-queue. Therefore, both can occur in the same command or in separated ones. If two or more REP bits are read before a PAUSE bit, this is an error case and the intermediary REP bits are ignored. 1: Indicates the start point of the sub-queue to be repeated. 0: It is not the start point of a loop.
3–4	Reserved
5 EB	External Buffer Bit A negated EB bit indicates that the command is sent to an internal CBuffer. 1: Not applicable to this configuration 0: Command is sent to an internal buffer
6 BN	Buffer Number Bit BN indicates which buffer the message will be stored in. Buffers 1 and 0 can be either internal or external depending on the EB bit setting. 1: Message stored in buffer 1 0: Message stored in buffer 0

Table 290. Read configuration command format for on-chip ADC operation – field descriptions

Field	Description
7 R/W	Read/Write bit An asserted R/W bit indicates a read configuration command. A negated R/W bit indicates a write configuration command. 1: Read 0: Write
8–11 MESSAGE_TAG [0:3]	MESSAGE_TAG Field The MESSAGE_TAG allows the eQADC to separate returning results into different RFIFOs. The field values and the corresponding MESSAGE_TAG meanings are as follows: 0b0000: Result is sent to RFIFO 0 0b0001: Result is sent to RFIFO 1 0b0010: Result is sent to RFIFO 2 0b0011: Result is sent to RFIFO 3 0b0100: Result is sent to RFIFO 4 0b0101: Result is sent to RFIFO 5 0b0110–0b0111: Reserved 0b1000: Null message received 0b1001: Reserved for customer use (*) 0b1010: Reserved for customer use (*) 0b1011–0b1111: Reserved When the eQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the external device/on-chip ADC returns the result with the same MESSAGE_TAG. The eQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data. (*) These messages are treated as null messages. Therefore, they must obey the format for incoming null messages and return valid BUSY0/1 fields. Refer to *Section 23.5.2.3.2.3: Null message format for external device operation*.
12–23	Reserved
24–31 ADC_REG_ADDRESS [0:7]	ADC Register Address The ADC_REG_ADDRESS field selects a register on the ADC register set to be written or read. Only half-word addresses can be used.

23.5.2.3.1.5 ADC result format for on-chip ADC operation

When the *FIFO Control Unit* receives a return data message, it decodes the MESSAGE_TAG field and stores the 16-bit data into the appropriate RFIFO. This section describes the *ADC result* portion of the *result message* returned by the on-chip ADCs. The 16-bit data stored in the RFIFOs can be:

- Data read from an ADC register with a read configuration command—In this case, the stored 16-bit data corresponds to the contents of the ADC register that was read.
- A time stamp—In this case, the stored 16-bit data is the value of the time base counter latched when the eQADC detects the end of the analog input voltage sampling. For

details see *Section 23.5.6.3: Time stamp feature*.

- A conversion result, coming directly from the ADCs—In this case, the stored 16-bit data contains a right justified 14-bit result data. The conversion result can be calibrated or not depending on the status of CAL bit in the command that requested the conversion[p]. When the CAL bit is negated, this 14-bit data is obtained by executing a 2-bit left-shift on the 12-bit data resultant from the resolution adjustment on the 8 or 10 or 12-bit data received from the ADC. The resolution adjustment consists of changing the conversion result input from 8, 10 or 12 bits right aligned to a 12-bit word left aligned—refer to *Section 23.5.6.5: ADC resolution selection feature* for details. When the CAL bit is asserted, this 14-bit data is the result of the calculations performed in the eQADC MAC unit using the 12-bit data result of the resolution adjustment and the calibration constants GCC and OCC, or ALTGCC and ALTOCC—refer to *Section 23.5.6.6: ADC calibration feature* for details. Then, this 14-bit data is further formatted into a 16-bit format according to the status of the FMT bit in conversion command of the standard configuration or FFMT bit in the conversion command of the alternate configurations[q]. When FMT/FFMT is asserted, the 14-bit result data is reformatted to look as if it was measured against an imaginary ground at VREF/2 (the MSB bit of the 14-bit result is inverted), and is sign-extended to a 16-bit format as in *Figure 442*. When FMT/FFMT is negated, the eQADC zero-extends the 14-bit result data to a 16-bit format as in *Figure 443*. Correspondence between the analog voltage in a channel and the calculated digital values is shown in *Table 292*.

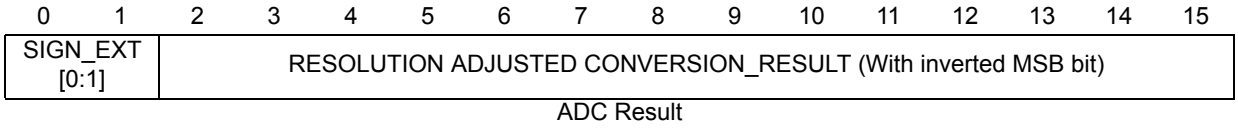


Figure 442. ADC result format when FMT = 1 (Right Justified Signed)

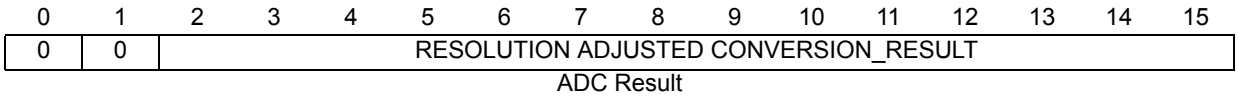


Figure 443. ADC result format when FMT = 0 (Right Justified Unsigned)

Table 291. ADC result format when FMT = 0 (Right Justified Unsigned) – field descriptions

Field	Description
0–1 SIGN_EXT[0:1]	Sign Extension field SIGN_EXT only has meaning when FMT is asserted. SIGN_EXT is 0b00 when CONVERSION_RESULT is positive, and 0b11 when CONVERSION_RESULT is negative.

p. In case the conversion result is routed through an on-chip DSP via side interface, the calibration is applied before the data is sent to the DSP.

q. For simplicity, the following text will refer to FMT only, but when using alternate configurations, refer to *Section 23.5.2.3.1.2: Conversion command format for alternate configurations*.

Table 291. ADC result format when FMT = 0 (Right Justified Unsigned) – field descriptions

Field	Description
CONVERSION_RESULT[0:13]	Conversion Result field CONVERSION_RESULT is a digital value corresponding to the analog input voltage in a channel when the conversion command was initiated. The two's complement representation is used to express negative values.

Table 292. Correspondence between analog voltages and digital values(1)(2)

Conversion type	Voltage level on channel (V)(3)	Corresponding result returned by the ADC			16-bit result sent to RFIFOs	
		8-bit conversion	10-bit conversion	12-bit conversion	(FMT = 0)(4)	(FMT = 1)(4)
Single-ended	5.12	—	—	0xFFF	0x3FFC	0x1FFC
		—	0x3FF	—	0x3FF0	0x1FF0
		0xFF	—	—	0x3FC0	0x1FC0
	5.12 - LSB	—	—	0xFFF	0x3FFC	0x1FFC
		—	0x3FF	—	0x3FF0	0x1FF0
		0xFF	—	—	0x3FC0	0x1FC0
	...	...	...	...	...	...
	2.56	—	—	0x800	0x2000	0x0000
		—	0x200	—	0x2000	0x0000
		0x80	—	—	0x2000	0x0000
	...	...	...	...	...	...
	1 LSB	—	—	0x001	0x0004	0xE004
		—	0x001	—	0x0010	0xE010
		0x01	—	—	0x0040	0xE040
	0	0x00	0x000	0x000	0x0000	0xE000

Table 292. Correspondence between analog voltages and digital values(1)(2) (continued)

Conversion type	Voltage level on channel (V)(3)	Corresponding result returned by the ADC			16-bit result sent to RFIFOs	
		8-bit conversion	10-bit conversion	12-bit conversion	(FMT = 0)(4)	(FMT = 1)(4)
Differential	2.56	—	—	0xFFF	0x3FFC	0x1FFC
		—	0x3FF	—	0x3FF0	0x1FF0
		0xFF	—	—	0x3FC0	0x1FC0
	2.56 - LSB	—	—	0xFFF	0x3FFC	0x1FFC
		—	0x3FF	—	0x3FF0	0x1FF0
		0xFF	—	—	0x3FC0	0x1FC0
	...	...	...	...	...	...
	0	—	—	0x800	0x2000	0x0000
		—	0x200	—	0x2000	0x0000
		0x80	—	—	0x2000	0x0000
	...	...	...	...	...	...
	2.56 - LSB	—	—	0x001	0x0004	0xE004
		—	0x001	—	0x0010	0xE010
		0x01	—	—	0x0040	0xE040
	–2.56	0x00	0x000	0x000	0x0000	0xE000

1. VREF = VRH – VRL = 5.12 V. Resulting in one 12-bit count (LSB) = 1.25 mV.
2. The two's complement representation is used to express negative values.
3. The maximum differential voltage (difference between DAN+ to DAN-) is 2.5V and the common mode voltage must be (VRH-VRL)/2.
4. Assuming uncalibrated conversion results.

23.5.2.3.2 Message formats for external device operation

This section describes the Command Messages, Data Messages, and Null Messages formats used for external device operation.

23.5.2.3.2.1 Command message format for external device operation

Figure 444 describes the command message format for external device operation. Command message formats for on-chip operation and for external device operation share the same CFIFO header format. However, there are no limitations regarding the format an ADC Command used to communicate to an arbitrary external device. Only the upper bit of an ADC Command has a fixed format (BN field) to indicate to the *FIFO Control Unit*/external device to which CBuffer the corresponding command should be sent. The remaining 25 bits can be anything decodable by the external device. Only the ADC Command portion of a command message is transferred to the external device.

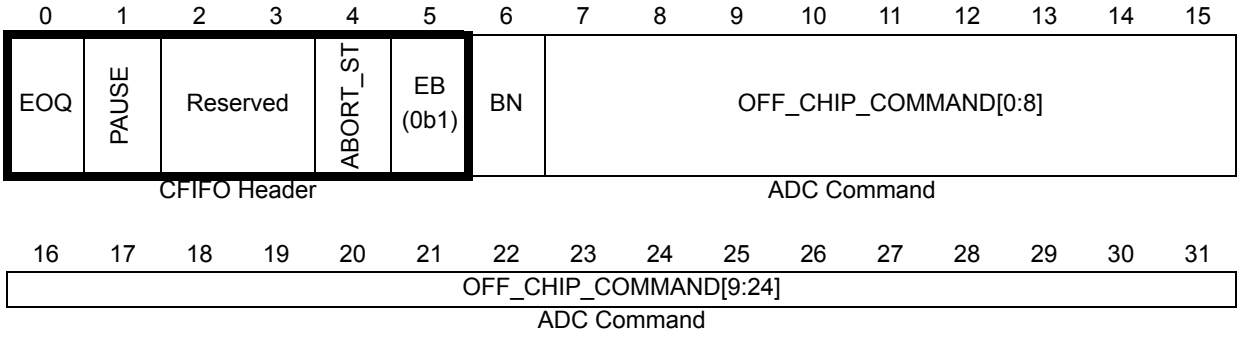


Figure 444. Command message format for external device operation

Table 293. Command message format for external device operation – field descriptions

Field	Description
0 EOQ	End Of Queue Bit The EOQ bit is asserted in the last command of a CQueue to indicate to the eQADC that a scan of the CQueue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to zero. Depending on the CFIFO mode of operation, the CFIFO status will also change upon the detection of an asserted EOQ bit on the last transferred command—see *Section 23.5.4.6: CFIFO scan trigger modes* for details. 1: Last entry of the CQueue. 0: Not the last entry of the CQueue. **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
1 PAUSE	Pause Bit The Pause bit allows software to create sub-queues within a CQueue. When the eQADC completes the transfer of a command with an asserted Pause bit, the CFIFO enters the WAITING FOR TRIGGER state. Refer to *Section 23.5.4.7.1: CFIFO operation status* for a description of the state transitions. The Pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 1: Enter WAITING FOR TRIGGER state after transfer of the current Command Message. 0: Do not enter WAITING FOR TRIGGER state after transfer of the current Command Message. **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
2–3	Reserved

Table 293. Command message format for external device operation – field descriptions

Field	Description
4 ABORT_ST	ABORT Serial Transmission Bit ABORT_ST indicates whether an on-going serial transmission should be aborted or not. All CFIFOs can abort null message transmissions when triggered but only CFIFO0 can abort command transmissions of lower priority CFIFOs. For more on serial transmission aborts see *Section 23.5.4.3: CFIFO common prioritization and command transfer*. 1: Abort current serial transmission. 0: Do not abort current serial transmission.
5 EB	External Buffer Bit An asserted EB bit indicates that the command is sent to an external CBuffer. 1: Command is sent to an external CBuffer 0: Not applicable to this configuration
6 BN	Buffer Number Bit BN indicates which buffer the message will be stored in. Buffers 1 and 0 can be either internal or external depending on the EB bit setting. 1: Message stored in buffer 1 0: Message stored in buffer 0
7–31 OFF_CHIP_COMMAND[0:24]	OFF-CHIP COMMAND Field The OFF_CHIP_COMMAND field can be anything decodable by the external device. It is 25 bits long and it is transferred together with the BN bit to the external device when the CFIFO is triggered. Refer to *Section 23.5.2.3.1.1: Conversion command format for the standard configuration* for a description of the command message used when interfacing with the on-chip ADCs.

23.5.2.3.2.2 Result message format for external device operation

Data is returned from the ADCs in the form of Result Messages. A result message is composed of an RFIFO header and an ADC Result. The *FIFO Control Unit* decodes the information contained in the RFIFO header and sends the contents of the ADC Result to the appropriate RFIFO. Only data stored on the ADC_RESULT field is stored in the RFIFOs/RQueues. The ADC result of any received message with a Null Data Message Tag will be ignored. The format of a Result Message returned from the external device is shown in *Figure 445*. It is 26 bits long, and is composed of a MESSAGE_TAG field, information about the status of the CBuffers (BUSY fields), and result data. The BUSY fields are needed to inform the eQADC about when it is appropriate to transfer commands to the external CBuffers.

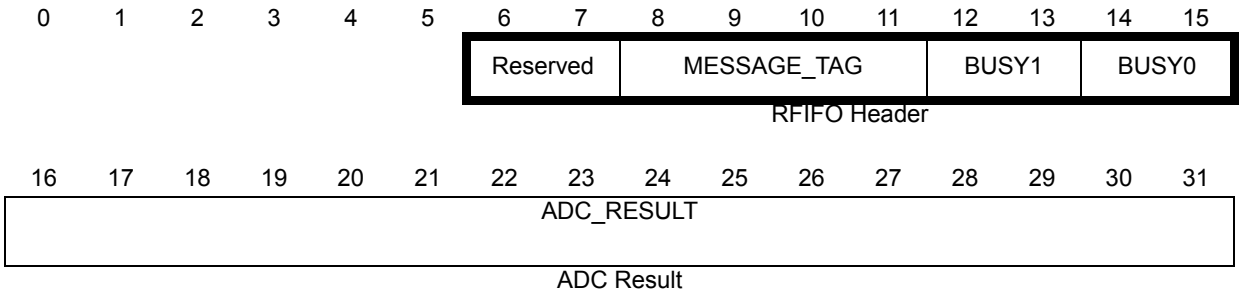


Figure 445. Result message format for external device operation

Table 294. Result message format for external device operation – field descriptions

Field	Description
8–11 MESSAGE_TAG[0:3]	MESSAGE_TAG Field The MESSAGE_TAG allows the eQADC to separate returning results into different RFIFOs. The field values and the corresponding MESSAGE_TAG meanings are as follows: 0b0000: Result is sent to RFIFO 0 0b0001: Result is sent to RFIFO 1 0b0010: Result is sent to RFIFO 2 0b0011: Result is sent to RFIFO 3 0b0100: Result is sent to RFIFO 4 0b0101: Result is sent to RFIFO 5 0b0110–0b0111: Reserved 0b1000: Null message received 0b1001: Reserved for customer use (*) 0b1010: Reserved for customer use (*) 0b1011–0b1111: Reserved When the eQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the external device/on-chip ADC returns the result with the same MESSAGE_TAG. The eQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data. (*) These messages are treated as null messages. Therefore, they must obey the format for incoming null messages and return valid BUSY0/1 fields. Refer to *Section 23.5.2.3.2.3: Null message format for external device operation*.
12–15 BUSY1[0:1] BUSY0[0:1]	BUSY Status field The BUSY fields indicate if the external device can receive more commands. The following values show the command BUFFERx BUSY status encoding of these two bits (BUSYx[0:1]): 0b00: Send available commands - CBuffer is empty 0b01: Send available commands 0b10: Send available commands 0b11: Do not send commands When an external device cannot accept any more new commands, it must set BUSYx to a value indicating “Do not send commands” in the returning message. The BUSY fields of values 0b01 and 0b10 can be freely encoded by the external device to allow visibility of the status of the external CBuffers for debug, they could indicate the number of entries in a external CBuffer for example. **Note:** After reset, the eQADC always assumes that the external CBuffers are full and cannot receive commands.
16–31 ADC_RESULT [0:15]	ADC RESULT Field ADC_RESULT is the result data received from the external device or on-chip ADC. This can be the result of a conversion command, data requested via a read configuration command, or time stamp value. The ADC_RESULT of any incoming message with a Null Message tag will be ignored. When the MESSAGE_TAG is for an RFIFO, the eQADC extracts the 16-bit ADC_RESULT from the raw message and stores it into the appropriate RFIFO.

23.5.2.3.2.3 Null message format for external device operation

Null Messages are only transferred through the serial interface to allow results and unsolicited control data, like the status of the external CBuffers, to return when there are no more commands pending to transfer. Null Messages are only transmitted when serial transmissions from the eQADC SSI are enabled (see ESSIE field in the eQADC Module Configuration Register (EQADC_MCR)), and when one of the following conditions apply:

1. there are no triggered CFIFOs with commands bound for external CBuffers, or;
2. there are triggered CFIFOs with commands bound for external CBuffers but the external CBuffers are full. The eQADC detected returning BUSYx fields indicating “Do not send commands.”

Figure 446 illustrates the null message send format. When the eQADC transfers a null message, it directly shifts out the 26-bit data content inside the eQADC Null Message Send Format Register (EQADC_NMSFR). The register must be programmed with the null message send format of the external device.

Figure 447 illustrates the null message receive format. It has the same fields found in a Result Message with the exception that the ADC result is not used. Refer to *Section 23.5.2.3.2.2: Result message format for external device operation* for more information. The MESSAGE_TAG field must be set to the Null Message tag (0b1000). The eQADC does not store into an RFIFO any incoming message with a Null Message tag.

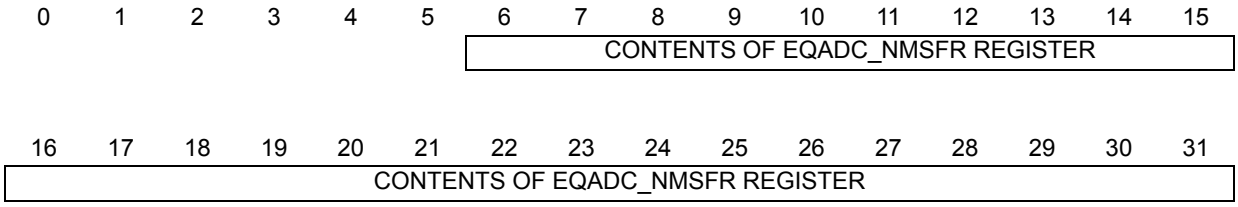


Figure 446. Null message send format for external device operation

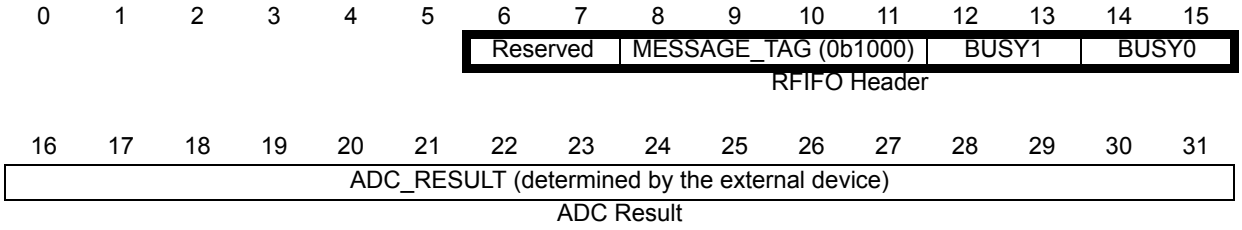


Figure 447. Null message receive format for external device operation

Table 295. Null message format for external device operation – field descriptions

Field	Description
8–11 MESSAGE_TAG [0:3]	MESSAGE_TAG Field The MESSAGE_TAG allows the eQADC to separate returning results into different RFIFOs. The field values and the corresponding MESSAGE_TAG meanings are as follows: 0b0000: Result is sent to RFIFO 0 0b0001: Result is sent to RFIFO 1 0b0010: Result is sent to RFIFO 2 0b0011: Result is sent to RFIFO 3 0b0100: Result is sent to RFIFO 4 0b0101: Result is sent to RFIFO 5 0b0110–0b0111: Reserved 0b1000: Null message received 0b1001: Reserved for customer use (*) 0b1010: Reserved for customer use (*) 0b1011–0b1111: Reserved When the eQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the external device/on-chip ADC returns the result with the same MESSAGE_TAG. The eQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data. (*) These messages are treated as null messages. Therefore, they must obey the format for incoming null messages and return valid BUSY0/1 fields. Refer to *Section 23.5.2.3.2.3: Null message format for external device operation*.
12–15 BUSY1[0:1] BUSY0[0:1]	BUSY Status field The BUSY fields indicate if the external device can receive more commands. The following values show the command BUFFERx BUSY status encoding of these two bits (BUSYx[0:1]): 0b00: Send available commands - CBuffer is empty 0b01: Send available commands 0b10: Send available commands 0b11: Do not send commands When an external device cannot accept any more new commands, it must set BUSYx to a value indicating “Do not send commands” in the returning message. The BUSY fields of values 0b01 and 0b10 can be freely encoded by the external device to allow visibility of the status of the external CBuffers for debug, they could indicate the number of entries in an external CBuffer for example. **Note:** After reset, the eQADC always assumes that the external CBuffers are full and cannot receive commands.
16–31 ADC_RESULT [0:15]	ADC RESULT Field ADC_RESULT is the result data received from the external device or on-chip ADC. This can be the result of a conversion command, data requested via a read configuration command, or time stamp value. The ADC_RESULT of any incoming message with a Null Message tag will be ignored. When the MESSAGE_TAG is for an RFIFO, the eQADC extracts the 16-bit ADC_RESULT from the raw message and stores it into the appropriate RFIFO.

23.5.3 Command/Result queues

The Command and Result queues (CQueues and RQueues) are actually part of the eQADC system although they are not hardware implemented inside the eQADC. Each CQueue entry is a 32-bit Command Message. The last entry of a CQueue has the EOQ bit asserted to indicate that it is the last entry of the CQueue. RQueue entry is a 16-bit data result.

See *Section 23.5.2.1: Overview and basic terminology* for a description of the message formats and their flow in eQADC.

Refer to *Section 23.6.5: CQueue and RQueues usage* for examples of how CQueues and RQueues can be used.

23.5.4 eQADC command FIFOs

23.5.4.1 CFIFO basic functionality

There are six prioritized CFIFOs located in the eQADC. Each CFIFO is four entries deep, except CFIFO0 that can be configured to eight entries deep in extended mode, and each CFIFO entry is 32 bits long. A CFIFO serves as a temporary storage location for the command messages stored on the CQueues in the system memory. When a CFIFO is not full, the eQADC sets the corresponding CFFF bit in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR). If CFFE is asserted in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR), the eQADC generates requests for more commands from a CQueue. An interrupt request, served by the host CPU, is generated when CFFS is negated, and a DMA request, served by the DMAC, is generated when CFFS is asserted. The host CPU or the DMAC respond to these requests by writing to the eQADC CFIFO Push Registers (EQADC_CFPR) to fill the CFIFO.

Note: The DMAC should be configured to write a single command (32-bit data) to the CFIFO push registers for every asserted DMA request it acknowledges. Refer to Section 23.6.2: eQADC/DMAC interface for DMAC configuration guidelines.

CFIFO0 can be configured to work in an alternative way called Streaming Mode. This mode is very different from the mode described here because it maintains some stored commands to execute them several times in sequence and in loop.

Only whole words must be written to EQADC_CFPR. Writing half-words or bytes to EQADC_CFPR will still push the whole 32-bit CF_PUSH field into the corresponding CFIFO, but undefined data will fill the areas of CF_PUSH that were not specifically designated as target locations for writing.

Figure 448 describes the important components in the CFIFO. Each CFIFO is implemented as a circular set of registers to avoid the need to move all entries at each push/pop operation. The Push Next Data Pointer points to the next available CFIFO location for storing data written into the eQADC Command FIFO Push Register. The Transfer Next Data Pointer points to the next entry to be removed from CFIFOx when it completes a transfer. The *CFIFO Transfer Counter Control Logic* counts the number of entries in the CFIFO and generates DMA or interrupt requests to fill the CFIFO. TNXTPTR in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) indicates the index of the entry that is currently being addressed by the Transfer Next Data Pointer, and CFCTR, in the same register, provides the number of entries stored in the CFIFO. Using TNXTPTR and CFCTR, the absolute addresses for the entries indicated by the Transfer Next Data Pointer and by the Push Next Data Pointer can be calculated using the following formulas:

Transfer Next Data Pointer Address = CFIFOx_BASE_ADDRESS + TNXTPTRx*4

Push Next Data Pointer Address = CFIFOx_BASE_ADDRESS + [(TNXTPTRx+CFCTRx) mod CFIFO_DEPTH] * 4

where

- *a mod b* returns the remainder of the division of *a* by *b*.
- CFIFOx_BASE_ADDRESS is the smallest memory mapped address allocated to a CFIFOx entry.
- CFIFO_DEPTH is the number of entries contained in a CFIFO—four in this implementation.

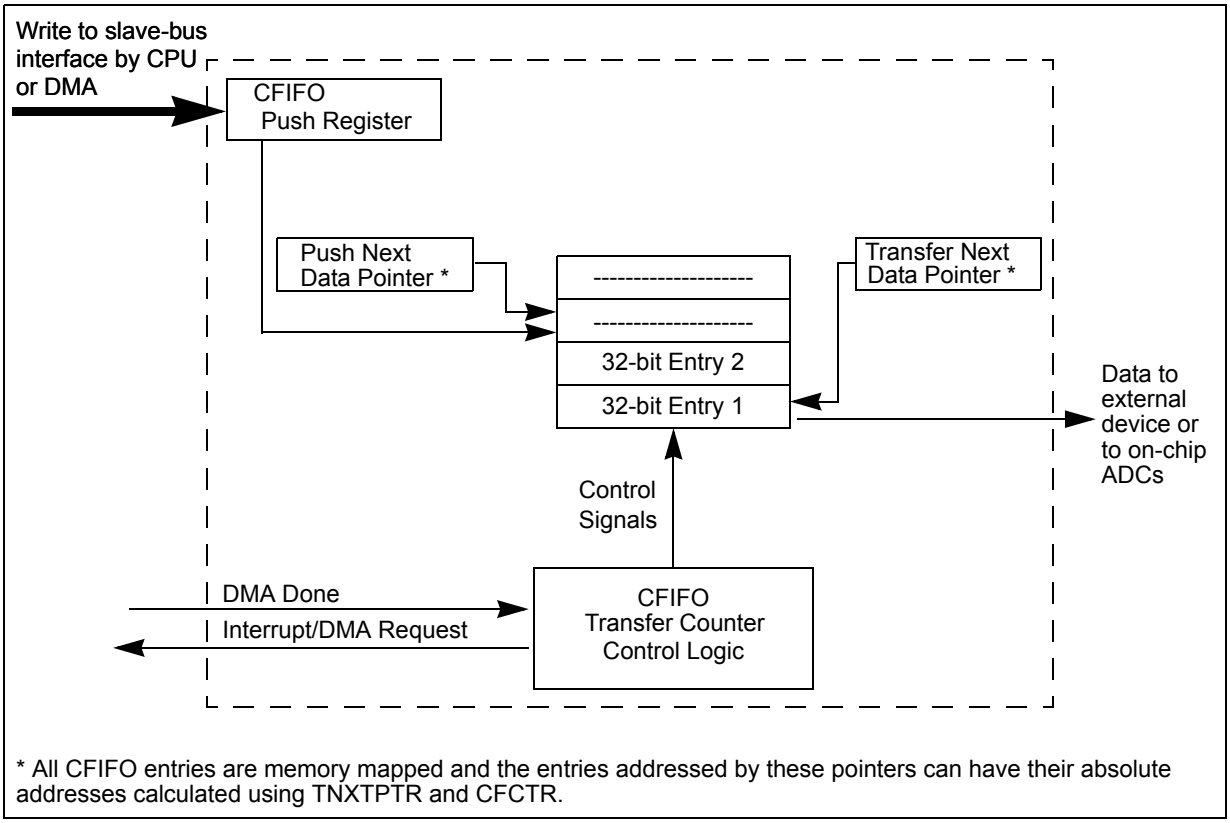
When CFS*x* in the eQADC CFIFO Status Register (EQADC_CFSR) is TRIGGERED, the eQADC generates the proper control signals for the transfer of the entry pointed by Transfer Next Data Pointer. CFUFx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is set when a CFIFOx underflow event occurs. A CFIFO underflow occurs when the CFIFO is in TRIGGERED state and it becomes empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. CFIFOx is empty when the Transfer Next Data Pointer x equals the Push Next Data Pointer x and CFCTRx is zero. CFIFOx is full when the Transfer Next Data Pointer x equals the Push Next Data Pointer x and CFCTRx is not zero.

When the eQADC completes the transfer of an entry from CFIFOx: the transferred entry is popped from CFIFOx, the CFIFO counter CFCTR in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is decremented by one, and Transfer Next Data Pointer x is incremented by one (or wrapped around) to point to the next entry in the CFIFO. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed.

When the EQADC_CFPRx is written and CFIFOx is not full, the CFIFO counter CFCTRx is incremented by one, and the Push Next Data Pointer x then is incremented by one (or wrapped around) to point to the next entry in the CFIFO.

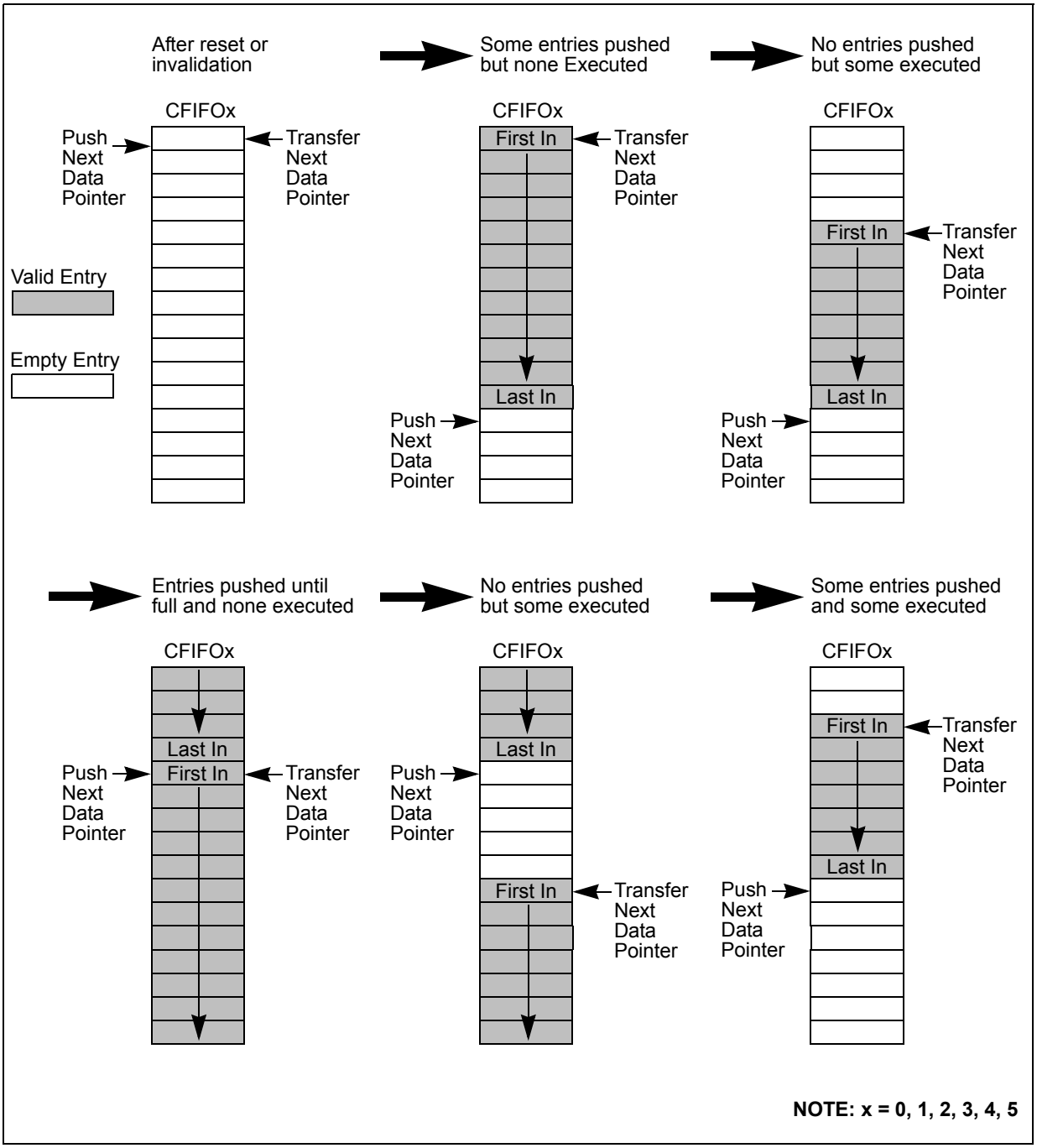
When the EQADC_CFPRx is written but CFIFOx is full, the eQADC will not increment the counter value and will not overwrite any entry in CFIFOx.

Figure 448. CFIFO diagram



The detailed behavior of the Push Next Data Pointer and Transfer Next Data Pointer is described in the example shown in *Figure 449* where a CFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four entries. In this example, CFIFOx with 16 entries is shown in sequence after pushing and transferring entries.

Figure 449. CFIFO entry pointer example



23.5.4.2 CFIFO0 streaming mode description

CFIFO0 can be configured to operate in streaming mode to allow repetition of a group of commands several times without the need of refilling the registers as in the normal mode of operation of CFIFOs. This mode makes use of the additional bit in the Conversion Command Word (CCW) called ‘Repeat’ (REP bit). The purpose of this bit is to mark in the command queue, where to start a repeating sequence. This location is stored in an additional pointer ‘Repeat Pointer’.

Streaming mode requires two trigger inputs. The standard queue 0 trigger, in this mode referred to as Repeat Trigger and a new internal trigger input to the eQADC called Advance Trigger (no filter available).

CFIFO0 is configured to operate in streaming mode by setting the bit STRME0 as described in *Section 23.4.2.6: eQADC CFIFO Control Registers (EQADC_CFCR)*. CFIFO0 is eight entries deep in extended mode by setting the bit EQADC_CFCR0[CFEEE0], and each entry is 32 bits long. This CFIFO0 serves as a local storage of a few commands that need to be executed sequentially as in a FIFO but can contain sub-queues that need to be executed several times. The CFFF0 bit in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is used to assure the CFIFO0 is not full and command messages are stored from address 0x0 to 0x7.

23.5.4.2.1 CFIFO0 operation in streaming mode

In Streaming mode, the CFIFO0 is filled with CCWs using the DMA exactly the same as existing modes. The CFIFO executes commands as per the existing modes until it executes a Conversion Command Word with the Repeat bit set. When this CCW is executed, the Repeat Pointer is set to point to this FIFO location and from this CCW onwards, CFIFO0 entries is not invalidated, that is, the Repeat Pointer prevents this and subsequent entries from being overwritten.

The queue continues to execute until a CCW with an asserted Pause bit is completed; then the queue stops and enters the Pause state, waiting for a trigger. This is the same as normal behavior.

The Pause state is exited in one of two ways: Repeat Trigger or Repeat Trigger with Advance Trigger. The Repeat trigger with no Advance trigger causes the Transfer Next Data Pointer to be loaded with the Repeat Pointer location and CCWs are then executed from the Repeat Pointer back to the Pause bit. This means that a section of the CFIFO0 is repeatedly executed every time a Repeat Trigger occurs.

The Repeat trigger with the Advance trigger pending causes all CCWs from the Repeat pointer to the Pause bit to be invalidated and the CCW after the pause bit to be executed. This is achieved by invalidating the Repeat Pointer. The effect is that the queue advances beyond the repeating section of the CFIFO0 to execute new CCWs.

Note that the Advance trigger can occur at any time between Repeat triggers, but is only actioned when the next Repeat trigger occurs. Prior to that it is pending.

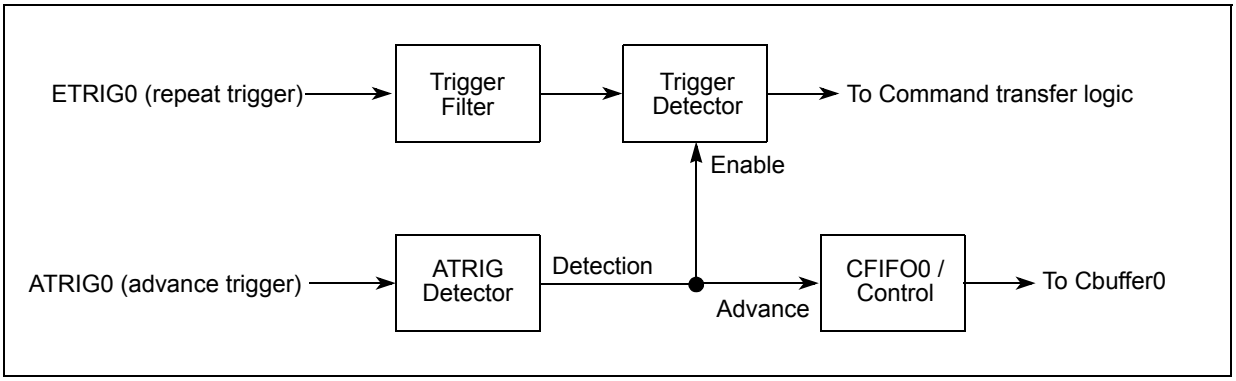
In a typical application, the queue is made of some configuration commands to the ADC (to flush the decimator or turn on pad pull-up/down) followed by a repeating section of ADC conversions on one or more ADC channels from one or more sensors; followed by a few more configuration commands; then more repeating ADC conversions, until the entire engine cycle is complete; when the queue is restarted. The mechanism described permits any number of repeating sub-queues to be loaded and executed, interspersed by configuration commands.

23.5.4.2.2 Triggering description in streaming mode

The additional trigger signal ATRIG0 is detected by a separate circuit that is configured by the bit field AMODE0 as described in *Section 23.4.2.6: eQADC CFIFO Control Registers (EQADC_CFCR)*. This trigger signal is used as an advance control of pop pointer of CFIFO0. In addition, it is used as the enable trigger for the Repeat trigger. This means it is necessary to have an Advance trigger first to enable the detection of the Repeat trigger. When the Repeat trigger is enabled, the Advance trigger is used to advance the pop pointer

beyond some loop sub-queue. The Advance trigger is also used to disable the Repeat trigger by executing a Pause without a previous REP bit. *Figure 450* and *Figure 451* illustrate these two triggers dependency.

Figure 450. Advance trigger and repeat trigger relationship block diagram



A typical sequence of events is presented below to describe the relationship between the triggers.

In Streaming mode, the CFIFO0 is filled with CCWs using the DMA as usual. The two triggers are configured to positive edge and single scan mode.

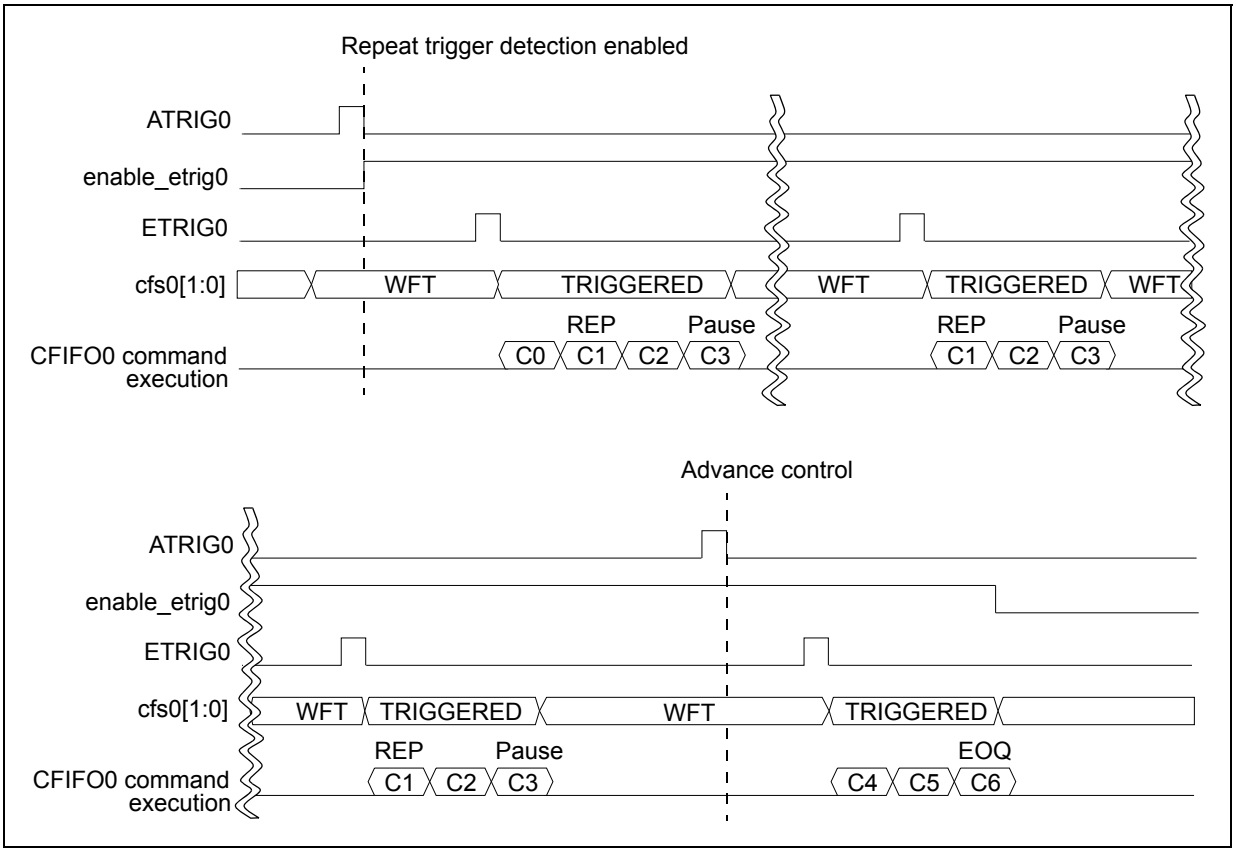
The SSS bit is asserted and the trigger detector of the Repeat trigger is disabled in the start of the queue. It is necessary to receive the first Advance trigger to enable the detector of the other trigger. This enable is useful when the Repeat trigger is received all the time and the trigger signal can be disabled when it is not desired.

The Advance trigger is received and detected and the Repeat trigger detector is enabled. No commands are executed until now.

The Repeat trigger is detected and the commands start to be executed in sequence. If a REP bit is decoded with the PAUSE bit, the loop is configured and the CFIFO0 commands stop to be executed. The next Repeat trigger is waited to start the execution of the loop again, or the Advance trigger can be detected to break the loop and advance the queue in CFIFO0. The Repeat trigger detector remains enabled.

If the Advance trigger is received and the next command in the CFIFO0 does not present the REP bit set, this means the CFIFO0 is not starting a new loop. In this case (outside a loop) if a PAUSE bit is decoded, this means to disable the Repeat trigger detector. This can be useful if the Repeat trigger is not required for some interval of time. The Repeat trigger detector is enabled again when the next Advance trigger event is detected.

Figure 451. Timing diagram of typical use of advance trigger and repeat trigger



23.5.4.2.3 CFIFO0 diagram description in streaming mode

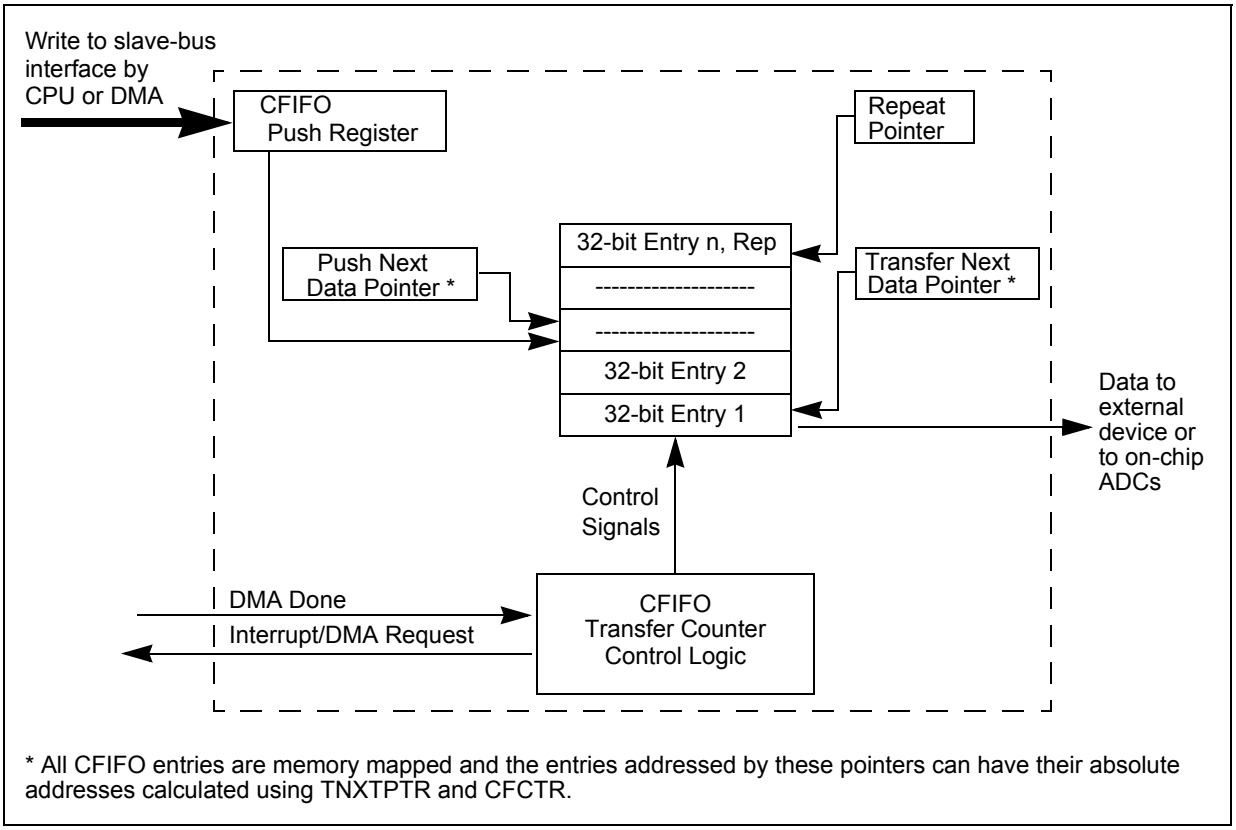
Figure 452 represents the main components of CFIFO0 in streaming mode. However, some signals behave in a different way from the common operation. The Push Next Data Pointer points to the next available CFIFO0 location for storing data written into the eQADC Command FIFO Push Register. The Transfer Next Data Pointer points to the next entry to be transferred to Cbuffer. The Repeat Pointer points to the first entry of the repeating sub-queue. TNXTPTR in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) indicates the index of the entry that is currently being addressed by the Transfer Next Data Pointer, and CFCTR, in the same register, provides the number of entries stored in the CFIFO.

When CFS0 in the eQADC CFIFO Status Register (EQADC_CFSR) is TRIGGERED, the eQADC generates the proper control signals for the transfer of the entry pointed by Transfer Next Data Pointer. CFUF0 in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is set when CFIFO0 underflow event occurs. A CFIFO underflow occurs when the CFIFO is in TRIGGERED state and it is empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. CFIFO0 is empty when CFCTR0 is zero. CFIFO0 is full when (CFCTR0 mod CFIFO_DEPTH) is zero but CFCTR0 is not zero.

When the eQADC completes the transfer of an entry from CFIFO0 in loop condition: the transferred entry is not popped from CFIFO0, the CFIFO counter CFCTR in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is not decremented by one, and

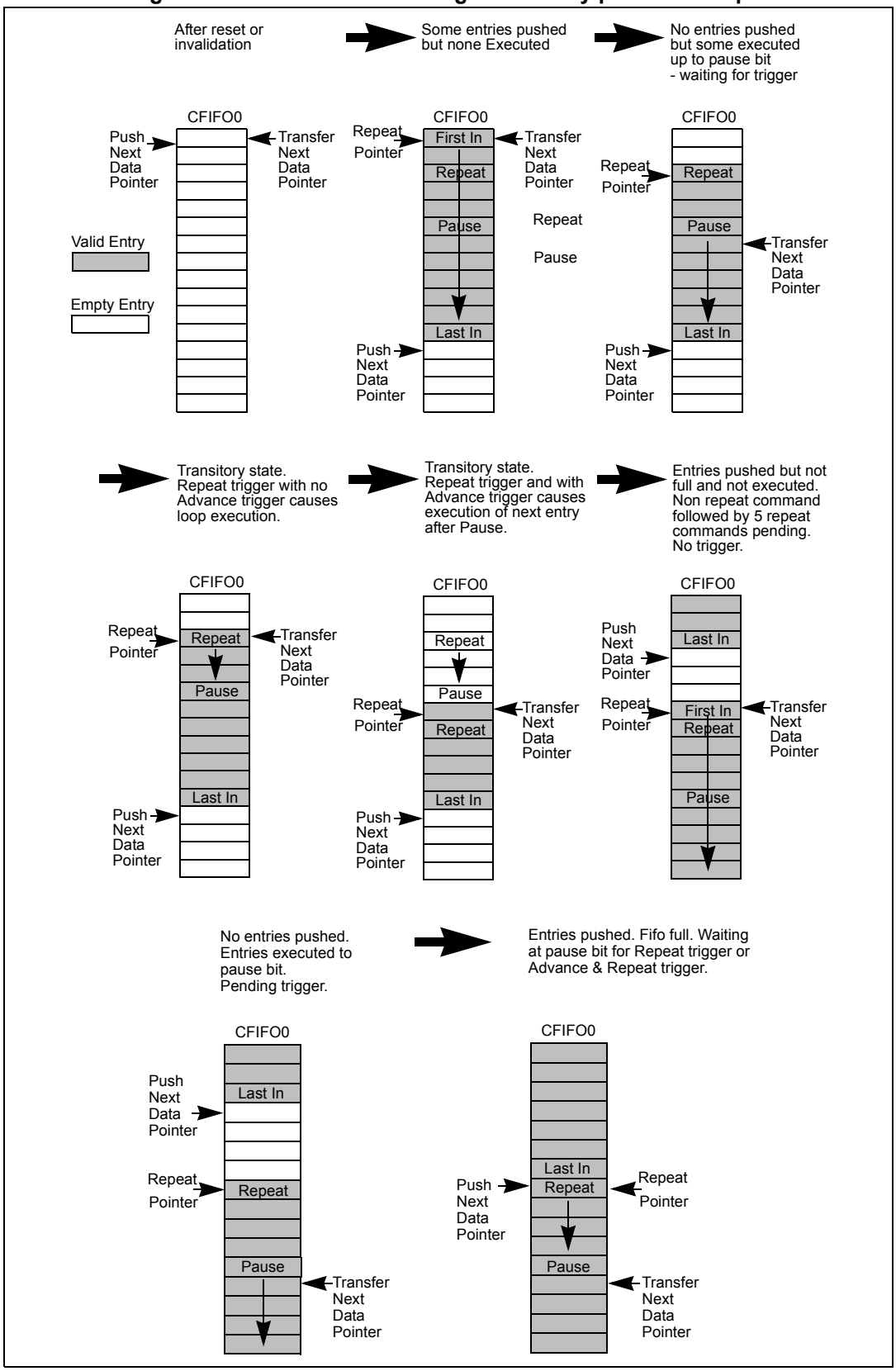
Transfer Next Data Pointer 0 is incremented by one (or wrapped around) to point to the next entry in the CFIFO0.

Figure 452. CFIFO0 in streaming mode diagram



The detailed behavior of the Push Next Data Pointer and Transfer Next Data Pointer is described in the example shown in *Figure 453* where a CFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four/eight entries. In this example, CFIFO0 with 16 entries is shown in sequence after pushing and transferring entries.

Figure 453. CFIFO0 in streaming mode entry pointer example



23.5.4.2.4 Streaming mode error conditions

In the repeat state, the existing error conditions still apply, but now there are new ways to trigger them. Now, the CCWs are not being invalidated so the DMA is not able to load more CCWs into those locations. So a queue overflow becomes more likely, and occurs if the repeat loop is longer than eight entries. If all CCWs in the CFIFO0 are executed and no Pause bit or EOQ bit is detected, the eQADC will signal an underflow error. In practice this may limit a repeating queue to seven entries since otherwise an underflow will occur at the point a Repeat with Advance trigger occurs, and there is no command in the CFIFO0 to execute. The exception is a final command with both a Pause and an EOQ bit set. The End of Queue bit EOQ continues to operate as in normal mode, unless the Repeat mode is enabled. In this case the Pause bit takes precedence and a Repeat trigger causes the jump back described. A Repeat trigger with Advance trigger causes the queue to end.

Another error condition occur when the repeat trigger is in the TRIGGERED state and a new repeat trigger is received. In this case, a trigger overflow occurs but the CFIFO0 is defined to not restart the loop. The trigger in this case is not used in the CFIFO0, but the overflow is indicated.

23.5.4.3 CFIFO common prioritization and command transfer

The CFIFO priority is fixed according to the CFIFO number. A CFIFO with a smaller number has a higher priority. When commands of distinct CFIFOs are bound for the same destination (CBuffer), the higher priority CFIFO is always served first. A TRIGGERED, not-underflowing CFIFO will start the transfer of its commands when:

- its commands are bound for an internal CBuffer that is not full, and it is the highest priority triggered CFIFO sending commands to that CBuffer.
- its commands are bound for an external CBuffer that is not full, and it is the highest priority triggered CFIFO sending commands to an external CBuffer that is not full.

A triggered CFIFO with commands bound for a certain CBuffer consecutively transfers its commands to it until:

- an asserted End Of Queue bit is reached, or;
- an asserted Pause bit is encountered and the CFIFO is configured for edge trigger mode, or;
- CFIFO is configured for level trigger mode and a closed gate is detected, or;
- in case its commands are bound for an internal CBuffer, a higher priority CFIFO that uses the same internal CBuffer is triggered, or;
- in case its commands are bound for an external CBuffer, a higher priority CFIFO that uses an external CBuffer is triggered.

The prioritization logic of the eQADC, depicted in *Figure 454*, is composed of three independent sub-blocks: one prioritizing CFIFOs with commands bound for CBuffer0, another prioritizing CFIFOs with commands for CBuffer1, and a last one prioritizing CFIFOs with commands for CBuffer2 and CBuffer3 which reside inside the external device. As these three sub-blocks are independent, simultaneous writes to CBuffer0, to CBuffer1, and to eQADC SSI transmit buffer are allowed. The hardware identifies the destination of a command by decoding the EB and BN bits in the command message—see *Section 23.5.2.3: Message format in eQADC* for details.

Note: Triggered but empty CFIFOs, underflowing CFIFOs, are not considered for prioritization. No data from these CFIFOs will be sent to the CBuffers and nor will they stop lower priority CFIFOs from transferring commands.

Whenever CBuffer0 is able to receive new entries, the prioritization sub-block selects the highest-priority triggered CFIFO with a command bound for CBuffer0, and writes its command into the buffer. In case CBuffer0 is able to receive new entries but there are no triggered CFIFOs with commands bound for it, nothing is written to the buffer. The sub-block prioritizing CBuffer1 usage behaves in the same way.

When the eQADC SSI is enabled and ready to start serial transmissions, the sub-block prioritizing eQADC SSI usage writes command or null messages into the eQADC SSI transmit buffer, data written to the eQADC SSI transmit buffer is subsequently transmitted to the external device through the eQADC SSI link. The sub-block writes commands to the eQADC SSI transmit buffer when there are triggered CFIFOs with commands bound for not-full external CBuffers. The command written to the transmit buffer belongs to the highest priority CFIFO sending commands to a external CBuffer that is not full. This implies that a lower priority CFIFO can have its commands sent if a higher priority CFIFO cannot send its commands due to a full CBuffer. The sub-block writes null messages to the eQADC SSI transmit buffer when there are no triggered CFIFOs with commands bound for external CBuffers, or when there are triggered CFIFOs with commands bound for external CBuffers but the external CBuffers are full. The eQADC monitors the status of the external CBuffers by decoding the BUSY fields of the incoming result messages from the external device—see *Section 23.5.2.3.2.2: Result message format for external device operation* for details.

Note: When a lower priority CFIFO is served first because a higher priority CFIFO cannot send its commands due to a full external CBuffer, there is a possibility that command transfers from the lower priority CFIFO will be interrupted and the CFIFO will become non-coherent, when the higher priority CFIFO again becomes ready to send commands. If the lower priority CFIFO becomes non-coherent or not depends on the rate at which commands on the external CBuffers are executed, on the rate at which commands are transmitted to the external CBuffers, and on the depth of those buffers.

Once a serial transmission is started, the sub-block monitors triggered CFIFOs and manages the abort of serial transmissions. In case a null message is being transmitted, the serial transmission is aborted when all following conditions are met:

- A not-underflowing CFIFO in TRIGGERED state has commands bound for an external CBuffer that is not full, and it is the highest priority CFIFO sending commands to an external CBuffer that is not full.
- The ABORT_ST bit of the command to be transmitted is asserted.
- The 26th bit of currently transmitting null message has not being shifted out.

The command from the CFIFO is then written into eQADC SSI transmit buffer, allowing for a new serial transmission to initiate.

In case a command is being transmitted, the serial transmission is aborted when all following conditions are met:

- CFIFO0 is in TRIGGERED state, is not underflowing, and its current command is bound for an external CBuffer that is not full.
- The ABORT_ST bit of the command to be transmitted is asserted.
- The 26th bit of currently transmitting command has not being shifted out.

The command from CFIFO0 is then written into eQADC SSI transmit buffer, allowing for a new serial transmission to initiate.

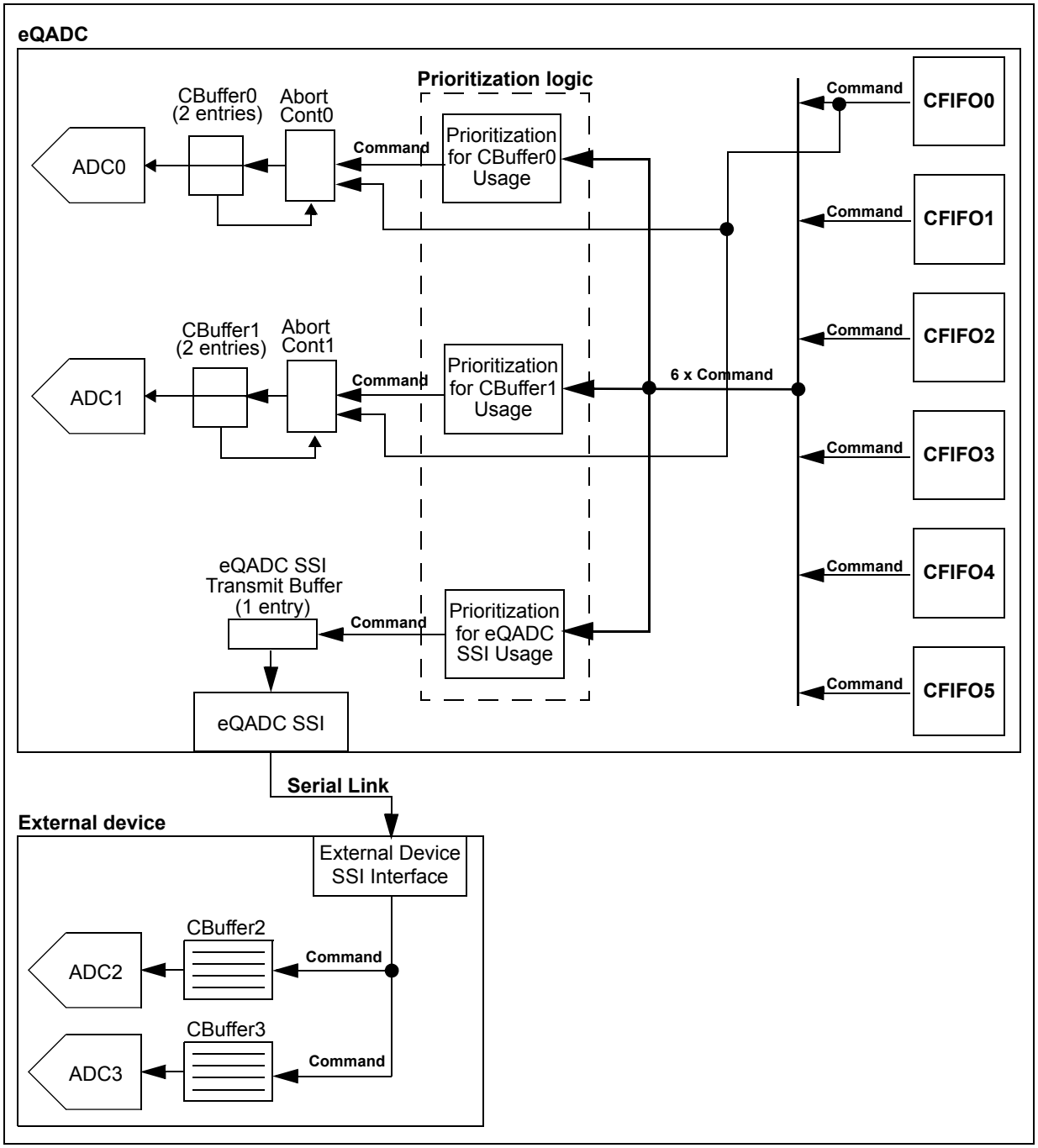
Note: The aborted command is not popped from the preempted CFIFO and will be retransmitted as soon as its CFIFO becomes the highest priority CFIFO sending commands to an external CBuffer that is not full.

After a serial transmission is completed, the eQADC prioritizes the CFIFOs and schedules a command or a null message to be sent in the next serial transmission. After the data for the next transmission has been defined and scheduled, the eQADC can, under certain conditions, stretch the SDS negation time in order to allow the schedule of new data for that transmission. This occurs when the eQADC acknowledges that the status of a higher-priority CFIFO changed to TRIGGERED and attempts to schedule that CFIFO command before SDS is asserted. Only commands of CFIFOs that have the ABORT_ST bit asserted can be scheduled in this manner. Under such conditions:

1. a CFIFO0 command is scheduled for the next transmission independently of the type of data that was previously scheduled. The time during which SDS is negated is stretched in order to allow the eQADC to load the CFIFO0 command and start its transmission.
2. CFIFO1–5 commands are only scheduled for the next transmission if the previously scheduled data was a null message. The time during which SDS is negated is stretched in order to allow the eQADC to load that command and start its transmission. However, if the previously scheduled data was a command, no rescheduling occurs and the next transmission starts without delays.

If a CFIFO becomes TRIGGERED while SDS is negated, but the eQADC only attempts to reschedule that CFIFO command after SDS is asserted, then the current transmission is aborted depending on if the conditions for that are met or not.

Figure 454. CFIFO prioritization logic



23.5.4.4 CFIFO prioritization in abort mode

The CFIFO priority does not change when the eQADC is configured to allow abortion of conversion execution in on-chip ADC analog blocks. However, CFIFO0 is the only one that can be enabled to abort conversions.

This feature is necessary when the timing of some conversion is very important. In normal priority scheme, when Cqueue0 is triggered, its conversion command can be put behind two pending conversion commands in the Cbuffer due to the queue structure. Considering that

these two pending commands are from lower priority Cqueues and that the delay between the trigger and the sampling of the command from Cqueue0 can be unacceptable, eQADC can be configured to permit immediate conversion commands from CFIFO0 with abort function.

When CFIFO0 is triggered and abort is enabled, up to two commands in Cbuffer0 or Cbuffer1 are stored in a side register. The abort request signal is generated to ADC0 or ADC1 and the confirmation of ADC reset/ready is waited to send the command from CFIFO0 to the decoded Cbuffer.

Note: The eQADC abort feature takes several system clock cycles to be effective. This delay, which is between the trigger event and the rise edge of the signal start command, is a function of the ADC Clock Prescaler and can be computed as follows:

Equation 3 Delay (in system clock cycles) = 13 + R; R>=4.

Where R represents the ADC clock prescaling factor (configured by the field ADCx_CLK_PS of the ADCx_CR where x={0;1})

For R<4, the delay is fixed to 17 system clock cycles.

After the transfer of all commands from CFIFO0, the recovery phase restores the up to two commands that were in Cbuffer when the abort occurred. After this recovery phase, it is established the normal process of prioritization of commands from CFIFOs.

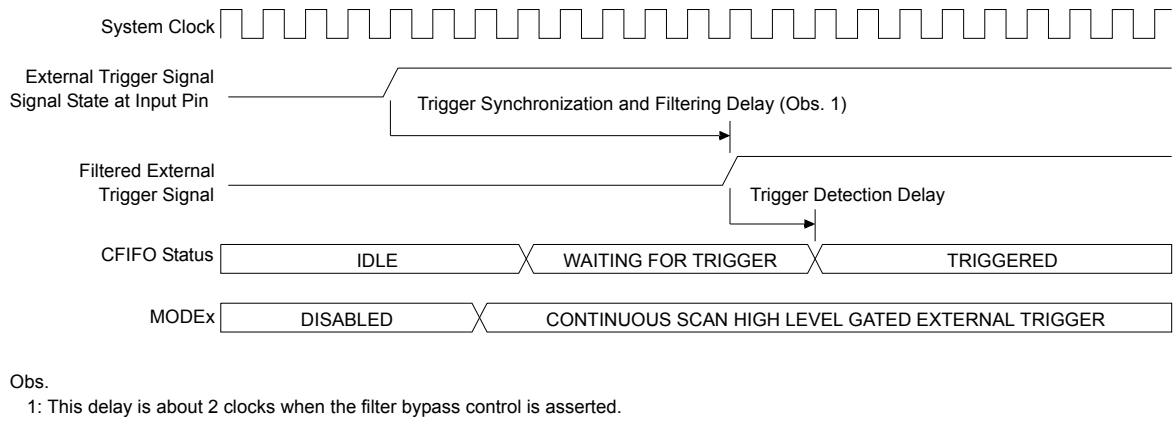
23.5.4.5 External trigger event detection

The digital filters for trigger signals can be individually bypassed by asserting the input control signals eqadc_intern_trig_sel5–0. When the filter is bypassed, the ETRIG input signal is not filtered and the logic after the filter receives a copy of this input trigger signal.

The Digital Filter Length field in the eQADC External Trigger Digital Filter Register (EQADC_ETDFR) specifies the minimum number of system clocks that the ETRIG0–5 signals must be held at a logic level to be recognized as valid. All ETRIG signals are filtered. A counter for each queue trigger is implemented to detect a transition between logic levels. The counter counts at the system clock rate. The corresponding counter is cleared and restarted each time the signal transitions between logic levels. When the corresponding counter matches the value specified by the Digital Filter Length field in the eQADC External Trigger Digital Filter Register (EQADC_ETDFR), the eQADC considers the ETRIG logic level to be valid and passes that new logic level to the rest of the eQADC.

The filter is only for filtering the ETRIG signal. Logic after the filter checks for transitions between filtered values, such as for detecting the transition from a filtered logic level zero to a filter logic level one in rising edge external trigger mode. The eQADC can detect rising edge, falling edge, or level gated external triggers. The digital filter will always be active independently of the status of the MODE*x* field in the eQADC CFIFO Control Registers (EQADC_CFCR), but the edge, level detection logic is only active when MODEx is set to a value different from disabled, and in case MODEx is set to single scan mode, when the SSS bit is asserted. Note that the time necessary for a external trigger event to result into a CFIFO status change is not solely determined by the DFL field in the eQADC External Trigger Digital Filter Register (EQADC_ETDFR). After being synchronized to the system clock and filtered, a trigger event is checked against the CFIFO trigger mode. Only then, after a valid trigger event is detected, the eQADC accordingly changes the CFIFO status. Refer to *Figure 455* for an example.

Figure 455. ETRIG event propagation example



Obs.
1: This delay is about 2 clocks when the filter bypass control is asserted.

23.5.4.6 CFIFO scan trigger modes

The eQADC supports two different scan modes, single-scan and continuous-scan. Refer to *Table 296* for a summary of these two scan modes. When a CFIFO is triggered, the eQADC scan mode determines whether the eQADC will stop command transfers from a CFIFO, and wait for software intervention to rearm the CFIFO to detect new trigger events, upon detection of an asserted EOQ bit in the last transfer. Refer to *Section 23.5.2.3: Message format in eQADC* for details about command formats.

CFIFOs can be configured in single-scan or continuous-scan mode. When a CFIFO is configured in single-scan mode, the eQADC scans the CQueue one time. The eQADC stops future command transfers from the triggered CFIFO after detecting the EOQ bit set in the last transfer. After a EOQ bit is detected, software involvement is required to rearm the CFIFO so that it can detect new trigger events.

When a CFIFO is configured for continuous-scan mode, no software involvement is necessary to rearm the CFIFO to detect new trigger events after an asserted EOQ is detected. In continuous-scan mode the whole CQueue is scanned multiple times.

The eQADC also supports different triggering mechanisms for each scan mode. The eQADC will not transfer commands from a CFIFO until the CFIFO is triggered. The combination of scan modes and triggering mechanisms allows the support of different requirements for scanning input channels. The scan mode and trigger mechanism are configured by programming the MODEx field in the eQADC CFIFO Control Registers (EQADC_CFCR).

Enabled CFIFOs can be triggered by software or external trigger events. The elapsed time from detecting a trigger to transferring a command is a function of clock frequency, trigger synchronization, trigger filtering or not, programmable trigger events, command transfer, CFIFO prioritization, CBuffer availability, etc. Fast and predictable transfers can be achieved by ensuring that the CFIFO is not underflowing and that the target CBuffer is not full when the CFIFO is triggered.

23.5.4.6.1 Disabled Mode

The MODE*x* field in the eQADC CFIFO Control Registers (EQADC_CFCR) for all of the CFIFOs can be changed from any other mode to disabled at any time. No trigger event can

initiate command transfers from an CFIFO which has its MODE field programmed to disabled.

Note: If MODEx is not disabled, it must not be changed to any other mode besides disabled. If MODEx is disabled and the CFIFO status is IDLE, MODEx can be changed to any other mode.

If MODE*x* is changed to disabled:

- The CFIFO execution status will change to IDLE. The timing of this change depends on whether a command is being transferred or not:
  - When no command transfer is in progress, the eQADC switches the CFIFO to IDLE status immediately.
  - When a command transfer to an on-chip CBuffer is in progress, the eQADC will complete the transfer, update TC_CF, and switch CFIFO status to IDLE. Command transfers to the internal CBuffers are considered completed when a command is written to the buffers.
  - When a command transfer to an external CBuffer is in progress, the eQADC will abort the transfer and switch CFIFO status to IDLE. If the eQADC cannot abort the transfer, that is when the 26th bit of the serial message has being already shifted out, the eQADC will complete the transfer, update TC_CF and then switch CFIFO status to IDLE.
- The CFIFOs are not invalidated automatically. The CFIFO still can be invalidated by writing a ‘1’ to the CFINVx bit in the eQADC CFIFO Control Registers (EQADC_CFCR). Certify that CFS has changed to IDLE before setting CFINVx.
- The TC_CFx value also is not reset automatically, but it can be reset by writing ‘0’ to it.
- The SSS bit in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is negated. The SSS bit can be set even if a ‘1’ is written to the SSE bit in the eQADC CFIFO Control Registers (EQADC_CFCR) in the same write that the MODEx field is changed to a value other than disabled.
- The trigger detection hardware is reset. If MODEx is changed from disabled to an edge trigger mode, a new edge, matching that edge trigger mode, is needed to trigger the command transfers from the CFIFO.

Note: CFIFO fill requests, which generated when CFFF is asserted, are not automatically halted when MODEx is changed to disabled. CFIFO fill requests will still be generated until CFFE is cleared in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR).

23.5.4.6.2 Single-scan Mode

In single-scan mode, a single pass through a sequence of command messages in a CQueue is performed.

In single-scan software trigger mode, the CFIFO is triggered by an asserted Single-Scan Status bit (SSS) in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR). The SSS bit is set by writing ‘1’ to the Single-Scan Enable bit (SSE) in the eQADC CFIFO Control Registers (EQADC_CFCR).

In single-scan edge- or level-trigger mode, the respective triggers are only detected when the SSS bit is asserted. When the SSS bit is negated, all trigger events for that CFIFO are ignored. Writing a ‘1’ to the SSE bit can be done during the same write cycle that the CFIFO operation mode is configured.

Only the eQADC can clear the SSS bit. Once SSS is asserted, it remains asserted until the eQADC completes the CQueue scan, or the CFIFO operation mode (MODEx) in the

eQADC CFIFO Control Registers (EQADC_CFCR) is changed to disabled. The SSSx bit will be negated while MODE*x* is disabled.

23.5.4.6.2.1 Single-scan software trigger

When single-scan software trigger mode is selected, the CFIFO is triggered by an asserted SSS bit. The SSS bit is asserted by writing '1' to the SSE bit. Writing to SSE while SSS is already asserted will not have any effect on the state of the SSS bit, nor will it cause a trigger overrun event.

The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer. When an asserted EOQ bit is encountered, the eQADC will clear the SSS bit. Setting the SSS bit is required for the eQADC to start the next scan of the queue.

The Pause bit has no effect in single-scan software trigger mode.

23.5.4.6.2.2 Single-scan edge trigger

When SSS is asserted and an edge triggered mode is selected for a CFIFO, an appropriate edge on the associated trigger signal causes the CFIFO to become TRIGGERED. For example, if rising-edge trigger mode is selected, the CFIFO becomes TRIGGERED when a rising edge is sensed on the trigger signal. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer.

When an asserted EOQ bit is encountered, the eQADC clears SSS and stops command transfers from the CFIFO. An asserted SSS bit and a subsequent edge trigger event are required to start the next scan for the CFIFO. When an asserted Pause bit is encountered, the eQADC stops command transfers from the CFIFO, but SSS remains set. Another edge trigger event is required for command transfers to continue. A trigger overrun happens when the CFIFO is in TRIGGERED state and an edge trigger event is detected.

23.5.4.6.2.3 Single-scan level trigger

When SSS is asserted and a level gated trigger mode is selected, the input level on the associated trigger signal puts the CFIFO in TRIGGERED state. When the CFIFO is asserted to high-level gated trigger, a high level signal opens the gate, and a low level closes the gate. When the CFIFO is set to low-level gated trigger mode, a low level signal opens the gate, and a high level closes the gate. If the corresponding level is already present, setting the SSS bit triggers the CFIFO. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or a not-full external CBuffer.

The eQADC clears the SSS bit and stops transferring commands from a TRIGGERED CFIFO when an asserted EOQ bit is encountered or when CFIFO status changes from TRIGGERED due to the detection of a closed gate. If a closed gate is detected while no command transfers are taking place and the CFIFO status is TRIGGERED, the CFIFO status is immediately changed to IDLE, the SSS bit is negated, and the PF flag is asserted. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes. Once the transmission is completed, the TC_CF counter is updated, the SSS bit is negated, the PF flag is asserted, and the CFIFO status is changed to IDLE. An asserted SSS bit and a level trigger are required to restart the CFIFO. Command transfers will restart from the point they have stopped.

If the gate closes and opens during the same serial transmission of a command to the external device, it will have no effect on the CFIFO status or on the PF flag, but the TORF flag will become asserted as was exemplified in *Figure 457*. Therefore, closing the gate for a period less than a serial transmission time interval does not guarantee that the closure will affect command transfers from a CFIFO.

The Pause bit has no effect in single-scan level-trigger mode.

23.5.4.6.3 Continuous-scan Mode

In continuous-scan mode, multiple passes looping through a sequence of command messages in a CQueue are executed. When a CFIFO is programmed for a continuous-scan mode, the SSE bit in the eQADC CFIFO Control Registers (EQADC_CFCR) does not have any effect.

23.5.4.6.3.1 Continuous-scan software trigger

When a CFIFO is programmed to continuous-scan software trigger mode, the CFIFO is triggered immediately. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer. When a CFIFO is programmed to run in continuous-scan software trigger mode, the eQADC will not halt transfers from the CFIFO until the CFIFO operation mode is modified to disabled or a higher priority CFIFO preempts it. Although command transfers will not stop upon detection of an asserted EOQ bit, the EOQF is set and, if enabled, an EOQ interrupt request is generated.

The Pause bit has no effect in continuous-scan software trigger mode.

23.5.4.6.3.2 Continuous-scan edge trigger

When rising, falling, or either edge trigger mode is selected for a CFIFO, a corresponding edge on the associated ETRIG signal places the CFIFO in TRIGGERED state. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer

When an EOQ or a Pause is encountered, the eQADC halts command transfers from the CFIFO and, if enabled, the appropriate interrupt requests are generated. Another edge trigger event is required to resume command transfers but no software involvement is required to rearm the CFIFO in order to detect such event.

A trigger overrun happens when the CFIFO is already in TRIGGERED state and a new edge trigger event is detected.

23.5.4.6.3.3 Continuous-scan level trigger

When high or low level gated trigger mode is selected, the input level on the associated trigger signal places the CFIFO in TRIGGERED state. When high-level gated trigger is selected, a high-level signal opens the gate, and a low level closes the gate. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer. Although command transfers will not stop upon detection of an asserted EOQ bit at the end of a command transfer, the EOQF is asserted and, if enabled, an EOQ interrupt request is generated.

The eQADC stops transferring commands from a TRIGGERED CFIFO when CFIFO status changes from TRIGGERED due to the detection of a closed gate. If a closed gate is detected while no command transfers are taking place and the CFIFO status is

TRIGGERED, the CFIFO status is immediately changed to WAITING FOR TRIGGER and the PF flag is asserted. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes. Once the transmission is completed, the TC_CF counter is updated, the PF flag is asserted, and the CFIFO status is changed to WAITING FOR TRIGGER. Command transfers will restart as the gate opens.

If the gate closes and opens during the same serial transmission of a command to the external device, it will have no effect on the CFIFO status or on the PF flag, but the TORF flag will become asserted as was exemplified in *Figure 457*. Therefore, closing the gate for a period less than a serial transmission time interval does not guarantee that the closure will affect command transfers from a CFIFO.

The Pause bit has no effect in continuous-scan level-trigger mode.

23.5.4.6.4 CFIFO scan trigger mode start/stop summary

Table 296 summarizes the start and stop conditions of command transfers from CFIFOs for all of the single-scan and continuous-scan trigger modes.

Table 296. CFIFO scan trigger mode - command transfer start/stop summary

Trigger mode	Requires asserted SSS to recognize trigger events?	Command transfer start/restart condition	Stop on asserted EOQ bit[1]?	Stop on asserted Pause bit[2]?	Other command transfer stop condition[3] [4]
Single Scan Software	Don't Care	Asserted SSS bit.	Yes	No	None
Single Scan Edge	Yes	A corresponding edge occurs when the SSS bit is asserted.	Yes	Yes	None
Single Scan Level	Yes	Gate is opened when the SSS bit is asserted.	Yes	No	eQADC also stops transfers from the CFIFO when CFIFO status changes from TRIGGERED due to the detection of a closed gate.[5]
Continuous Scan Software	No	CFIFO starts automatically after being configured into this mode.	No	No	None
Continuous Scan Edge	No	A corresponding edge occurs.	Yes	Yes	None
Continuous Scan Level	No	Gate is opened.	No	No	eQADC also stops transfers from the CFIFO when CFIFO status changes from TRIGGERED due to the detection of a closed gate.[5]

1. Refer to *Section 23.5.4.7.2: CQueue completion status* for more information on EOQ.
2. Refer to *Section 23.5.4.7.3: Pause status* for more information on Pause.
3. eQADC always stops command transfers from a CFIFO when the CFIFO operation mode is disabled.
4. eQADC always stops command transfers from a CFIFO when a higher priority CFIFO is triggered. Refer to *Section 23.5.4.3: CFIFO common prioritization and command transfer* for information on CFIFO priority.

5. If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes.

23.5.4.7 CFIFO and trigger status

23.5.4.7.1 CFIFO operation status

Each CFIFOs has its own CFIFO status field. CFIFO status (CFS) can be read from the eQADC CFIFO Status Register (EQADC_CFSR). *Figure 456* and *Table 297* indicate the CFIFO status switching condition. Refer to *Table 268* for the meaning of each CFIFO operation status. The last CFIFO to transfer a command to an on-chip CBuffer can be read from the LCFTCB*n* (n = 0, 1) fields in the eQADC CFIFO Status Snapshot Registers (EQADC_CFSSR). The last CFIFO to transfer a command to a specific external CBuffer can be identified by reading the LCFTSSI and ECBNI fields in the eQADC CFIFO Status Snapshot Registers (EQADC_CFSSR).

Figure 456. State machine of CFIFO status

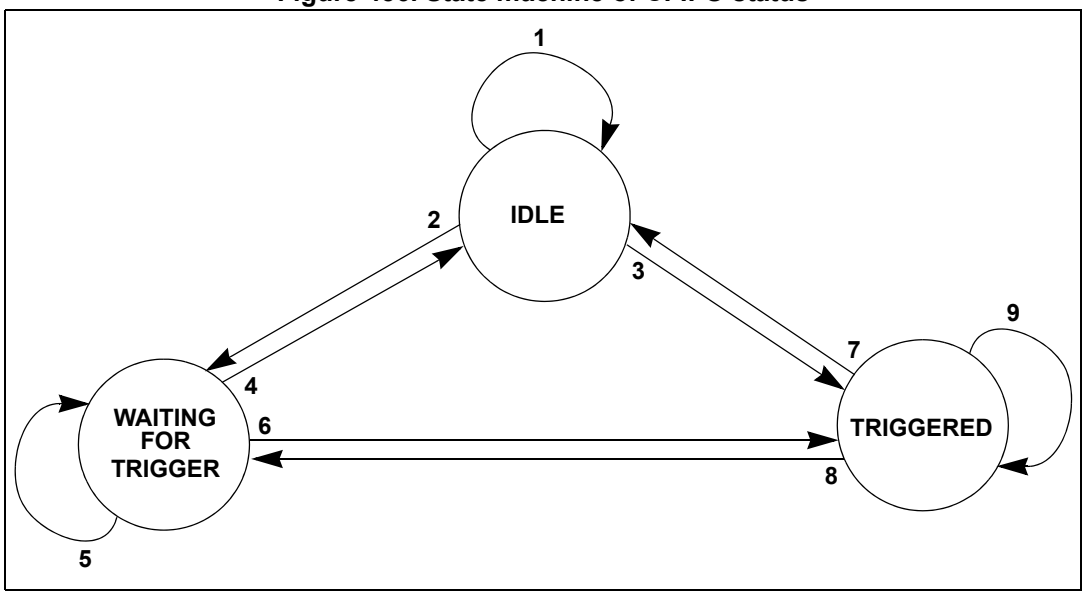


Table 297. Command FIFO status switching condition

No.	From current CFIFO status (CFS)	To new CFIFO status (CFS)	Status switching condition
1	IDLE (00)	IDLE (0b00)	– CFIFO Mode is programmed to disabled, OR – CFIFO Mode is programmed to single-scan edge or level trigger mode and SSS is negated.
2		WAITING FOR TRIGGER (0b10)	– CFIFO Mode is programmed to continuous-scan edge or level trigger mode, OR – CFIFO Mode is programmed to single-scan edge or level trigger mode and SSS is asserted, OR – CFIFO Mode is programmed to single-scan software trigger mode.
3		TRIGGERED (0b11)	CFIFO Mode is programmed to continuous-scan software trigger mode
4	WAITING FOR TRIGGER (10)	IDLE (0b00)	CFIFO Mode is modified to disabled mode.
5		WAITING FOR TRIGGER (0b10)	No trigger occurred.
6		TRIGGERED (0b11)	– Appropriate edge or level trigger occurred, OR – CFIFO Mode is programmed to single-scan software trigger mode and SSS bit is asserted.

Table 297. Command FIFO status switching condition (continued)

No.	From current CFIFO status (CFS)	To new CFIFO status (CFS)	Status switching condition
7	TRIGGERED (11)	IDLE (0b00)	– CFIFO in single-scan mode, eQADC detects the EOQ bit asserted at end of command transfer, and CFIFO Mode is not modified to disabled, OR – CFIFO, in single-scan level trigger mode, and the gate closes while no commands are being transferred from the CFIFO, and CFIFO Mode is not modified to disabled, OR – CFIFO, in single-scan level trigger mode, and eQADC detects a closed gated at end of command transfer, and CFIFO Mode is not modified to disabled, OR – CFIFO Mode is modified to disabled mode and CFIFO was not transferring commands. – CFIFO Mode is modified to disabled mode while CFIFO was transferring commands, and CFIFO completes or aborts the transfer.
8		WAITING FOR TRIGGER (0b10)	– CFIFO in single or continuous-scan edge trigger mode, eQADC detects the Pause bit asserted at the end of command transfer, the EOQ bit in the same command is negated, and CFIFO Mode is not modified to disabled, OR – CFIFO in continuous-scan edge trigger mode, eQADC detects the EOQ bit asserted at the end of command transfer, and CFIFO Mode is not modified to disabled, OR – CFIFO, in continuous-scan level trigger mode, and the gate closes while no commands are being transferred from the CFIFO, and CFIFO Mode is not modified to disabled, OR – CFIFO, in continuous-scan level trigger mode, and eQADC detects a closed gated at end of command transfer, and CFIFO Mode is not modified to disabled.
9		TRIGGERED (0b11)	No event to switch to IDLE or WAITING FOR TRIGGER status has happened.

23.5.4.7.2 CQueue completion status

The End of Queue Flag (EOQF) in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is asserted when the eQADC completes the transfer of a CFIFO entry with an asserted EOQ bit. Software sets the EOQ bit in the last Command Message of a CQueue to indicate that this entry is the end of the CQueue—see *Section 23.5.2.3: Message format in eQADC* for information on command message formats. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The transfer of entries bound for the external device is considered completed when the serial transmission of the entry is completed.

The command with a EOQ bit asserted is valid and will be transferred. When EOQIE in the eQADC CFIFO Control Registers (EQADC_CFCR) and EOQF are asserted, the eQADC will generate an End of Queue interrupt request.

In single-scan modes, command transfers from the corresponding CFIFO will cease when the eQADC completes the transfer of a entry with an asserted EOQ. Software involvement is required to rearm the CFIFO so that it can detect new trigger events.

Note: An asserted EOQFx only implies that the eQADC has finished transferring a command with an asserted EOQ bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

23.5.4.7.3 Pause status

In edge trigger mode, when the eQADC completes the transfer of a CFIFO entry with an asserted Pause bit, the eQADC will stop future command transfers from the CFIFO and set the corresponding Pause Flag (PF) in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR). Refer to *Section 23.5.2.3: Message format in eQADC* for information on command message formats. The eQADC ignores the Pause bit in command messages in any software and external level trigger mode. The eQADC sets the PF flag upon detection of an asserted Pause bit only in single or continuous-scan edge trigger mode. When the PF flag is set for a CFIFO in single-scan edge trigger mode, the SSS bit will not be cleared in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR).

In level trigger mode, the definition of the PF flag has been redefined. In level trigger mode, when CFIFOx is in TRIGGERED status, PFx is set when CFIFO status changes from TRIGGERED due to detection of a closed gate. The pause flag interrupt routine can be used to verify if the a complete scan of the CQueue was performed. If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes.

When PIE in the eQADC CFIFO Control Registers (EQADC_CFCR) and PF are asserted, the eQADC will generate a Pause interrupt request.

Note: In edge trigger mode, an asserted PFx only implies that the eQADC finished transferring a command with an asserted PAUSE bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

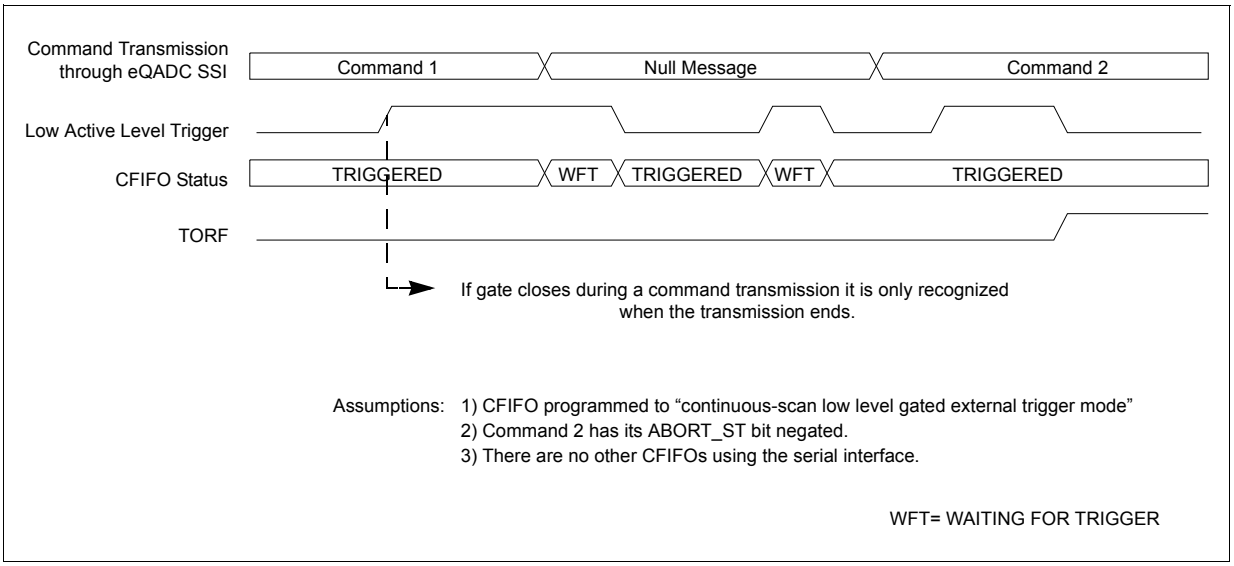
Note: In software or level trigger mode, when the eQADC completes the transfer of an entry from CFIFOx with an asserted Pause bit, PFx will not be set and command transfers will continues without pausing.

23.5.4.7.4 Trigger overrun status

When a CFIFO is configured for edge- or level-trigger mode and is in TRIGGERED state, an additional trigger occurring for the same CFIFO results in a trigger overrun. The trigger overrun bit for the corresponding CFIFO will be set (TORF*x* = 1) in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR). When TORIE in the eQADC CFIFO Control Registers (EQADC_CFCR) and TORF are asserted, the eQADC generates a trigger overrun interrupt request.

For CFIFOs configured for level-trigger mode, a trigger overrun event is only detected when the gate closes and opens during a single serial command transmission as shown in *Figure 457*.

Figure 457. Trigger overrun on level-trigger mode CFIFOs



Note: The trigger overrun flag will not set for CFIFOs configured for software trigger mode.

23.5.4.7.5 Command sequence non-coherency detection

The eQADC provides a mechanism to indicate if a command sequence has been completely executed without interruptions. A command sequence is defined as a group of consecutive commands bound for the same CBuffer and it is expected to be executed without interruptions. A command sequence is coherent if its commands are executed in order without interruptions. Since commands are stored in the CBuffers before being executed in the eQADC, a command sequence is coherent if, while it is transferring commands to a CBuffer, the buffer is only fed with commands from that sequence without ever becoming empty.

A command sequence starts when:

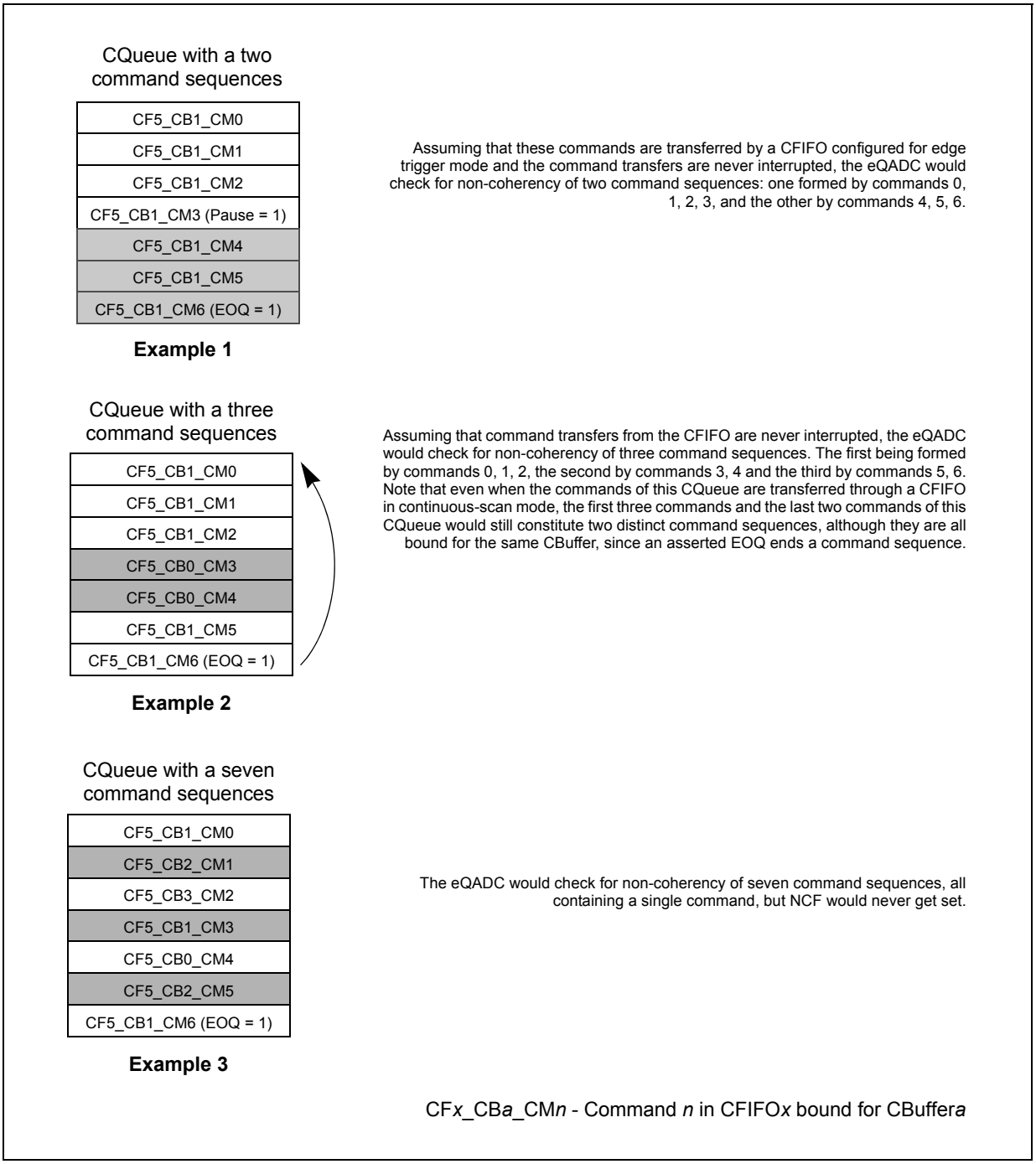
- a CFIFO in TRIGGERED state transfers its first command to CBuffer.
- the CFIFO is constantly transferring commands and the previous command sequence ended.
- the CFIFO resumes command transfers after being interrupted.

And a command sequence ends when:

- an asserted EOQ bit is detected on the last transferred command.
- CFIFO is in edge-trigger mode and asserted PAUSE bit is detected on the last transferred command.
- the CBuffer to which the next command is bound is different from the one to which the last command was transferred.

Figure 458 shows examples of how the eQADC would detect command sequences when transferring commands from a CFIFO to a CBuffer. The smallest possible command sequence can have a single command as shown in example 3 of *Figure 458*.

Figure 458. Command sequence examples



The NCF flag is used to indicate command sequence non-coherency. When the NCFx flag is asserted, it indicates that the command sequence being transferred through CFIFOx became non-coherent. The NCF flag only becomes asserted for CFIFOs in TRIGGERED state.

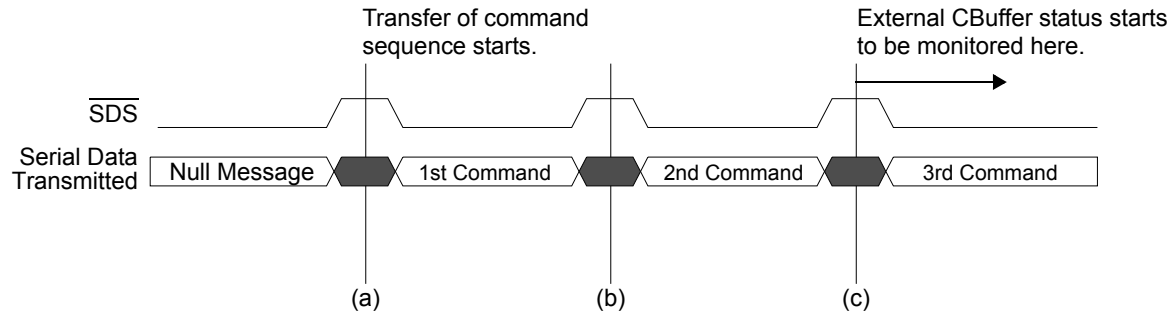
A command sequence is non-coherent when, after transferring the first command of a sequence from a CFIFO to a CBuffer, it cannot successively send all the other commands of the sequence before any of the following conditions are true:

- The CFIFO through which commands are being transferred is preempted by a higher priority CFIFO which sends commands to the same CBuffer. The NCF flag becomes asserted immediately after the first command transfer from the preempting CFIFO, that is the higher priority CFIFO, to the CBuffer in use is completed. See *Figure 460*.
- The external CBuffer in use becomes empty[r]. This case happens when different CFIFOs attempt to use different external CBuffers and the higher priority CFIFO bars the lower priority one from sending new commands to its CBuffer—see *Figure 461*. An external CBuffer is considered empty when the corresponding BUSY field in the last result message received from external device is encoded as “Send available commands – CBuffer is empty”. Refer to *Section 23.5.2.3.2.2: Result message format for external device operation*. The NCF flag becomes asserted immediately after the eQADC detects that the external CBuffer in use becomes empty.

Note: After the transfer of a command sequence to an external CBuffer starts, the eQADC ignores, for non-coherency detection purposes, the BUSY fields captured at the end of the first serial transmission. Thereafter, all BUSY fields captured at the end of consecutive serial transmissions are used to check the fullness of that external CBuffer. This is done because the eQADC only updates its external CBuffers status record when it receives a serial message, resulting that the record kept by the eQADC is always outdated by, at least, the length of one serial transmission. This prevents a CFIFO from immediately becoming non-coherent when it starts transferring commands to an empty external CBuffer. Refer to Figure 459 for an example.

r. Only the fullness of external CBuffers is monitored because the fill rate for internal CBuffers is many times faster than the drain rate, and each has a dedicated priority engine.

Figure 459. External CBuffer status detection at command sequence transfer start



External CBuffer status

Capture point at eQADC	CBuffer status		Used for NCF detection on eQADC?
	at external device	as captured by eQADC	
(a)	EMPTY	EMPTY	Don't care
(b)	1 ENTRY	EMPTY	No
(c)	2 ENTRY	1 ENTRY	Yes

Assumptions:

1. The CFIFO starts sending commands to an empty external CBuffer when triggered.
2. Execution of a command on the external device takes longer than the time to complete three serial transmissions.

Once a command sequence starts to be transferred, the eQADC will check for the command sequence coherency until the command sequence ends or until one of the conditions below becomes true.

- The command sequence became non-coherent.
- The CFIFO status changed from TRIGGERED.
- The CFIFO underflowed.

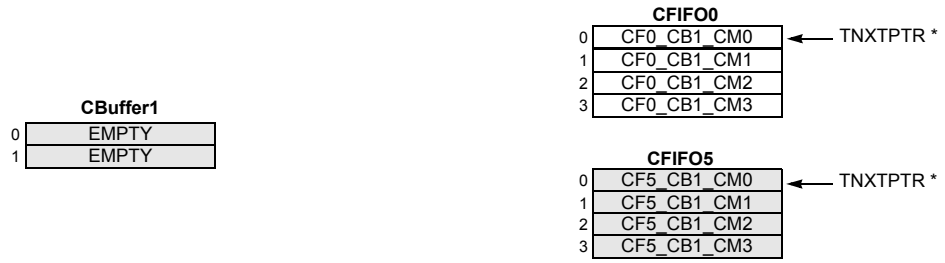
Note: The NCF flag still becomes asserted if an external CBuffer empty event is detected at the same time the eQADC stops checking for the coherency of a command sequence.

Once command transfers restart/continue, the non-coherency hardware will behave as if the command sequence started from that point. *Figure 462* depicts how the non-coherency hardware will behave when a non-coherency event is detected.

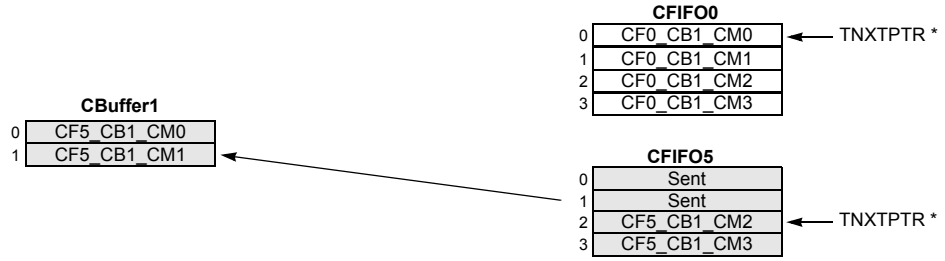
Note: If MODEx is changed to disabled while a CFIFO is transferring commands, the NCF flag for that CFIFO will not become asserted.

Note: When the eQADC enters debug or stop mode while a command sequence is being executed, the NCF will become asserted if an empty external CBuffer is detected after debug/stop mode is exited.

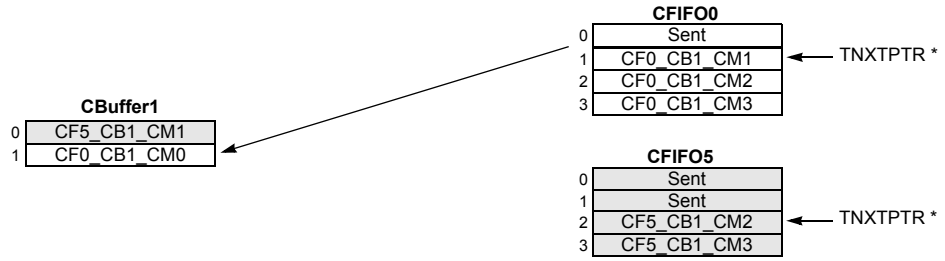
Figure 460. Non-coherency event when different CFIFOs use the same CBuffer



(a) CFIFO0 and CFIFO5 both have commands to be sent to CBuffer1, and both are not triggered



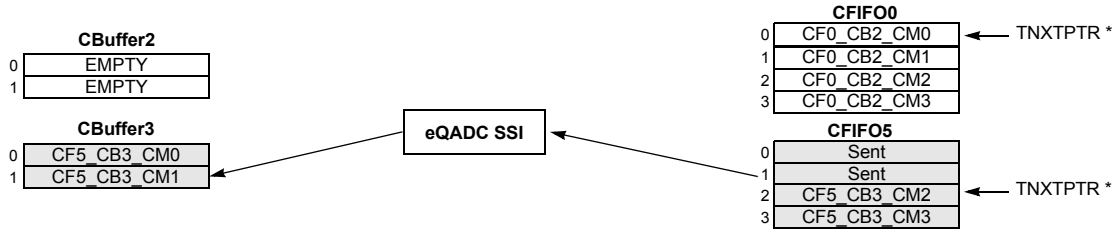
(b) CFIFO5 becomes triggered and transfers two commands to CBuffer1



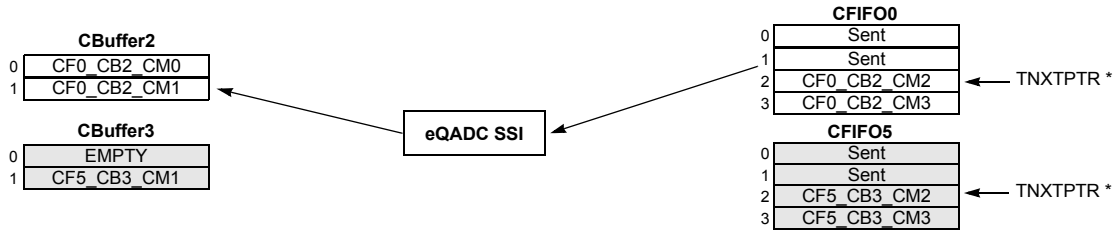
(c) CFIFO0 becomes triggered and transfers a command to CBuffer1. The sequence sent through CFIFO5 becomes non-coherent.

* TNXTPTR - Transfer Next Data Pointer
CF*x*_CB*a*_CM*n* - Command *n* in CFIFO*x* bound for CBuffer*a*

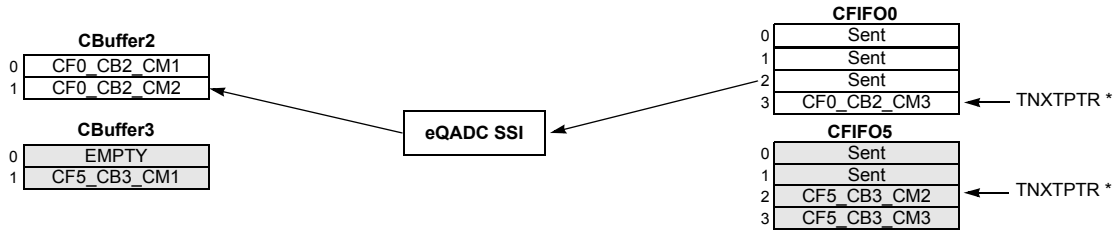
Figure 461. Non-coherency event when different CFIFOs are using different external CBuffers



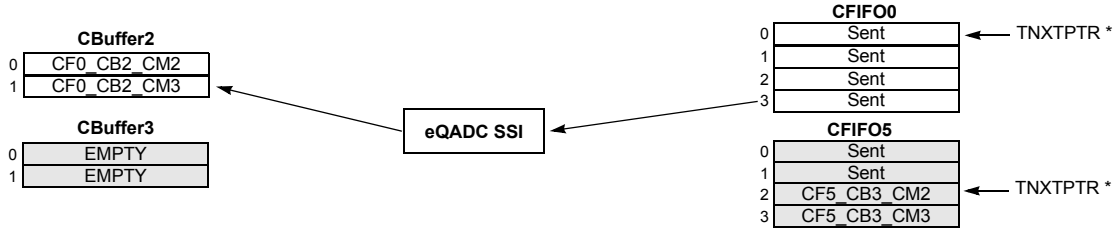
(a) CFIFO0 and CFIFO5 both have commands to be sent to external CBuffers. CFIFO0 is not triggered. CFIFO5 is triggered and sent two commands to CBuffer3



(b) CFIFO0 is triggered and sent two commands to CBuffer2. CFIFO5 cannot send commands to CBuffer3 because the eQADC SSI is busy transferring commands from CFIFO0. Execution of first command of CFIFO5 is completed.



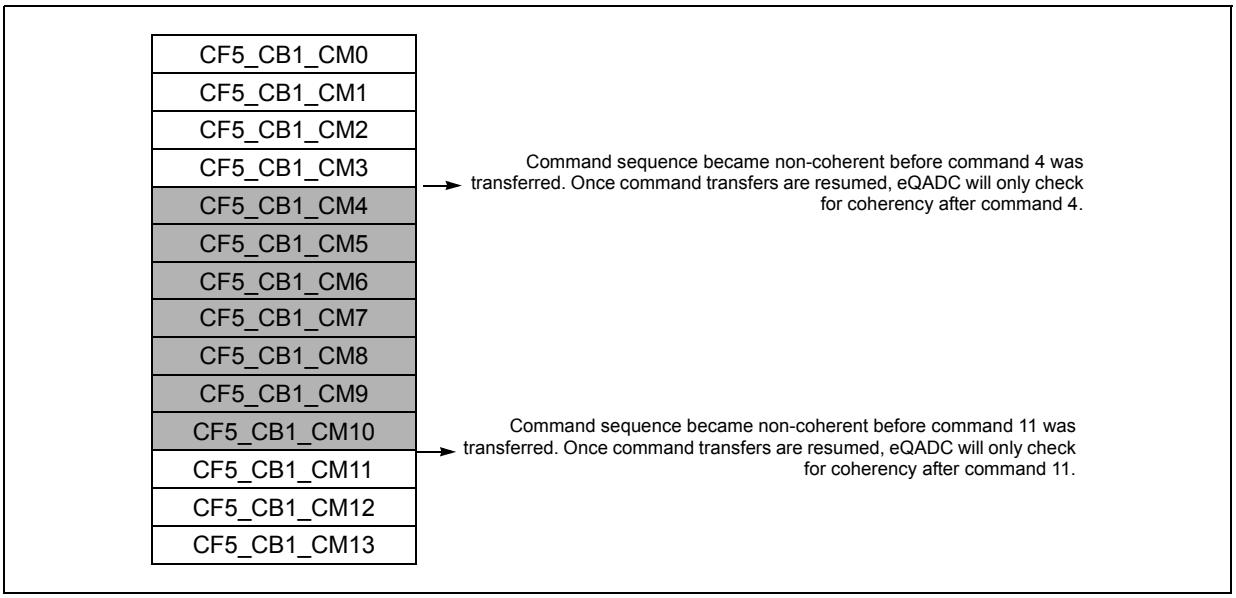
(c) Execution of first command of CFIFO0 is completed and CFIFO0 sends new command to CBuffer2.



(d) Second command in CBuffer3 completes. CBuffer3 became empty before the complete command sequence in CFIFO5 is sent to it. NCF5 becomes asserted when the eQADC receives an indication that CBuffer3 is empty, by the BUSY fields in the returning serial message.

* TNXTPTR - Transfer Next Data Pointer
CF*x*_CB*a*_CM*n* - Command *n* in CFIFO*x* bound for CBuffer*a*

Figure 462. Non-coherency detection when transfers from a command sequence are interrupted



23.5.5 eQADC result FIFOs

23.5.5.1 RFIFO basic functionality

There are six RFIFOs located in the eQADC. Each RFIFO is four entries deep, and each RFIFO entry is 16 bits long. Each RFIFO serves as a temporary storage location for the one of the RQueues allocated in system memory. Result data is saved in the RFIFOs before being moved into the system RQueues. When an RFIFO is not empty, the eQADC sets the corresponding RFDF bit in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR). If RFDE is asserted in the eQADC Interrupt and DMA Control Registers (EQADC_IDCR), the eQADC generates a request so that an RFIFO entry is moved to the RQueue. An interrupt request, served by the host CPU, is generated when RFDS is negated, and a DMA request, served by the DMAC, is generated when RFDS is asserted. The host CPU or the DMAC responds to these requests by reading the eQADC Result FIFO Pop Registers (EQADC_RFPR) to retrieve data from the RFIFO.

Note: The DMAC should be configured to read a single result (16-bit data) from the RFIFO pop registers for every asserted DMA request it acknowledges. Refer to Section 23.6.2: eQADC/DMAC interface for DMAC configuration guidelines.

Note: Reading a word, a half-word, or any bytes from EQADC_RFPRx will pop an entry from RFIFOx, and the RFCTRx field will be decremented by one.

Figure 463 describes the important components in the RFIFO. Each RFIFO is implemented as a circular set of registers to avoid the need to move all entries at each push/pop operation. The Pop Next Data Pointer always points to the next RFIFO message to be retrieved from the RFIFO when reading EQADC_RFPR. The Receive Next Data Pointer points to the next available RFIFO location for storing the next incoming message from the on-chip ADCs or from the external device. The *RFIFO Counter Logic* counts the number of entries in RFIFO and generates interrupt or DMA requests to drain the RFIFO.

POPNXTPTR in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) indicates which entry is currently being addressed by the Pop Next Data Pointer, and RFCTR, in the

same register, provides the number of entries stored in the RFIFO. Using POPNXTPTR and RFCTR, the absolute addresses for Pop Next Data Pointer and Receive Next Data Pointer can be calculated using the following formulas:

Pop Next Data Pointer Address =

RFIFOx_BASE_ADDRESS + POPNXTPTRx * 4

Receive Next Data Pointer Address =

RFIFOx_BASE_ADDRESS + [(POPNXTPTRx + RFCTRx) mod RFIFO_DEPTH] * 4

where

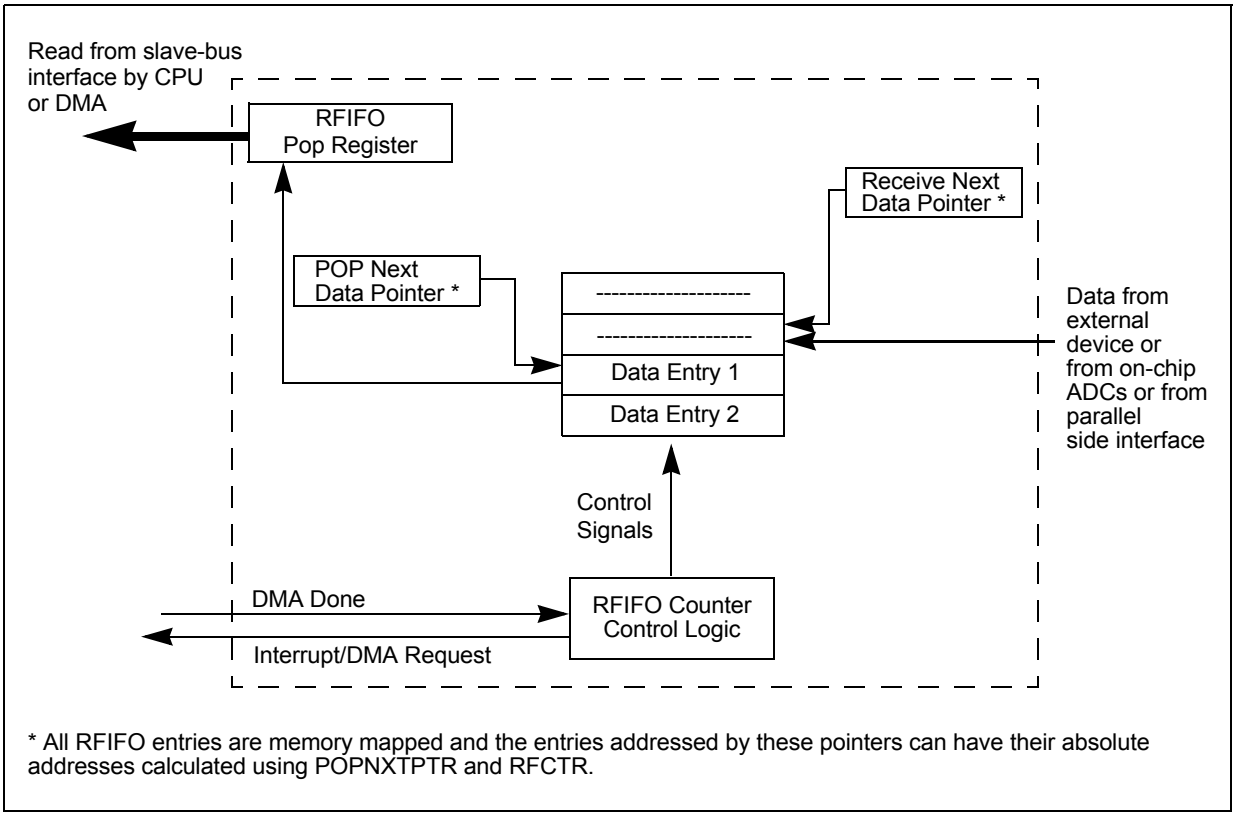
- *a mod b* returns the remainder of the division of *a* by *b*.
- RFIFOx_BASE_ADDRESS is the smallest memory mapped address allocated to an RFIFOx entry.
- RFIFO_DEPTH is the number of entries contained in a RFIFO—four in this implementation.

When a new message arrives and RFIFOx is not full, the eQADC copies its contents into the entry pointed by the Receive Next Data Pointer. The RFIFO counter RFCTRx in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR) is incremented by one, and the Receive Next Data Pointer x is also incremented by one (or wrapped around) to point to the next empty entry in RFIFOx. However, if the RFIFOx is full, the eQADC sets the RFOF in the eQADC FIFO and Interrupt Status Registers (EQADC_FISR). The RFIFOx will not overwrite the older data in the RFIFO, the new data will be ignored, and the Receive Next Data Pointer x is not incremented or wrapped around. RFIFOx is full when the Receive Next Data Pointer x equals the Pop Next Data Pointer x and RFCTRx is not zero. RFIFOx is empty when the Receive Next Data Pointer x equals the Pop Next Data Pointer x and RFCTRx is zero.

When the eQADC RFIFO Pop Register x is read and the RFIFOx is not empty, the RFIFO counter RFCTRx is decremented by one, and the POP Next Data Pointer is incremented by one (or wrapped around) to point to the next RFIFO entry.

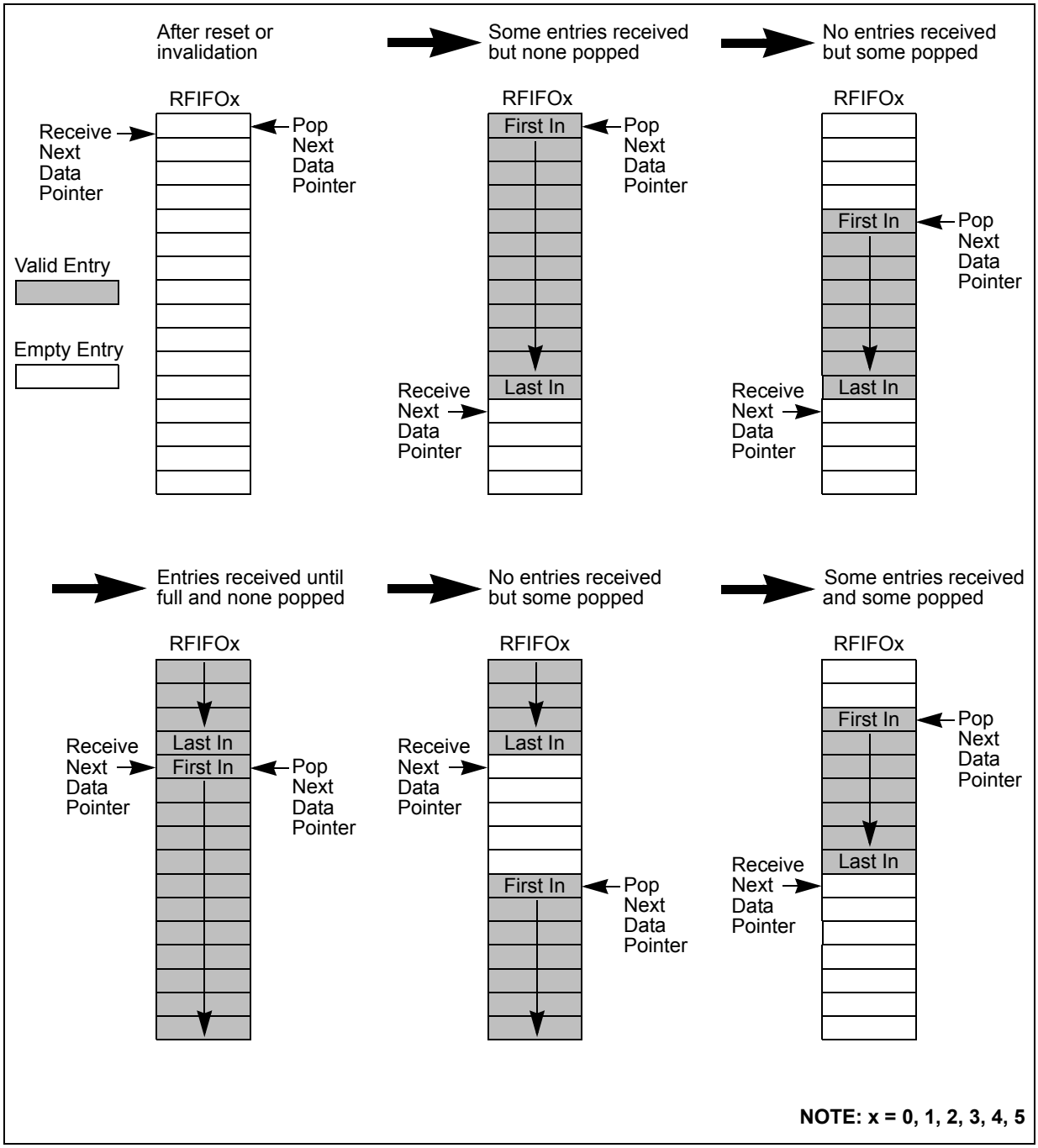
When the eQADC RFIFO Pop Register x is read and RFIFOx is empty, the eQADC will not decrement the counter value and the POP Next Data Pointer x will not be updated. The read value will be undefined.

Figure 463. RFIFO diagram



The detailed behavior of the Pop Next Data Pointer and Receive Next Data Pointer is described in the example shown in *Figure 464* where an RFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four entries. In this example, RFIFOx with 16 entries is shown in sequence after popping or receiving entries.

Figure 464. RFIFO entry pointer example



23.5.5.2 Distributing result data into RFIFOs

Data to be moved into the RFIFOs can come from four sources: from ADC0, from ADC1, from the external device or from the on-chip companion module through the PSI. All result data comes with a MESSAGE_TAG field and a DEST field defining what should be done

with the received data. The eQADC hardware decodes the MESSAGE_TAG and DEST fields and:

- stores the 16-bit data into the appropriate RFIFO if the MESSAGE_TAG indicates a valid RFIFO number, or;
- sends the 16-bit data, the MESSAGE_TAG and the DEST data through the PSI to an on-chip companion module (as a decimation filter), or;
- ignores the data in case of a null or “reserved for customer use” MESSAGE_TAG.

In general received data is moved into RFIFOs as they become available, while an exception happens when multiple results from different sources become available at the same time. In that case, result data from ADC0 is processed first, result data from ADC1 is only process after all ADC0 data is processed, result data from the external device is only processed after all data from ADC0/1 is processed, and finally returned data from companion module is only processed after all data from ADC0/1 and external device is processed.

When time-stamped results return from the on-chip ADCs, the conversion result and the time stamp are always moved to the RFIFOs in consecutive clock cycles in order to guarantee they are always stored in consecutive RFIFO entries.

23.5.6 On-chip ADC configuration and control

23.5.6.1 Enabling and disabling the on-chip ADCs

The on-chip ADCs have an enable bit (ADC0/1_EN) in the ADC0/1 Control Registers (ADC0_CR and ADC1_CR) which allows the enabling of the ADCs only when necessary. When the enable bit for an ADC is negated, the clock input to that ADC is stopped. The ADCs are disabled out of reset—ADC0/1_EN bits are negated—to allow for their safe configuration. The ADC must only be configured when its enable bit is negated. Once the enable bit of an ADC is asserted, clock input the ADC is started.

Note: Conversion commands sent to the CBuffer of a disabled ADC are ignored by the ADC control hardware.

Note: An 8 ms wait time from VDDA power up to enabling ADC is required to precharge the external 100 nF capacitor on REFBYPC pin. This time must be guaranteed by crystal startup time plus reset duration or user.

Note: Due to legacy reasons, the eQADC will always wait 120 ADC clocks before issuing the first conversion command following the enabling of one of on-chip ADCs, or the exiting of stop mode. There are two independent counters checking for this delay: one clocked by ADC0_CLK and another by ADC1_CLK. Conversion commands can start to be executed whenever one of these counters completes counting 120 ADC clocks.

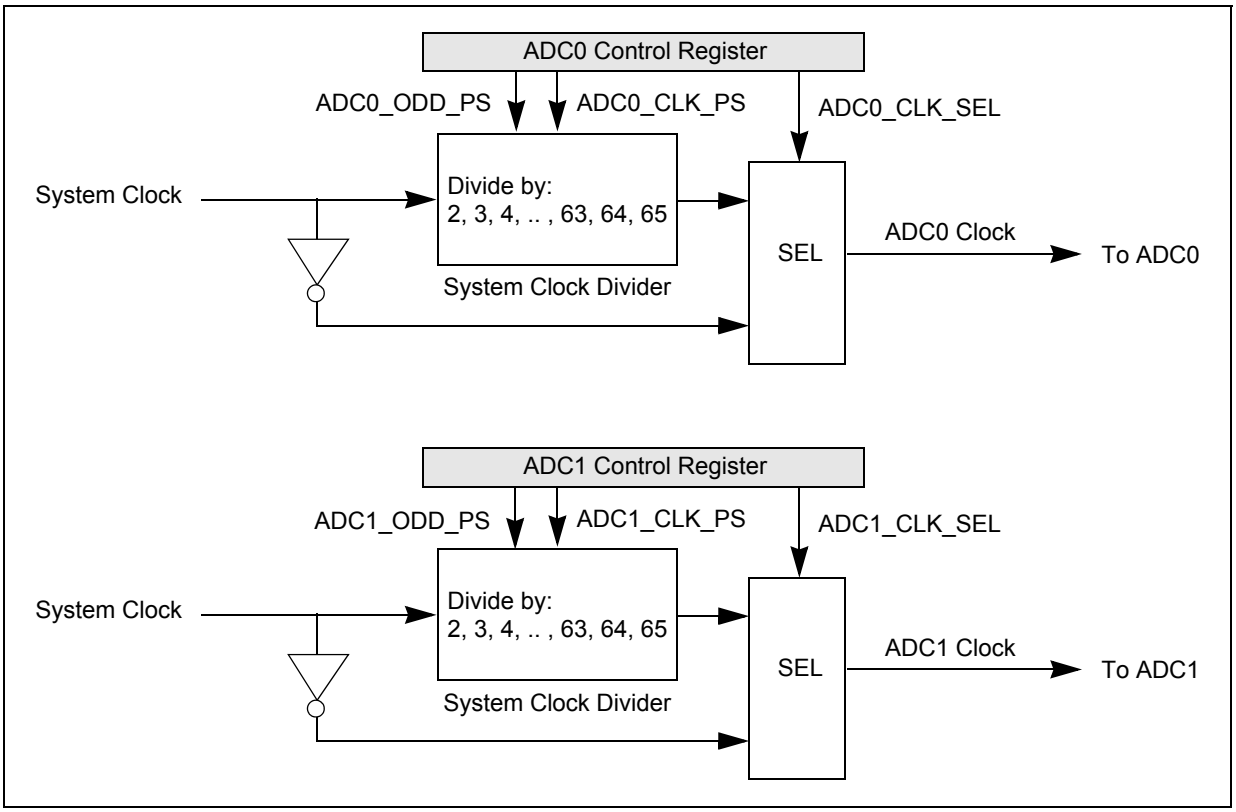
23.5.6.2 ADC clock and conversion speed

The clock input to the ADCs is defined by setting the ADC0/1_ODD_PS, the ADC0/1_CLK_SEL and the ADC0/1_CLK_PS fields in the ADC0/1 Control Registers (ADC0_CR and ADC1_CR). When the ADC0/1_CLK_SEL is set, the ADC clock frequency is the same as the system clock, but it has the inverted phase. When it is clear, the ADC0/1_ODD_PS and the ADC0/1_CLK_PS fields select the clock divide factor by which the system clock will be divided as shown in *Table 277*. The ADC clock frequency is calculated as below and it must not exceed 15 MHz. This is also the maximum frequency of system clock when the ADC0/1_CLK_SEL is asserted.

$$ADCClockFrequency = \frac{SystemClockFrequency(MHz)}{SystemClockDivideFactor}; (ADCClockFrequency \leq 15MHz)$$

Figure 465 depicts how the ADC clocks for ADC0 and ADC1 are generated.

Figure 465. ADC0/1 clock generation



The ADC conversion speed (in K samples per second—Ksps) is calculated by the following formula. The *number of sampling cycles* is determined by the LST bits in the command message—see *Section 23.5.2.3.1.1: Conversion command format for the standard configuration*—and it can take one of the following values: 2, 8, 64, or 128 ADC clock cycles. The *number of AD conversion cycles* is 13 for differential conversions and 14 for single-ended conversions. The maximum conversion speed is achieved when the ADC Clock frequency is set to its maximum (15 MHz) and the number of sampling cycles set to its minimum (2 cycles). The maximum conversion speed for differential and single-ended conversions are 1 Msps and 937.5 Ksps, respectively.

$$ADCConversionSpeed = \frac{ADCClockFrequency(MHz)}{(NumberOfSamplingCycles + NumberOfADConversionCycles)}$$

Table 298 shows an example of how the ADC0/1_CLK_PS can be set when using a 120 MHz system clock and the corresponding conversion speeds for all possible ADC clock frequencies. The table also shows that according to the system clock frequency, certain clock divide factors are invalid (2, 4, 6, 8 clock divide factors in the example) since their use

would result in a ADC clock frequency higher than the maximum one supported by the ADC. ADC clock frequency must not exceed 15 MHz.

Table 298. ADC clock configuration example (system clock frequency = 120 MHz)

ADC0/1_CLK_PS [0:4]	ADC0/1_ ODD_PS	System clock divide factor	ADC clock (system clock = 120 MHz)	Conversion speed with default sampling time (2 cycles)	
				Differential	Single-ended
0b00000	0	2	—	—	—
	1	3	—	—	—
0b00001	0	4	—	—	—
	1	5	—	—	—
0b00010	0	6	—	—	—
	1	7	—	—	—
0b00011	0	8	15.0 MHz	1.0 Msps	938 Ksps
	1	9	13.3 MHz	889 Ksps	833 Ksps
0b00100	0	10	12.0 MHz	800 Ksps	750 Ksps
	1	11	10.9 MHz	727 Ksps	682 Ksps
0b00101	0	12	10.0 MHz	667 Ksps	625 Ksps
	1	13	9.23 MHz	615 Ksps	577 Ksps
0b00110	0	14	8.57 MHz	571 Ksps	536 Ksps
	1	15	8.0 MHz	533 Ksps	500 Ksps
0b00111	0	16	7.5 MHz	500 Ksps	469 Ksps
	1	17	7.06 MHz	471 Ksps	441 Ksps
0b01000	0	18	6.67 MHz	444 Ksps	417 Ksps
	1	19	6.32 MHz	421 Ksps	395 Ksps
0b01001	0	20	6.0 MHz	400 Ksps	375 Ksps
	1	21	5.71 MHz	381 Ksps	357 Ksps
0b01010	0	22	5.45 MHz	364 Ksps	341 Ksps
	1	23	5.22 MHz	348 Ksps	326 Ksps
0b01011	0	24	5.0 MHz	333 Ksps	313 Ksps
	1	25	4.80 MHz	320 Ksps	300 Ksps
0b01100	0	26	4.62 MHz	308 Ksps	288 Ksps
	1	27	4.44 MHz	296 Ksps	278 Ksps
0b01101	0	28	4.29 MHz	286 Ksps	268 Ksps
	1	29	4.14 MHz	276 Ksps	259 Ksps
0b01110	0	30	4.0 MHz	267 Ksps	250 Ksps
	1	31	3.87 MHz	258 Ksps	242 Ksps

Table 298. ADC clock configuration example (system clock frequency = 120 MHz) (continued)

ADC0/1_CLK_PS [0:4]	ADC0/1_ ODD_PS	System clock divide factor	ADC clock (system clock = 120 MHz)	Conversion speed with default sampling time (2 cycles)	
				Differential	Single-ended
0b01111	0	32	3.75 MHz	250 Ksps	234 Ksps
	1	33	3.64 MHz	242 Ksps	227 Ksps
0b10000	0	34	3.53 MHz	235 Ksps	221 Ksps
	1	35	3.43 MHz	229 Ksps	214 Ksps
0b10001	0	36	3.33 MHz	222 Ksps	208 Ksps
	1	37	3.24 MHz	216 Ksps	203 Ksps
0b10010	0	38	3.16 MHz	211 Ksps	198 Ksps
	1	39	3.08 MHz	205 Ksps	192 Ksps
0b10011	0	40	3.0 MHz	200 Ksps	188 Ksps
	1	41	2.93 MHz	195 Ksps	183 Ksps
0b10100	0	42	2.86 MHz	190 Ksps	179 Ksps
	1	43	2.79 MHz	186 Ksps	174 Ksps
0b10101	0	44	2.73 MHz	182 Ksps	170 Ksps
	1	45	2.67 MHz	178 Ksps	167 Ksps
0b10110	0	46	2.61 MHz	174 Ksps	163 Ksps
	1	47	2.55 MHz	170 Ksps	160 Ksps
0b10111	0	48	2.5 MHz	167 Ksps	156 Ksps
	1	49	2.45 MHz	163 Ksps	153 Ksps
0b11000	0	50	2.4 MHz	160 Ksps	150 Ksps
	1	51	2.35 MHz	157 Ksps	147 Ksps
0b11001	0	52	2.31 MHz	154 Ksps	144 Ksps
	1	53	2.26 MHz	151 Ksps	142 Ksps
0b11010	0	54	2.22 MHz	148 Ksps	139 Ksps
	1	55	2.18 MHz	145 Ksps	136 Ksps
0b11011	0	56	2.14 MHz	143 Ksps	134 Ksps
	1	57	2.11 MHz	140 Ksps	132 Ksps
0b11100	0	58	2.07 MHz	138 Ksps	129 Ksps
	1	59	2.03 MHz	136 Ksps	127 Ksps
0b11101	0	60	2.0 MHz	133 Ksps	125 Ksps
	1	61	1.97 MHz	131 Ksps	123 Ksps
0b11110	0	62	1.94 MHz	129 Ksps	121 Ksps
	1	63	1.90 MHz	127 Ksps	119 Ksps

Table 298. ADC clock configuration example (system clock frequency = 120 MHz) (continued)

ADC0/1_CLK_PS [0:4]	ADC0/1_ ODD_PS	System clock divide factor	ADC clock (system clock = 120 MHz)	Conversion speed with default sampling time (2 cycles)	
				Differential	Single-ended
0b11111	0	64	1.88 MHz	125 Ksps	117 Ksps
	1	65	1.85 MHz	123 Ksps	115 Ksps

23.5.6.3 Time stamp feature

The on-chip ADCs can provide a time stamp for the conversions they execute. A time stamp is the value of the time base counter latched when the eQADC detects the end of the analog input voltage sampling. A time stamp for a conversion command is requested by setting the TSR bit in the corresponding command. When TSR is negated, that is a time stamp is not requested, the ADC returns a single result message containing the conversion result. When TSR is asserted, that is a time stamp is requested, the ADC returns two result messages; one containing the conversion result, and afterwards another containing the time stamp for that conversion. The result messages are sent in this order to the RFIFOs and both messages are sent to the same RFIFO was specified in the MESSAGE_TAG field of the executed conversion command.

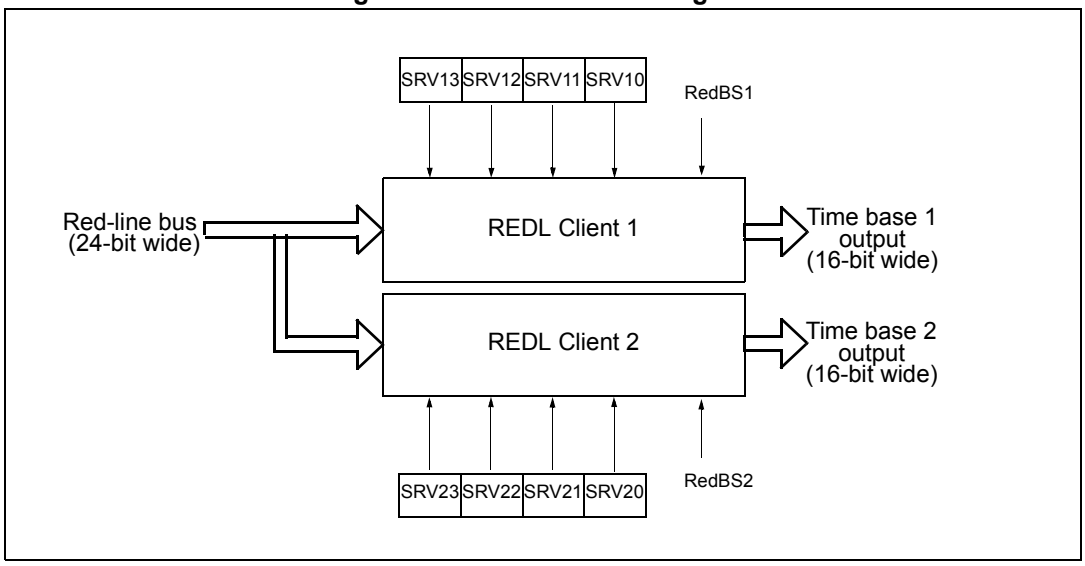
The time stamp can be provided by an external source using the STAC interface (more details in *Section 23.5.6.3.1: STAC client submodule (REDLC)*) or by the internal time base counter. The selection between the two sources is done by field ADC0/1_TBSEL in the ADC0/1_CR or by field ATBSEL in registers ADC_ACR1–8. Refer to *Table 276* and *Table 283* for selection details.

The time base counter is a 16-bit up counter that wraps after reaching 0xFFFF. It is disabled after reset and it is enabled according to the setting of TBC_CLK_PS field in the ADC Time Stamp Control Register (ADC_TSCR). TBC_CLK_PS defines if the counter is enabled or disabled, and, if enabled, at what frequency it is incremented. The time stamps are returned regardless of whether the time base counter is enabled or disabled. The time base counter can be reset by writing 0x0000 to the ADC Time Base Counter Registers (ADC_TBCR) with a write configuration command.

23.5.6.3.1 STAC client submodule (REDLC)

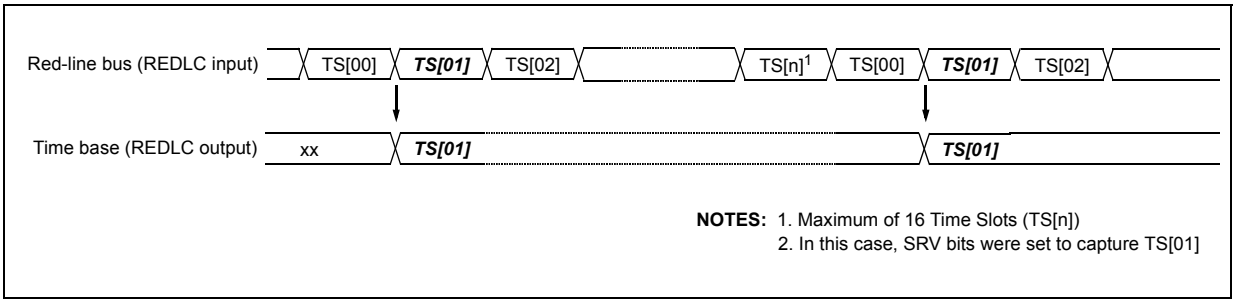
The STAC Client sub-block REDLC extracts the external time base from the STAC bus to the eQADC. *Figure 466* provides a block diagram for the REDLC sub-module that contains actually two slot selectors.

Figure 466. REDLC block diagram



The STAC data stream is composed of several identified time slots. Each time slot contains a time value from a specified source. The bits SRV1/2[0:3] in the EQADC_REDLCCR are used to select the desired time slots of the Red-line bus to be used internally by the eQADC. *Figure 467* shows a timing diagram for the REDLC.

Figure 467. Timing diagram for the STAC bus and REDLC output



Every time the selected time slot change, the REDLC outputs are updated. As this time base is an external data to the eQADC, this output is not affected by the stop or by the debug internal state.

After the slot selection is done and the timebase data is extracted, the REDBS1/2 bits select 16 bits from the original 24-bit timebase data. These selected bits are the timebase to be used internal to the eQADC.

23.5.6.4 ADC pre-gain feature

Each ADC can be configured to have a selectable input gain as defined in *Section 23.4.3.6: Alternate Configuration 1–8 Control Registers (ADC_ACR1–8)*. This means the input signal is sampled and the result is amplified by factor 2, or 4 before the conversion phase. In present implementation of this feature, the conversion is 1 or 2 ADC clock cycles longer for gain 2 or gain 4, respectively.

Note: This feature is valid only for differential channels.

23.5.6.5 ADC resolution selection feature

The ADC conversion resolutions can be 8 bits, 10 bits or 12 bits as described in *Section 23.4.3.6: Alternate Configuration 1–8 Control Registers (ADC_ACR1–8)*. For conversions at a resolution less than 12, the ADC is executing less operations and the conversion time is smaller. In this ADC, it is verified that there is 1 ADC clock cycle for each bit of resolution. Therefore, for the same ADC clock frequency, the ADC sample frequency is higher for lower resolutions.

When a conversion is undertaken at a resolution less than 12, the result is presented by the ADC in right justified format in the 12-bit input bus, for example 0000xxxxxxxx for 8 bits and 00xxxxxxxxxx for 10 bits. The eQADC inverts the result to left justified format, that is xxxxxxxx0000 for 8 bits and xxxxxxxxxx00 for 10 bits. This is because the same calibration coefficients in the MAC can then be used. The left shift operation is done just after the conversion result enters the eQADC, in the Resolution Adjustment block prior to the MAC, as illustrated in *Figure 470*.

23.5.6.6 ADC calibration feature

23.5.6.6.1 Overview

There are three sets of calibration coefficients for each ADC. Each set is composed of a gain factor and an offset factor: GCCn/OCCn, ALTGCCn1/ALTGCCn1, and ALTGCCn2/ALTGCCn2, where n is the ADC number 0 or 1. The pair GCCn/OCCn is selected when it is used the normal configuration or the alternate configurations 3 to 8. The pair ALTGCCn1/ALTGCCn1 is used only when the alternate configuration 1 is selected. And the pair ALTGCCn2/ALTGCCn2 is for the alternate configuration 2. The description below is for a generic pair of gain/offset GCC/OCC.

The eQADC provides a calibration scheme to remove the effects of gain and offset errors from the results generated by the on-chip ADCs. Only results generated by the on-chip ADCs are calibrated. The results generated by ADCs on the external device are directly sent to RFIFOs unchanged. The main component of calibration hardware is a Multiply-and-Accumulate (MAC) unit, one per on-chip ADC, that is used to calculate the following transfer function which relates a calibrated result to a raw, uncalibrated one.

CAL_RES = GCC * RAW_RES + OCC+2;

where:

- CAL_RES is the calibrated result corresponding the input voltage V_i.
- GCC is the gain calibration constant.
- RAW_RES is the raw, uncalibrated result with resolution adjustment corresponding to an specific input voltage V_i.
- OCC is the offset calibration constant.
- The addition of two reduces the maximum quantization error of the ADC. See *Section 23.6.6.3: Quantization error reduction during calibration*.

Calibration constants GCC and OCC are determined by taking two samples of known reference voltages and using these samples to calculate the values for the constants. For details and an example about how to calculate the calibration constants and use them in result calibration refer to *Section 23.6.6: ADC result calibration*. Once calculated, GCC is stored in the ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR), and OCC in the ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR), from where their values are fed to the MAC unit. The

alternate gain values are stored in the ADC0/1 Alternate Gain Registers (ADC0_AGR1–2 and ADC1_AGR1–2), and the alternate offset values in the ADC0/1 Alternate Offset Register (ADC0_AOR1–2 and ADC1_AOR1–2). Since the analog characteristics of each on-chip ADCs differs, each ADC has an independent pair of calibration constants.

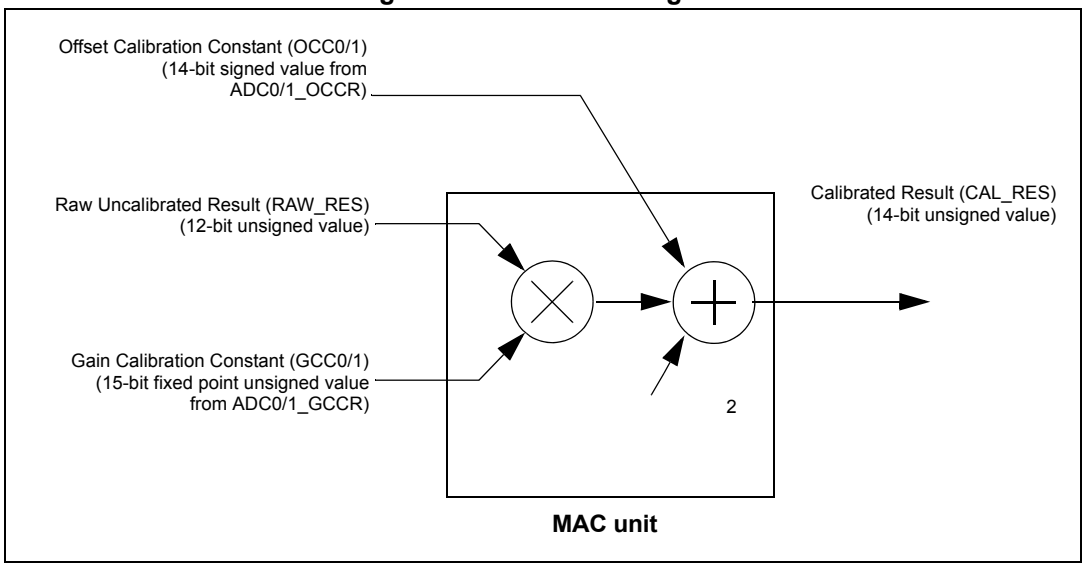
A conversion result is calibrated according to the status of CAL bit in the command that initiated the conversion. If the CAL bit is asserted, the eQADC will automatically calculate the calibrated result before sending the result to the appropriate RFIFO or companion module. If the CAL bit is negated, the result is not calibrated, it bypasses the calibration hardware, and is directly sent to the appropriate RFIFO or companion module.

23.5.6.6.2 MAC unit and operand data format

The MAC unit diagram is shown in *Figure 468*. Each on-chip ADC has a separate MAC unit to fine-tune its conversion results. The description below considers the general calibration constant registers but it is the same for the alternate calibration constants.

The OCC0/1 operand is a 14-bit signed value stored in the ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR). The RAW_RES operand is the raw uncalibrated result, and it is the direct output from the on-chip ADCs but passing through the resolution adjustment block. The GCC0/1 operand is a 15-bit fixed point unsigned value stored in the ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR). The GCC is expressed in the *GCC_INT.GCC_FRAC* binary format. The integer part of the GCC (GCC_INT = GCC[1]) contains a single binary digit while its fractional part (GCC_FRAC = GCC[2:15]) contains 14 bits—see *Figure 469*. The gain constant equivalent decimal value ranges from 0 to 1.999938..., as shown in *Table 300*. Two is always added to the MAC output—see *Section 23.6.6.3: Quantization error reduction during calibration*. CAL_RES output is the calibrated result, and it is a 14-bit unsigned value. CAL_RES is truncated to 0x3FFF, in case of a overflow, and to 0x0000, in case of an underflow.

Figure 468. MAC unit diagram



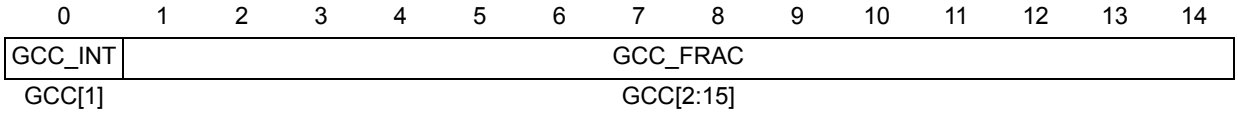


Figure 469. Gain calibration constant format

Table 299. Gain calibration constant format field descriptions

Field	Description
GCC_INT	Integer part of the gain calibration constant for ADC0/1 GCC_INT is the integer part of the gain calibration constant for ADC0/1.
GCC_FRAC [1:14]	Fractional part of the gain calibration constant for ADC0/1 GCC_FRAC is the fractional part of the gain calibration constant for ADC0/1. GCC_FRAC can express decimal values ranging from 0 to 0.999938... (see *Table 300*).

Table 300. Binary and decimal representations of the gain constant

Gain constant (GCC_INT.GCC_FRAC binary format)	Corresponding decimal value
0.0000_0000_0000_00	0
...	...
0.1000_0000_0000_00	0.5
...	...
0.1111_1111_1111_11	0.999938...
1.0000_0000_0000_00	1
...	...
1.1100_0000_0000_00	1.75
...	...
1.1111_1111_1111_11	1.999938...

23.5.6.7 *ADC Control Logic* overview and command execution

Figure 470 shows the basic logic blocks involved in the ADC Control and how they interact. CFIFOs/RFIFOs interact with CBuffers/*Abort Cont*/*Result Message Return Logic* through the *FIFO Control Unit*. The EB and BN bits in the Command Message uniquely identify the CBuffer to which a command should be sent. The *FIFO Control Unit* decodes these bits and sends the ADC command to the proper CBuffer. Other blocks of logic are the *Resolution Adjustment*, *Result Format and Calibration Sub-Block*, the *Time Stamp Logic,* and the *MUX Control Logic*.

The *Resolution Adjustment Sub-Block* receives the 12-bit data bus directly from the ADC and changes the received conversion results from right aligned format of ADC to the left aligned format depending on the selected resolution of the conversion. This operation helps the calibration processing to use the calibration coefficients always with the same format.

The *Result Format and Calibration Sub-Block* formats the returning data into Result Messages and sends them to the RFIFOs(s). The returning data can be data read from an

ADC register, a conversion result, or a time stamp. The formatting and calibration of conversion results also take place inside this sub-block.

The *Time Stamp Logic* latches the value of the time base counter or the STAC time base when detecting the end of the analog input voltage sampling, and sends it to the *Result Format and Calibration Sub-Block* as time stamp information.

The *MUX Control Logic* generates the proper MUX control signals and, when the ADC0/1_EMUX bits are asserted, the MA signals based on the channel numbers extracted from the ADC Command.

When the on-chip ADC abort feature is not enabled, ADC Commands are stored in the CBuffers as they come and they are executed in the first-in-first-out basis. After the execution of a command in ENTRY1 finishes all commands are shifted one entry. After the shift, ENTRY0 is always empty and ready to receive a new command. Execution of configuration commands only start when they reach ENTRY1. Consecutive conversion commands are pipelined and their execution can start while in ENTRY0. This is explained below.

AD conversion accuracy can be affected by the settling time of the input channel multiplexers. Some time is required for the channel multiplexers' internal capacitances to settle after the channel number is changed. If the time prior to sampling is not long enough to absorb this settling, then the settling time will shorten ADC sampling time which may result in inaccurate sampling and ultimately compromise conversion result accuracy—see *Figure 471* (a). This could be avoided by switching the multiplexers in preparation for the next command's sampling during the AD conversion phase of the current command as shown in *Figure 471* (b). In eQADC, this is done in the following way: when a conversion command is in buffer ENTRY1 and another conversion command is identified in ENTRY0, then the channel number of ENTRY0 is sent to the *MUX Control Logic* some cycles before the sampling phase of the command in ENTRY0 starts. In this way, sampling for the next command can promptly start after the current conversion finishes because the internal capacitance of the multiplexers will be settled by that time, allowing for more accurate sampling. This is especially important for applications that require high conversion speeds, that is with the ADC running at maximum clock frequency and with the analog input voltage sampling time set to a minimum (2 ADC clock cycles), when the short sampling time does not allow the multiplexers to completely settle. The second advantage of pipelining conversion commands is that precise conversion intervals are provided, which means the time intervals between two consecutive conversions are the same. This is important for any digital signal process application.

When the on-chip ADC abort feature is enabled, ADC Commands from CFIFO0 should be considered immediately, even stopping the execution of some command that is already in ENTRY1. When the abort request is sent to the ADC, the already stored commands in the CBuffers are copied in a temporary set of registers. The first ADC command from CFIFO0 is sent after the abort acknowledge indication from ADC. The process is the same as usual until the transfer of the last command from CFIFO0. Then the temporarily stored commands that were postponed by the abortion are recovered and they are pipelined for execution. After the last command from this temporary memory is transferred, the next commands are pipelined from the CFIFOs.

s. The result messages may also be routed to an on-chip companion module via the side interface, and then fed back to the RFIFOs.

Figure 470. On-chip ADC control scheme

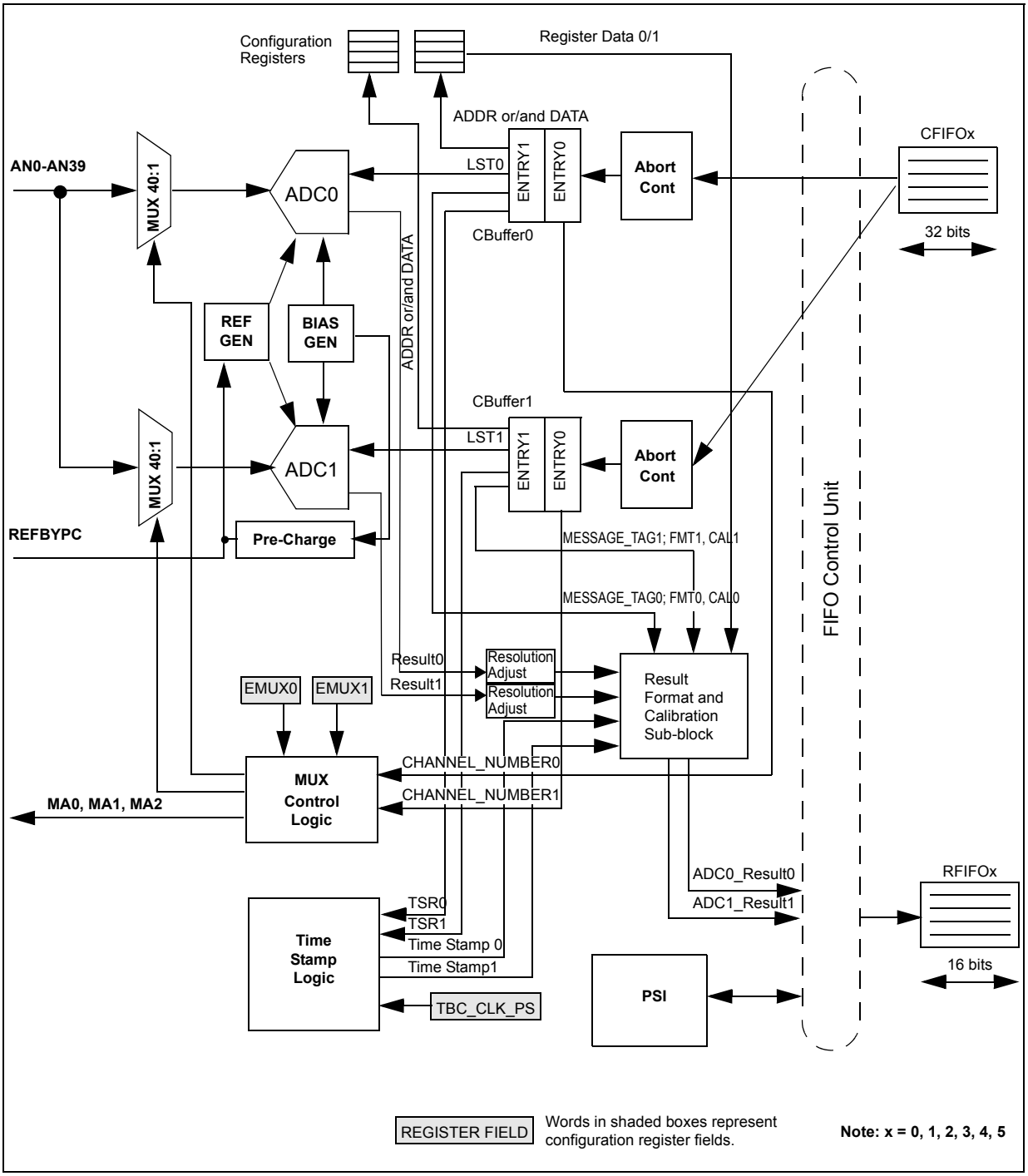
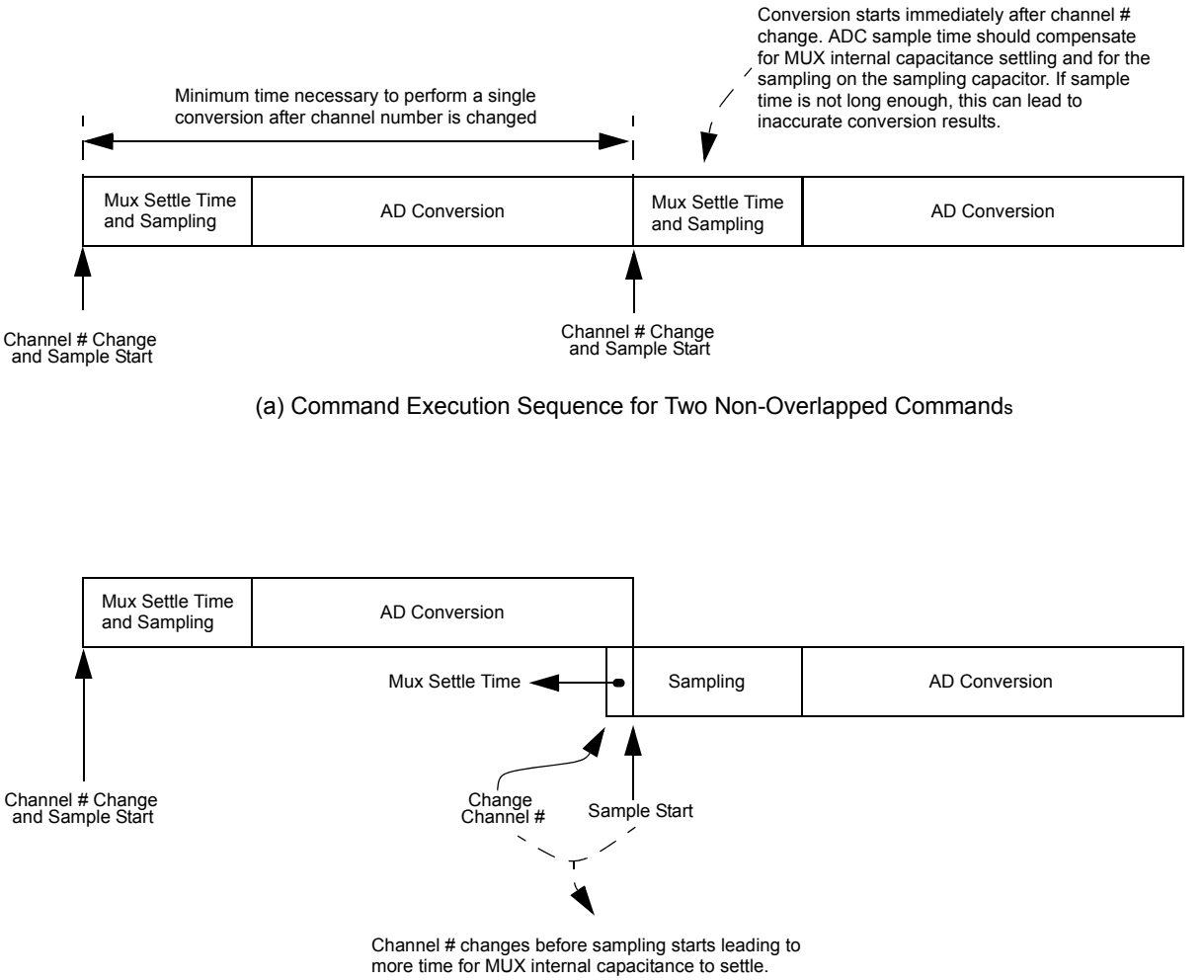


Figure 471. Overlapping consecutive conversion commands



23.5.7 Internal/External multiplexing

23.5.7.1 Channel assignment

The internal analog multiplexers select one of the 40 analog input pins for conversion, based on the CHANNEL_NUMBER field of a Command Message. The analog input pin channel number assignments and the pin definitions vary depending on how the ADC0/1_EMUX are configured. Allowed combinations of ADC0/1_EMUX bits are shown in *Table 301* together with references to tables indicating how CHANNEL_NUMBER field of each conversion command must be set to avoid channel selection conflicts.

During differential conversions the analog multiplexer passes differential signals to both the positive and negative terminals of the ADC. The differential conversions can only be initiated on four channels: DAN0, DAN1, DAN2, and DAN3. Refer to *Table 302* and *Table 303* for the channel numbers used to select differential conversions.

Table 301. ADC0/1_EMUX bit combinations

ADC0_EMUX	ADC1_EMUX	CHANNEL_NUMBER should be set as in	
		ADC0	ADC1
0	0	Refer to *Table 302*	Refer to *Table 302*
0	1	Refer to *Table 302*	Refer to *Table 303*
1	0	Refer to *Table 303*	Refer to *Table 302*
1	1	Reserved(1)	

1. ADC0_EMUX and ADC1_EMUX must not be asserted at the same time.

Table 302 shows the channel number assignments for the non-multiplexed mode. The 43 single-ended channels and 4 differential pairs are shared between the two ADCs.

Table 302. Non-multiplexed channel assignments(1)

Input pins			ADC	Channel number in CHANNEL_NUMBER field	
Analog pin name	Other functions	Conversion type		Binary	Decimal
AN0 to AN39		Single-ended	ADC0/ADC1	0000_0000 to 0010_0111	0 to 39
VRH	—	Single-ended	ADC0/ADC1	0010_1000	40
VRL	—	Single-ended	ADC0/ADC1	0010_1001	41
—	50% VREF(2)	Single-ended	ADC0/ADC1	0010_1010	42
—	75% VREF(2)	Single-ended	ADC0/ADC1	0010_1011	43
—	25% VREF(2)	Single-ended	ADC0/ADC1	0010_1100	44
INA_ADC0/1_0	Bandgap	Single-ended	ADC0/ADC1	0010_1101	45
Reserved				0010_1110 to 0101_1111	46 to 95
DAN0+ and DAN0–	—	Differential	ADC0/ADC1	0110_0000	96
DAN1+ and DAN1–	—	Differential	ADC0/ADC1	0110_0001	97
DAN2+ and DAN2–	—	Differential	ADC0/ADC1	0110_0010	98
DAN3+ and DAN3–	—	Differential	ADC0/ADC1	0110_0011	99
Reserved				0110_0100 to 0111_1111	100 to 127
INA_ADC0/1_1	Temperature sensor	Single-ended	ADC0/ADC1	1000_0000	128
INA_ADC0/1_2	Spare	Single-ended	ADC0/ADC1	1000_0001	129
Reserved				1000_0010 to 1010_0001	130 to 161

Table 302. Non-multiplexed channel assignments(1) (continued)

Input pins			ADC	Channel number in CHANNEL_NUMBER field	
Analog pin name	Other functions	Conversion type		Binary	Decimal
Reserved			ADC1	1010_0010 to 1010_0111	162 to 167
INA_ADC0_3	VDDEH1B 50%	Single-ended	ADC0	1010_0010	162
INA_ADC0_4	VDDEH1B 50%	Single-ended	ADC0	1010_0011	163
INA_ADC0_5	VDDEH6A 50%	Single-ended	ADC0	1010_0100	164
INA_ADC0_6	VDDEH6A 50%	Single-ended	ADC0	1010_0101	165
INA_ADC0_7	VDDEH6B 50%	Single-ended	ADC0	1010_0110	166
INA_ADC0_8	VDDEH7 50%	Single-ended	ADC0	1010_0111	167
Reserved				1010_1000 to 1100_0001	168 to 193
Reserved			ADC0	1100_0010 to 1100_0111	194 to 199
INA_ADC1_3	Reserved	Single-ended	ADC1	1100_0010	194
INA_ADC1_4	Reserved	Single-ended	ADC1	1100_0011	195
INA_ADC1_5	VDD33	Single-ended	ADC1	1100_0100	196
INA_ADC1_6	VDD33	Single-ended	ADC1	1100_0101	197
INA_ADC1_7	VDD12	Single-ended	ADC1	1100_0110	198
INA_ADC1_8	VDDEH1A 50%	Single-ended	ADC1	1100_0111	199
Reserved				1100_1000 to 1111_1111	200 to 255

1. The two on-chip ADCs can access the same analog input pins but simultaneous conversions are not allowed. Also, when one ADC is performing a differential conversion on a pair of pins, the other ADC must not access either of these two pins as single-ended channels.
2. VREF = VRH – VRL. After calibration, the 50% reference point will read approximately 20 mV below 50%(VRH - VRL). The 50% reference point should not be used to calibrate the EQADC. Use the 25% and 75% reference points to calibrate the EQADC.

Table 303 shows the channel number assignments for multiplexed mode. The ADC with the ADC0/1_EMUX bit asserted can access 4 differential pairs, 39 single-ended, and, at most, 64 externally multiplexed channels. Refer to *Section 23.5.7.1.1: External multiplexing* for a detailed explanation about how external multiplexing can be achieved.

Table 303. Multiplexed channel assignments(1)

Input pins			ADC	Channel number in CHANNEL_NUMBER field	
Analog pin name	Other functions	Conversion type		Binary	Decimal
AN0 to AN39(2)	—	Single-ended	ADC0/ADC1	0000_0000 to 0010_0111	0 to 39

Table 303. Multiplexed channel assignments[1] (continued)

Input pins			ADC	Channel number in CHANNEL_NUMBER field	
Analog pin name	Other functions	Conversion type		Binary	Decimal
VRH	—	Single-ended	ADC0/ADC1	0010_1000	40
VRL	—	Single-ended	ADC0/ADC1	0010_1001	41
—	50% VREF[3]	Single-ended	ADC0/ADC1	0010_1010	42
—	75% VREF[3]	Single-ended	ADC0/ADC1	0010_1011	43
—	25% VREF[3]	Single-ended	ADC0/ADC1	0010_1100	44[4]
INA_ADC0/1_0	Bandgap	Single-ended	ADC0/ADC1	0010_1101	45
Reserved				0010_1110 to 0011_1111	46 to 63
ANW	—	Single-ended	ADC0/ADC1	0100_0xxx	64 to 71
ANX	—	Single-ended	ADC0/ADC1	0100_1xxx	72 to 79
ANY	—	Single-ended	ADC0/ADC1	0101_0xxx	80 to 87
ANZ	—	Single-ended	ADC0/ADC1	0101_1xxx	88 to 95
DAN0+ and DAN0–	—	Differential	ADC0/ADC1	0110_0000	96
DAN1+ and DAN1–	—	Differential	ADC0/ADC1	0110_0001	97
DAN2+ and DAN2–	—	Differential	ADC0/ADC1	0110_0010	98
DAN3+ and DAN3–	—	Differential	ADC0/ADC1	0110_0011	99
Reserved				0110_0100 to 0111_1111	100 to 127
INA_ADC0/1_1	Temperature sensor	Single-ended	ADC0/ADC1	1000_0000	128
INA_ADC0/1_2	Spare	Single-ended	ADC0/ADC1	1000_0001	129
Reserved				1000_0010 to 1000_1111	130 to 143
Reserved			ADC1	1001_0000 to 1001_0011	144 to 147
INA_ADC0_3	Buffered Bandgap	Single-ended	ADC0	1001_0000	144
DAN4+	Reference Voltage for 1.2V LVD	Differential	ADC0	1001_0001	145
INA_ADC0_4	Reference for 1.2V Regulator	Single-ended	ADC0	1001_0010	146
DAN5+	Reference Voltage for 3.3V LVD	Differential	ADC0	1001_0011	147
Reserved				1001_0100 to 1010_0001	148 to 161
Reserved			ADC1	1010_0010 to 1010_0111	162 to 167
INA_ADC0_5	VDDEH1B 50%	Single-ended	ADC0	1010_0010	162
INA_ADC0_6	VDDEH1B 50%	Single-ended	ADC0	1010_0011	163
INA_ADC0_7	VDDEH6A 50%	Single-ended	ADC0	1010_0100	164
INA_ADC0_8	VDDEH6A 50%	Single-ended	ADC0	1010_0101	165

Table 303. Multiplexed channel assignments(1) (continued)

Input pins			ADC	Channel number in CHANNEL_NUMBER field	
Analog pin name	Other functions	Conversion type		Binary	Decimal
INA_ADC0_9	VDDEH6B 50%	Single-ended	ADC0	1010_0110	166
INA_ADC0_10	VDDEH7 50%	Single-ended	ADC0	1010_0111	167
Reserved				1010_1000 to 1100_0001	168 to 193
Reserved			ADC0	1100_0010 to 1100_0111	194 to 199
INA_ADC1_3	Reserved	Single-ended	ADC1	1100_0010	194
INA_ADC1_4	Reserved	Single-ended	ADC1	1100_0011	195
INA_ADC1_5	VDD33	Single-ended	ADC1	1100_0100	196
INA_ADC1_6	VDD33	Single-ended	ADC1	1100_0101	197
INA_ADC1_7	VDD12	Single-ended	ADC1	1100_0110	198
INA_ADC1_8	VDDEH1A 50%	Single-ended	ADC1	1100_0111	199
Reserved				1100_1000 to 1101_1111	200 to 223

1. The two on-chip ADCs can access the same analog input pins but simultaneous conversions are not allowed. Also, when one ADC is performing a differential conversion on a pair of pins, the other ADC must not access either of these two pins as single-ended channels.
2. Old version has reserved values for channel numbers 8 to 11 when EMUX = 1. Therefore, now the behavior is different because it is converted the signal at AN8 to AN11, respectively.
3. VREF = VRH – VRL. After calibration, the 50% reference point will read approximately 20 mV below 50%(VRH - VRL). The 50% reference point should not be used to calibrate the EQADC. Use the 25% and 75% reference points to calibrate the EQADC.
4. This channel should be converted using the Long Sample Time (LST) setting for either 64 or 128 ADC sample cycles in the ADC Conversion Command Message (LST = 0b10 or 0b11).

23.5.7.1.1 External multiplexing

The eQADC can use from one to eight external multiplexer chips to expand the number of analog signals that may be converted. Up to 64 analog channels can be converted through external multiplexer selection. The externally multiplexed channels are automatically selected by the CHANNEL_NUMBER field of a Command Message, in the same way done with internally multiplexed channels. The software selects the external multiplexed mode by setting the ADC0/1_EMUX bit in either ADC0_CR or ADC1_CR depending on which ADC will perform the conversion. *Table 303* shows the channel number assignments for the multiplexed mode. There are 4 differential pairs, 39 single-ended, and, at most, 64 externally multiplexed channels which can be selected. Only one ADC can have its ADC0/1_EMUX bit asserted at a time.

Figure 472 shows the maximum configuration of eight external multiplexer chips connected to the eQADC. The external multiplexer chip selects one of eight analog inputs and connects it to a single analog output, which is fed to a specific input of the eQADC. The eQADC provides three multiplexed address signals (MA0, MA1, and MA2) to select one of eight inputs. These three multiplexed address signals are connected to all eight external multiplexer chips. The analog output of the eight multiplex chips are each connected to eight separate eQADC inputs: ANR, ANS, ANT, ANU, ANW, ANX, ANY, and ANZ. The MA pins correspond to the three least significant bits of the channel number that selects ANR, ANS,

ANT, ANU, ANW, ANX, ANY, and ANZ with MA0 being the most significant bit—see *Table 304*.

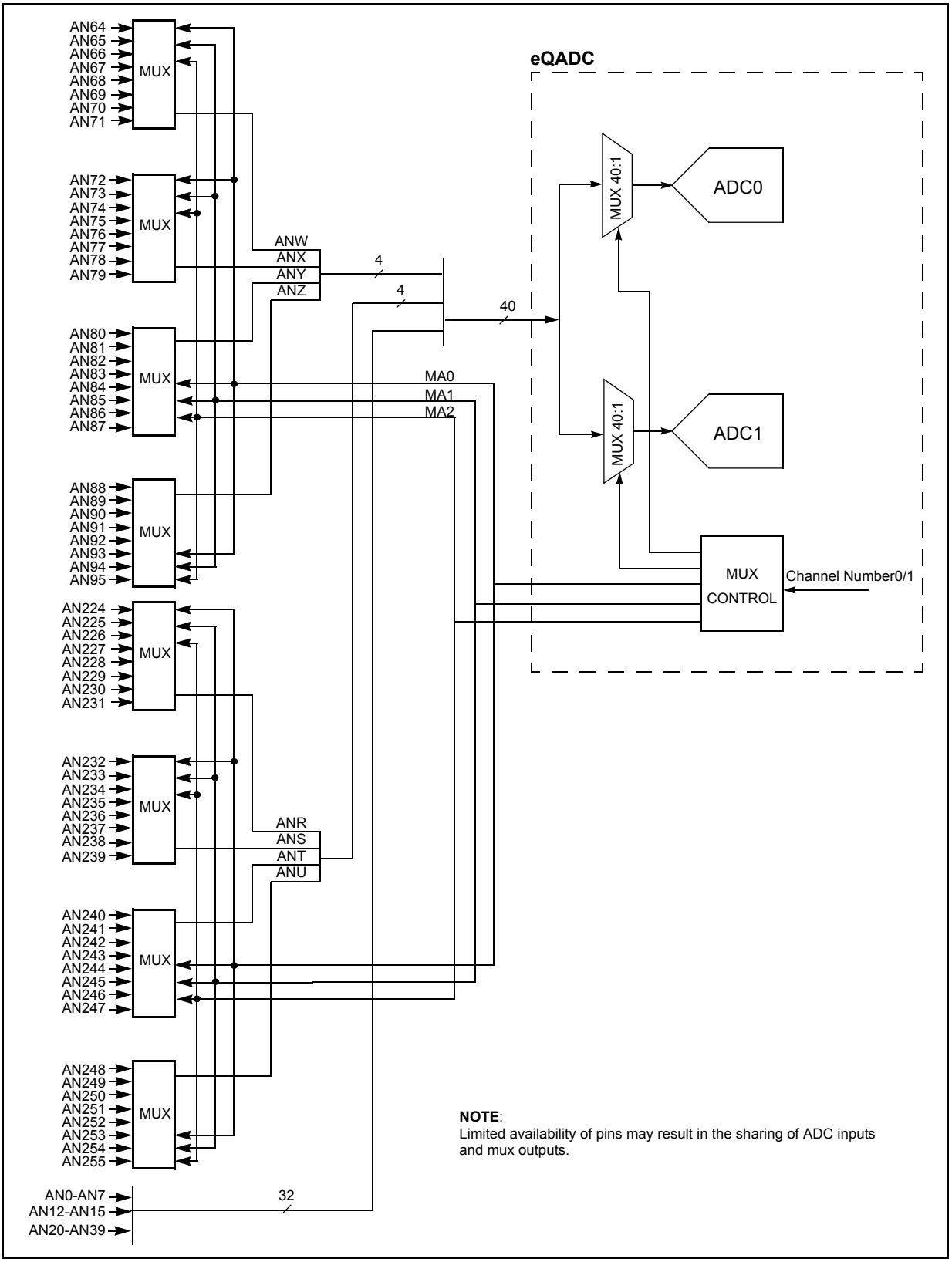
Table 304. Encoding of MA pins(1)

Channel number selecting ANR, ANS, ANT, ANU, ANW, ANX, ANY, ANZ (decimal)								MA0	MA1	MA2
ANR	ANS	ANT	ANU	ANW	ANX	ANY	ANZ			
224	232	240	248	64	72	80	88	0	0	0
225	233	241	249	65	73	81	89	0	0	1
226	234	242	250	66	74	82	90	0	1	0
227	235	243	251	67	75	83	91	0	1	1
228	236	244	252	68	76	84	92	1	0	0
229	237	245	253	69	77	85	93	1	0	1
230	238	246	254	70	78	86	94	1	1	0
231	239	247	255	71	79	87	95	1	1	1

1. ‘0’ means pin is driven LOW and ‘1’ that pin is driven HIGH.

When the external multiplexed mode is selected for either ADC, the eQADC automatically creates the MA output signals from CHANNEL_NUMBER field of a Command Message. The eQADC also converts the proper input channel (ANR, ANS, ANT, ANU, ANW, ANX, ANY, and ANZ) by interpreting the CHANNEL_NUMBER field. As a result, up to 64 externally multiplexed channels appear to the conversion queues as directly connected signals.

Figure 472. Example of external multiplexing



23.5.8 eQADC DMA/Interrupt request

Table 305 lists methods to generate interrupt requests in the eQADC queuing control and triggering control. The DMA/interrupt request select bits and the DMA/interrupt enable bits are described in *Section 23.4.2.7: eQADC Interrupt and DMA Control Registers (EQADC_IDCR)* and the interrupt flag bits are described in *Section 23.4.2.8: eQADC FIFO and Interrupt Status Registers (EQADC_FISR)*. *Figure 473* depicts all interrupts and DMA requests generated by the eQADC.

Table 305. eQADC FIFO interrupt summary(1)

Interrupt	Condition	Clearing mechanism
Non Coherency Interrupt	NCIEx = 1 NCFx = 1	Clear NCFx bit by writing a '1' to the bit.
Result FIFO Overflow Interrupt(2)	RFOIEx = 1 RFOFx = 1	Clear RFOFx bit by writing a '1' to the bit.
Command FIFO Underflow Interrupt(2)	CFUIEx = 1 CFUFx = 1	Clear CFUFx bit by writing a '1' to the bit.
Result FIFO Drain Interrupt	RFDEx = 1 RFDSx = 0 RFDFx = 1	Clear RFDFx bit by writing a '1' to the bit.
Command FIFO Fill Interrupt	CFFEx = 1 CFFSx = 0 CFFFx = 1	Clear CFFFx bit by writing a '1' to the bit.
End of Queue Interrupt	EOQIEx = 1 EOQFx = 1	Clear EOQFx bit by writing a '1' to the bit.
Pause Interrupt	PIEx = 1 PFx = 1	Clear PFx bit by writing a '1' to the bit.
Trigger Overrun Interrupt(2)	TORIEx = 1 TORFx = 1	Clear TORFx bit by writing a '1' to the bit.

1. For details refer to *Section 23.4.2.8: eQADC FIFO and Interrupt Status Registers (EQADC_FISR)* and *Section 23.4.2.7: eQADC Interrupt and DMA Control Registers (EQADC_IDCR)*.
2. Apart from generating an independent interrupt request for when a RFIFO Overflow Interrupt, a CFIFO Underflow Interrupt, and a CFIFO Trigger Overrun Interrupt occurs, the eQADC also provides a combined interrupt request at which these requests from ALL CFIFOs are ORed. Refer to *Figure 473* for details.

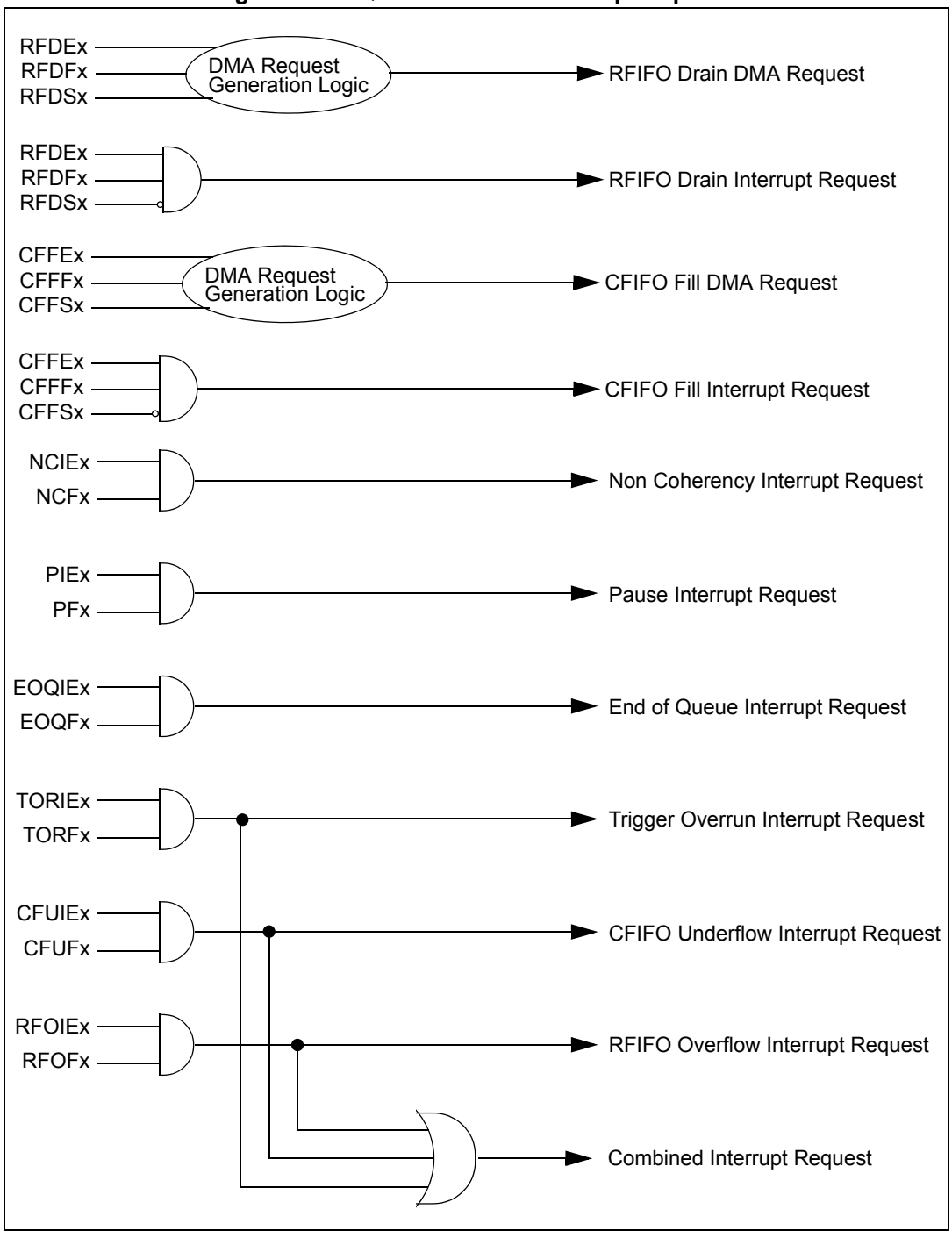
Table 306 describes a list of methods to generate DMA requests in the eQADC.

Table 306. eQADC FIFO DMA summary(1)

DMA request	Condition	Clearing mechanism
Result FIFO Drain DMA Request	RFDEx = 1 RFDSx = 1 RFDFx = 1	The eQADC automatically clears the RFDFx when RFIFOx becomes empty. Writing ‘1’ to the RFDFx bit is not allowed.
Command FIFO Fill DMA Request	CFFEx = 1 CFFSx = 1 CFFFx = 1	The eQADC automatically clears the CFFFx when CFIFOx becomes full. Writing ‘1’ to the CFFFx bit is not allowed.

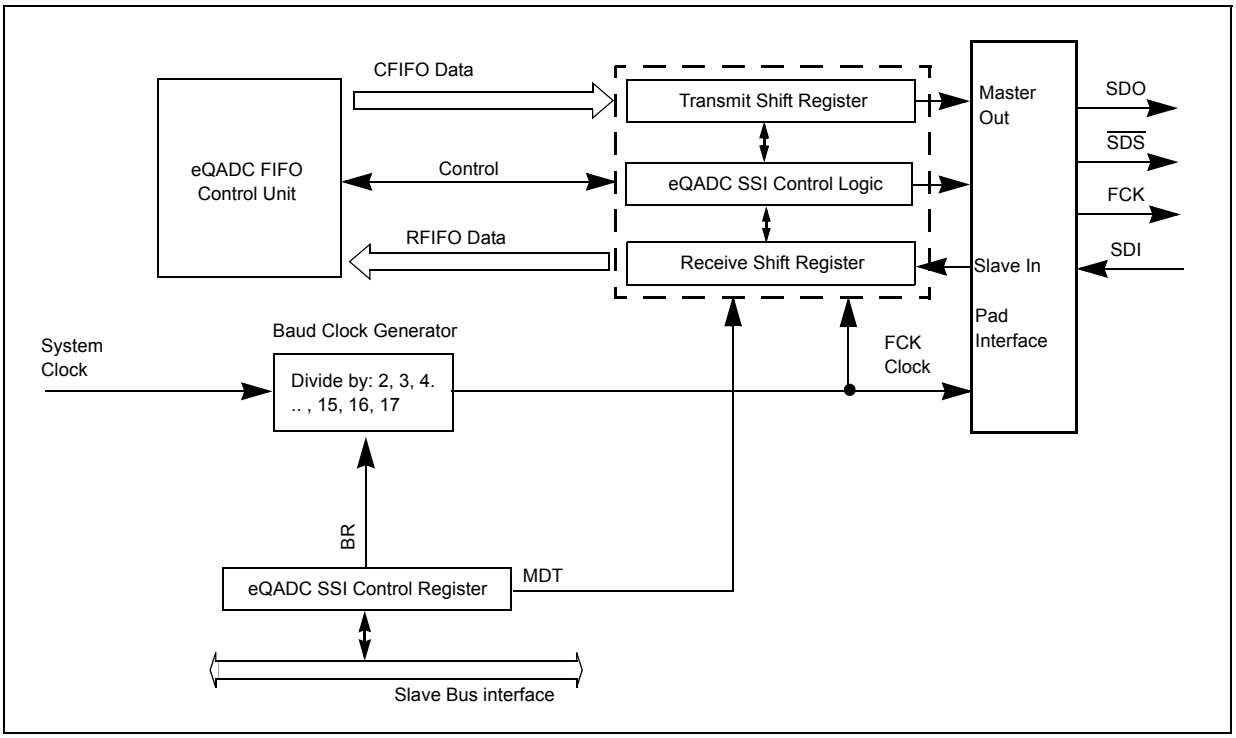
1. For details refer to *Section 23.4.2.8: eQADC FIFO and Interrupt Status Registers (EQADC_FISR)* and *Section 23.4.2.7: eQADC Interrupt and DMA Control Registers (EQADC_IDCR)*.

Figure 473. eQADC DMA and interrupt requests



23.5.9 eQADC synchronous serial interface (SSI) sub-block

Figure 474. eQADC synchronous serial interface block diagram



The eQADC SSI protocol allows for a full duplex, synchronous, serial communication between the eQADC and a single external device. *Figure 474* shows the different components inside the eQADC SSI block. The eQADC SSI sub-block on the eQADC is always configured as a master. The eQADC SSI has four associated port pins:

- Free-running Clock (FCK)
- Serial Data Select ($\overline{\text{SDS}}$)
- Serial Data In (SDI)
- Serial Data Out (SDO)

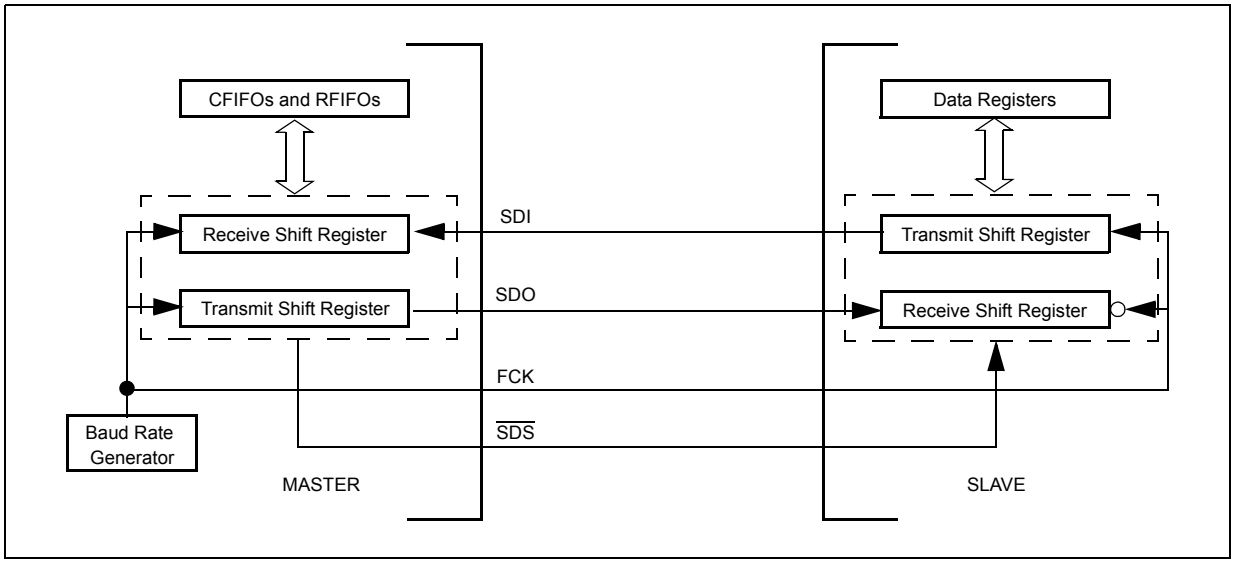
The FCK clock signal times the shifting and sampling of the two serial data signals and it is free-running between transmissions, allowing it to be used as the clock for the external device. The $\overline{\text{SDS}}$ signal will be asserted to indicate the start of a transmission, and negated to indicate the end or the abort of a transmission. SDI is the master serial data input and SDO the master serial data output.

The eQADC SSI sub-block is enabled by setting the ESSIE field in the eQADC Module Configuration Register (EQADC_MCR). When enabled, the eQADC SSI can be optionally capable of starting serial transmissions. When serial transmissions are disabled (ESSIE set to 0b10), no data will be transmitted to the external device but FCK will be free-running. This operation mode permits the control of the timing of the first serial transmission, and can be used to avoid the transmission of data to an unstable external device, for example, a device that is not fully reset. This mode of operation is specially important for the reset procedure of an external device that uses the FCK as its main clock.

The main elements of the eQADC SSI block are the shift registers. The 26-bit transmit shift register in the master and 26-bit receive shift register in the slave are linked by the SDO pin.

In a similar way, the 26-bit transmit shift register in the slave and 26-bit receive shift register in the master are linked by the SDI pin. See *Figure 475*. When a data transmission operation is performed, data in the transmit registers is serially shifted 26 bit positions into the receive registers by the FCK clock from the master; data is exchanged between the master and the slave. Data in the master transmit shift register in the beginning of a transmission operation becomes the output data for the slave, and data in the master receive shift register after a transmission operation is the input data from the slave.

Figure 475. Full duplex pin connection



23.5.9.1 eQADC SSI data transmission protocol

Figure 476 shows the timing of an eQADC SSI transmission operation. The main characteristics of this protocol are:

- FCK is free-running; it does not stop between data transmissions. FCK will be driven low:
  - When the serial interface is disabled
  - In stop/debug mode
  - Immediately after reset
- Frame size is fixed to 26 bits
- MSB bit is always transmitted first
- Master drives data on the positive edge of FCK and latches incoming data on the next positive edge of FCK
- Slave drives data on the positive edge of FCK and latches incoming data on the negative edge of FCK

Master initiates a data transmission by driving $\overline{SDS}$ low, and its MSB bit on SDO on the positive edge of FCK. Once an asserted $\overline{SDS}$ is detected, the slave shifts its data out, one bit at a time, on every FCK positive edge. Both the master and the slave drive new data on the serial lines on every FCK positive edge. This process continues until all the initial 26 bits in the master shift register are moved into the slave shift register. t_{DT} is the delay between two consecutive serial transmissions, time during which $\overline{SDS}$ is negated. When ready to start of the next transmission, the slave must drive the MSB bit of the message on every positive edge of FCK regardless of the state of the $\overline{SDS}$ signal. On the next positive edge,

the second bit of the message is conditionally driven according to if an asserted $\overline{SDS}$ was detected by the slave on the preceding FCK negative edge. This is an important requisite since the $\overline{SDS}$ and the FCK are not synchronous. The $\overline{SDS}$ signal is not generated by FCK, rather both are generated by the system clock, so that it is not guaranteed that FCK edges will precede $\overline{SDS}$ ones. While $\overline{SDS}$ is negated, the slave continuously drives its MSB bit on every positive edge of FCK until it detects an asserted $\overline{SDS}$ on the immediately next FCK negative edge. See *Figure 477* for three situations showing how the slave should behave according to when $\overline{SDS}$ is asserted.

Note: On the master, the FCK is not used as a clock. Although, the eQADC SSI behavior is described in terms of the FCK positive and negative edges, all eQADC SSI related signals (SDI, SDS, SDO, and FCK) are synchronized by the system clock on the master side. There are no restrictions regarding the use of the FCK as a clock on the slave device.

23.5.9.1.1 Abort feature

The master indicates it is aborting the current transfer by negating $\overline{SDS}$ before the whole data frame has being shifted out, that is the 26th bit of data being transferred has not being shifted out. The eQADC ignores the incompletely received message. The eQADC resends the aborted message whenever the corresponding CFIFO becomes again the highest priority CFIFO with commands bound for not-full external CBuffer. Refer to *Section 23.5.4.3: CFIFO common prioritization and command transfer* for more information on aborts and CFIFO priority.

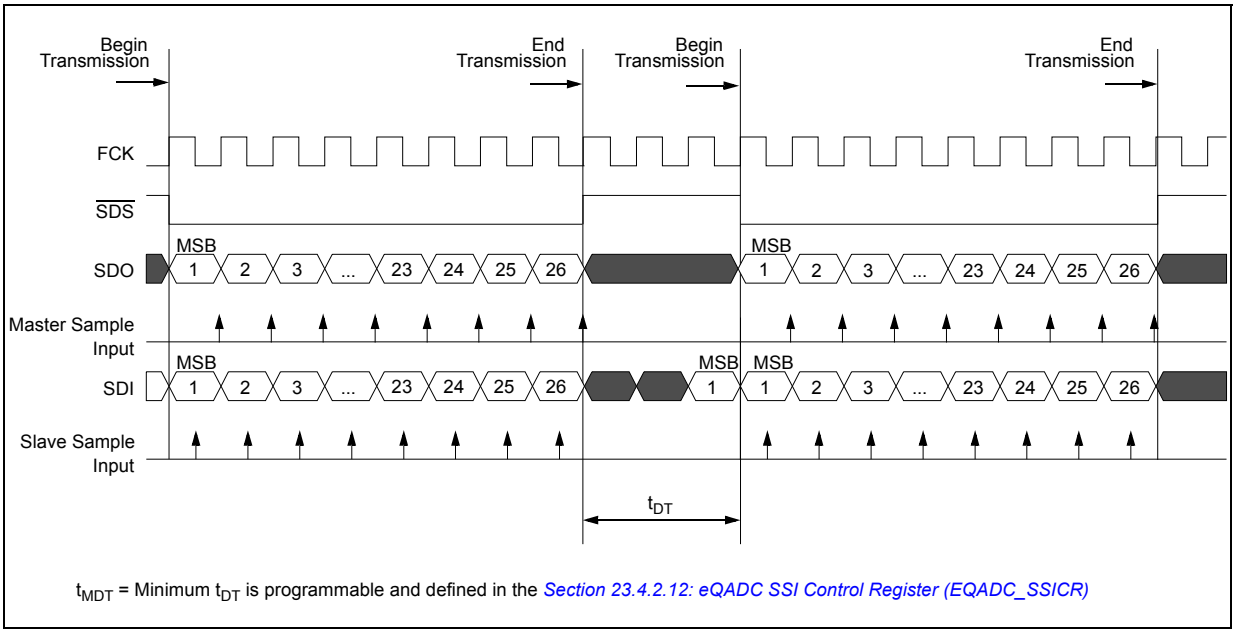
23.5.9.2 Baud clock generation

As shown in *Figure 474*, the baud clock generator divides the system clock to produce the baud clock. The BR field in the eQADC SSI Control Register (EQADC_SSICR) selects the system clock divide factor as in *Table 269*.[t]

$$\text{BaudClockFrequency} = \frac{\text{SystemClockFrequency(MHz)}}{\text{SystemClockDivideFactor}}$$

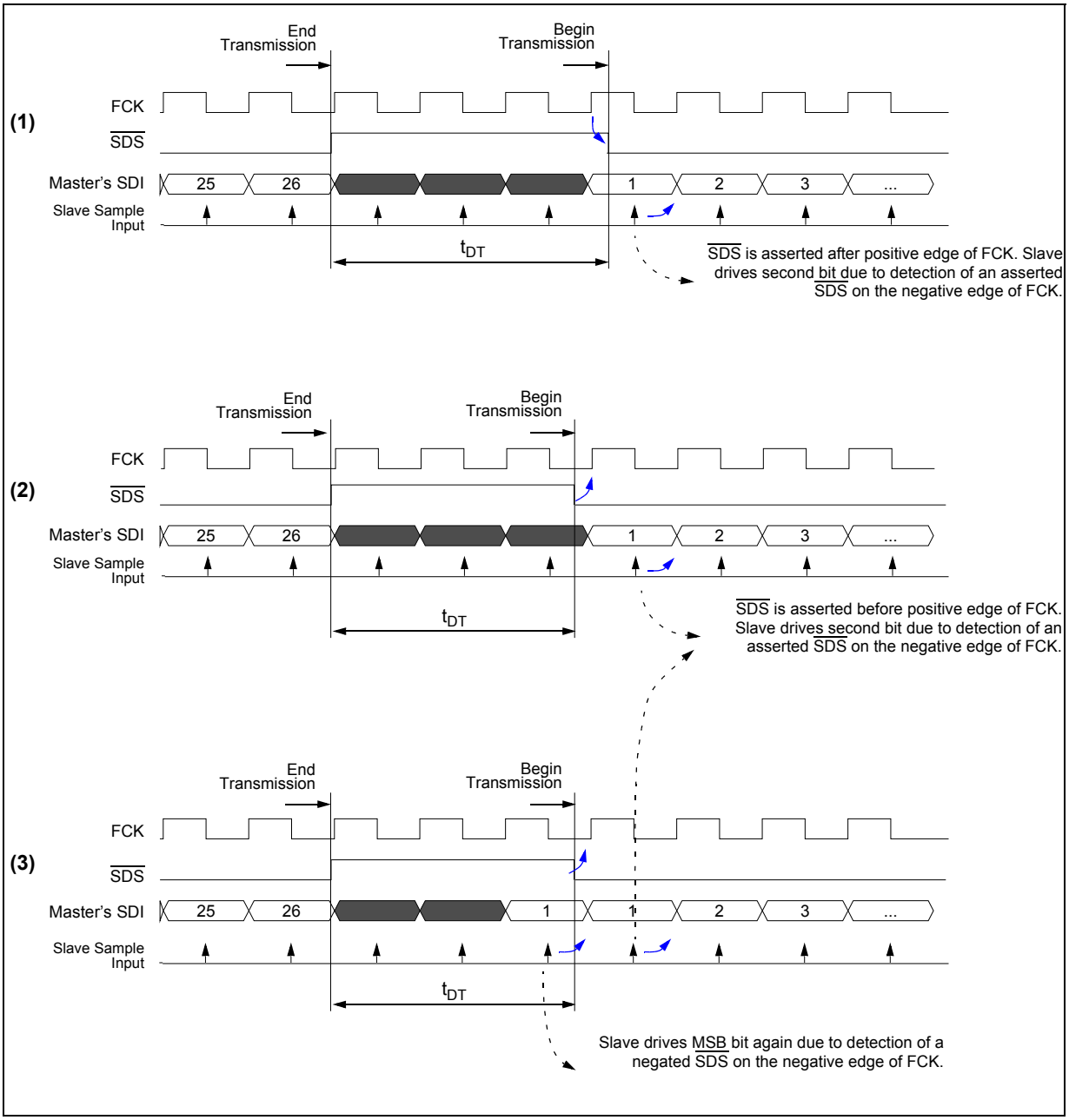
t. Maximum FCK frequency is highly dependable on track delays, master pad delays, and slave pad delays.

Figure 476. Synchronous serial interface protocol timing



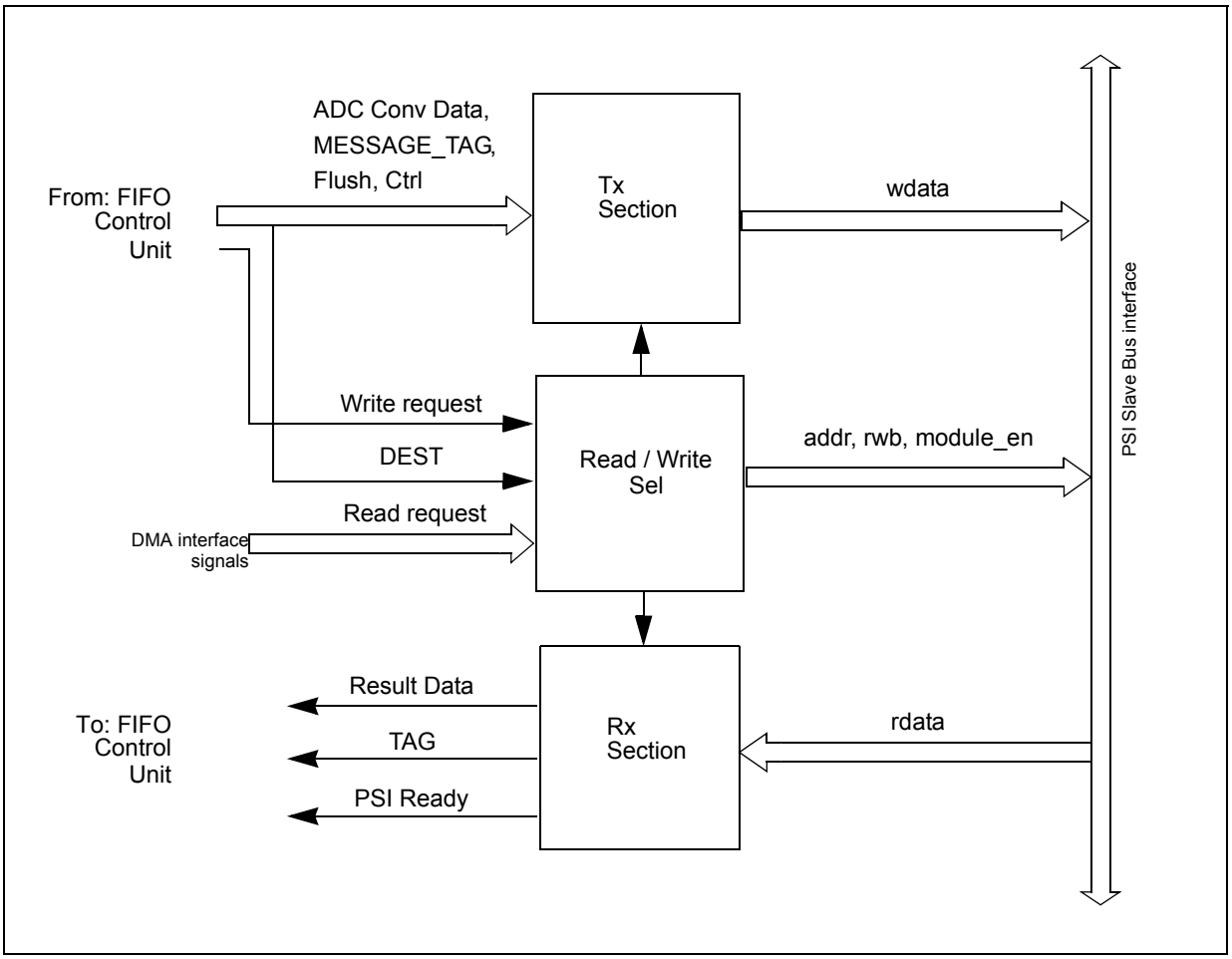
t_{MDT} = Minimum t_{DT} is programmable and defined in the *Section 23.4.2.12: eQADC SSI Control Register (EQADC_SSICR)*

Figure 477. Slave driving the MSB and consecutive bits in a data transmission



23.5.10 eQADC parallel side interface (PSI) sub-block

Figure 478. eQADC parallel side interface block diagram



The eQADC PSI sub-block allows communication between the eQADC and the companion modules through a local slave bus that has the eQADC block as the master and the companion modules as the slaves. However, due to the eQADC's limited processing capability, no more than 15 slaves can be enabled.

The conversion result goes through the Result Formatting and Calibration sub-block and the corresponding MESSAGE_TAG and DEST fields are decoded to decide the destination of the conversion result data. When the DEST field is not zero as described in *Section 23.4.3.6: Alternate Configuration 1–8 Control Registers (ADC_ACR1–8)*, the MESSAGE_TAG bits, and some control bits, and the conversion result data are sent to the transmission section of the PSI sub-block. This set of data is processed and sent to the corresponding on-chip companion module.

The companion module can also send back to eQADC the result of some processing in the received data. In this case, the receiver section of he PSI sub-block treats this request and receives the result data with the corresponding MESSAGE_TAG or TAG field. This TAG is used by the decoder to select which RFIFO will be written with the received companion module data.

23.5.10.1 Input/Output signals description

The information content of the input and the output data buses from/to the companion modules are described in *Figure 479*. The input data format is used in the Read cycle and corresponds to the data supplied by some companion module. The output data format is used in the Write cycle of this interface block and contains information / controls to some companion module.

Read (Input) / Write (Output) Data Buses Content

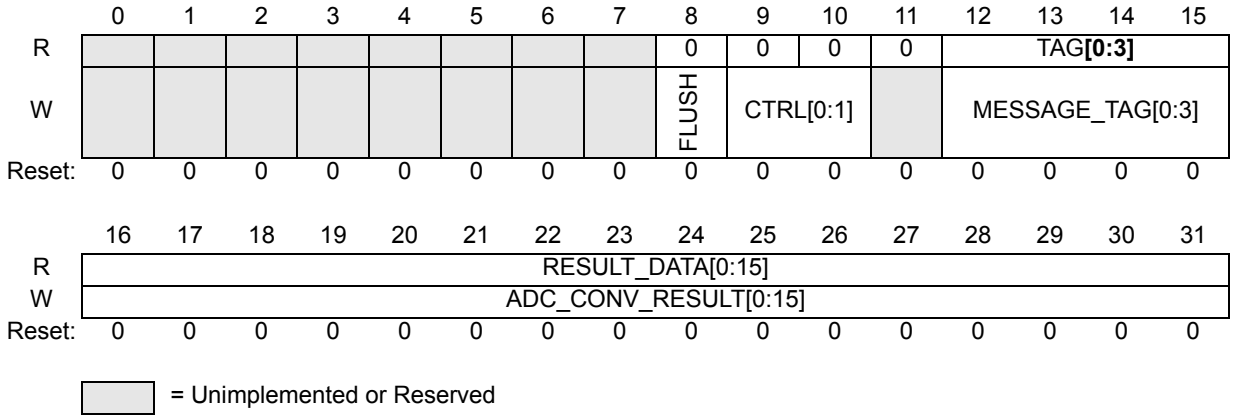


Figure 479. PSI Input and Output Data Buses Content

Table 307. PSI Input and Output Data Buses Content

Field	Description
FLUSH	Master block Flush request/control bit The FLUSH signal is used to request a Flush flow in the slave block. More details are presented in item *Section 23.5.2.3.1.2: Conversion command format for alternate configurations*. 1: Flush request 0: No flush request
CTRL[0:1]	Control bits This field is used for the control of the companion module. Field functions: 00: INH_RET or PREFILL—This mode indicates that the companion module should not send back some RESULT_DATA corresponding to the accompanying ADC_CONV_RESULT data. When the slave module is the Decimation Filter, this control enables the PREFILL filter mode. 01: CONVERSION RESULT—Indicates that the ADC_CONV_RESULT field should be treated as a valid sample data. This control mode is useful to put the Decimation Filter in normal mode instead of prefill mode. 10: TIME STAMP or REGISTER READ—A time stamp indicates that the ADC_CONV_RESULT field should follow a bypass flow in the companion module, returning back to the eQADC without any modification. 11: Reserved

Table 307. PSI Input and Output Data Buses Content (continued)

Field	Description
MESSAGE_TAG [0:3]	Message tag bits field This field indicates the RFIFO destination associated with the ADC_CONV_RESULT sample. This value is stored by the companion module and is used to address the destination RFIFO register when a RESULT_DATA is generated due to that ADC_CONV_RESULT sample. TAG[0:3] — Companion module tag bits This bit field is used to address the appropriate destination RFIFO in the eQADC for the accompanying RESULT_DATA bits. In eQADC application, this is used to address the appropriate RFIFO in the eQADC block. In this case, the possible values are only from 0000 to 0101.
ADC_CONV_RESULT [0:15]	ADC Conversion Result Data This bit field is the ADC conversion result data after passing through the calibration and formatting block.
RESULT_DATA[0:15]	Companion Module Result Data This bit field corresponds to the companion module data processing result to eQADC.

23.5.10.2 PSI transmitter / write section

The transmission sub-block formats the data bus from RFIFO control sub-block to send to the PSI slave wdata bus. The transmission data is registered and its content is described in *Section 23.5.10.1: Input/Output signals description*. The transmission has higher priority than reception. This is done to avoid the use of memory to store transmission data and it is not used waiting time for transmission.

The destination companion module for the transmission data is obtained by decoding the DEST[0:3] bits. The not null decimal value of DEST[0:3] is used to uniquely set the corresponding module enable signal. For example, DEST[0:3] value equal to 0xF that corresponds to the decimal value 15 is going to set only the module enable 15. All other module enable from 1 to 14 are not set.

23.5.10.3 PSI receiver / read section

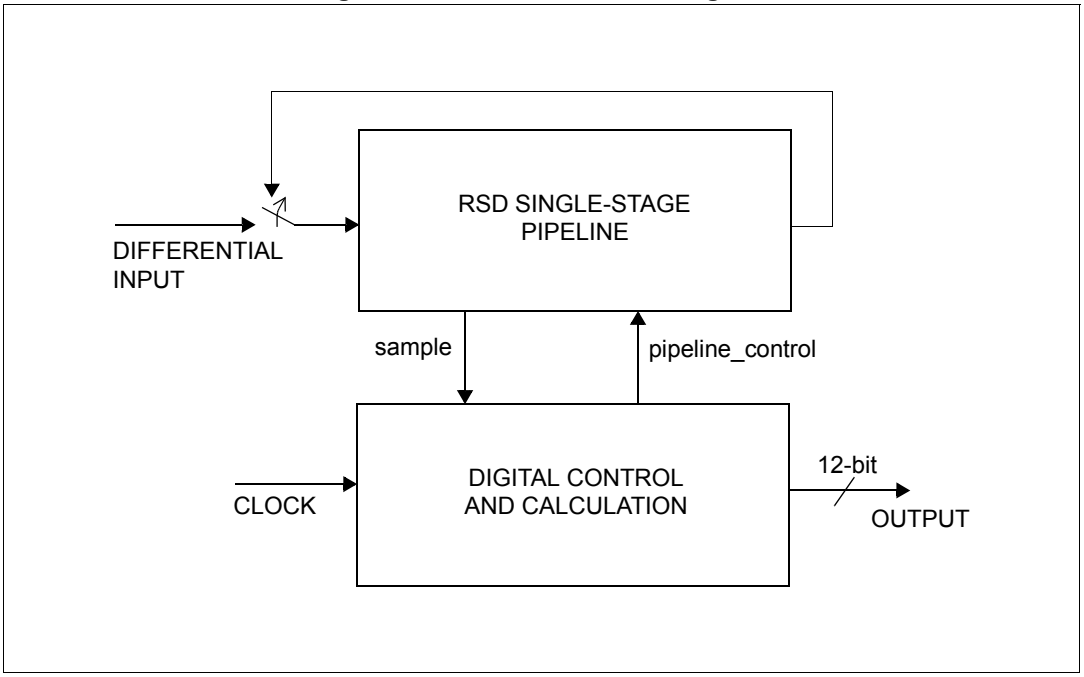
The receiver sub-block receives data from some companion module using the PSI slave bus interface. The companion module sends a read request to eQADC using the DMA read request line. The PSI logic sends a read command if there is no transmission request. The received data has the structure described in item *Section 23.5.10.1: Input/Output signals description*.

23.5.11 Analog sub-block

23.5.11.1 Analog-to-digital converter (ADC)

23.5.11.1.1 ADC architecture

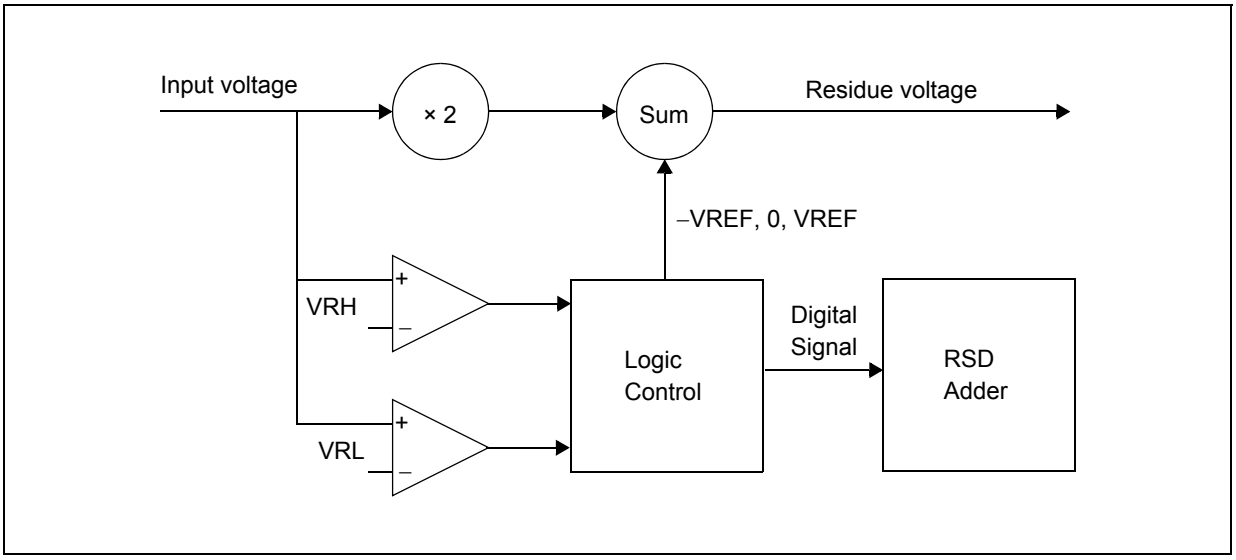
Figure 480. RSD ADC block diagram



The redundant signed digit (RSD) cyclic ADC consists of two main portions: the analog RSD stage, and the digital control and calculation block, as shown in *Figure 480*. To begin an analog-to-digital conversion, a differential input is passed into the analog RSD stage. The signal is passed through the RSD stage, and then from the RSD stage output, back to its input to be passed again. To complete a 12-bit conversion, the signal must pass through the RSD stage 12 times. For 10-bit and 8-bit resolution, the signal must pass 10 or 8 times through the RSD. Each time an input signal is read into the RSD stage, a digital sample is taken by the digital control/calculation block. The digital control/calculation block uses this sample to tell the analog block how to condition the signal. The digital block also saves each successive sample and adds them according to the RSD algorithm at the end of the entire conversion cycle.

23.5.11.1.2 RSD overview

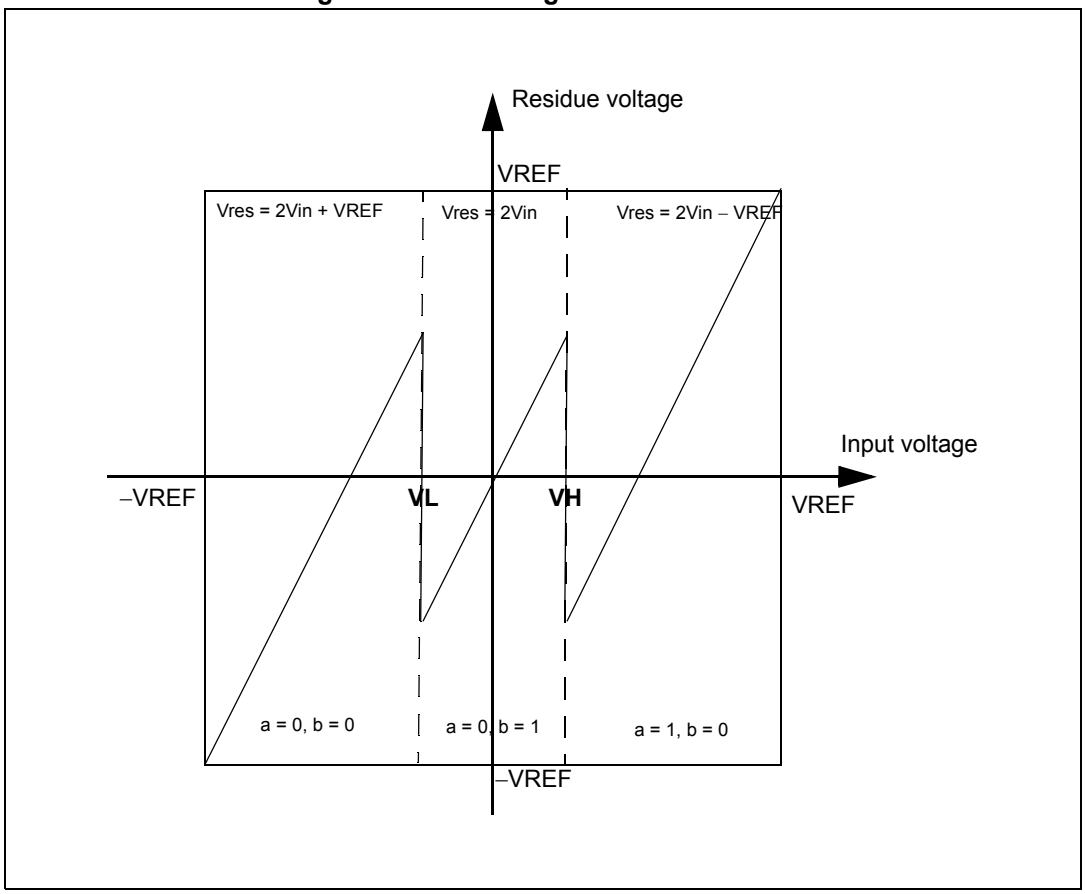
Figure 481. RSD stage block diagram



On each pass through the RSD stage, the input signal will be multiplied by exactly two, and summed with either –VREF, 0, or VREF, depending on the Logic Control. The Logic Control will determine –VREF, 0, or VREF depending on the two comparator inputs. As the Logic Control sets the summing operation, it also sends a digital value to the RSD adder. Each time an analog signal passes through the RSD single-stage, a digital value is collected by the RSD adder. At the end of an entire AD conversion cycle, the RSD adder uses these collected values to calculate the 12-bit/10-bit/8-bit digital output.

Figure 482 shows the transfer function for the RSD stage. Note how the digital value (a, b) is dependent on the two comparator inputs.

Figure 482. RSD stage transfer function

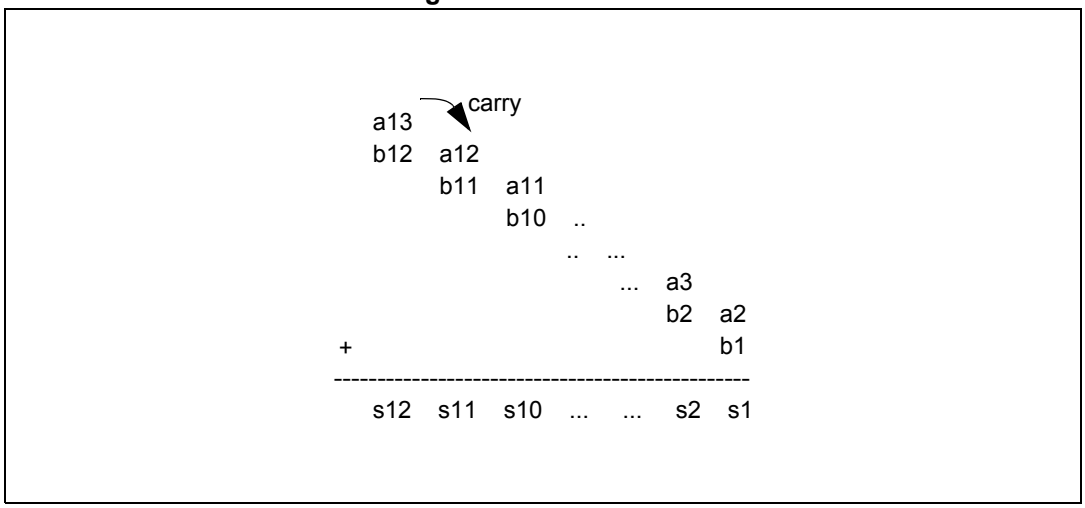


In each pass through the RSD stage, the residue will be sent back to be the new input, and the digital signals, a and b, will be stored. For the 12-bit ADC, input signal is sampled during the input phase, and after each of the 12 passes through the RSD stage. Thus, 13 total a and b values are collected. Upon collecting all these values, they will be added according to the RSD algorithm to create the 12-bit digital representation of the original analog input. The bits are added in the following manner:

23.5.11.1.3 RSD adder

The array, s1 to s12,will be the digital output of the RSD ADC with s1 being the MSB and s12 being the LSB.

Figure 483. RSD adder



23.5.11.1.4 Variable gain amplification (VGA) for pre-gain

The VGA starts after sampling completes. It is enabled by a 2-bit signal PRE_GAIN described in *Section 23.4.3.6: Alternate Configuration 1–8 Control Registers (ADC_ACR1–8)*.

The ADC takes 2, 8, 64 or 128 clock cycles to do sampling which is selected by the LST[0:1] field in the conversion command message. After the sampling, if 2x VGA is enabled, there is a 2x gain stage without comparison before the regular conversion cycles. When 4x VGA is enabled, there are the 2x gain stage without comparison by 2 times before the normal conversion processing.

23.6 Initialization/Application information

23.6.1 Multiple queues control setup example

This section provides an example of how to configure multiple CQueues. *Table 308* describes how each CQueue can be used for a different application. Also documented in this section are general guidelines on how to initialize the on-chip ADCs and the external device, and how to configure the CQueues and the eQADC.

Table 308. Application of each CQueue

CQueue number	CQueue type	Running speed	Number of contiguous conversions	Example
0	Very fast burst time-based CQueue	Every 2 µs for 200 µs; pause for 300 µs and then repeat	2	Injector current profiling
1	Fast hardware-triggered CQueue	Every 900 µs	3	Current sensing of PWM controlled actuators
2	Fast repetitive time-based CQueue	Every 2 ms	8	Throttle position

Table 308. Application of each CQueue (continued)

CQueue number	CQueue type	Running speed	Number of contiguous conversions	Example
3	Software-triggered CQueue	Every 3.9 ms	3	Command triggered by software strategy
4	Repetitive angle-based CQueue	Every 625 µs	7	Airflow read every 30 degrees at 8000 RPM
5	Slow repetitive time-based CQueue	Every 100 ms	10	Temperature sensors

23.6.1.1 eQADC initialization

The following steps provide an example about how to configure the eQADC controls and how to initialize the on-chip ADCs and the external device. In this example, all conversion commands will be transferred through CFIFO0.

1. Load all required configuration commands in the RAM in such way that they form a queue; this data structure will be referred below as CQueue0. *Figure 484* shows an example of a CQueue able to configure the on-chip ADCs and external device at the same time. Although, this example uses the DMAC to store commands in CFIFO0, configuration commands could have also been directly written to the CFIFO0 push register.
2. Select source driving eQADC hardware trigger ports (ETRIG). Before proceeding to next step, allow some time (minimum of two system clocks—filter period is set to minimum after reset) so that the logic level at the source is filtered and reaches the eQADC control logic.

Note: ETRIG ports could be driven by an external pin or by the output port of other blocks in the device, such as timers. In order to avoid unexpected triggering of CFIFOs in hardware trigger modes, the source driving the ETRIG port must be selected and set to a known logic level before putting the CFIFOs into the WAITING FOR TRIGGER state.

Note: The trigger filter bypass control inputs must be set considering the characteristics of the trigger signal. A particular case to assert the bypass control is when a device's internal signal with one clock width pulse is used.

3. Configure eQADC External Trigger Digital Filter Register (EQADC_ETDFR).
4. Configure eQADC Null Message Send Format Register (EQADC_NMSFR).
5. Configure eQADC SSI Control Register (EQADC_SSICR) to communicate with the external device.
6. Enable the eQADC SSI by programming the ESSIE field in the eQADC Module Configuration Register (EQADC_MCR).
   a) Write 0b10 to ESSIE field to enable the eQADC SSI. FCK is free running but serial transmissions are not started.
   b) Wait until the external device becomes stable after reset.
   c) Write 0b11 to ESSIE field to enable the eQADC SSI to start serial transmissions.
7. Configure the DMAC to transfer data from CQueue0 to CFIFO0 in the eQADC.
8. Configure eQADC Interrupt and DMA Control Registers (EQADC_IDCR).
   a) Set CFFS0 to configure the eQADC to generate a DMA request to load commands from CQueue 0 to the CFIFO0.
   b) Set CFFE0 to enable the eQADC to generate a DMA request to transfer commands from CQueue0 to CFIFO0; command transfers from the RAM to the CFIFO0 will start immediately.
   c) Set EOQIE0 to enable the eQADC to generate an interrupt after transferring all of the commands of CQueue0 through CFIFO0.
9. Configure eQADC CFIFO Control Registers (EQADC_CFCR).
   a) Write 0b0001 to the MODE0 field in EQADC_CFCR0 to program CFIFO0 for software single-scan mode.
   b) Write '1' to SSE0 to assert SSS0 and trigger CFIFO0.
10. Since CFIFO0 is in single-scan software mode and it is also the highest priority CFIFO, the eQADC starts to transfer configuration commands to the on-chip ADCs and to the external device.
11. When all of the configuration commands have been transferred, CF0 in eQADC FIFO and Interrupt Status Registers (EQADC_FISR) will be set. The eQADC generates a End of Queue interrupt. The initialization procedure is complete.

Figure 484. Example of a CQueue configuring the on-chip ADCs/external device

CQueue in system memory

Command Address	
0x0	Configuration Command to CBuffer0 - Ex: Write ADC0_CR
0x1	Configuration Command to CBuffer0 - Ex: Write ADC_TSCR
0x2	Configuration Command to CBuffer1 - Ex: Write ADC1_CR
0x3	Configuration Command to CBuffer2 - Ex: Write to external device configuration register

The initialization procedure described above does not generate ADC clocks that are in phase because the timing at which the ADC0/1_EN bits in the ADC0/1 Control Registers (ADC0_CR and ADC1_CR) are set is different. Below follows an example on how to simultaneously set these bits so that in-phase ADC clocks are generated. In this example, ADC0/1_CLK are configured to the same frequency.

1. Push an ADC0_CR write configuration command in CFIFO0 that enables ADC0 (ADC0_EN = 1) and that sets the ADC0_CLK_PS to an appropriate value. For example, 0x80800801.
2. Push an ADC1_CR write configuration command in CFIFO1 that enables ADC1 (ADC1_EN = 1) and that sets the ADC1_CLK_PS to an appropriate value. For example, 0x82800801.
3. Configure CFIFO0 and CFIFO1 to single scan software trigger mode and simultaneously trigger them by writing 0x04100410 to the EQADC_CFCR0—see *Section 23.4.2.6: eQADC CFIFO Control Registers (EQADC_CFCR)*.

23.6.1.2 Configuring eQADC for applications

This section provides an example based on the applications in *Table 308*. The example describes how to configure multiple CQueues to be used for those applications and provides a step-by-step procedure to configure the eQADC and the associated CQueue structures. In the example, the “Fast hardware-triggered CQueue”, described on the second row of *Table 308*, will have its commands transferred to CBuffer1; the conversion commands will be executed by ADC1. The generated results will be returned to RFIFO3 before being transferred to the RQueues in the RAM by the DMAC.

Note: There is no fixed relationship between CFIFOs and RFIFOs with the same number. The results of commands being transferred through CFIFO1 can be returned to any RFIFO, regardless of its number. The destination of a result is determined by the MESSAGE_TAG field of the command that requested the result. See Section 23.5.2.3: Message format in eQADC for details.

Step One: Setup the CQueues and RQueues.

1. Load the RAM with configuration and conversion commands. *Table 309* is an example of how CQueue1 commands should be set.
   a) Each trigger event will cause four commands to be executed. When the eQADC detects the Pause bit asserted, it will wait for another trigger to restart transferring commands from the CFIFO.
   b) At the end of the CQueue, the EOQ bit is asserted as shown in *Table 309*.
   c) Results will be returned to RFIFO3 as specified in the MESSAGE_TAG field of commands.
2. Reserve memory space for storing results.

Table 309. Example of CQueue commands(1)

Bit #	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
Bit name	EOQ	PAUSE	Reserved		ABORT_ST	ER	BN	Reserved	MESSAGE_TAG				ADC COMMAND																			
CMD 1	0	0	0		0	0	1	0	0b0011				Conversion Command																			
CMD 2	0	0	0		0	0	1	0	0b0011				Conversion Command																			
CMD 3	0	0	0		0	0	1	0	0b0011				Conversion Command																			
CMD 4	0	1	0		0	0	1	0	0b0011 (2)				Configure peripheral device for next conversion sequence																			
CMD 5	0	0	0		0	0	1	0	0b0011				Conversion Command																			
CMD 6	0	0	0		0	0	1	0	0b0011				Conversion Command																			
CMD 7	0	0	0		0	0	1	0	0b0011				Conversion Command																			
CMD 8	0	1	0		0	0	1	0	0b0011 (2)				Configure peripheral device for next conversion sequence																			
									etc.																							
CMD EOQ	1	0	0		0	0	1	0	0b0011				EOQ Message																			
	CFIFO Header								ADC Command																							

1. Fields LST, TSR, FMT, and CHANNEL_NUMBER are not shown for clarity. See *Section 23.5.2.3.1.1: Conversion command format for the standard configuration* for details.
2. MESSAGE_TAG field is only defined for read configuration commands.

Step Two: Configure the DMAC to handle data transfers between the CQueues/RQueues in RAM and the CFIFOs/RFIFOs in the eQADC.

1. For transferring, set the source address of the DMAC to point to the start address of CQueue1. Set the destination address of the DMAC to point to EQADC_CFPR1. Refer to *Section 23.4.2.4: eQADC CFIFO Push Registers (EQADC_CFPR)*.
2. For receiving, set the source address of the DMAC to point to EQADC_RFPR3. Refer to *Section 23.4.2.5: eQADC Result FIFO Pop Registers (EQADC_RFPR)*. Set the destination address of the DMAC to point to the starting address of RQueue1.

Step Three: Configure the eQADC Control Registers.

3. Configure eQADC Interrupt and DMA Control Registers (EQADC_IDCR).
   a) Set EOQIE1 to enable the End of Queue Interrupt request.
   b) Set CFFS1 and RFDS3 to configure the eQADC to generate DMA requests to push commands into CFIFO1 and to pop result data from RFIF03.
   c) Set CFINV1 to invalidate the contents of CFIFO1.
   d) Set RFDE3 and CFFE1 to enable the eQADC to generate DMA requests. Command transfers from the RAM to the CFIFO1 will start immediately.
   e) Set RFOIE3 to indicate if RFIFO3 overflows.
   f) Set CFUIE1 to indicate if CFIFO1 underflows.
4. Configure MODE1 to continuous-scan rising edge external trigger mode in eQADC CFIFO Control Registers (EQADC_CFCR).

Step Four: Command transfer to ADCs and Result data reception.

When an external rising edge event occurs for CFIFO1, the eQADC automatically will begin transferring commands from CFIFO1 when it becomes the highest priority CFIFO trying to send commands to CBuffer1. The received results will be placed in RFIFO3 and then moved to RQueue1 by the DMAC.

23.6.2 eQADC/DMAC interface

This section provides an overview about the eQADC/DMAC interface and general guidelines about how the DMAC should be configured in order for it to correctly transfer data between the queues in system memory and the eQADC FIFOs.

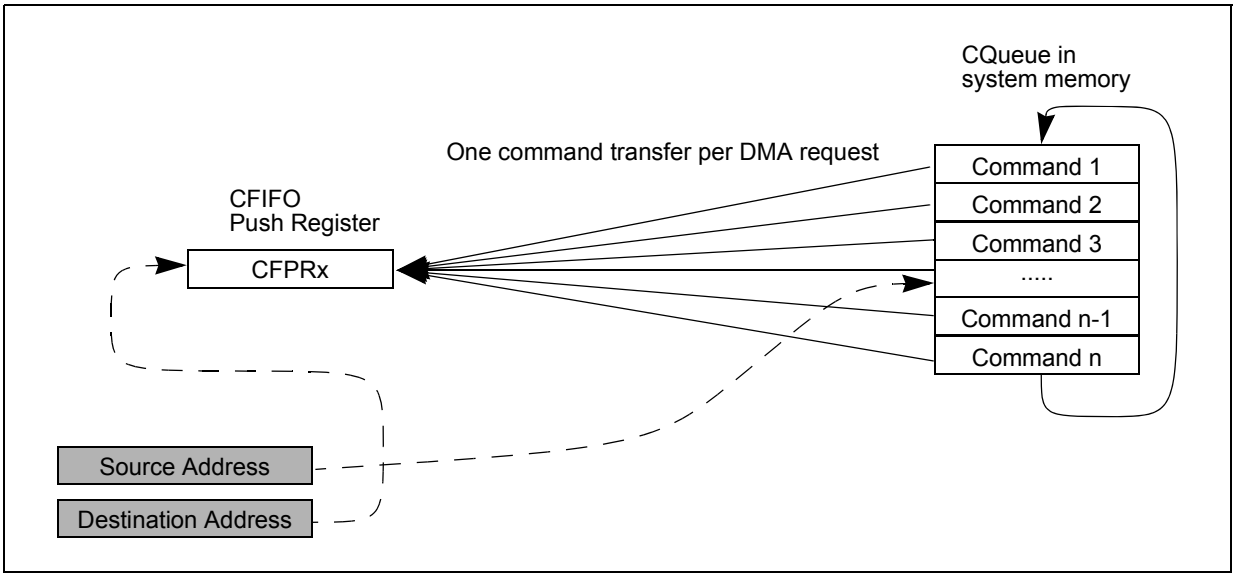
Note: Advanced DMACs provide more functionality then the ones discussed in this section.

23.6.2.1 CQueue/CFIFO transfers

In transfers involving CQueues and CFIFOs, the DMAC moves data from a queued source to a single destination as shown in *Figure 485*. The location of the data to be moved is indicated by the source address, and the final destination for that data, by the destination address. The DMAC contains a data structure containing these addresses and other parameters used in the control of data transfers. For every DMA request issued by the eQADC, the DMAC has to be configured to transfer a single command (32-bit data) from the CQueue, pointed to by the source address, to the CFIFO push register, pointed to by the destination address. After the service of a DMA request is completed, the source address has to be updated to point to the next valid command. The destination address remains unchanged. When the last command of a queue is transferred one of the following actions is recommended. Refer to *Chapter 7: Direct Memory Access (DMA)* for details.

- The corresponding DMA channel should be disabled. This might be desirable for CFIFOs in single scan mode.
- The source address should be updated to pointed to a valid command which can be the first command in the queue that has just been transferred (cyclic queue), or the first command of any other CQueue. This is desirable for CFIFOs in continuous scan mode, and at some cases, for CFIFOs in single scan mode.

Figure 485. CQueue/CFIFO interface

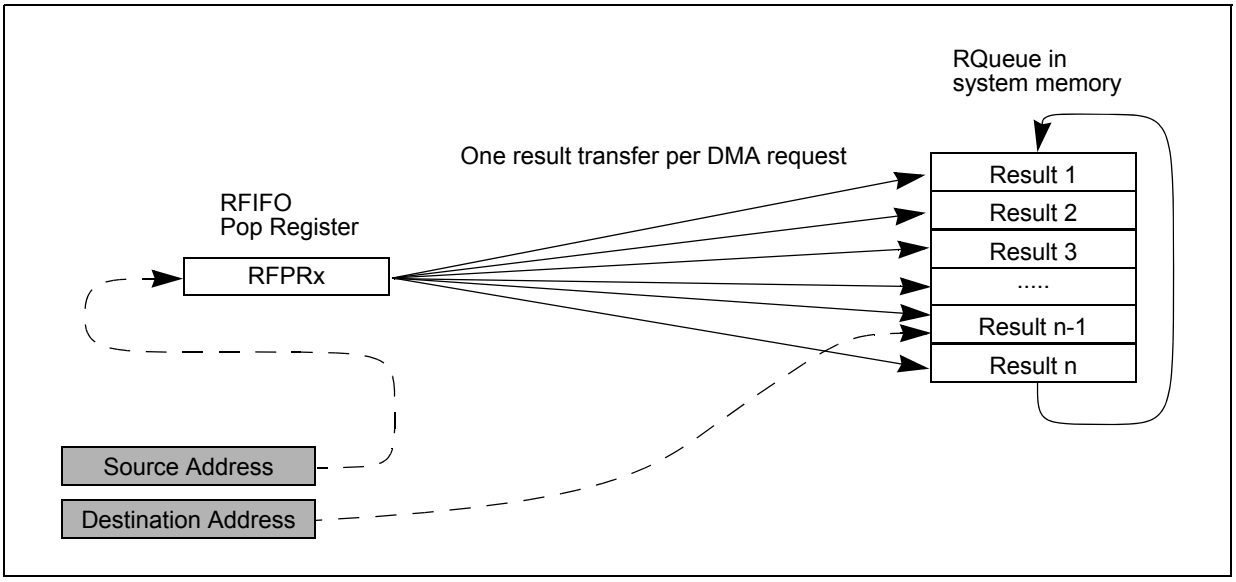


23.6.2.2 RQueue/RFIFO transfers

In transfers involving RQueues and RFIFOs, the DMAC moves data from a single source to a queue destination as shown in *Figure 486*. The location of the data to be moved is indicated by the source address, and the final destination for that data, by the destination address. For every DMA request issued by the eQADC, the DMAC has to be configured to transfer a single result (16-bit data), pointed to by the source address, from the RFIFO pop register to the RQueue, pointed to by the destination address. After the service of a DMA request is completed, the destination address has to be updated to point to the location where the next 16-bit result will be stored. The source address remains unchanged. When the last expected result is written to the RQueue, one of the following actions is recommended. Refer to *Chapter 7: Direct Memory Access (DMA)* for details about how this functionality is supported.

- The corresponding DMA channel should be disabled.
- The destination address should be updated pointed to the next location where new coming results are stored, which can be the first entry of the current RQueue (cyclic queue), or the beginning of a new RQueue.

Figure 486. RQueue/RFIFO interface



23.6.3 Sending immediate command setup example

In the eQADC, there is no immediate command register for sending a command immediately after writing to that register. However, a CFIFO can be configured to perform the same function as an immediate command register. The following steps illustrate how to configure CFIFO5 as an immediate command CFIFO. The results will be returned to RFIFO5.

1. Configure the eQADC Interrupt and DMA Control Registers (EQADC_IDCR).
   a) Clear CFIFO Fill Enable5 (CFFE5 = 0) in EQADC_IDCR2.
   b) Clear CFIFO Underflow Interrupt Enable5 (CFUIE5 = 0) in EQADC_IDCR2.
   c) Clear RFDS5 to configure the eQADC to generate interrupt requests to pop result data from RFIF05.
   d) Set RFIFO Drain Enable5 (RFDE5 = 1) in EQADC_IDCR2.
2. Configure the eQADC CFIFO Control Registers (EQADC_CFCR).
   a) Write ‘1’ to CFINV5 in EQADC_FCR2. This will invalidate the contents of CFIFO5.
   b) Set MODE5 to Continuous-Scan Software Trigger mode in EQADC_CFCR2.
3. To transfer a command, write it to eQADC CFIFO Push Register 5 (EQADC_CFPR5) with Message Tag = 0b0101. Refer to *Section 23.4.2.4: eQADC CFIFO Push Registers (EQADC_CFPR)*.
4. Up to four commands can be queued in CFIFO5. Check the CFCTR5 status in EQADC_FISR5 before pushing another command to avoid overflowing the CFIFO. Refer to *Section 23.4.2.8: eQADC FIFO and Interrupt Status Registers (EQADC_FISR)*.
5. When the eQADC receives a conversion result for RFIFO5, it generates an interrupt request. RFIFO Pop Register 5 (EQADC_RFPR5) can be popped to read the result. Refer to *Section 23.4.2.5: eQADC Result FIFO Pop Registers (EQADC_RFPR)*.

23.6.4 Modifying queues

More CQueues may be needed than the six supported by the eQADC. These additional CQueues can be supported by interrupting command transfers from a configured CFIFO, even if it is TRIGGERED and transferring, modifying the corresponding CQueue in the RAM or associating another CQueue to it, and restarting the CFIFO. More details on disabling a CFIFO are described in *Section 23.5.4.6.1: Disabled Mode*.

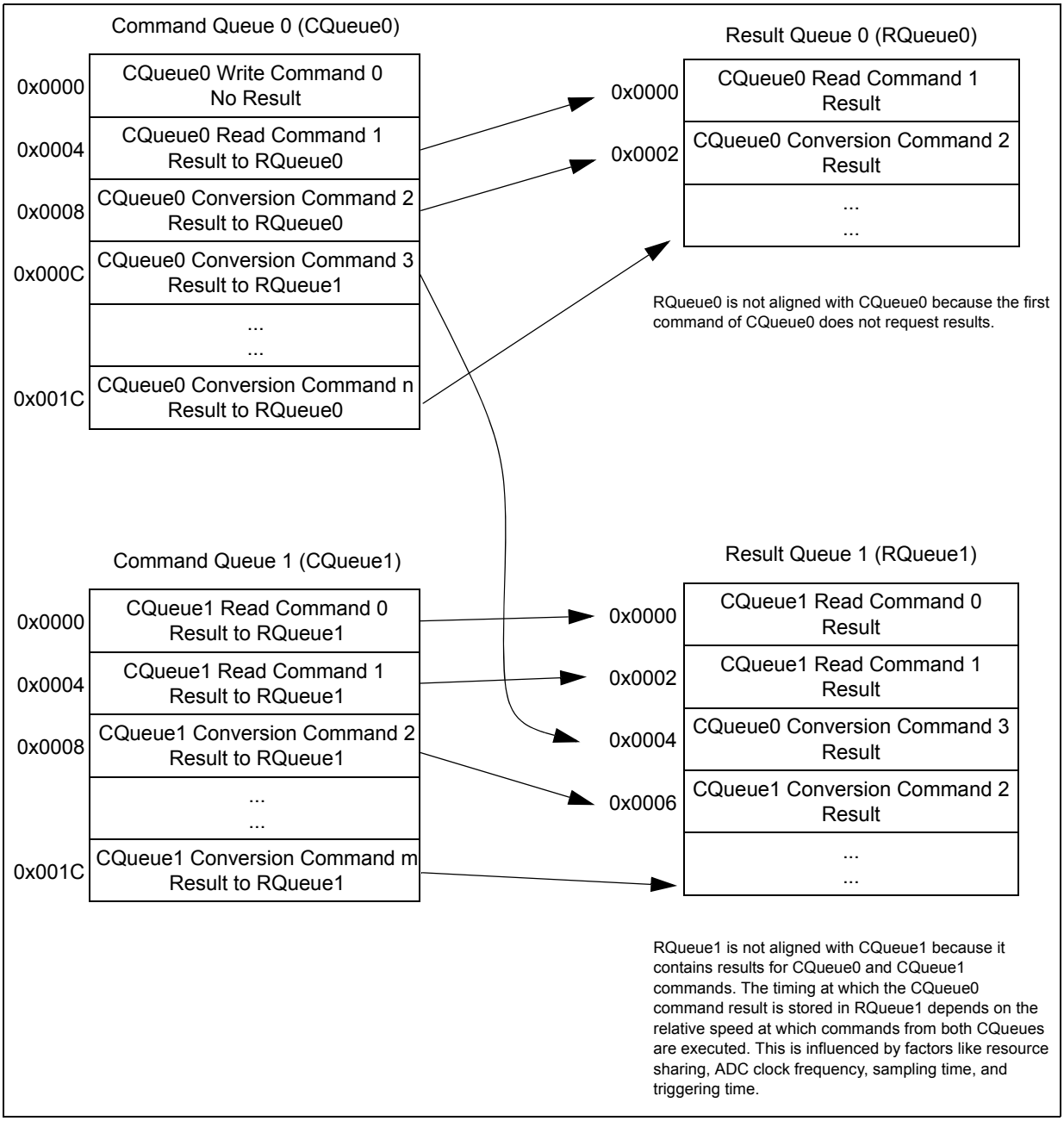
1. Determine the resumption conditions when later resuming the scan of the CQueue at the point before it was modified.
   a) Change MODEx in eQADC CFIFO Control Registers (EQADC_CFCR) to Disabled. Refer to *Section 23.5.4.6.1: Disabled Mode* for a description of what happens when MODEx is changed to Disabled.
   b) Poll CFSx until it becomes IDLE in eQADC CFIFO Status Register (EQADC_CFSR).
   c) Read and save TC_CFx in eQADC CFIFO Transfer Counter Registers (EQADC_CFTCR) for later resuming the scan of the queue. The TC_CFx provides the point of resumption.
   d) Since all result data may not have being stored in the appropriate RFIFO at the time MODEx is changed to disable, wait for all expected results to be stored in the RFIFO/RQueue before reconfiguring the DMAC to work with the modified RQueue. The number of results that must return can be estimated from the TC_CFx value obtained above.
2. Disable the DMAC from responding to the DMA request generated by CFFFx and RFDFx in eQADC FIFO and Interrupt Status Registers (EQADC_FISR).
3. Write ‘0x0000’ to the TC_CFx field.
4. Load the new configuration and conversion commands into RAM. Configure the DMAC to support the new CQueue/RQueue, but do not configure it yet to respond to DMA requests from CFIFOx/RFIFOx.
5. If necessary, modify eQADC Interrupt and DMA Control Registers (EQADC_IDCR) to suit the modified CQueue.
6. Write ‘1’ to CFINVx in eQADC CFIFO Control Registers (EQADC_CFCR) to invalidate the entries of CFIFOx. Perform any other modifications to EQADC_CFCR except changing MODEx from Disabled.
7. Configure the DMAC to respond to DMA requests generated by CFFFx and RFDFx.
8. Change MODEx to the modified CFIFO operation mode. Write ‘1’ to SSEx to trigger CFIFOx if MODEx is software trigger.

23.6.5 CQueue and RQueues usage

Figure 487 is an example of CQueue and RQueue usage. It shows the CQueue0 commands requesting results that will be stored in RQueue0 and RQueue1, and CQueue1 commands requesting results that will be stored only in RQueue1. Some Command Messages request data to be returned from the on-chip ADC/external device, but some only configure them and do not request returning data. When a CQueue contains both write and read commands like CQueue0, the CQueue and RQueue entries will not be aligned; as shown in *Figure 487*, the result for the second command of CQueue0 is the first entry of RQueue0. The figure also shows that CQueue and RQueue entries can also become unaligned even if all commands in a CQueue request data as CQueue1. CQueue1 entries became unaligned to RQueue1 entries because a result requested by the forth CQueue0

command was sent to RQueue1. This happens because the system can be configured so that several CQueues can have its results sent to a single RQueue.

Figure 487. eQADC command and result queues



23.6.6 ADC result calibration

The ADC result calibration process consists of two steps: determining the gain and offset calibration constants, and calibrating the raw results generated by the on-chip ADCs by solving the following equation discussed in *Section 23.5.6.6: ADC calibration feature*.

Equation 4 CAL_RES = GCC * RAW_RES + OCC+2;

The calibration constants GCC and OCC can be calculated from *Equation 4* provided that two pairs of expected (CAL_RES) and measured (RAW_RES) result values are available for two different input voltages. Most likely calibration points to be used are 25% VREF[u] and 75% VREF since they are far apart but not too close to the end points of the full input voltage range. This allows for calculations of more representative calibration constants. The eQADC provides these voltages via channel numbers 43 and 44. The raw, uncalibrated results for these input voltages are obtained by converting these channels with conversion commands that have the CAL bit negated.

The transfer equations for when sampling these reference voltages are:

$CAL_RES_{75\%VREF} = GCC * RAW_RES_{75\%VREF} + OCC+2;$

$CAL_RES_{25\%VREF} = GCC * RAW_RES_{25\%VREF} + OCC+2;$

Thus,

Equation 5 $GCC = (CAL_RES_{75\%VREF} - CAL_RES_{25\%VREF}) / (RAW_RES_{75\%VREF} - RAW_RES_{25\%VREF});$

Equation 6 $OCC = CAL_RES_{75\%VREF} - GCC*RAW_RES_{75\%VREF} - 2;$

or

Equation 7 $OCC = CAL_RES_{25\%VREF} - GCC*RAW_RES_{25\%VREF} - 2;$

After being calculated, the GCC and OCC values must be written to ADC registers: ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR) and ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR), using write configuration commands.

The eQADC will automatically calibrate the results, according to *Equation 4*, of every conversion command that has its CAL bit asserted using the GCC and OCC values stored in the ADC calibration registers.

Note: For accurate calibration, the 25% VREF channel must be converted using the Long Sample Time (LST) setting for either 64 or 128 ADC sample cycles in the ADC Conversion Command Message (LST = 0b10 or 0b11).

23.6.6.1 MAC configuration procedure

The following steps illustrate how to configure the calibration hardware, namely, determining the values of the gain and offset calibration constants, and the writing of these constants to the calibration registers. The procedure below should be performed for ADC0 and for ADC1.

1. Convert channel 44 with a command that has its CAL bit negated and obtain the raw, uncalibrated result for 25%VREF ($RAW_RES_{25\%VREF}$).
2. Convert channel 43 with a command that has its CAL bit negated and obtain the raw, uncalibrated result for 75%VREF ($RAW_RES_{75\%VREF}$).
3. Since the expected values for the conversion of these voltages are known ($CAL_RES_{25\%VREF}$ and $CAL_RES_{75\%VREF}$), GCC and OCC values can be calculated from *Equation 5* and *Equation 6* using these values, and the ones determined in steps

u. VREF = VRH – VRL

1 and 2.

4. Reformat GCC and OCC to the proper data formats as specified in *Section 23.5.6.6.2: MAC unit and operand data format*. GCC is an unsigned 15-bit fixed point value and OCC is a signed 14-bit value.
5. Write GCC value to ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR), and OCC value to ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR), using write configuration commands.

23.6.6.2 Example

The raw results obtained when sampling reference voltages 25%VREF and 75%VREF were, respectively, 3798 and 11592. The results that should have been obtained from the conversion of these reference voltages are, respectively, 4096 and 12288. Therefore, using *Equation 5* and *Equation 6*, the gain and offset calibration constants are:

GCC = (12288 – 4096) / (11592 – 3798) = 1.05106492 -> 1.05102539 = 0x4344

OCC = 12288 – 1.05106492 × 11592 – 2 = 102.06 -> 102 = 0x0066

Table 310 shows, for this particular case, examples of how the result values change according to GCC and OCC when result calibration is executed (CAL = 1) and when it is not (CAL = 0).

Table 310. Calibration example

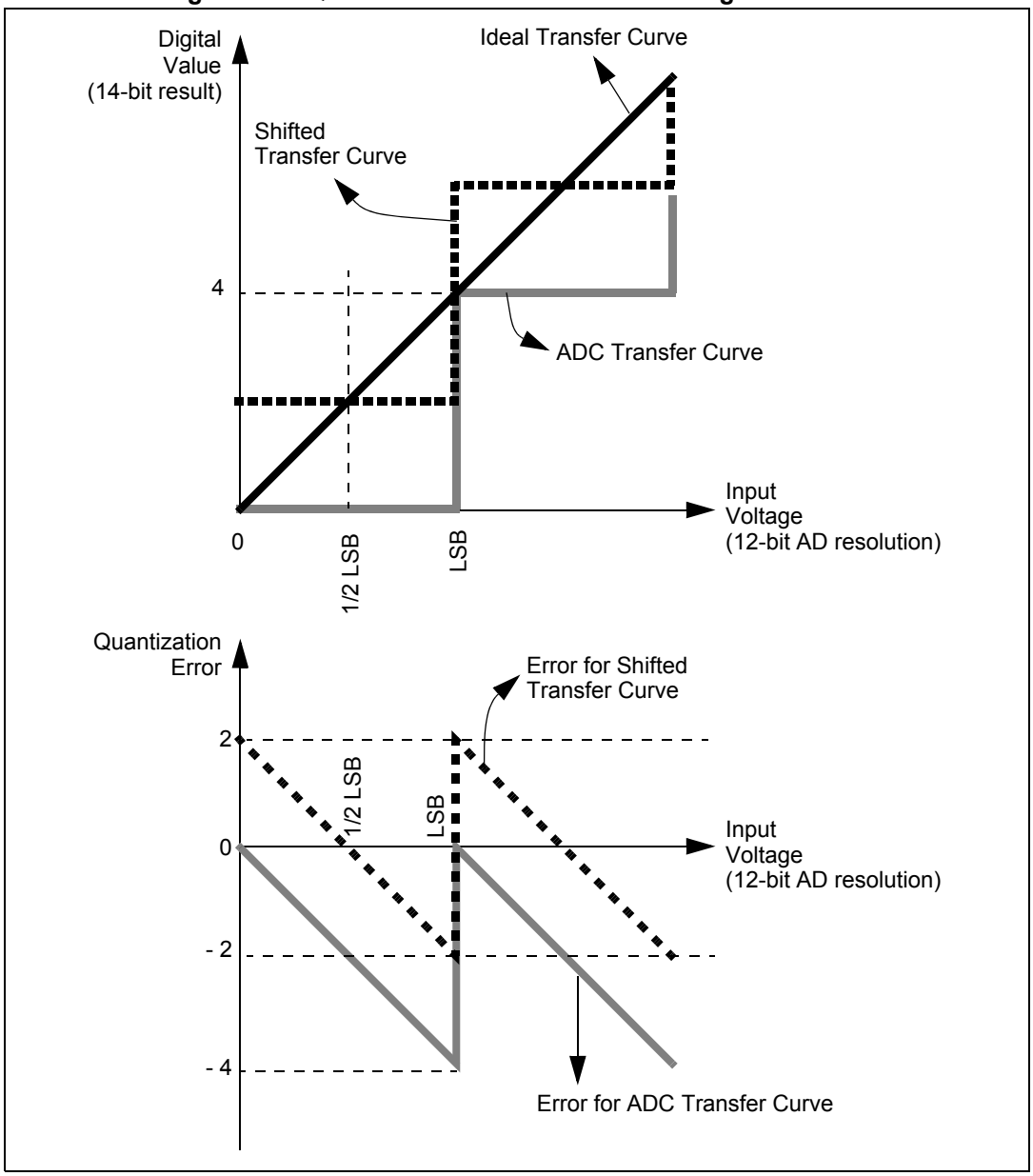
Input voltage	Raw result (CAL = 0)		Calibrated result (CAL = 1)	
	Hexadecimal	Decimal	Hexadecimal	Decimal
25% VREF(1)	0x0ED6	3798	0x1000	4095.794
75% VREF	0x2D48	11592	0x3000	12287.486

1. For accurate calibration, the 25% VREF channel must be converted using the Long Sample Time (LST) setting for either 64 or 128 ADC sample cycles in the ADC Conversion Command Message (LST = 0b10 or 0b11).

23.6.6.3 Quantization error reduction during calibration

Figure 488 shows how the ADC transfer curve changes due to the addition of two to the MAC output during the calibration—see MAC output equation at *Section 23.5.6.6.1: Overview*. The maximum absolute quantization error is reduced by half leading to an increase in accuracy.

Figure 488. Quantization error reduction during calibration



23.6.7 eQADC versus QADC

This section describes how the eQADC upgrades the QADC functionality. The section also provides a comparison between the eQADC and QADC in terms of their functionality. This section targets the users familiar with terminology in QADC. *Figure 489* is an overview of a QADC. *Figure 490* is an overview of the eQADC system.

Figure 489. QADC overview

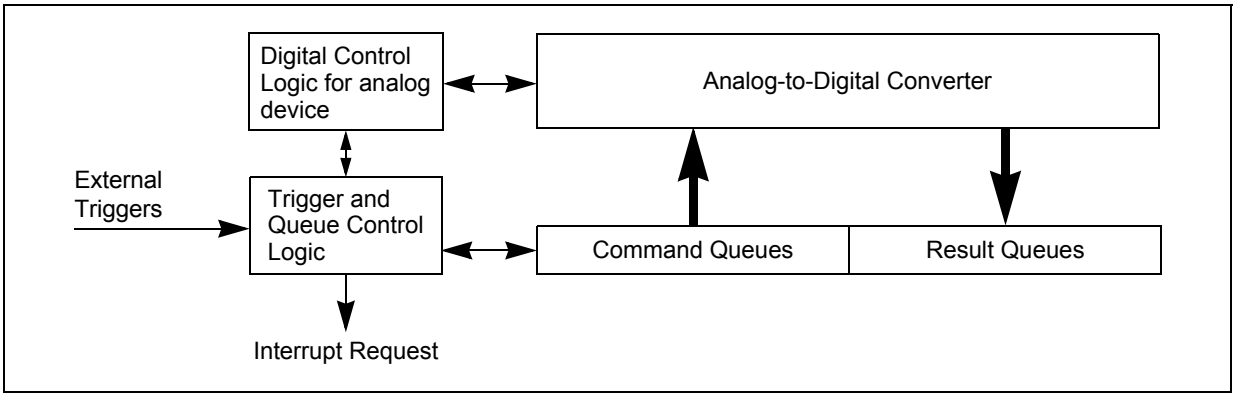
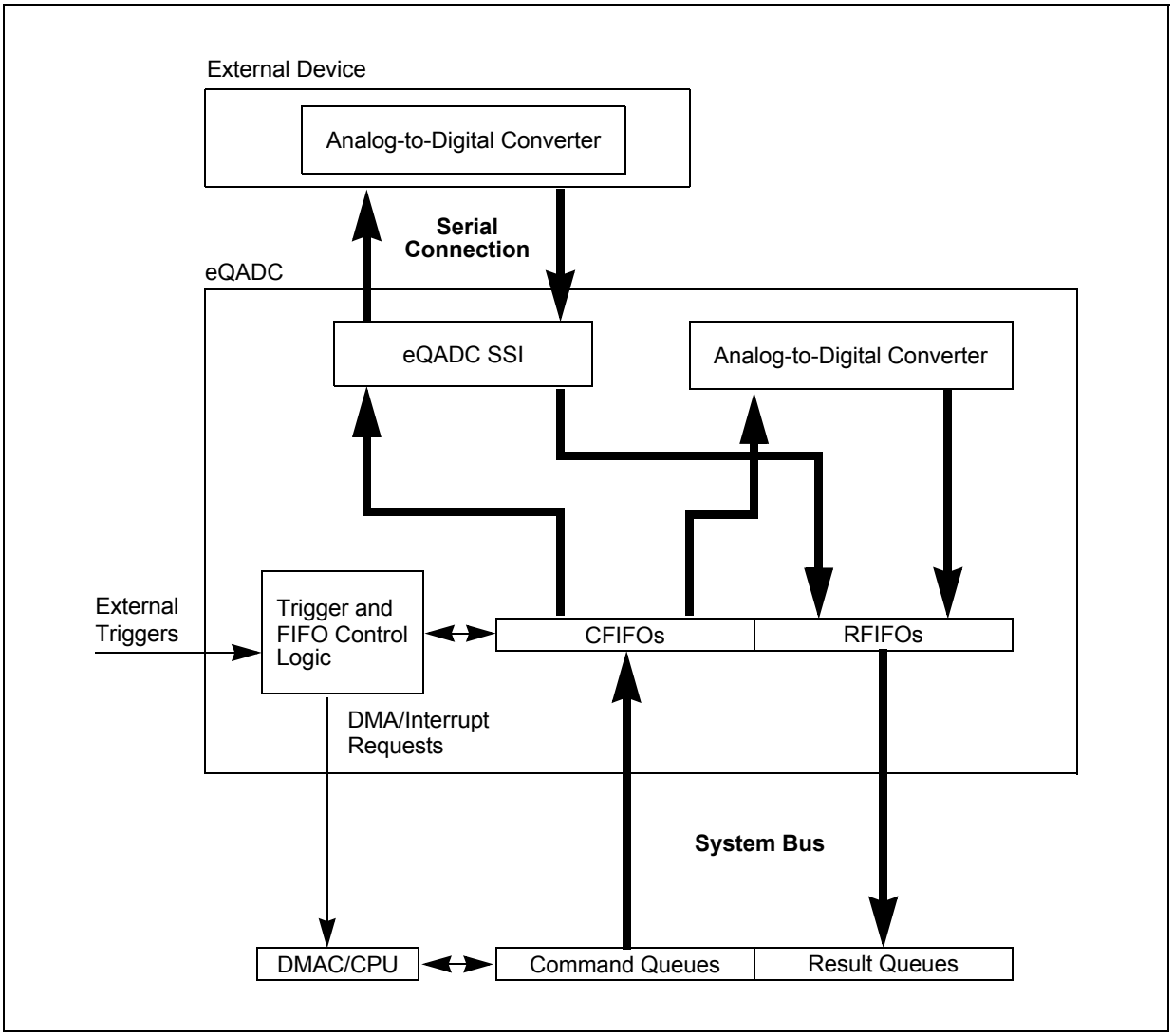


Figure 490. eQADC system overview



The eQADC system consists of four parts: queues in RAM, the eQADC, on-chip ADCs, and an external device. As compared with the QADC, the eQADC system requires two pieces of extra hardware.

1. A DMA or an MCU is required to move data between the eQADC's FIFOs and Queues in the system memory.
2. A serial interface (eQADC Synchronous Serial Interface—eQADC SSI) is implemented to transmit and receive data between the eQADC and the external device.

Since there are only FIFOs inside the eQADC, much of the terminology or use of the register names, register contents, and signals of the eQADC involve "FIFO" instead of "Queue". These register names, register contents, and signals are functionally equivalent to the "Queue" counterparts in the QADC. *Table 311* lists how the eQADC register, register contents, and signals are related to QADC.

Table 311. Terminology comparison between QADC and eQADC

QADC terminology	eQADC terminology	Function
CCW	Command Message	In the QADC, the hardware only executes conversion command words. In the eQADC, not all commands are conversion commands; some are configuration commands.
Queue Trigger	CFIFO Trigger	In the QADC, a trigger event is required to start the execution of a queue. In the eQADC, a trigger event is required to start command transfers from a CFIFO. When a CFIFO is TRIGGERED and transferring, commands are continuously moved from CQueues to CFIFOs. Thus, the trigger event initiates the "execution of a queue" indirectly.
Current Word Pointer Queue x (CWPQx)	Counter Value of Commands Transferred from Command FIFOx (TC_CFx)	In the QADC, CWPQx allows the last executed command on queue x to be determined. In the eQADC, the TC_CFx value allows the last transferred command on CQueue x to be determined.
Queue Pause Bit (P)	CFIFO Pause Bit	In the QADC, detecting a pause bit in the CCW will pause the queue execution. In the eQADC, detecting a pause bit in the Command will pause command transfers from a CFIFO.
Queue Operation Mode (MQx)	CFIFO Operation Mode (MODEx)	The eQADC supports all queue operation modes in the QADC except operation modes related to a periodic timer. A timer elsewhere in the system can provide the same functionality if it is connected to ETRIGx.
Queue Status (QS)	CFIFO Status (CFSx)	In the QADC, the Queue Status is read to check whether a queue is idle, active, paused, suspended, or trigger pending. In the eQADC, the CFIFO Status is read to check whether a queue is IDLE, WAITING FOR TRIGGER (idle or paused in QADC), or TRIGGERED (suspended or trigger pending in QADC). Which CFIFO is currently "active" can be determined by reading the LCFTCB*n* field in the EQADC_CFSSRs.

The eQADC and QADC also have similar procedures for the configuration or execution of applications. *Table 312* shows the steps required for the QADC versus the steps required for the eQADC system.

Table 312. Usage comparison between QADC and eQADC system

Procedure	QADC	eQADC system
Analog Control Configuration	Configure analog device by writing to the QADC registers	Program configuration commands into command queues
Prepare Scan Sequence	Program scan commands into command queues	Program scan commands into command queues
Queue Control Configuration	Write to the QADC Control Registers	Write to the eQADC Control Registers
Data Transferred between Queues and Buffers	Not required	Program the DMAC or the CPU to handle the data transfer
Serial Interface Configuration	Not required	Write to the eQADC SSI Registers
Queue Execution	Require Software or External Trigger events to start queue execution	Require Software or External Trigger events to start command transfers from a CFIFO

24 Decimation Filter

24.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

24.1.1 Device-specific features

The decimation filter allows ADC samples to be collected and streamed to the decimation filter in the time domain, but allows the results to be pulled from the decimation filter in the angle domain.

Not all features described in this chapter are included in this device. Specifically, the following features are **not** included:

- DECFILTER_IB register field INTAG
- MCR signals (ipp_integ_*)
- Level-sensitive trigger mode (TMODE configurations ‘01’ and ‘11’)
- Cascaded mode (only one instance of decimation filter is supported)

24.2 Introduction

24.2.1 Overview

The decimation filter is a dedicated hardware block, designed to decimate fixed point sample conversion results, generated by master block, usually an eQADC. A dedicated parallel side interface (PSI) provides bi-directional communication between the master block and the filter. A second interface is provided for use by the CPU, allowing setup of the filter parameters and read/write of the configuration registers.

The decimation filter receives data samples from the master block (eQADC) in the PSI RX subblock. Each sample arrives at the decimation filter with an identifier tag and associated commands. The input information is decoded by the PSI RX and control logic subblocks. When receiving a filtering command, the data is transferred to the filter TAP (test access port) register’s subblock and is processed by the filter using the MAC, the coefficient register, and the control logic subblocks. Then the result is returned to the master block by the PSI TX subblock. This result is accompanied by the corresponding tag information that provides an address for the data.

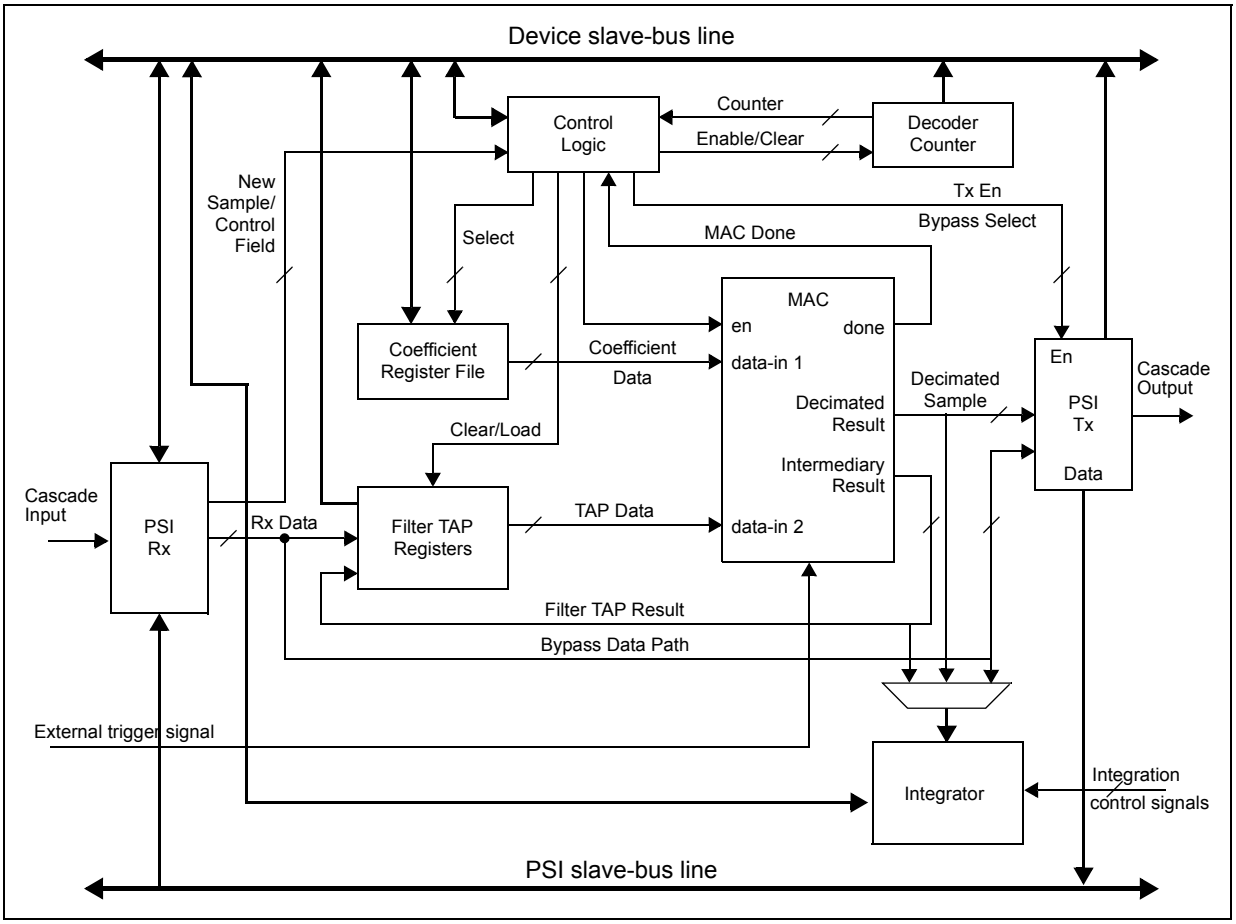
To summarize normal mode, the decimation filter works as a slave block on this second slave-bus line, and there is a PSI master block such as the eQADC to send and read data. This is illustrated in the application example in *Section 24.7: Application information*.

The decimation filter can also work in a standalone mode. In this mode, the input data is supplied and the output results are read by the chip core processor (CPU) using status and interrupt signals or DMA requests.

All signals in the interface are generated in the system clock domain.

Figure 491 is the block diagram for the decimation filter.

Figure 491. Decimation filter block diagram



24.2.2 Features

decimation filter block includes these distinctive features:

- Selectable 4th order Infinite Impulse Response (IIR) filter, or an 8th order Finite Impulse Response (FIR) filter
  - Input/output with 16-bit (fixed point) two's complement signed values
  - Internal TAPs with 16-bit (feed-forward portion of first IIR) and 24-bit (feedback portion) resolutions (fixed point) for two's complement signed value
  - 24-bit programmable filter coefficients (fixed point) for two's complement signed value
  - MAC unit with 51-bit fixed point accumulator
  - Convergent rounding methodology
  - Two's complement overflow or saturation selection
  - 58 clock cycles to process the input
- Implements a local slave-bus interface to a master block (e.g. the eQADC block)
- Input and output buffers with DMA capability
- Slave-bus interface to device
- Filter TAPs access for debug
- Filter initialization (flush) and stabilization (prefill) commands
- Timestamp support
- Decimation controlled by an internal counter or from an on-chip independent trigger signal (triggered output result).

24.2.3 Modes of operation

This section describes the operation modes of the FMPLL. The modes are selected using the DECFILTER_MCR fields MDIS, FREN, FRZ, ISEL, MIXM, and CASCD (see *Section 24.4.2.1: Decimation Filter Module Configuration Register (DECFILTER_MCR)*). The mode selection is summarized in *Table 313*.

Table 313. Operation mode selection

Mode	MDIS	FREN, FRZ	ISEL	MIXM	CASCD[1:0]
Normal	0	(0, 0) or (0, 1) or (1, 0)	0	0	00
Standalone			1	0	
PSI Input Mixed(1)			0	1	
PSI Output Mixed(1)			1	1	
Cascade			0 or 1	0	01 or 10 or 11
Freeze(2)		1, 1	X	X	X
Low Power	1	X	X	X	X

1. PSI input and PSI output mixed modes are not included on this device.
2. Freeze mode can also be activated from outside the decimation filter, depending on the MCU, if FREN = 1.

24.2.3.1 Normal Mode

This is the default operational mode of the decimation filter block. It corresponds to the prefill/filter operation with input data supplied through the PSI slave-bus interface (i.e. its input data is the ADC conversion result), with output going to the same PSI.

24.2.3.2 Standalone Mode

Standalone mode differs from normal mode because the input data is not supplied by the master block through the PSI slave-bus interface. In this case, the data is provided by the central processor using the device slave-bus interface or DMA interface signals. Once the data is filtered the decimated result is available in the Output Buffer register. The filter output is also consumed by a CPU or DMA mastering the same device slave-bus interface. This operation mode can be used to debug the filter stability or to decimate data in System RAM.

24.2.3.3 Low Power Mode

Low power mode corresponds to the module disable mode or stop mode. In the module disable mode the PSI slave-bus line is disabled and it is not possible to enter Freeze mode. The system clock is stopped. And in stop mode, the system clock is also stopped.

24.2.3.4 Freeze Mode

This mode is also known as debug mode. All filter action is frozen, either through software or by the hardware SoC debug request signal. If a freeze request comes when the filter is processing an input, it enters freeze mode only after the processing finishes.

24.3 External signal description

Note: The decimation filter does not provide metastability protection nor filtering for these signals.

24.3.1 Decimation trigger signal

This signal is used to control the output of the decimation filter, allowing decimation to be driven externally. For more details, see *Section 24.5.4.2: Triggered output result description*.

24.4 Memory map and register definition

This section provides the memory maps and detailed descriptions of all registers. Accesses to reserved areas of the memory map can return a bus error, depending on the device integration.

This module communicates with two distinct slave-bus lines. One is related to the device integration and the second is related to a PSI master block for data transfers. *Table 314* and *Table 324* describe the two memory maps.

24.4.1 Decimation filter device memory map

The addresses of the decimation filter registers are specified as offsets from the module's base address, described in *Table 314*. The registers allocated in this memory map are sufficient for a 4th order IIR filter implementation.

Table 314. Block memory map

Address offset	Register	Access	Reset value	Location
0x000	Decimation Filter Module Configuration Register (DECFILTER_MCR)	R/W	—(1)	on page 835
0x004	Decimation Filter Module Status Register (DECFILTER_MSR)	R/W	0x0000_0000	on page 840
0x00C–0x00F	Reserved	—	—	—
0x010	Decimation Filter Interface Input Buffer Register (DECFILTER_IB)	R/W	0x0000_0000	on page 843
0x014	Decimation Filter Interface Output Buffer Register (DECFILTER_OB)	R	0x0000_0000	on page 844
0x018–0x01F	Reserved	—	—	—
0x020	Decimation Filter Coefficient 0 Register (DECFILTER_COEF0)	R/W	0x0000_0000	on page 844
0x024	Decimation Filter Coefficient 1 Register (DECFILTER_COEF1)	R/W	0x0000_0000	on page 844
0x028	Decimation Filter Coefficient 2 Register (DECFILTER_COEF2)	R/W	0x0000_0000	on page 844
0x02C	Decimation Filter Coefficient 3 Register (DECFILTER_COEF3)	R/W	0x0000_0000	on page 844
0x030	Decimation Filter Coefficient 4 Register (DECFILTER_COEF4)	R/W	0x0000_0000	on page 844
0x034	Decimation Filter Coefficient 5 Register (DECFILTER_COEF5)	R/W	0x0000_0000	on page 844
0x038	Decimation Filter Coefficient 6 Register (DECFILTER_COEF6)	R/W	0x0000_0000	on page 844
0x03C	Decimation Filter Coefficient 7 Register (DECFILTER_COEF7)	R/W	0x0000_0000	on page 844
0x040	Decimation Filter Coefficient 8 Register (DECFILTER_COEF8)	R/W	0x0000_0000	on page 844
0x044–0x077	Reserved	—	—	—
0x078	Decimation Filter TAP0 Register (DECFILTER_TAP0)(2)	R	0x0000_0000	on page 845
0x07C	Decimation Filter TAP1 Register (DECFILTER_TAP71)	R	0x0000_0000	on page 845
0x080	Decimation Filter TAP2 Register (DECFILTER_TAP2)	R	0x0000_0000	on page 845
0x084	Decimation Filter TAP3 Register (DECFILTER_TAP3)	R	0x0000_0000	on page 845
0x088	Decimation Filter TAP4 Register (DECFILTER_TAP4)	R	0x0000_0000	on page 845
0x08C	Decimation Filter TAP5 Register (DECFILTER_TAP5)	R	0x0000_0000	on page 845

Table 314. Block memory map (continued)

Address offset	Register	Access	Reset value	Location
0x090	Decimation Filter TAP76 Register (DECFILTER_TAP6)	R	0x0000_0000	*on page 845*
0x094	Decimation Filter TAP7 Register (DECFILTER_TAP7)	R	0x0000_0000	*on page 845*
0x098–0x0CF	Reserved	—	—	—
0x0D0	Decimation Filter Interface Enhanced Debug Input Data Register (DECFILTER_EDID)	R	0x0000_0000	*on page 845*
0x0D4–0x0DF	Reserved	—	—	—
0x0F0–0x1FF	Reserved	—	—	—

1. See register description
2. The TAP register stores, on each filter node, the input sample data and, for the IIR type, the filter intermediary results.

24.4.2 Decimation filter register descriptions

All registers are 32-bit wide.

24.4.2.1 Decimation Filter Module Configuration Register (DECFILTER_MCR)

The decimation filter module configuration register provides configuration control bits for the decimation filter internal logic. One must not write to this register when the status bit DECFILTER_MSR[BSY] is set.

Address: DECFILTER_BASE + 0x000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FREN	0	FRZ	0	CASCD[1:0]		IDEN	ODEN	ERREN	0	FTYPE[1:0]		0	SCAL[1:0]	
W					SRES											
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	SAT	ISEL	MIXM	DEC_RATE[3:0]					DSEL	IBIE	OBIE	EDME	TORE	TMODE	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 492. Decimation Filter Module Configuration Register (DECFILTER_MCR)

1. Reset value is defined by the MDIS_DEFAULT parameter value.

Table 315. DECFILTER_MCR field descriptions

Field	Description
MDIS	Module Disable The MDIS bit puts the decimation filter in low power mode. Communication through the PSI slave-bus Interface is ignored in this mode. Writes to the configuration register are allowed with the exception of writes to the FREN and SRES bits, which are ignored. Writes to the Coefficient registers are also allowed. The decimation filter cannot enter Freeze mode once in disable mode. Once the module is disabled it no longer receives the system clock. 1: Low Power Mode 0: Normal Mode
FREN	Freeze Enable The FREN bit enables the decimation filter to enter freeze mode if the SoC debug request signal or the FRZ bit is asserted. See *Section 24.5.13: Freeze Mode description* for more details. 1: Decimation filter Freeze mode enabled 0: Decimation filter Freeze mode disabled
FRZ	Freeze Mode The FRZ bit controls the freeze mode of the decimation filter. For this bit to take effect the FREN freeze enable bit also needs to be asserted. While in freeze mode the MAC operations are halted. See *Section 24.5.13: Freeze Mode description* for more details. 1: Decimation filter in Freeze Mode 0: Decimation filter in Normal Mode
SRES	Software-reset bit The SRES is a self-negated bit which provides the CPU with the capability to initialize the decimation filter through the slave-bus interface. This bit always reads as zero. See *Section 24.5.10: Soft-Reset command description* for more details. 1: Software-Reset 0: No action
CASCD[1:0]	Cascade Mode Configuration The CASCD[1:0] bit field configures the block to work in cascade mode of operation as shown below. 00: No cascade mode (single block) 01: Cascade Mode, Head block configuration 10: Cascade Mode, Tail block configuration 11: Cascade Mode, Middle block configuration
IDEN	Input Data Interrupt Enable The IDEN bit enables the decimation filter to generate interrupt requests on all new input data written to the Interface Input Buffer register or Input/Output Buffers register. 1: Input Data Interrupt Enabled 0: Input Data Interrupt Disabled
ODEN	Output Data Interrupt Enable The ODEN bit enables the decimation filter to generate interrupt requests on all new data written to the filter Output buffer. It is independent of how ISEL and MIXM are set. 1: Output Data Interrupt Enabled 0: Output Data Interrupt Disabled

Table 315. DECFILTER_MCR field descriptions (continued)

Field	Description
ERREN	Error Interrupt Enable The ERREN bit enables the decimation filter to generate interrupt requests based on the assertion of the DECFILTER_MSR error flags OVF, DIVR, OVR or IVR. 1: Error Interrupts Enabled 0: Error Interrupts Disabled
FTYPE[1:0]	Filter Type Selection bits The FTYPE[1:0] bits select the filter type as shown in *Table 316*.
SCAL[1:0]	Filter Scaling Factor The SCAL[1:0] bit field selects the scaling factor used by the filter algorithm as shown below. 00: Scaling Factor = 1 01: Scaling Factor = 4 10: Scaling Factor = 8 11: Scaling Factor = 16
SAT	Saturation Enable The SAT bit enables the saturation of the filter output. See *Section 24.5.6.2: Saturation* for more details. 1: Enable Saturation 0: Disables Saturation
ISEL	Input Selection The ISEL bit selects the source of input data to the filter. Possible data sources are the master block of the PSI slave-bus interface, or the CPU/DMA on the device slave-bus interface. Each device slave-bus write to the Interface Input Buffer register or DMA transfer to the input buffer is interpreted as a new sample to be processed by the filter. The output interface used is the same as the one selected by ISEL if the output selection bit MIXM = 0. When MIXM = 1, the output selection (slave-bus or device slave-bus) is contrary to the input selection (see *Table 317* and the MIXM bit definition), configuring a mixed mode operation. The slave-bus interface can always read the input/output buffers, however the PSI slave-bus interface can only read the output buffer by request of the decimation filter, in normal or input mixed modes. This behavior is outlined in detail in *Table 317*. 1: Filter input from the device slave-bus interface 0: Filter input from PSI slave-bus interface **Note:** ISEL completely selects the output when the filter is configured as cascade tail, and is ignored when it is configured as cascade middle. ISEL must not be modified during the filter operation (when the status bit DECFILTER_MSR[BSY] is set). In addition, the interface not selected by ISEL must not be used to write into the input buffer.
MIXM	Mixed Mode The MIXM field selects the interface used for filter output, either device slave-bus or the PSI slave-bus, in relation to the interface selected by ISEL (see ISEL bit definition and *Table 317* for more details): 1: Interface NOT selected by ISEL is used for output, configuring mixed mode. 0: Interface selected by ISEL is used for output, configuring normal or standalone mode **Note:** MIXM must be set to 0 (zero) when the filter is configured as cascade mode.

Table 315. DECFILTER_MCR field descriptions (continued)

Field	Description
DEC_RATE[3:0]	Decimation Rate Selection The DEC_RATE[3:0] field selects the decimation rate used by the decimation filter. The decimation rate defines the number of data samples from the master block that is required to generate one decimated result in the decimation filter output. 0000: No Decimation: one filter output for each sample input 0001–1111: One filter output for each (DEC_RATE+1) sample inputs
DSEL	DMA Selection The DSEL bit determines whether the data transfers — to the input buffer (write to) and from the output buffer (read from) — are performed by DMA requests or by interrupt requests. This bit can also be active when PSI input is selected with Enhanced Debug (ISEL = 0, EDME = 1), in which case, the input buffer generates read requests only (see *Section 24.5.14: Enhanced Debug Monitor description*). 1: DMA requests are generated 0: Interrupt requests are generated
IBIE	Input Buffer Interrupt Request Enable The IBIE bit enables the decimation filter to generate interrupt requests when: – device slave-bus input is selected (ISEL = 1) and DSEL = 0 when the input buffer is available to receive new data; – PSI input is selected with Enhanced debug (ISEL = 0, EDME = 1) and DSEL = 0 when the input buffer has data to be read by the device CPU. 1: Input Buffer Interrupt Request Enabled 0: Input Buffer Interrupt Request Disabled
OBIE	Output Buffer Interrupt Request Enable The OBIE bit enables the decimation filter interrupt requests when outputs are directed to the device slave-bus (ISEL != MIXM) and DMA is not selected (DSEL = 0). 1: Output Buffer Interrupt Request Enabled 0: Output Buffer Interrupt Request Disabled
EDME	Enhanced Debug Monitor Enable The EDME bit defines the enhanced debug monitor when input selection is from PSI (ISEL = 0). In this case, the raw data fed from the PSI Master block is also, in parallel, made available in the register DECFILTER_EDID (see *Section 24.5.14: Enhanced Debug Monitor description*), generating and input interrupt or DMA request. 1: Enhanced debug monitor enabled (read requests of input data from master block enabled) 0: Enhanced debug monitor disabled
TORE	Triggered Output Result Enable The TORE bit enables an input trigger signal to force the decimation filter to send the next result of the filter back to the master block. For more details, see *Section 24.5.4.2: Triggered output result description*. 1: Output buffer update using an external signal is enabled 0: Output buffer update using an external signal is disabled **Note:** TORE must only be asserted when PSI is selected as output (normal or PSI output mixed modes). TORE must not be asserted with the filter bypassed (FTYPE = 00).

Table 315. DECFILTER_MCR field descriptions (continued)

Field	Description
TMODE[1:0]	Trigger Mode The TMODE field selects the way the trigger signal controls the output result sampling function enabled by the TORE bit: 00: Output is posted at the rising edge of the trigger signal 01: Output is posted whenever the trigger signal is a logical 0 10: Output is posted at the falling edge of the trigger signal 11: Output is posted whenever the trigger signal is a logical 1 **Note:** The TMODE definition replaces, and is upward compatible with, the TRFE bit definition found in previous versions of the decimation filter.

Table 316. FTYPE[1:0] configuration—Filter type selection

FTYPE[1:0]	Description
00	Filter bypass(1)
01	IIR filter—1 × 4th order
10	FIR filter—1 × 8th order
11	Reserved

1. In Bypass configuration the filter is disabled.

Table 317. ISEL/MIXM definition – read/write to input/output buffers

ISEL	MIXM	Mode	Operation	Device slave-bus interface		PSI slave-bus interface	
				Input buffer	Output buffer	Input buffer	Output buffer
0	0	Normal	Read	Always, by DMA or interrupt(1) request if EDME = 1	Always, no DMA or interrupt	Forbidden, write only	By decfil request(2)
	1	PSI Input Mixed			Always, issues DMA or interrupt(1)		Disabled
	0	Normal	Write	Forbidden	No effect, read only	Enabled	Read only
	1	PSI Input Mixed					
1	0	Standalone	Read	Always, no DMA or interrupt	Always, issues DMA or interrupt(1)	Forbidden, write only	Disabled
	1	PSI Output Mixed			Always, no DMA or interrupt		By decfil request(2)
	0	Standalone	Write	Enabled, by DMA or interrupt request(1)	No effect, read only	Forbidden	Read only
	1	PSI Output Mixed					

1. Bit DSEL selects between interrupt or DMA request.
2. Decimation filter issues a read request to the master block.

24.4.2.2 Decimation Filter Module Status Register (DECFILTER_MSR)

Address: DECFILTER_BASE + 0x004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BSY	0	DEC_COUNTER[3:0]				0	0	0	0	0	0	0	0	0	0
W							IDFC	ODFC		IBIC	OBIC		DIVRC	OVFC	OVRC	IVRC
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	IDF	ODF	0	IBIF	OBIF	0	DIVR	OVF	OVR	IVR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 493. Decimation Filter Module Status Register (DECFILTER_MSR)

Table 318. DECFILTER_MSR field descriptions

Field	Description
BSY	Decimation Filter Busy indication The BSY bit indicates that the decimation filter is processing new input data from the master block in normal mode or from the device core in standalone mode. BSY is not asserted when the filter is disabled (FTYPE = 00). However, the BSY bit is asserted when the soft reset is executed. 1: Decimation filter busy 0: Decimation filter idle
DEC_COUNTER[3:0]	Decimation Counter The DEC_COUNTER[3:0] field indicates the current value of the DEC_COUNTER Decimation Counter (see *Figure 491*), which counts the number of input data samples received by the decimation filter. When the value of this counter matches the DEC_RATE[3:0] configuration register field, one decimated result is generated and the DEC_COUNTER counter is re-initialized at zero. This register is cleared by a soft reset or a flush command.
IDFC	Input Data Flag Clear bit The IDFC bit clears the IDF Flag bit in the status register. This bit is self negated, therefore it is always read as zero. 1: Clears IDF 0: No action
ODFC	Output Data Flag Clear bit The ODFC bit clears the ODF Flag bit in the status register. This bit is self negated, therefore it is always read as zero. 1: Clears ODF 0: No action
IBIC	Input Buffer Interrupt Request Clear bit The IBIC bit clears the IBIF Flag bit in the status register. This bit is self negated, therefore it is always read as zero. 1: Clears IBIF 0: No action

Table 318. DECFILTER_MSR field descriptions (continued)

Field	Description
OBIC	Output Buffer Interrupt Request Clear bit The OBIC bit clears the OBIF Flag bit in the status register. This bit is self negated, therefore it is always read as zero. 1: Clears OBIF 0: No action
DIVRC	DIVR Clear bit The DIVRC bit clears the DIVR Debug Filter Input Data Read Overrun indication bit in the status register. This bit is self negated, therefore it is always read as zero. 1: Clears DIVR 0: No action
OVFC	OVF Clear bit The OVFC bit clears the OVF Output Overflow bit in the status register. This bit is self negated, therefore it is always read as zero. 1: Clears OVF 0: No action
OVRC	OVR Clear bit The OVRC bit clears the OVR Output Overrun bit in the status register. This bit is self negated, therefore it is always read as zero. 1: Clears OVR 0: No action
IVRC	IVR Clear bit The IVRC bit clears the IVR Filter Input Overrun indication bit in the status register. This bit is self negated, therefore it is always read as zero. 1: Clears IVR 0: No action
IDF	Input Data Flag The IDF bit flag indicates when new data is available at the DECFILTER_IB register or at the DECFILTER_IOB register. This flag generates an Interrupt Request if enabled by the IDEN bit in the configuration register. This Flag is cleared by the IDFC Status bit or by a soft reset of the decimation filter. 1: New Sample received 0: Sample not received **Note:** OBS: This flag is not used for read / write requests. It is used only to announce the input data event. For read / write request flag, refer to IBIF.
ODF	Output Data Flag The ODF bit flag indicates when a new decimated sample is available at the DECFILTER_OB register or at the DECFILTER_IOB register. This flag generates an Interrupt Request if enabled by the ODEN bit in the configuration register. This flag is cleared by the ODFC Status bit or by a soft reset of the decimation filter. 1: New Decimated Output Sample available 0: No new Decimated Output Sample available **Note:** OBS: This flag is not used for read requests. It is used only to announce the output data event. For read request flag, refer to OBIF.

Table 318. DECFILTER_MSR field descriptions (continued)

Field	Description
IBIF	Input Buffer Interrupt Request Flag The IBIF bit flag indicates that the input buffer DECFILTER_IB is available to be filled with new data, when Enhanced Debug Monitor is off. In Enhanced Debug Monitor, it indicates the input buffer DECFILTER_IB was filled with a new sample and is ready to be read. IBIF assertion also asserts the interrupt signal when enabled by the IBIE bit in the configuration register when DMA is not selected (DSEL = 0) and the input buffer requires access from the device slave-bus (ISEL != EDME). This flag is cleared by the IBIC Status bit or by a soft reset of the decimation filter. 1: New Sample is requested (ISEL = 1, EDME = 0) or new sample is available in Enhanced Debug Monitor (ISEL = 0, EDME = 1). 0: No action
OBIF	Output Buffer Interrupt Request Flag The OBIF bit flag indicates that either a new decimated sample is available at the DECFILTER_OB register. This flag generates an Interrupt Request if enabled by the OBIE bit in the configuration register and with ISEL != MIXM and DSEL = 0. This flag is cleared by the OBIC Status bit or by a soft reset of the decimation filter. 1: New Decimated Output available 0: No new Decimated Output available
DIVR	Enhanced Debug Monitor Input Data Read Overrun The DIVR bit indicates that a received sample in the Filter Interface Input Register was overwritten by a new sample and was not read by the Core. This flag generates an Interrupt Request if enabled by DECFILTER_MCR[ERREN]. This flag is cleared by the DIVRC Status bit or by a soft reset of the decimation filter. 1: Enhanced Debug Monitor Input Data Read Overrun occurred 0: Input Data Read Overrun did not occur in Enhanced Debug monitor
OVF	Filter Overflow Flag The OVF bit indicates that an overflow occurred in the filtered sample result. This flag generates an Interrupt Request if enabled by DECFILTER_MCR[ERREN]. This flag is cleared by the OVFC Status bit or by a soft reset of the decimation filter. 1: Overflow occurred 0: No overflow
OVR	Output Interface Buffer Overrun The OVR bit indicates that a decimated sample was overwritten by a new sample in the Interface Output Buffer Register. This flag generates an Interrupt Request if enabled by DECFILTER_MCR[ERREN]. This flag is cleared by the OVRC Status bit or by a soft reset of the decimation filter. 1: Filter Output Overrun occurred 0: No Output Overrun
IVR	Input Interface Buffer Overrun The IVR bit indicates that a received sample in the Filter Interface Input Register was overwritten by a new sample. This was probably caused by a violation of the decimation filter maximum throughput. This flag generates an Interrupt Request if enabled by DECFILTER_MCR[ERREN]. This flag is cleared by the IVRC Status bit or by a soft reset of the decimation filter. 1: Input Buffer Overrun occurred 0: Input Buffer Overrun did not occur

24.4.2.3 Decimation Filter Interface Input Buffer Register (DECFILTER_IB)

The Input Buffer Register provides access to the Input buffer of the decimation filter when the filter is in the standalone mode of operation. Writes to this register are interpreted as requests to the decimation filter to process new sample data. Writes to this register when ISEL = 0 are not allowed.

Address: DECFILTER_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	INTAG[3:0]				0	0	0	0	0	0	PREFILL	FLUSH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 494. Decimation Filter Interface Input Buffer Register (DECFILTER_IB)

Table 319. DECFILTER_IB field descriptions

Field	Description
INTAG[3:0]	Decimation filter input tag bits The INTAG[3:0] bit field is defined as a selector signal and it is used to identify different destinations for the INBUF[15:0] data. When the PSI master block is an eQADC, it is used in PSI output mixed mode to address the appropriate RFIFO in the eQADC block.
PREFILL	Decimation Filter Prefill/Filter control bit The PREFILL bit selects the decimation filter operation mode. For more details, see *Section 24.5.7: Filter prefill control description*. 1: Decimation filter prefill sample 0: Decimation filter normal sample
FLUSH	Decimation Filter Flush control bit Assertion of the FLUSH bit initializes the decimation filter to a initial state, as defined in *Section 24.5.9: Flush Command description*. This bit is self negated and it is cleared only when the data is read and the flush is executed. 1: Flush request 0: No flush request
INPBUF[15:0]	Input Buffer Data The INPBUF[15:0] bit field carries the sample data to be filtered. This data buffer can be written from the PSI slave-bus interface or by the device slave-bus interface. See *Section 24.5.3: Input buffer description* for more details.

24.4.2.4 Decimation Filter Interface Output Buffer Register (DECFILTER_OB)

Address: DECFILTER_BASE + 0x014 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	OUTTAG[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 495. Decimation Filter Interface Output Buffer Register (DECFILTER_OB)

Table 320. DECFILTER_OB field descriptions

Field	Description
OUTTAG[3:0]	Decimation filter output tag bits The OUTTAG[3:0] bit field is defined as a selector signal and it is used to identify different destinations for the OUTBUF[15:0] data. When an eQADC is the PSI block master, it holds the same number used to address the destination RFIFO.
OUTBUF[15:0]	Output Buffer Data The OUTPBUF[15:0] bit field is the result data in the decimation filter Output Buffer. It represents a fixed point signed number in two's complement format and is updated only when a decimated result is ready to be transmitted, meaning it contains the last decimated result from the filter.

24.4.2.5 Decimation Filter Coefficient n Register (DECFILTER_COEFn)

Address: DECFILTER_BASE + 0x020–0x040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	8{COEFn[23]}								COEFn[23:16]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 496. Decimation Filter Coefficient n Register (DECFILTER_COEFn)

Table 321. DECFILTER_COEFn field descriptions

Field	Description
COEFn[23:0]	Coefficient n field The COEFn[23:0] bit fields are the digital filter coefficients registers. The coefficients are fractional signed values in two's complement format, in the range (-1 ≤ coef < 1). **Note:** Reads to this register are sign-extended, meaning the coefficient's sign bit is copied to all 8 most significant register bits. **Note:** Writing to these fields when DECFILTER_MSR[BSY] = 1 is not allowed.

24.4.2.6 Decimation Filter TAPn Register (DECFILTER_TAPn)

Address: DECFILTER_BASE + 0x078–0x094 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	8{TAPn[23]}								TAPn[23:16]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 497. Decimation Filter TAPn Register (DECFILTER_TAPn)

Table 322. DECFILTER_TAPn field descriptions

Field	Description
TAPn[23:0]	TAPn Register The read-only TAPn[23:0] bit fields shows the contents of the digital filter TAP registers, as fractional signed values in two's complement format, in the range (-1 ≤ coef < 1). **Note:** Reads to this register are sign-extended, meaning the coefficient's sign bit is copied to all 8 most significant register bits. **Note:** The content of these registers is meaningless when DECFILTER_MSR[BSY] = 1.

24.4.2.7 Decimation Filter Interface Enhanced Debug Input Data Register (DECFILTER_EDID)

The Enhanced Debug Input Data Register provides read-only access to the sample data received by the decimation filter when the input is selected from the device slave-bus (ISEL = 0), allowing the monitoring of filter operation. See *Section 24.5.14: Enhanced Debug Monitor description* for more details. Writes to this register are not allowed.

Address: DECFILTER_BASE + 0x0D0 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	SAMP_DATA[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 498. Decimation Filter Interface Enhanced Input Buffer Register (DECFILTER_EDID)

Table 323. DECFILTER_EDID field descriptions

Field	Description
SAMP_DATA[15:0]	Conversion Sample Data The SAMP_DATA[15:0] bit field carries the data that was loaded in the decimation filter to be processed by the FIR/IIR subblock. This conversion data is supplied by the PSI slave-bus interface only. See *Section 24.5.11: Interrupts requests description* and *Section 24.5.12: DMA requests description* for more details.

24.4.3 Decimation filter memory map for parallel side interface

The decimation filter exchanges data with the master block through the PSI. The master block sends data to the input buffers and reads data from the output buffers of the filter. To implement this exchange, only a single register is required as described in *Table 324*, therefore the PSI address is ignored.

Table 324. Parallel side interface memory map for decimation filter data exchange

Decimation filter address	Description	Access
0x0	DECFILTER_IOB—Decimation Filter(1) input/output Registers	R/W

1. The input registers are addressed for write only and output registers are addressed for read only.

24.4.4 PSI register description

This register is defined as 24-bit.

24.4.4.1 Decimation Filter Input/Output Buffers Register (DECFILTER_IOB)

The Input/Output Buffers Register is used by the PSI master block to access the Input and Output buffers of the decimation filter. Writes to this register are interpreted as requests to the decimation filter to process new sample data or to bypass timestamp data. Reads from this register frees the decimation filter output buffer from filtered data or from bypass timestamp data.

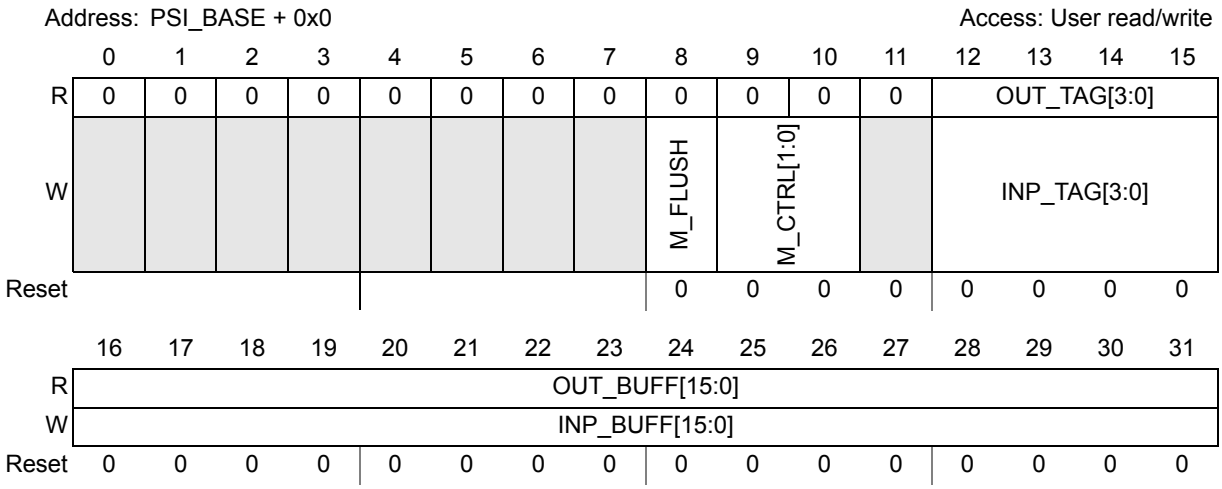


Figure 499. Decimation Filter Interface Input/Output Buffers Register (DECFILTER_IOB)

Table 325. DECFILTER_IOB field descriptions

Field	Description
M_FLUSH	Master block Flush request/control bit Assertion of the M_FLUSH bit initializes the decimation filter to a initial state, as defined in *Section 24.5.9: Flush Command description*. The sample or timestamp data written with a flush request in the same IOB register write is processed normally after the flush. 1: Flush request 0: No flush request
M_CTRL[1:0]	Decimation Filter mode control bits *Table 326* describes the M_CTRL[1:0] field functions. This field is used for control of the decimation filter.
INP_TAG[3:0]	Decimation filter input tag bits The INP_TAG[3:0] field indicates the destination associated with the INPBUFF[15:0] sample. This value is stored by the decimation filter and used to address the destination register when a decimated sample is available to be read by the master block. Since several input data samples can be received before a decimated result is generated, the INP_TAG[3:0] used for the decimated sample corresponds to the latest INP_TAG[3:0] received. Therefore it is expected that the tag field be constant during the decimation process.
OUT_TAG[3:0]	Decimation filter output tag bits The OUT_TAG[3:0] bit field is used to address the appropriate destination register in the master block for the accompanying OUTBUF[15:0] data. When this value is updated, it is a copy of the INP_TAG[3:0] value that was received with the last processed input data. When an eQADC is the PSI master block, this is used to address the appropriate RFIFO in the eQADC block.

Table 325. DECFILTER_IOB field descriptions (continued)

Field	Description
INP_BUFF[15:0]	Input Buffer Data The INP_BUFF[15:0] bit field is the data input from the master block. The input register can be written with this data when ISEL = 0. This data can be timestamp information that is not processed by the filter, or sample data that is processed by the digital filter. In this case, the information is a signed signal in two's complement format.
OUT_BUFF[15:0]	Output Buffer Data The OUT_BUFF[15:0] bit field corresponds to the data result of the decimation filter that has been processed to the master block. This data can be timestamp information or a digital filter result. In this case, the information is a signed signal in two's complement format.

Table 326. M_CTRL[1:0] field configuration

M_CTRL[1:0]	Description
00	PREFILL—A prefill indicates to the decimation filter to accept INP_BUFF[15:0] as valid data but no decimated samples are generated out of these master samples. The prefill function is used to initialize and stabilize the decimation filter without generating decimated samples.
01	CONVERSION RESULT—A conversion result indicates that the INP_BUFF[15:0] field is data to be treated as valid sample data and it is considered for decimation counting and output buffer update.
10	TIMESTAMP—A timestamp indicates that the INP_BUFF[15:0] field has data that bypasses the flow in the decimation filter logic, returning back to the master block without any modification when: – the previous accompanying data is not for prefill, and – when the previous accompanying input data is generating decimated filter output. Also, bit M_FLUSH is always 0 for timestamp data type.
11	Reserved

24.5 Functional description

24.5.1 Overview

Figure 491 shows the block diagram of the decimation filter. The Control Logic provides the control signals for all other submodules. The PSI data interface is subdivided into two submodules, transmitter and receiver, that are accessed by the PSI slave-bus interface. The bypass path is used when the filter is disabled and the incoming data can be transmitted back to the master block without being processed by the Filter algorithm. The filter hardware is implemented in such a way that an IIR (1 × 4 poles) or FIR filter type can be implemented. The selection between the two types of filter algorithms is implemented by the Control Logic subblock.

The Coefficient register file provides the digital filter coefficients. This block is a register bank with read/write access by the device slave-bus interface. The filter TAP registers are also accessed through the device slave-bus line interface, providing additional debug capabilities to the decimation filter block. The MAC (multiplier/accumulator) subblock executes the filter arithmetic operations controlled by the Control Logic. The MAC results

are routed to the filter TAP registers and to the output buffer when the result is a decimated filter sample.

24.5.2 Parallel Side Interface (PSI) description

This section describes the operation of the Parallel Side Interface (PSI) subblock which is responsible for communication and data exchange between the master block (for instance, the eQADC block) and the decimation filter block.

The decimation filter receives sample data from the master block. The input data bus format is presented in *Figure 499*. The sample data arrives along with the control bits. These control bits are decoded and the proper action is decided in the Control Logic subblock. When the decimation filter finishes its processing and a result is available, the read request signal is issued to the master block. This data transfer request remains set until the result is read by the master block.

When using two or more decimation filter blocks in the device, the output of the second block is connected to the input data of the next block, and the output of the first block is connected to the read data input of the PSI master block.

24.5.3 Input buffer description

The decimation filter receives data samples for filtering from a master block (e.g. eQADC) using the PSI, or from the CPU using the device slave-bus interface. The data source is selected by the ISEL bit of the module configuration register DECFILTER_MCR.

When the device slave-bus interface is selected and DMA operation is chosen (DECFILTER_MCR[DSEL] = 1), the input data request signal is asserted when the input buffer is empty. When DMA operation is not chosen (DSEL = 0) in standalone mode, the logic asserts an input interrupt request and the input buffer waits for data from the device slave-bus.

Input buffer filling is flagged by DECFILTER_MSR[IDF]. The IDF flag remains set, even after the input data has been consumed and the buffer is free, until it is cleared by software.

Input buffer overrun is detected and flagged by bit DECFILTER_MSR[IVR]. The idle (DECFILTER_MSR[BSY] = 0) decimation filter is able to receive two consecutive input writes without input overrun. For more details, see *Section 24.5.3.1: Input buffer overrun*.

When the selected input source is the PSI master block, the PSI master may send timestamp data after related sample data for filtering. The timestamp is sent back to the PSI following the respective filter output, using the same request mechanism. As the decimation filter takes several clock cycles to process a sample, the timestamp is copied into an internal timestamp register until the filtered output is sent out, therefore freeing the input buffer for yet another sample data.

When the input buffer is loaded with a sample data coming from the PSI and enhanced debug monitor enabled is enabled (DECFILTER_MCR bits ISEL = 0, EDME = 1), input DMA or interrupt requests are asserted, so that the PSI input samples can be monitored from the device slave-bus interface. These interrupt or DMA requests result in input read accesses from the DECFILTER_IB register, unlike write accesses needed when the same requests are made and the device slave-bus interface is selected for input (ISEL = 1). Only the sample data input (not timestamps), can be monitored in this way. See also *Section 24.5.14: Enhanced Debug Monitor description*.

Soft reset clears the input buffer indication flags and any data write/read request that was generated in any mode.

24.5.3.1 Input buffer overrun

An input overrun occurs when the input buffer is holding input data and new data is received by the filter. See *Section 24.5.3: Input buffer description* for details of the input buffer.

The input buffer overrun can occur in the following cases:

- When the input buffer has sample data to be processed but the filter is busy and another input (data or timestamp) is received.
- When the input buffer has a timestamp, the internal timestamp register is loaded and the next input data is received.

As an example of the input data sequence, assume that the filter is enabled and not busy, and all registers are empty. Then a word of sample data is received followed by a timestamp and another word of sample data. No input overrun occurs in this case, because the first sample is immediately transferred to the TAP input register, the timestamp is immediately transferred to the internal timestamp storage register, and the second sample can be held in the input buffer until the end of the processing of the sample data by the filter. The input overrun may occur if more input is received before the end of the processing, or if the filter is busy at the beginning of the received sequence.

When the filter is in bypass/disable mode (DECFILTER_MCR field FTYPE = 00), the data from the input buffer is transferred to the output buffer, if it is not already full. If the output buffer is full, the input buffer is loaded, and another word of input data is sent, then an input overrun occurs.

Note: Configuring ISEL = 1, MIXM = 1 and FTYPE = 00 (bypass), writes to the DECFILTER_IB are routed directly to the PSI output.

24.5.4 Output buffer description

The decimation filter has an output buffer to send filtering results to a master block using the PSI, or to the CPU using the device slave-bus, as selected through the DECFILTER_MCR bits ISEL and MIXM bits.

Filtering of prefill inputs do not update the output buffer, so the flag ODF is not set.

When filter types IIR and FIR are selected and the input source is the PSI master block (normal mode), the output buffer receives data from the MAC subblock or from the timestamp storage register. The result from the MAC is written immediately after the processing if the decimation is enabled. However, the timestamp data is enabled to be written in the output buffer only when the output buffer is empty, the decimation count is reached, and the corresponding data was also ready to be transmitted. When a new word of data is available in the output buffer, a read request signal is sent to the master block. The master block has to send the corresponding read commands to clear the read request signal. In this configuration, the core can always read the output buffer. The flag ODF is set when the buffer is updated.

When in filter operation mode with input from the device slave-bus (ISEL = 1), only sample data is processed by the filter, as there is no way to input timestamps. When the filter result from the MAC is ready, this is immediately written to the output buffer when the decimation count is reached. The flag ODF is set when the buffer is updated. It also generates a DMA read request if DECFILTER_MCR[DSEL] = 1 in standalone mode.

When the filter is bypassed (FTYPE = 00), and PSI is selected as output (normal mode), the data written into the input buffer waits until the output buffer is empty before passing the data. This is needed because the master block takes some clock cycles to send the read commands to the decimation filter after the read request signal is asserted. When the filter is bypassed and the device slave-bus is selected as output (standalone mode), the data written into the input buffer is immediately written into the output buffer. The flag ODF is set when the buffer is updated and a DMA read request is asserted if DSEL = 1.

Soft reset clears the output buffer, as well as any data read request generated in any mode.

24.5.4.1 Output buffer overrun

An output overrun occurs when the output buffer is holding output data (sample or timestamp) that has not been read is overwritten with another word of data (sample or timestamp). See *Section 24.5.4: Output buffer description* for details of the output buffer. Output overruns are flagged by the DECFILTER_MSR bit OVR.

Prefill inputs do not cause IIR or FIR output overrun in standalone and normal modes.

When PSI is selected as output (normal mode) and the filter is bypassed (FTYPE = 00), output overruns do not occur because the input buffer waits until the output buffer is empty to transfer the data to the output, but an input overrun may still occur (see *Section 24.5.3.1: Input buffer overrun*).

In standalone mode with output interrupt (DSEL = 0), an output overrun is flagged if a new output is ready and the output data flag OBIF is not cleared, even if the output buffer has not been read.

When the device slave-bus is selected for output and DSEL = 1 (DMA enabled), an output overrun is flagged if new output is ready and the previous result has not yet been read by the DMA controller, independently of ODF and OBIF flags. In bypass, the output overrun does not occur because the data written into the input buffer is written into the output buffer only when this buffer is empty, but an input overrun may still occur.

24.5.4.2 Triggered output result description

The posting of a filter output, either to the PSI or to the device slave-bus can be controlled by an additional input trigger signal (see *Section 24.3.1: Decimation trigger signal*). It allows the decimation to be controlled by another circuit, instead of the internal decimation counter.

Triggered output operation is enabled by the field DECFILTER_MCR[TORE]. When triggered output is enabled, the decimation count configured by DECFILTER_MCR field DEC_COUNTER is ignored. The decimation filter detects the rising edge or the falling edge of the trigger signal as selected by the configuration field TMODE in the DECFILTER_MCR. When the corresponding edge is detected, the output buffer is enabled to receive the next filter result.

The input trigger signal can also be used as a simple filtered output enable: in this trigger mode, when the input trigger signal is asserted, every filtered output is posted to the output buffer; no output is posted when the signal is negated. This trigger mode and the assertion polarity (active 0 or 1) of the input trigger signal is also defined in the DECFILTER_MCR field TMODE.

24.5.5 Bypass configuration description

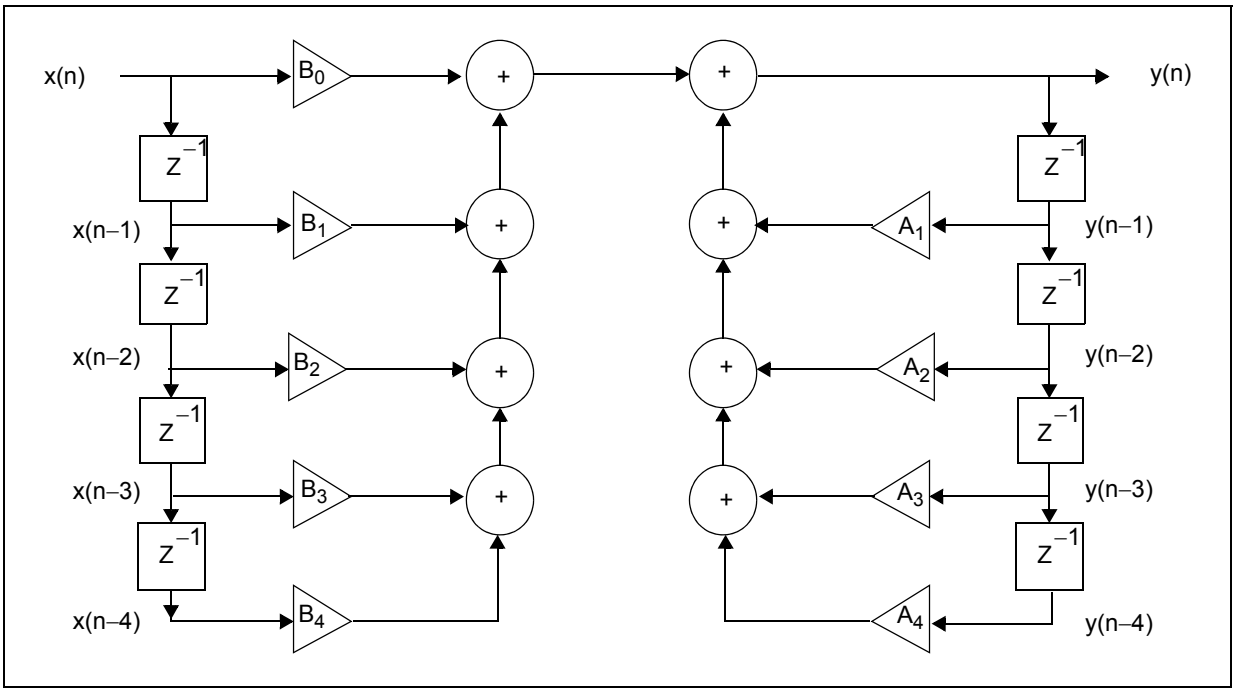
Bypass operation is configured by setting the field FTYPE[1:0] of module configuration register DECFILTER_MCR to '00'. In this case, the input sample and tag are sent to the output with no change. This behavior is independent of the ISEL/MIXM setting. The following applies to the bypass configuration:

- Flush is ignored
- Prefill is ignored
- Trigger or counted decimation is ignored
- DECFILTER_MSR[BSY] bit is not set
- Input and output flags are set

24.5.6 IIR and FIR filter

This section describes the IIR filters implemented in the decimation filter block. *Figure 500* shows the filters functional diagrams. The figure shows an IIR filter of fourth order (4 poles). The filter topology, not the hardware, is represented. The hardware implementation is based on a MAC unit controlled by an FSM which implements the filter algorithm.

Figure 500. 1 × 4 poles IIR filter functional diagram



The difference equation for the IIR filter of *Figure 500* can be written as:

Equation 8

$$y(n) = \sum_{i=0}^{N} B_i x(n-i) + \sum_{j=1}^{M} A_j y(n-j)$$

where *x(n)* is the filter input at time *n, y(n)* is the filter output at time *n*, *N* is the number of feed-forward filter coefficients minus one, B_i are the feed-forward filter coefficients, *M* is the number of feed-back filter coefficients, and A_j are the feedback filter coefficients.

Equation 8 can be written as:

Equation 9

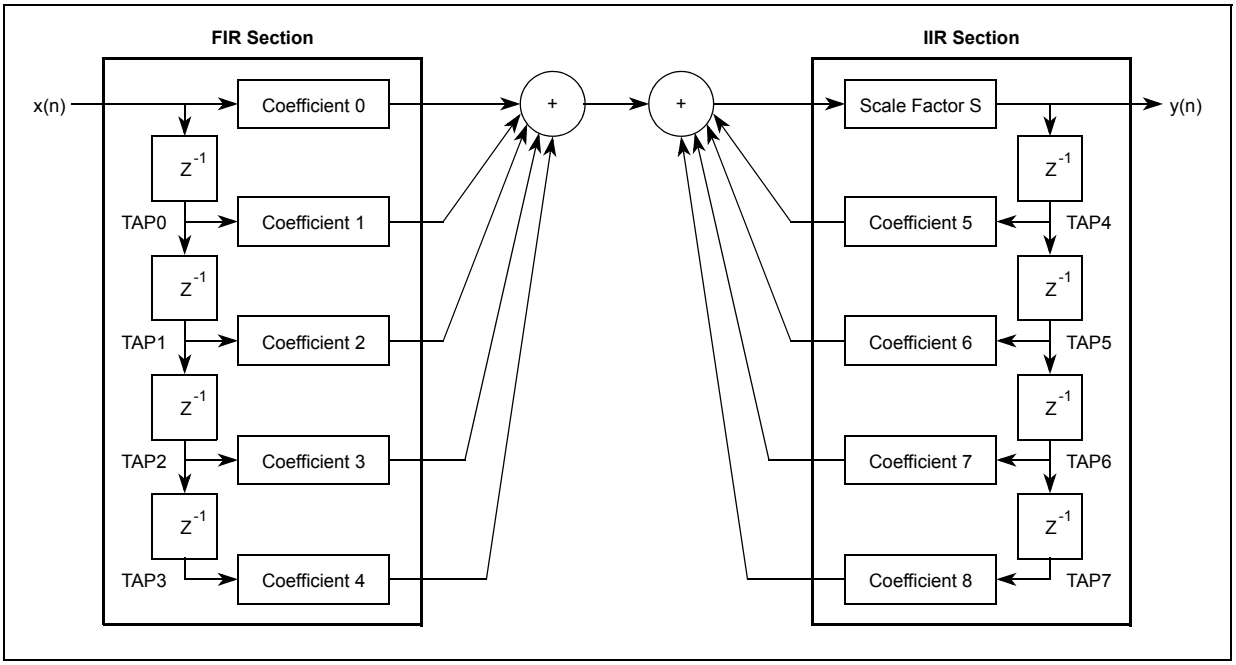
$$w(n) = \sum_{i=0}^{N} \frac{B_i}{S} x(n-i)$$

Equation 10

$$y(n) = S\left\{ w(n) + \sum_{j=1}^{M} \frac{A_j}{S} y(n-j) \right\}$$

Where all the coefficients are scaled down by S. The block diagram for *Equation 9* and *Equation 10* is shown in *Figure 501* in a fourth-order IIR filter implementation where the coefficients A_j and B_i are called *coefficient n*, where *n* = 0–8.

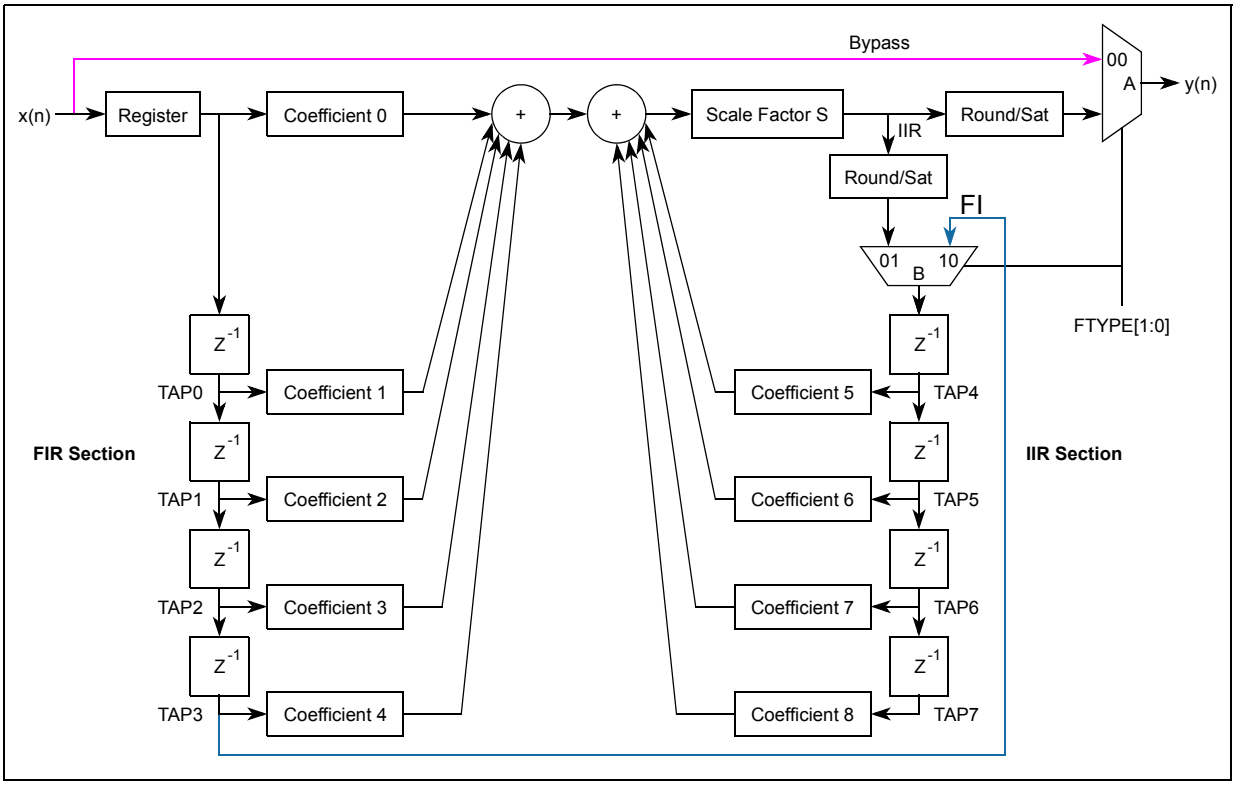
Figure 501. Fourth order IIR filter implementation block diagram



The fourth order IIR filter is implemented with a FIR section followed by an IIR section. If the FIR type filter mode is selected, the IIR section is converted into an FIR section. In this case the order of the FIR filter is twice the IIR filter order, since all the TAP and coefficient registers are allocated for the FIR section. The Filter configuration paths are shown in *Figure 502*. Multiplexer A controls the *bypass* filter path and multiplexer B controls/selects the filter mode of operation, to either IIR mode or FIR mode. The selection is controlled by

the FTYPE[1:0] bits in the Filter Module Configuration register. The order of the filter can be controlled by setting the appropriate *filter coefficients* to *zero.*

Figure 502. Filter configuration paths (FIR or 1 × 4 poles IIR)

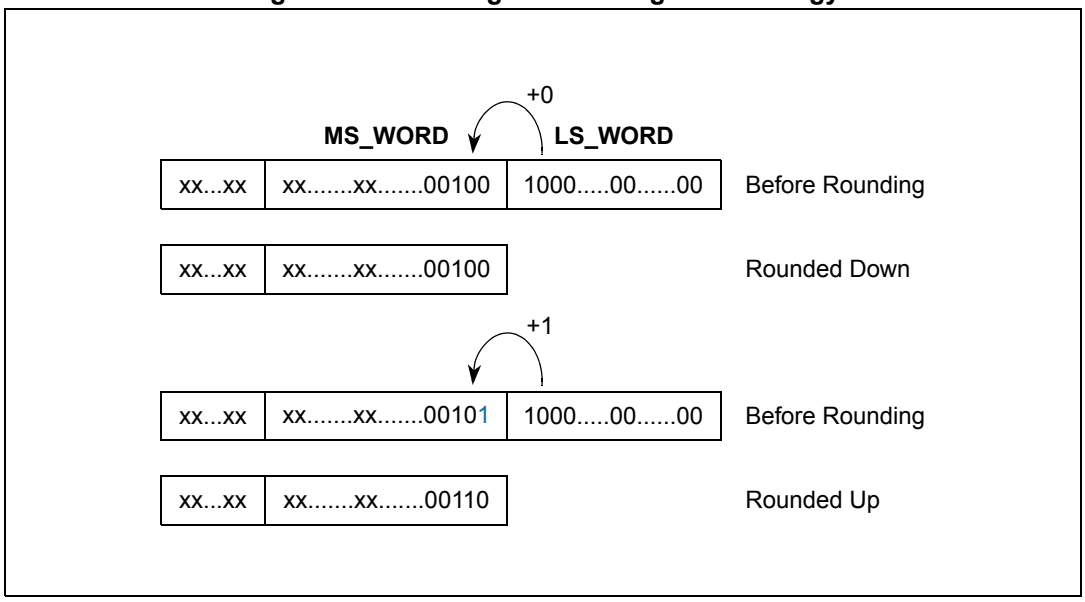


24.5.6.1 Rounding

The decimation filter implements the Convergent Rounding methodology (also known as round-to-nearest even number) in two different locations, as shown in *Figure 502*. In this method the decision on rounding up or down is based on the value of the lower portion of data to be rounded (LS_WORD). The rounding up/down condition is equal to the traditional rounding except when the LS_WORD has the format {1000...00}. In this particular case, the rounding procedure is like the example of *Figure 503*. If the MS_WORD is odd, the value is rounded up. Otherwise the value is rounded down.

Rounding is applied to obtain the filter output result with 16 bits, and also to obtain the IIR feedback result to be stored in TAP4 registers with 24 bits.

Figure 503. Convergent rounding methodology



24.5.6.2 Saturation

Filter output saturation occurs when an overflow or underflow condition of the filter is detected by dedicated logic, and if it is enabled by the SAT control bit of the configuration register DECFILTER_MCR. In this condition, the filter output is set to a saturated value equal to the maximum or minimum value that can be represented by the 16-bit output port. Also, for the IIR filter an equivalent logic is used to assert the saturation for the feedback result, which has 24 bits.

24.5.7 Filter prefill control description

A prefill indicates that the input data should be accepted by the decimation filter, but no decimated output should be generated while the control field indicates prefill. Therefore the prefill function is used in the beginning of the filter operation to initialize and stabilize the decimation filter without generating decimated samples. In addition, the prefill does not operate when the filter is in bypass (FTYPE = 0b00).

The prefill is controlled by the value in the M_CTRL[1:0] field in the DECFILTER_IOB register. When ISEL = 0, the M_CTRL[1:0] field in the DECFILTER_IOB register controls the prefill. When ISEL = 1, it is controlled by the PREFILL field in the DECFILTER_IB register. The prefill control is usually activated only in a certain number of words of sample data in the beginning of the input data sequence.

When the prefill control is set, the decimation filter block operates as follows:

- Input data is processed normally by the digital filter and TAP values are updated.
- The decimation counter is maintained in reset value.
- The output buffer is not updated and no output interrupt or read request is generated.
- The accompanying timestamp for the identified prefill conversion data is not bypassed.
- The overflow detector/flag operates normally and the error interrupt request is set if enabled.

24.5.8 Timestamp data transmission

The timestamp information is identified by the master block using the M_CTRL[1:0] bit field of the register DECFILTER_IOB. For timestamp data, the input data and tag values that come with M_CTRL[1:0] bits set to 0b10 are sent back to the master block without changing. However, some additional conditions are considered:

- The timestamp is additional information that accompanies a sample conversion data. The eQADC block sends the decimation filter the conversion data with control bits for either prefill or filter operation. This data may optionally be followed by the corresponding timestamp data. When the corresponding conversion data is marked for prefill, the timestamp data is not sent to the output buffer. This occurs because the filter result is not sent to the output buffer.
- Similarly, when the filter is decimating the results, the timestamp is only sent to the output buffer if the corresponding received conversion data has generated a filter output that is selected by the decimation counter to be sent to the output buffer. Other received timestamps that come with data not selected by the decimator are discarded.
- Sending two consecutive words of timestamp data is not allowed. In normal operation, the filter should receive only one timestamp for each word of conversion data to be filtered.

24.5.9 Flush Command description

The flush signal is used by the decimation filter to execute a partial reset of the filter. This is useful when the same filter is used on a new set of data samples after finishing the filtering of another set of data.

When the flush control is detected, all filter TAPs are cleared and the DEC_COUNTER[3:0] field in the DECFILTER_MSR is reset.

The flush function does not clear the coefficient registers file in the decimation filter, thus it is not required to rewrite these registers after a flush. The integrator accumulator and sample count are not cleared either. The output buffer also keeps the last result and may be retrieved until the next output is posted.

The flush control precedes the input data to be filtered. Therefore, the corresponding sample data is processed by the block after the flush. When ISEL = 0, the field M_FLUSH in the DECFILTER_IOB register is processed. When ISEL = 1, the field FLUSH in the DECFILTER_IB register is processed. Note that a word of valid sample data can be available at the same time the flush signal is asserted. In this case the flush is executed and the sample is processed after the flush.

When ISEL = 0, flush bit M_FLUSH must not be asserted when the input data is a timestamp (M_CTRL = 10).

When the filter is disabled by the FTYPE[1:0] control bit field, the flush command is not executed.

24.5.10 Soft-Reset command description

The Soft-Reset command is requested through the self-negated bit DECFILTER_MCR[SRES] and provides the CPU with the capability to initialize the decimation filter through the slave-bus interface. After the Soft-Reset command is issued, all internal Filter TAP registers, the decimation counter, and the state machine are put in the initial state. The DECFILTER_MSR is also cleared. The coefficient registers are not affected

by the SRES. In case there is some filter processing, the filter process is aborted and the last sample is discarded. In addition, data in the input buffer waiting to be processed, and data in the output buffer waiting to be read, are discarded (the requests of service are cleared). The Soft-Reset command has high priority and the DECFILTER_MSR[BSY] bit is set during its operation.

The DECFILTER_MCR is also not affected by a soft reset, except the bit SRES that is self-negated and is always read as zero.

When in debug or freeze mode, the soft reset is executed but the filter remains in debug or freeze mode.

Note: It is recommended to clear the IBIE bit before a soft reset, especially if ISEL changes, in order to avoid unwanted interrupt requests.

24.5.11 Interrupts requests description

24.5.11.1 Block interrupt request

There are several interrupt request events that can be enabled using the module configuration register DECFILTER_MCR. Basically, the interrupt request can be issued under any of the following conditions:

- when a word of input data is received
- when a word of output data is available
- when an error has occurred.

The input data flag IDF is set when a word of data is received from the CPU when in standalone mode, or when a word of data is received in the PSI when it is not in standalone mode (normal mode). It is not used to generate the read or write requests (as defined in *Section 24.5.11.2: Input buffer interrupt request*) when DSEL = 0.

Output data is available and its flag (ODF) is set when the input data sample is processed by the filter and the decimation counter matches the decimation rate value. It is not used to generate the read requests (as defined in *Section 24.5.11.3: Output buffer interrupt request*) when DSEL = 0.

An error event in the decimation filter block is defined as one of these events:

- Overflow in the filter, flagged by OVF
- Overrun in the decimation filter input, flagged by IVR
- Overrun in the decimation filter output, flagged by OVR
- Overrun in enhanced debug monitor, flagged by DIVR

An overflow occurs when the two's-complement result value from the multiplier/accumulator is out of the range of values that can be stored in TAP register 4 (IIR) or in the output register.

An input overrun occurs when the input buffer is holding a word of input data and one more word of data is received by the filter. See *Section 24.5.3.1: Input buffer overrun* for more details.

An output overrun occurs when a new word of data is sent to the output buffer but the previous word of data has not been handled yet. See *Section 24.5.4.1: Output buffer overrun* for more details.

These flags can be set by the PSI events, however they are only cleared by

- the CPU, or
- by the soft reset command in the DECFILTER_MCR, or
- by the clear flag fields in the DECFILTER_MSR.

24.5.11.2 Input buffer interrupt request

This interrupt is enabled by DECFILTER_MCR[IBIE] and is asserted only when DSEL = 0. This request is flagged by DECFILTER_MSR[IBIF].

In standalone mode, the input buffer interrupt request is asserted when the input buffer is available to receive a conversion sample and DMA operation is not selected (DSEL = 0), meaning the block is requesting data to be written into the input buffer. The interrupt request is cleared when the CPU writes a ‘1’ in field IBIC of the DECFILTER_MSR, or by the soft reset command.

When in normal mode with enhanced debug enabled (ISEL = 0 and EDME = 1, DSEL = 0), the input sample data can be read by the CPU when this interrupt request is asserted. The interrupt is asserted just when a new word of sample data is supplied to the filter subblock to be processed. As this filter register is overwritten by the next word of sample data, an input read overrun event can occur (bit DECFILTER_MSR[IVR] is asserted) if the interrupt request is not cleared before, or at the same time as, the new sample arrives to set the interrupt. This DIVR bit is cleared by the DIVRC bit in the status register DECFILTER_MSR. However, in enhanced debug monitoring, the set condition has higher priority than the clear. This means that if the set condition and the clear bit IBIC occur at the same time, the interrupt remains asserted.

24.5.11.3 Output buffer interrupt request

This interrupt is enabled by field DECFILTER_MCR[OBIE] and is asserted only when DSEL = 0. This request is also indicated in the field DECFILTER_MSR[OBIF].

When in standalone mode, the output buffer can be read by the CPU with the DMA disabled. The output buffer interrupt request is asserted when the output buffer receives a new result from the filter subblock. This means the block is requesting data to be read by the CPU.

This interrupt request is cleared when a ‘1’ is written in the bit OBIC of the DECFILTER_MSR, or by the soft reset command.

24.5.12 DMA requests description

The DMA function for input and output buffers is enabled using DECFILTER_MCR[DSEL] = 1.

24.5.12.1 Input buffer DMA request

This DMA request is enabled by the ISEL and DSEL bits in the DECFILTER_MCR.

When in standalone mode, the input buffer can be written by the DMA. The input buffer DMA request is asserted when the input buffer is available to receive a conversion sample (it is not holding a word of data). This DMA request is cleared when an input data word is written in the input buffer. Therefore, the DMA request is always cleared before it is asserted again.

When in normal mode with enhanced debug enabled (ISEL = 0 and EDME = 1, DSEL = 1), the input sample data can be read by the DMA when this DMA request is asserted. The

request is asserted when a new word of sample data is written into the input buffer to be processed. As this filter register is overwritten by the next word of sample data, a DMA read overrun event can occur (the DIVR bit in DECFILTER_MSR is asserted) if the DMA request is not cleared before, or at the same time as, a new sample arrives to set the DMA request. This DIVR bit is cleared by the DIVRC bit in the status register DECFILTER_MSR or by soft reset.

This DMA request is cleared when the DMA transfer is complete or by soft reset.

24.5.12.2 Output buffer DMA request

This DMA request is enabled by the ISEL, MIXM and DSEL bits in the DECFILTER_MCR.

When in standalone mode, the output buffer can be read using the DMA. The output buffer DMA request is asserted when the output buffer receives a new result from the internal filter subblock. This DMA request is cleared when the output buffer is read by the processor.

The output buffer DMA request is also asserted when an integrator output is ready, if DECFILTER_MXCR[SDMAE] = 1 and the filter output is directed to the PSI (ISEL = MIXM).

The DMA request is also cleared by soft reset.

24.5.13 Freeze Mode description

The freeze mode operation is asserted using the FREN enable bit and FRZ bit in the DEC_FILTER_MCR or by the modules debug input.

It is not possible to enter freeze mode when the module is disabled by the configuration bit MDIS.

In case of a freeze mode request during the processing of an input sample, the current processing is finished and then the module enters freeze mode.

24.5.14 Enhanced Debug Monitor description

This feature is enabled by field DECFILTER_MCR[EDME]. The monitoring operation works in normal mode, when the sample data is supplied by a master block through the PSI slave-bus interface (ISEL = 0). The Enhanced Debug Monitor feature makes the input sample data also available in the DECFILTER_EDID register. A DMA or interrupt request (selected by DSEL) indicates a new input was fed and DECFILTER_EDID was updated. The input is processed normally by the filter.

An Enhanced Debug Input Data Register (DECFILTER_EDID) overrun can occur if a sample is not read by the CPU or DMA before overwritten by a new sample. The overrun is indicated in a separate flag DIVR in the status register DECFILTER_MSR. If DECFILTER_MCR[ERREN] is set in the, this overrun asserts the module interrupt request.

24.6 Initialization information

Following are some simple initialization steps to be done before using the decimation filter block. These steps assume that the user has already calculated the filter coefficients using a filter design tool.

24.6.1 Initialization procedure

The sequence of steps for the block initialization is as follows:

1. Program the configuration registers DECFILTER_MCR and DECFILTER_MXCR as desired for your application.
2. Write all filter coefficient registers DECFILTER_COEFn with the previously calculated values.
3. Run a soft-reset cycle if necessary.
4. The module is ready to receive data from PSI or from the device slave-bus interface.

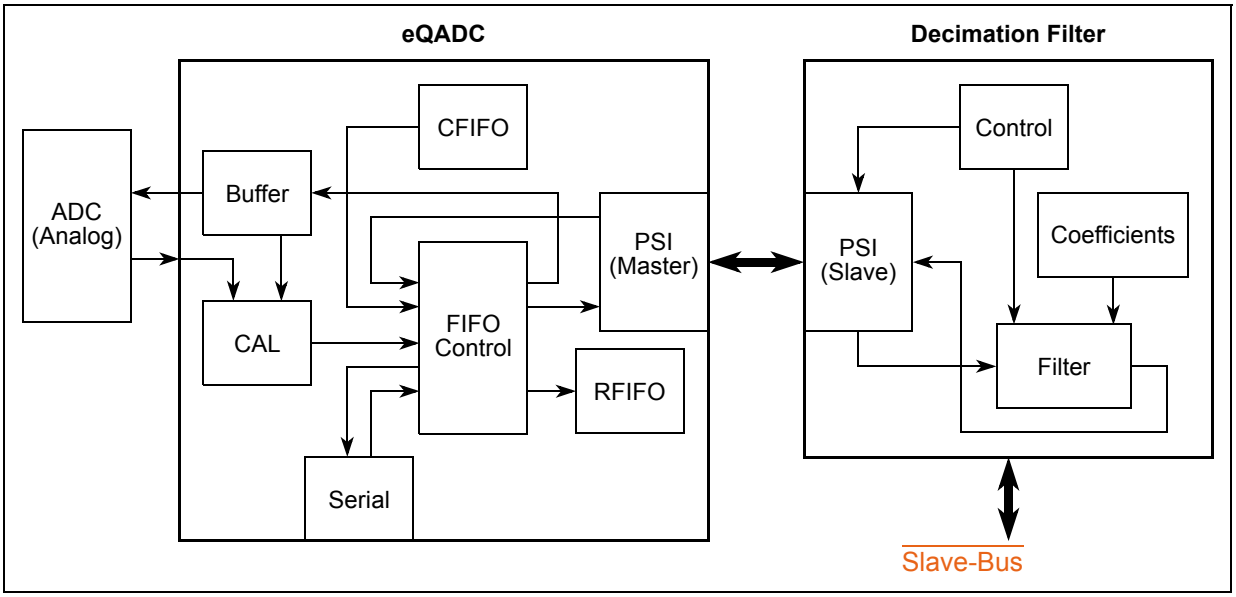
24.7 Application information

24.7.1 eQADC IP as the master block

The system block diagram for the eQADC application is shown in *Figure 504*. In this case, the decimation filter receives conversion results generated by the eQADC block. These results can be generated from eight different ADC setup configurations which are identified by a specific eQADC Control address within a Conversion command. Conversion commands with Register Address set to zero use the standard configuration setup. The samples generated by the standard configuration setup are sent to one of the local eQADC RFIFO buffers. The samples generated by the alternate configurations, with an address from 1–8, can be sent to the internal RFIFO or to the eQADC dedicated slave-bus interface (PSI—Parallel Side Interface) to communicate with the external decimation filter IP block or any other block that can communicate with this interface. A bit field in the corresponding Alternate Configuration n Control Register (ADC_ACRn [n = 1..8]) selects the Internal RFIFO or this slave-bus interface as the destination for the conversion result. The eQADC can send either conversion data or timestamp data. The conversion data is filtered by the decimation filter and the timestamp is bypassed and sent back to the eQADC.

In the eQADC application, the TAG field is used to address the appropriate RFIFO in the eQADC block. In this case, only addresses 0–5 are used since there are only six RFIFOS available in the eQADC block.

Figure 504. Decimation filter/eQADC interface



24.8 Filter example simulation

The decimation filter block operation was checked in a Verilog simulation using calculated filter coefficient values and noisy input data. The expected output values and the RMS error were then calculated.

24.8.1 Coefficients calculation

The coefficients were calculated using a digital filter design tool. We have supplied some hypothetical filter parameters to the tool and obtained the filter coefficients. The input parameters are:

- Filter characteristics: elliptic/low pass
- Filter type: 4th order IIR
- Input sample rate: 800k sample/s
- Passband edge: 100 kHz
- Stopband edge: 150 kHz
- Passband attenuation: ≤ 1 dB

The software tool gives the IIR filter coefficients in the Z-transform format expressed by *Equation 11*:

Equation 11

$$\frac{Y(s)}{X(s)} = \frac{B_0 + B_1 s + B_2 s^2 + B_3 s^3 + B_4 s^4}{A_0 - A_1 s - A_2 s^2 - A_3 s^3 - A_4 s^4}$$

The coefficient results in fixed point decimal representation are shown in *Table 327*.

Table 327. Coefficient values given by SPW digital filter design tool

Coefficient	Decimal value	Coefficient	Decimal value
B0	0.0221455	A0	1.0

Table 327. Coefficient values given by SPW digital filter design tool (continued)

Coefficient	Decimal value	Coefficient	Decimal value
B1	0.00445582948893748	A1	–2.69772868375858
B2	0.0318517846509088	A2	3.234056294853
B3	0.00445582948893748	A3	–1.92028561712454
B4	0.0221455	A4	0.47939080709495

Comparing *Equation 10* with *Equation 11*, we obtain the relationship between the calculated values of coefficients and the values to be loaded in the DECFILTER_COEFn registers. See *Table 328* to obtain the coefficients. A scale factor of eight is used, being the smallest divider factor to have all coefficient values in the range (–1 ≤ Coef < +1). Also note that the signal of the An coefficients has signals inverted.

Table 328. Coefficient values for decimation filter

SCALE	S = 1	S = 8	
COEFn	Decimal value	Decimal value	Hexadecimal values (24 bits)
Coef0 = B0/S	0.0221455	0.00276815891266	0x005AB5
Coef1 = B1/S	0.00445582948893748	0.00055694580078	0x001240
Coef2 = B2/S	0.0318517846509088	0.00398147106171	0x008277
Coef3 = B3/S	0.00445582948893748	0.00055694580078	0x001240
Coef4 = B4/S	0.0221455	0.00276815891266	0x005AB5
Coef5 = –A1/S	2.69772868375858	0.33721613883972	0x2B29E6
Coef6 = –A2/S	–3.234056294853	–0.40425717830658	0xCC414E
Coef7 = –A3/S	1.92028561712454	0.24003565311432	0x1EB97D
Coef8 = –A4/S	–0.47939080709495	–0.05992400646210	0xF8546A

24.8.2 Input data calculation

The 24-bit words of input data are samples of a sum of two tones: one tone of 30 kHz and another tone of 250 kHz. The samples were calculated at the rate of 800k samples per second. The tones have the same amplitude and it is assured that the resultant amplitude is smaller than 1 so as to obtain samples in the range (–1 ≤ sample < +1). It is supposed the input data are signed values in the two's complement format in the range (–1 ≤ sample < +1).

24.8.3 Filter results

The decimation filter block was used in a Verilog simulator using the calculated coefficients and the input data samples. A scaling factor of eight in the configuration register DECFILTER_MCR, and no decimation factor, was used to obtain the maximum of output results from the filter.

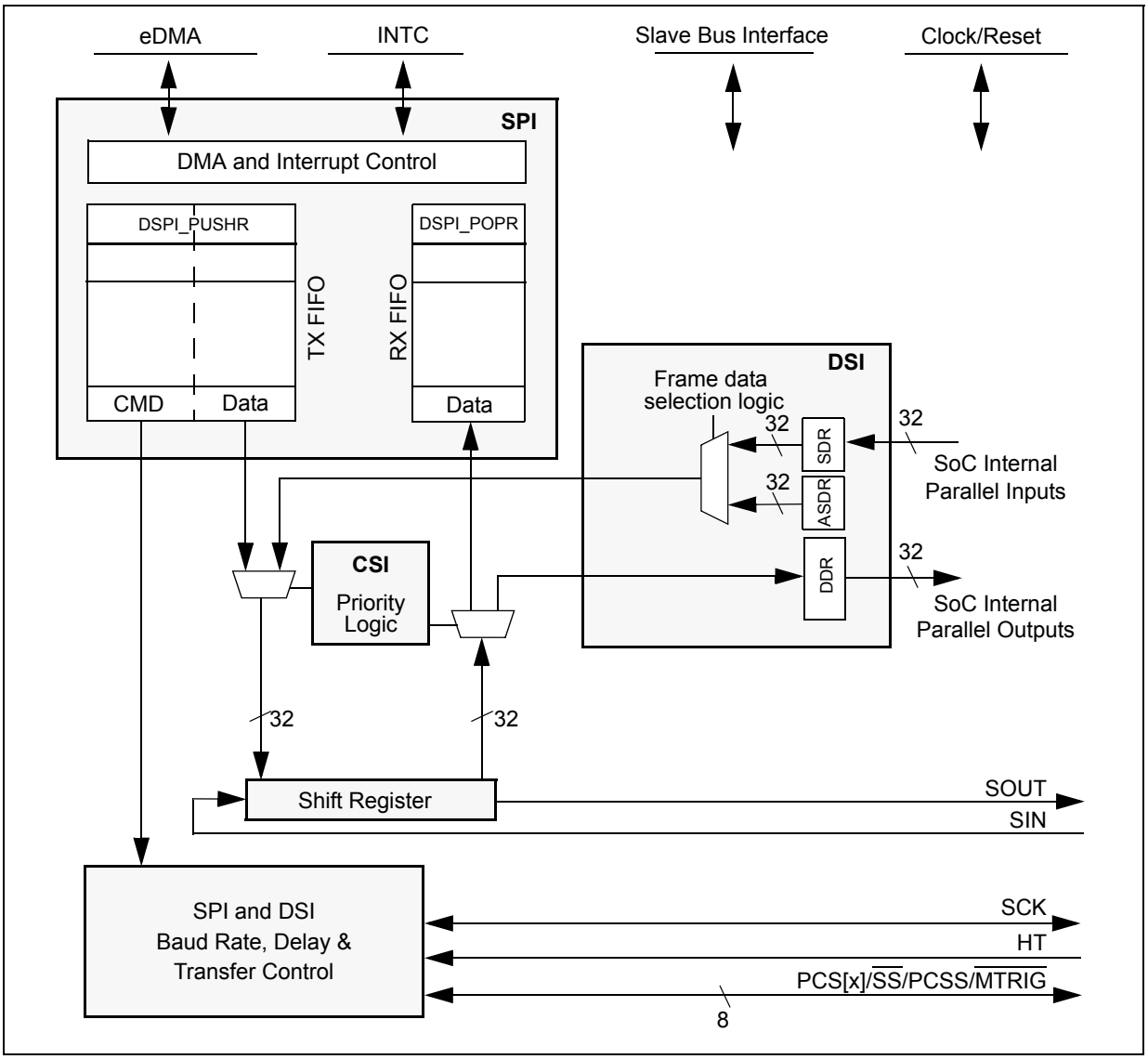
The theoretical expected values from this filter were also calculated, and these results were compared with those from the decimation filter. The resultant RMS error when considering about 500 samples was about –97 dB.

25 Deserial Serial Peripheral Interface (DSPI)

25.1 Introduction

Figure 505 is a block diagram of the Deserial Serial Peripheral Interface (DSPI) block.

Figure 505. DSPI block diagram



25.1.1 Overview

The Deserial Serial Peripheral Interface (DSPI) module provides a synchronous serial bus for communication between an SoC and an external peripheral device. The DSPI supports SoC pin count reduction through serialization and deserialization of SoC internal signals transmitted over the SPI serial link, if this feature is enabled in the block.

25.1.2 Features

The DSPI supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave modes
  - Data streaming operation in the slave mode with continuous slave selection
- Buffered transmit operation using the TX FIFO with parameterized depth of 1 to 16 entries
- Buffered receive operation using the RX FIFO with parameterized depth of 1 to 16 entries
- TX and RX FIFOs can be disabled individually for low-latency updates to SPI queues
- Visibility into TX and RX FIFOs for ease of debugging
- Programmable transfer attributes on a per-frame basis:
  - Parameterized number of transfer attribute registers (from two to eight)
  - Serial clock with programmable polarity and phase
  - Various programmable delays
  - Programmable serial frame size of 4 to 32 bits, expandable by software control
  - Continuously held chip select capability
  - Parity control
- 8 Peripheral Chip Selects, expandable to 256 with external demultiplexer
- Deglitching support for up to 128 Peripheral Chip Select with external demultiplexer
- DMA support for adding entries to TX FIFO and removing entries from RX FIFO:
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- 7 Interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - Attempt to transmit with an empty Transmit FIFO (TFUF)
  - RX FIFO is not empty (RFDF)
  - Frame received while Receive FIFO is full (RFOF)
  - Parity Error (SPEF)
- Global interrupt request line
- Modified SPI transfer formats for communication with slower peripheral devices
- Power-saving architectural features
  - Support for stop mode
  - Support for doze mode

The DSPI also supports pin reduction through serialization and deserialization if enabled for the module.

- 2 sources of serialized data:
  - DSPI memory-mapped register
  - Parallel Input signals
  - Programmable selection of source data on bit basis
- Deserialized data is provided as Parallel Output signals and as bits in a memory-mapped register
- 2 Interrupt conditions:
  - Deserialized data matches pre-programmed pattern (DDIF)
  - Parity Error (DPEF)
- Transfer initiation conditions:
  - Continuous
  - Edge sensitive hardware trigger
  - Change in data
- Support for parallel and serial chaining of several DSPI modules inside the SoC
- Pin serialization/deserialization with interleaved SPI frames for control and diagnostics
- Continuous serial communications clock

DSPI supports a combined SPI and DSI mode of operation (that is, Combined Serial Interface or CSI) for the downstream Micro Second Channel in either Timed Serial Bus (TSB) configuration or Interleaved Frames Configuration (configurable by software). Both TSB and Interleaved TSB modes have the following common features:

- Transmission of frames is done at frame boundaries
- Frames from SPI & DSI are identifiable by a bit transmitted at the start of each frame
- Separate interrupts for frame completion from SPI and DSI

25.1.3 DSPI configurations

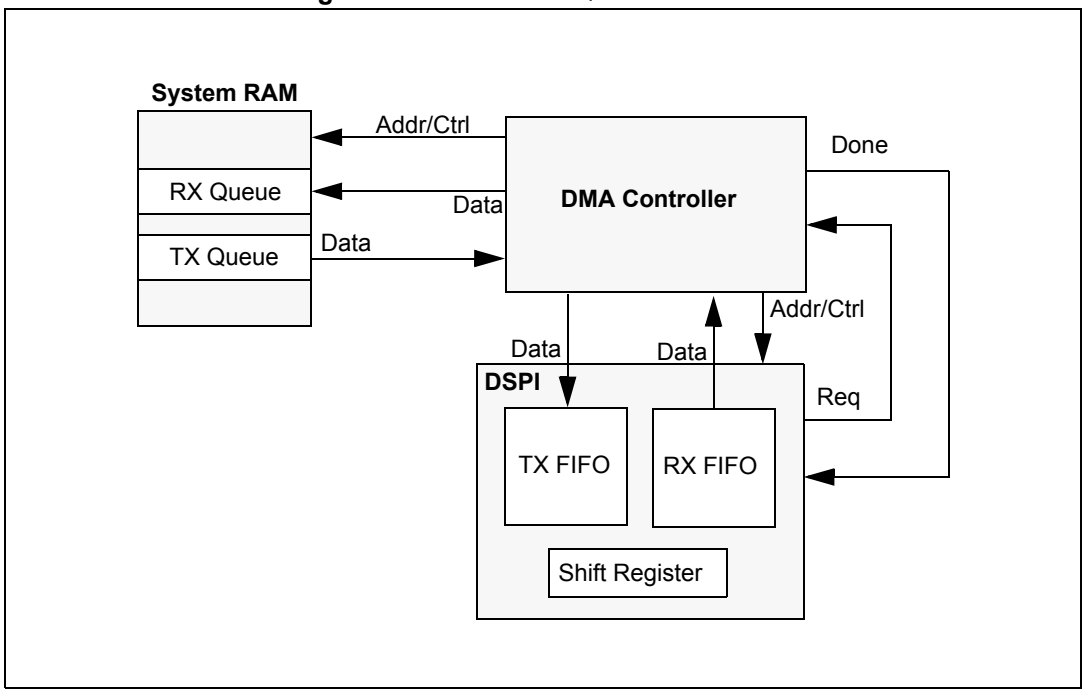
The DSPI module can operate in three configurations: SPI, DSI and CSI.

25.1.3.1 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 506* shows a system example with DMA, DSPI and external queues in system RAM.

Figure 506. DSPI with Queues and DMA



25.1.3.2 DSI configuration

In the DSI configuration the DSPI serializes up to 32 Parallel Input signals or register bits. The DSPI also deserializes the received data to Parallel Output signals or to a memory-mapped register. The data is transferred using an SPI-like protocol.

TSB mode, detailed in *Section 25.4.9: Timed Serial Bus (TSB)*, provides the Micro Second downstream Channel support (MSC), serializing from 4 to 32 Parallel Input signals or register bits.

25.1.3.3 CSI configuration

The CSI configuration is a combination of the SPI and DSI configurations. In this configuration the DSPI interleaves DSI data frames with SPI data frames. Interleaving is done on the frame boundaries.

There are two configurations available to the user in this mode. In the TSB configuration, transmission of SPI data has higher priority than DSI data. The other configuration available is the Interleaved TSB (ITSB) mode where frames from SPI & DSI are interleaved without priority and on every trigger, frames from DSI are sent when there are no frames in the SPI or the previous transmission was a frame from SPI. ITSB is detailed in *Section 25.4.10: Interleaved TSB (ITSB) Mode*.

25.1.4 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.

25.1.4.1 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.

25.1.4.2 Slave Mode

The slave mode allows the DSPI to communicate with SPI/DSI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The SCK signal and the PCS[0]/$\overline{SS}$ signal are configured as inputs and driven by a SPI bus master.

25.1.4.3 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.

25.1.4.4 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.

25.1.4.5 Debug Mode

The debug mode is used for system development and debugging. The DSPI_MCR[FRZ] bit controls DSPI behavior in the debug mode. If the bit is set, the DSPI stops all serial transfers, when the SoC in the debug mode. If the bit is cleared the SoC debug mode has no effect on the DSPI.

25.2 External signal description

25.2.1 Overview

Table 329 lists the signals that may connect off chip depending on SoC implementation.

Table 329. 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]/$\overline{MTRIG}$	Output	Peripheral Chip Select 4	Master Trigger
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)
HT	Input	Hardware Trigger	Hardware Trigger

25.2.2 Detailed signal description

25.2.2.1 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.

25.2.2.2 PCS[1]–PCS[3] — Peripheral Chip Selects 1–3

PCS[1]–PCS[3] are Peripheral Chip Select output signals in master mode. In slave mode these signals are unused.

25.2.2.3 PCS[4]/$\overline{MTRIG}$ — Peripheral Chip Select 4/Master Trigger

In master mode, PCS[4] is a Peripheral Chip Select output signal.

In slave mode, the active low $\overline{MTRIG}$ is an output trigger signal that indicates that a change in data to be serialized has occurred. The $\overline{MTRIG}$ provides a pulse in DSI Configuration when a change in data to be serialized occurs. The $\overline{MTRIG}$ pulse is four system clock cycles in duration. If the DSPI is in slave mode and the MTO is disabled, the PCS[4]/$\overline{MTRIG}$ signal is unused.

25.2.2.4 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.

25.2.2.5 PCS[6]–PCS[7] — Peripheral Chip Selects 6–7

PCS[6]–PCS[7] are Peripheral Chip Select output signals in master mode. In slave mode these signals are not used.

25.2.2.6 SIN — Serial Input

SIN is a serial data input signal.

25.2.2.7 SOUT — Serial Output

SOUT is a serial data output signal.

25.2.2.8 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.

25.2.2.9 HT — Hardware Trigger

HT is a trigger input signal that is used with Multiple Transfer Operations in DSI Configuration.

In master mode while in DSI or CSI Configurations, the HT signal initiates a data transfer when the TRRE bit in the DSPI_DSICR is set and a rising or falling edge is detected on HT. Which edge to trigger on is determined by the TPOL bit in the DSPI_DSICR.

In slave mode, the DSPI generates a trigger pulse on the $\overline{\text{MTRIG}}$ pin, when a rising or falling edge is detected on HT. Which edge that generates an output pulse is selected by the TPOL bit in the DSPI_DSICR.

25.3 Memory map and register definition

25.3.1 Memory map

Register accesses to memory addresses that are reserved or undefined result in a transfer error. Write access to the DSPI_POPR also result in a transfer error.

Table 330 shows the DSPI memory map.

Table 330. DSPI memory map

Offset from DSPI_BASE	Register name	Location
0x0	DSPI Module Configuration Register (DSPI_MCR)	*on page 871*
0x8	DSPI Transfer Count Register (DSPI_TCR)	*on page 873*
0xC – 0x28	DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR0–DSPI_CTAR7)	*on page 874*
0x2C	DSPI Status Register (DSPI_SR)	*on page 880*
0x30	DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)	*on page 882*
FIFO Registers		

Table 330. DSPI memory map (continued)

Offset from DSPI_BASE	Register name	Location
0x34	DSPI PUSH TX FIFO Register (DSPI_PUSHR)	on page 884
0x38	DSPI POP RX FIFO Register (DSPI_POPR)	on page 887
0x3C – 0x78	DSPI Transmit FIFO Registers 0–15 (DSPI_TXFR0–DSPI_TXFR15)	on page 887
0x7C – 0xB8	DSPI Receive FIFO Registers 0–15 (DSPI_RXFR0–DSPI_RXFR15)	on page 888
DSI Registers(1)		
0xBC	DSPI DSI Configuration Register (DSPI_DSICR)	on page 889
0xC0	DSPI DSI Serialization Data Register (DSPI_SDR)	on page 891
0xC4	DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)	on page 892
0xC8	DSPI DSI Transmit Comparison Register (DSPI_COMPR)	on page 892
0xCC	DSPI DSI Deserialization Data Register (DSPI_DDR)	on page 893
0xD0	DSPI DSI Configuration Register 1 (DSPI_DSICR1)	on page 893

1. DSI registers and DSI features may be not available in particular DSPI instances of the SoC, if I/O serialization/deserialization is not implemented.

25.3.2 Register descriptions

25.3.2.1 DSPI Module Configuration Register (DSPI_MCR)

The DSPI_MCR contains bits, which configure various attributes associated with DSPI operation. The HALT and MDIS bits can be changed at any time, but only take effect on the next frame boundary. Only the HALT and MDIS bits in the DSPI_MCR are allowed to be changed, while the DSPI is in the Running state.

Address: DSPI_BASE

	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	DOZE	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 507. DSPI Module Configuration Register (DSPI_MCR)

Table 331. DSPI_MCR field descriptions

Field	Description
0 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
1 CONT_SCKE	Continuous SCK Enable The CONT_SCKE bit enables the Serial Communication Clock (SCK) to run continuously. See *Section 25.4.7: Continuous Serial Communications Clock* for details. 0: Continuous SCK disabled 1: Continuous SCK enabled
2–3 DCONF[0:1]	DSPI Configuration The DCONF field selects between the three different configurations of the DSPI: 00: SPI 01: DSI 10: CSI 11: Reserved
4 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
5 MTFE	Modified Timing Format Enable The MTFE bit enables a modified transfer format to be used. See *Section 25.4.6.4: Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 1)* for more information. 0: Modified SPI transfer format disabled 1: Modified SPI transfer format enabled
6 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 25.4.5.5: 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
7 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 25.4.13.6: Receive FIFO Overflow Interrupt Request* for more information. 0: Incoming data is ignored 1: Incoming data is shifted in to the shift register
8–15 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
16 DOZE	Doze Enable The DOZE bit provides support for externally controlled Doze mode power-saving mechanism. See *Section 25.4.14: Power saving features* for details. 0: SoC Doze mode has no effect on DSPI. 1: SoC Doze mode disables DSPI.

Table 331. DSPI_MCR field descriptions (continued)

Field	Description
17 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 25.4.14: Power saving features* for more information. The reset value of the MDIS bit is parameterized, with a default reset value of '0'. 0: Enable DSPI clocks. 1: Allow external logic to disable DSPI clocks.
18 DIS_TXF	Disable Transmit FIFO When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See *Section 25.4.2.3: FIFO Disable Operation* for details. 0: TX FIFO is enabled 1: TX FIFO is disabled
19 DIS_RXF	Disable Receive FIFO When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See *Section 25.4.2.3: FIFO Disable Operation* for details. 0: RX FIFO is enabled 1: RX FIFO is disabled
20 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
21 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
22–23 SMPL_PT	Sample Point SMPL_PT field controls when the DSPI master samples SIN in Modified Transfer Format. *Figure 534* 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
24–30	Reserved, should be cleared.
31 HALT	Halt The HALT bit starts and stops DSPI transfers. See *Section 25.4.1: Start and Stop of DSPI Transfers* for details on the operation of this bit. 0: Start transfers 1: Stop transfers

25.3.2.2 DSPI Transfer Count Register (DSPI_TCR)

The DSPI_TCR contains a counter, that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. Do not write the DSPI_TCR, when the DSPI is in the Running state.

Address: DSPI_BASE + 0x8

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

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

Figure 508. DSPI Transfer Count Register (DSPI_TCR)

Table 332. 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.

25.3.2.3 DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR0–DSPI_CTAR7)

The DSPI_CTAR registers are used to define different transfer attributes. The number of CTAR registers is parameterized in the RTL and can be from two to eight registers. Do not write to the DSPI_CTAR registers while the DSPI is in the Running state.

In master mode, the DSPI_CTAR0–DSPI_CTAR7 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 is used. When the DSPI is configured as a SPI bus slave, the DSPI_CTAR0 register is used.

When the DSPI is configured as a DSI master, the DSICTAS field in the DSPI DSI Configuration Register (DSPI_DSICR) selects which DSPI_CTAR is used. When the DSPI is configured as a DSI bus slave, the DSPI_CTAR1 register is used.

In CSI Configuration, the transfer attributes are selected based on whether the current frame is SPI data or DSI data. SPI transfers in CSI Configuration follow the protocol described for SPI Configuration, and DSI transfers in CSI Configuration follow the protocol described for DSI Configuration. CSI Configuration is only valid in conjunction with master mode. See *Section 25.4.4: Combined Serial Interface (CSI) Configuration* for more details.

TSB mode sets some limitations on transfer attributes:

- Clock phase is forced to be CPHA = 1 and the CPHA bit setting has no effect.
- PCS lines are driven at the driving edge of the SCK clock together with SOUT, so PCS assertion and negation delays control is unavailable and PCSSCK, PASC, CSSCK and ASC fields have no effect.
- Delay after transfer can be set from 1 to 64 serial clocks with help of PDT and DT fields.

Address: DSPI_BASE + 0xC–DSPI_BASE + 0x28

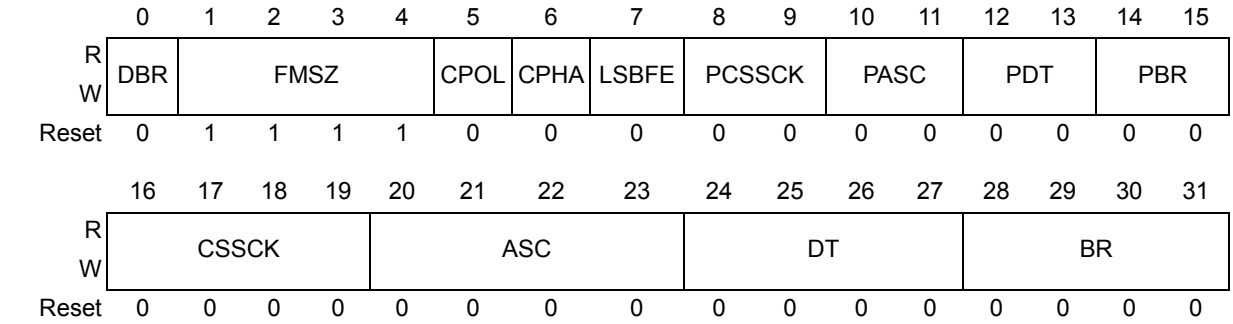


Figure 509. DSPI Clock and Transfer Attributes Register 0–7 (DSPI_CTAR0–DSPI_CTAR7)—master mode

Address: DSPI_BASE + 0xC /0x10

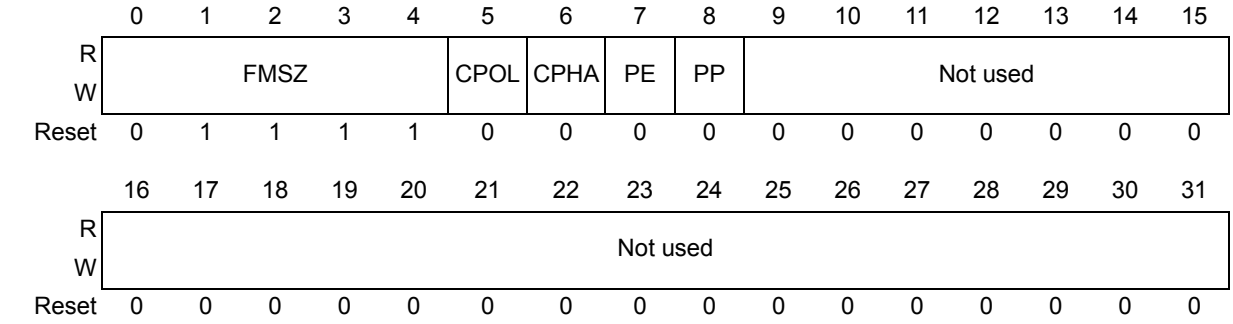


Figure 510. DSPI Clock and Transfer Attributes Register 0, 1 (DSPI_CTAR0, DSPI_CTAR1)—slave mode

Table 333. DSPI_CTAR*n* field descriptions in master mode

Field	Descriptions
0 DBR	Double Baud Rate The DBR bit doubles the effective baud rate of the Serial Communications Clock (SCK). This field is only used in master mode. It effectively halves the Baud Rate division ratio supporting faster frequencies and odd division ratios for the Serial Communications Clock (SCK). When the DBR bit is set, the duty cycle of the Serial Communications Clock (SCK) depends on the value in the Baud Rate Prescaler and the Clock Phase bit as listed in *Table 334*. See the BR field description for details on how to compute the baud rate. If the overall baud rate is divide by two or divide by three of the system clock then neither the Continuous SCK Enable or the Modified Timing Format Enable bits should be set. 0: The baud rate is computed normally with a 50/50 duty cycle 1: The baud rate is doubled with the duty cycle depending on the Baud Rate Prescaler

Table 333. DSPI_CTAR*n* field descriptions in master mode (continued)

Field	Descriptions
1–4 FMSZ[0:3]	Frame Size The number of bits transferred per frame is equal FMSZ field value plus 1. Minimum valid FMSZ field value is 3. When operating in TSB mode, detailed in *Section 25.4.9: Timed Serial Bus (TSB)* the FMSZ field value plus 1 is equal the data frame bit number, where control of the PCS assertion switches from the DSPI_DSICR to the DSPI_DSICR1 register.
5 CPOL	Clock Polarity The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock polarities. When the Continuous Selection Format is selected, switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. 0: The inactive state value of SCK is low 1: The inactive state value of SCK is high
6 CPHA	Clock Phase The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock phase settings. In Continuous SCK mode or TSB mode the bit value is ignored and the transfers are done as CPHA bit is set to 1. 0: Data is captured on the leading edge of SCK and changed on the following edge 1: Data is changed on the leading edge of SCK and captured on the following edge
7 LSBFE	LSB First The LSBFE bit selects if the LSB or MSB of the frame is transferred first. When operating in TSB configuration, this bit should be set to be compliant to MSC specification. 0: Data is transferred MSB first 1: Data is transferred LSB first
8–9 PCSSCK[0:1]	PCS to SCK Delay Prescaler The PCSSCK field selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. See the CSSCK field description how to compute the PCS to SCK Delay. In the TSB mode the PCSSCK field has no effect. 00: PCS to SCK Prescaler value is 1 01: PCS to SCK Prescaler value is 3 10: PCS to SCK Prescaler value is 5 11: PCS to SCK Prescaler value is 7
10–11 PASC[0:1]	After SCK Delay Prescaler The PASC field selects the prescaler value for the delay between the last edge of SCK and the negation of PCS. See the ASC field description how to compute the After SCK Delay. In the TSB mode the PASC field has no effect. 00: After SCK Delay Prescaler value is 1 01: After SCK Delay Prescaler value is 3 10: After SCK Delay Prescaler value is 5 11: After SCK Delay Prescaler value is 7

Table 333. DSPI_CTAR*n* field descriptions in master mode (continued)

Field	Descriptions
12–13 PDT[0:1]	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. In the TSB mode the PDT field defines two MSB bits of the Delay after Transfer. See the DT field description for details on how to compute the Delay after Transfer. 00: Delay after Transfer Prescaler value is 1 01: Delay after Transfer Prescaler value is 3 10: Delay after Transfer Prescaler value is 5 11: Delay after Transfer Prescaler value is 7
14–15 PBR[0:1]	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
16–19 CSSCK[0:3]	PCS to SCK Delay Scaler The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in master mode. The PCS to SCK Delay is the delay between the assertion of PCS and the first edge of the SCK. *Table 335* 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 12** $t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$ See *Section 25.4.5.2: PCS to SCK Delay (t_{CSC})* for more details. In the TSB mode the field has no effect.
20–23 ASC[0:3]	After SCK Delay Scaler The ASC field selects the scaler value for the After SCK Delay. This field is only used in master mode. The After SCK Delay is the delay between the last edge of SCK and the negation of PCS. *Table 335* 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 13** $t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$ See *Section 25.4.5.3: After SCK Delay (t_{ASC})* for more details. In the TSB mode the field has no effect.

Table 333. DSPI_CTAR*n* field descriptions in master mode (continued)

Field	Descriptions
24–27 DT[0:3]	Delay after Transfer Scaler The DT field selects the Delay after Transfer Scaler. This field is only used in master mode. The Delay after Transfer is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. *Table 335* 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 14** $t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$ In the TSB mode the Delay after Transfer is equal to a number formed by concatenation of PDT and DT fields plus 1 of the SCK clock periods. See *Section 25.4.5.4: Delay after Transfer (tDT)* for more details.
28–31 BR[0:3]	Baud Rate Scaler The BR field selects the scaler value for the baud rate. This field is only used in master mode. The prescaled system clock is divided by the Baud Rate Scaler to generate the frequency of the SCK. *Table 336* lists the Baud Rate Scaler values.The baud rate is computed according to the following equation: **Equation 15** $\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$ See *Section 25.4.5.1: Baud Rate Generator* for more details.

Table 334. 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 335. 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 336. 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 337. DSPI_CTAR0, DSPI_CTAR1 field descriptions in slave mode

Field	Descriptions
0–4 FMSZ[0:4]	Frame Size The number of bits transferred per frame is equal FMSZ field value plus 1. Minimum valid FMSZ field value is 3.
5 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
6 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
7 PE	Parity Enable PE bit enables parity bit transmission and reception for the frame 0: No parity bit included/checked. 1: Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.

Table 337. DSPI_CTAR0, DSPI_CTAR1 field descriptions in slave mode (continued)

Field	Descriptions
8 PP	Parity Polarity PP bit controls polarity of the parity bit transmitted and checked 0: Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is odd. 1: Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is even.
29–31 —	Not used, write always zero to keep software compatible with future updates.

25.3.2.4 DSPI Status Register (DSPI_SR)

The DSPI_SR contains status and flag bits. The bits reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI_SR by writing a ‘1’ to it. Writing a ‘0’ to a flag bit has no effect. This register may not be writable in module disable mode due to the use of power saving mechanisms.

Address: DSPI_BASE + 0x2C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	DSITCF	EOQF	TFUF	SPITCF	TFFF	0	0	DPEF	SPEF	DDIF	RFOF	0	RFDF	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c		w1c	
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

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

Figure 511. DSPI Status Register (DSPI_SR)

Table 338. DSPI_SR field descriptions

Field	Description
0 TCF	Transfer Complete Flag The TCF bit indicates that all bits in a frame have been shifted out. The TCF bit remains set until cleared by writing 1 to it. 0: Transfer not complete 1: Transfer complete
1 TXRXS	TX & RX Status The TXRXS bit reflects the run status of the DSPI. See *Section 25.4.1: 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)

Table 338. DSPI_SR field descriptions (continued)

Field	Description
2 DSITCF	DSI Frame Transfer Complete Flag Same as TCF except that this bit is asserted only on DSI frame transmission completion 0: Transfer not complete 1: Transfer complete
3 EOQF	End of Queue Flag The EOQF bit indicates that the last entry in a queue has been transmitted when the DSPI in the master mode. The EOQF bit is set when TX FIFO entry has the EOQ bit set in the command halfword and the end of the transfer is reached. The EOQF bit remains set until cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0: EOQ is not set in the executed command 1: EOQ bit is set in the executed SPI command
4 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
5 SPITCF	SPI Frame Transfer Complete Flag Same as TCF except that this bit is asserted only on SPI frame transmission completion 0: Transfer not complete 1: Transfer complete
6 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
7–8	Reserved, should be cleared.
9 DPEF	DSI Parity Error Flag The DPEF flag indicates that a DSI frame with parity error had been received. The bit remains set until cleared by writing 1 to it. 0: Parity Error has not occurred 1: Parity Error has occurred
10 SPEF	SPI Parity Error Flag The SPEF flag indicates that a SPI frame with parity error had been received. The bit remains set until cleared by writing 1 to it. 0: Parity Error has not occurred 1: Parity Error has occurred
11 DDIF	DSI data received with active bits The DDIF flag indicates that DSI frame had been received with bits, selected by DSPI_DIMR with active polarity, defined by DSPI_DPIR. The bit remains set until cleared by writing 1 to it. 0: No DSI data with active bits was received 1: DSI data with active bits was received

Table 338. DSPI_SR field descriptions (continued)

Field	Description
12 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
13	Reserved, should be cleared.
14 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
15	Reserved.
16–20 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.
20–23 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 25.4.13.4: Transmit FIFO Underflow Interrupt Request* for more details.
24–27 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.
28–31 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 25.4.2.5: Receive First In First Out (RX FIFO) Buffering Mechanism* for more details.

25.3.2.5 DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

The DSPI_RSER controls DMA and interrupt requests. Do not write to the DSPI_RSER while the DSPI is in the Running state.

Address: DSPI_BASE + 0x30

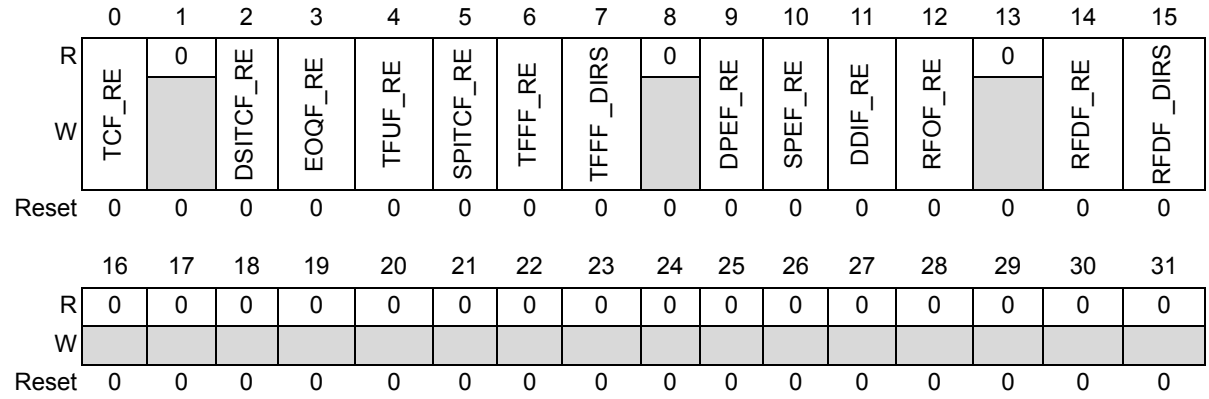


Figure 512. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

Table 339. DSPI_RSER field descriptions

Field	Description
0 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
1	Reserved, should be cleared.
2 DSITCF_RE	DSI Frame Transmission Complete Request Enable The DSITCF_RE bit enables DSITCF flag in the DSPI_SR to generate an interrupt request. 0: DSITCF interrupt requests are disabled 1: DSITCF interrupt requests are enabled
3 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
4 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
5 SPITCF_RE	SPI Frame Transmission Complete Request Enable The SPITCF_RE bit enables SPITCF flag in the DSPI_SR to generate an interrupt request. 0: SPITCF interrupt requests are disabled 1: SPITCF interrupt requests are enabled
6 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

Table 339. DSPI_RSER field descriptions (continued)

Field	Description
7 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 is set, this bit selects between generating an interrupt request or a DMA request. 0: Interrupt request is generated 1: DMA request is generated
8	Reserved, should be cleared.
9 DPEF_RE	DSI Parity Error Request Enable The DPEF_RE bits enables DPEF flag in the DSPI_SR to generate an interrupt requests. 0: DPEF interrupt requests are disabled 1: DPEF interrupt requests are enabled
10 SPEF_RE	SPI Parity Error Request Enable The SPEF_RE bits enables SPEF flag in the DSPI_SR to generate an interrupt requests. 0: SPEF interrupt requests are disabled 1: SPEF interrupt requests are enabled
11 DDIF_RE	DSI data received with active bits Request Enable The DDIF_RE bit enables the DDIF flag in the DSPI_SR to generate an interrupt requests. 0: DDIF interrupt requests are disabled 1: DDIF interrupt requests are enabled
12 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
13	Reserved, should be cleared.
14 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
15 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 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

25.3.2.6 DSPI PUSH TX FIFO Register (DSPI_PUSHR)

The DSPI_PUSHR provides means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. See *Section 25.4.2.4: 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.

Address: DSPI_BASE + 0x34

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CONT	CTAS			EOQ	CTCNT	PE/MASC	PP/MCSC	PCS7	PCS6	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 513. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode

Table 340. DSPI_PUSHR field descriptions in master mode

Field	Descriptions
0 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 25.4.6.5: Continuous Selection Format* for more information. 0: Return Peripheral Chip Select signals to their inactive state between transfers 1: Keep Peripheral Chip Select signals asserted between transfers
1–3 CTAS[0:2]	Clock and Transfer Attributes Select The CTAS field selects number of the DSPI_CTAR 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.
4 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
5 CTCNT	Clear Transfer Counter The CTCNT bit clears the SPI_TCNT field in the DSPI_TCR. The SPI_TCNT field is cleared before transmission of the current SPI frame begins. 0: Do not clear SPI_TCNT field in the DSPI_TCR 1: Clear SPI_TCNT field in the DSPI_TCR
6 PE	Parity Enable PE bit enables parity bit transmission and parity reception check for the SPI frame 0: No parity bit included/checked. 1: Parity bit is transmitted instead of last data bit in frame, parity checked for received frame. **Note:** This bit functions as Parity Enable (PE) when DSPI_MCR[2] = ‘0’.

Table 340. DSPI_PUSHR field descriptions in master mode (continued)

Field	Descriptions
7 PP	Parity Polarity PP bit controls polarity of the parity bit transmitted and checked 0: Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is odd. 1: Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is even. **Note:** This bit functions as Parity Polarity (PP) when DSPI_MCR[2] = ‘0’.
6 MASC	Mask T_{ASC} delay in the current frame The current frame will have the “after SCK” delay masked if this bit is asserted. See *Section 25.4.6.6: Fast Continuous Selection Format* for more details. 0: T_{ASC} delay is not masked and the current frame will have the after SCK delay 1: T_{ASC} delay is masked in the current frame. **Note:** This bit is used as Mask T_{ASC} in the Fast Continuous PCS mode i.e. when DSPI_MCR[2] = ‘1’.
7 MCSC	Mask T_{CSC} delay in the next fame The next frame will have the “PCS to SCK” delay masked if this bit is asserted. See *Section 25.4.6.6: Fast Continuous Selection Format* for more details. 0: T_{CSC} delay is not masked and the next frame will have the PCS to SCK delay 1: T_{CSC} delay is masked in the next frame. **Note:** This bit is used as Mask T_{CSC} in the Fast Continuous PCS mode i.e. when DSPI_MCR[2] = ‘1’.
8–15 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
16–31 TXDATA[0:15]	Transmit Data The TXDATA field holds SPI data to be transferred according to the associated SPI command.

Address: DSPI_BASE + 0x34

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

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

Figure 514. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in slave mode

Table 341. DSPI_PUSHR field descriptions in slave mode

Field	Descriptions
0–31 TXDATA[0:31]	Transmit Data The TXDATA field holds SPI data to be transferred.

25.3.2.7 DSPI POP RX FIFO Register (DSPI_POPR)

The DSPI_POPR provides means to read the RX FIFO. See *Section 25.4.2.5: 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.

Address: DSPI_BASE + 0x38

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

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

Figure 515. DSPI POP RX FIFO Register (DSPI_POPR)

Table 342. 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.

25.3.2.8 DSPI Transmit FIFO Registers 0–15 (DSPI_TXFR0–DSPI_TXFR15)

The DSPI_TXFR0–DSPI_TXFR15 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. The number of registers used to implement the TX FIFO is SoC specific. If a four entry TX FIFO is implemented, DSPI_TXFR0–DSPI_TXFR3 are accessible.

Address: DSPI_BASE+0x3C–DSPI_BASE+0x78

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

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

Figure 516. DSPI Transmit FIFO Register 0–15 (DSPI_TXFR0–DSPI_TXFR15)

Table 343. 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 25.3.2.6: DSPI PUSH TX FIFO Register (DSPI_PUSHR)* for details on the command field. In slave mode the TXDATA contains 16 MSB bits of the SPI data to be shifted out.
16–31 TXDATA[16:31]	Transmit Data The TXDATA field contains the SPI data to be shifted out.

25.3.2.9 DSPI Receive FIFO Registers 0–15 (DSPI_RXFR0–DSPI_RXFR15)

The DSPI_RXFR0–DSPI_RXFR15 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. The number of registers used to implement the RX FIFO is SoC specific. If a four entry RX FIFO is implemented, DSPI_RXFR0–DSPI_RXFR3 exist, for example.

Address: DSPI_BASE + 0x7C–DSPI_BASE + 0xB8

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

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

Figure 517. DSPI Receive FIFO Registers 0–15 (DSPI_RXFR0–DSPI_RXFR15)

Table 344. DSPI_RXFR*n* field descriptions

Field	Description
0–31 RXDATA[0:31]	Receive Data The RXDATA field contains the received SPI data.

25.3.2.10 DSPI DSI Configuration Register (DSPI_DSICR)

The DSI Configuration Register selects various attributes associated with DSI and CSI Configurations. Do not write to the DSPI_DSICR, while the DSPI is in the Running state.

Address: DSPI_BASE + 0xBC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTOE	FMSZ[4]	MTOCNT						0	TRG	ITSB	TSBC	TXSS	TPOL	TRRE	CID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DCONT	DSICTAS			DMS	PES	PE	PP	DPCS7	DPCS6	DPCS5	DPCS4	DPCS3	DPCS2	DPCS1	DPCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 518. DSPI DSI Configuration Register (DSPI_DSICR)

Table 345. DSPI_DSICR field descriptions

Field	Description
0 MTOE	Multiple Transfer Operation Enable The MTOE bit enables multiple DSPIs to be connected in a parallel or serial configuration. See *Section 25.4.3.6: Multiple Transfer Operation (MTO)* for more information. 0: Multiple Transfer Operation disabled 1: Multiple Transfer Operation enabled The MTOE and TSB bits should not be set simulteneously.
1	MSB of the Frame Size in master mode if the bit is set, 16 is added to the frame size, defined by DSPI_CTARn[FMSZ] filed. DSPI_CTARn register is selected by the DSICTAS field.
2–7 MTOCNT[0:5]	Multiple Transfer Operation Count The MTOCNT field selects number of bits to be shifted out during a transfer in Multiple Transfer Operation. The field sets the number of SCK cycles that the bus master will generate to complete the transfer. The number of SCK cycles used will be one more than the value in the MTOCNT field. The number of SCK cycles defined by MTOCNT must be equal to or greater than the frame size. When TSBC is set, MTOCNT field has no effect.
8	Reserved bit
9 TRG	Trigger Source for Interleaved TSB mode This bit decides the source for trigger. 0: Internal Trigger—Trigger is generated inside the IP. DSPI_DSICR1[15:8] provides the trigger period for the ICSI operation. 1: External Trigger—Trigger is generated external to IP at SoC level. Trigger period decided at SoC level. This bit is writeable only when DSPI_MCR[HALT] bit is asserted and is used when DSPI_DSICR[ITSB] bit is asserted. See *Section 25.4.10: Interleaved TSB (ITSB) Mode* for more details.

Table 345. DSPI_DSICR field descriptions (continued)

Field	Description
10 ITSB	Interleaved TSB mode Enables the Interleaved TSB mode of operation. When enabled, there is no priority among frames from the SPI/DSI. DSI frames are always sent when either the previous transmission was a SPI frame or if the TX_FIFO is empty. Frames are transmitted on every trigger whose source is selected by the TRG bit setting. The ITSB mode requires the TSB bit to be set and is writeable only when DSPI_MCR[HALT] bit is asserted. 0: Interleaved mode is disabled 1: Interleaved mode is enabled See *Section 25.4.10: Interleaved TSB (ITSB) Mode* for mode details.
11 TSBC	Timed Serial Bus Configuration The TSBC bit enables the Timed Serial Bus Configuration. This configuration allows 32-bit data to be used. It also allows t_{DT} to be programmable. See *Section 25.4.9: Timed Serial Bus (TSB)* for detailed information. 0: Timed Serial Bus Configuration disabled 1: Timed Serial Bus Configuration enabled If this bit is clear the DSPI_DSICR1 register value has no effect.
12 TXSS	Transmit Data Source Select The TXSS bit selects the source of data to be serialized. The source can be either data from host Software written to the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), or Parallel Input pin states latched into the DSPI DSI Serialization Data Register (DSPI_SDR). 0: Source of serialized data is the DSPI_SDR 1: Source of serialized data is the DSPI_ASDR
13 TPOL	Trigger Polarity The TPOL bit selects the active edge of the hardware trigger input signal (HT). initiating DSI frames transfer. See *Section 25.4.3.5: DSI Transfer Initiation Control* for more information. 0: Falling edge will initiate a transfer 1: Rising edge will initiate a transfer
14 TRRE	Trigger Reception Enable The TRRE bit enables the DSPI to initiate DSI frames transfer with external trigger signall. See *Section 25.4.3.5: DSI Transfer Initiation Control* for more information. 0: Trigger signal reception disabled 1: Trigger signal reception enabled
15 CID	Change In Data Transfer Enable The CID bit enables a change in serialization data to initiate DSI frames transfer. in DSI and CSI configurations. When the CID bit is set, DSI frames are initiated when the current DSI data differs from the previous DSI data shifted out. Refer to *Section 25.4.3.5: DSI Transfer Initiation Control* for more information.
16 DCONT	DSI Continuous Peripheral Chip Select Enable The DCONT bit enables the PCS signals to remain asserted between transfers. The DCONT bit only affects the PCS signals in DSI master mode. See *Section 25.4.6.5: Continuous Selection Format* for details. When TSBC bit is set, DCONT bit has no effect. 0: Return Peripheral Chip Select signals to their inactive state after transfer is complete 1: Keep Peripheral Chip Select signals asserted after transfer is complete

Table 345. DSPI_DSICR field descriptions (continued)

Field	Description
17–19 DSICTAS[0:2]	DSI Clock and Transfer Attributes Select The DSICTAS field selects which of the DSPI_CTAR is used to provide transfer attributes for DSI frames. The DSICTAS field is used in DSI master mode. In DSI slave mode, the DSPI_CTAR1 is always selected.
20 DMS	Data Match Stop DMS bit if set stops DSI frames transmissions if DDIF flag is set in the DSPI_SR. 0: DDIF flag does not have effect on DSI frames transmissions. 1: DDIF flag stops DSI frame transmissions.
21 PES	Parity Error Stop PES bit if set stops DSI operation if the parity error had happened in received DSI frame. 0: parity error does not stop DSI frame transmissions. 1: parity error stops all DSI frame transmissions.
22 PE	Parity Enable PE bit enables parity bit transmission and parity reception check for the DSI frames 0: No parity bit included/checked. 1: Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.
23 PP	Parity Polarity PP bit controls polarity of the parity bit transmitted and checked 0: Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[DPEF] bit is set if in the received frame number of “1” bits is odd. 1: Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[DPEF] bit is set if in the received frame number of “1” bits is even.
24–31 DPCS*x*	DSI Peripheral Chip Select 0–7 The DPCS bits select which of the PCS signals to assert during a DSI master mode transfer. 0: Negate PCS[x] 1: Assert PCS[x]

25.3.2.11 DSPI DSI Serialization Data Register (DSPI_SDR)

The DSPI_SDR contains the states of the Parallel Input signals. The states of the Parallel Input signals are latched into the DSPI_SDR on the rising edge of every system clock. The DSPI_SDR is read-only. When the TXSS bit in the DSPI_DSICR is cleared, the data in the DSPI_SDR is used as the source of the DSI frames.

Address: DSPI_BASE + 0xC0

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

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

Figure 519. DSPI DSI Serialization Data Register (DSPI_SDR)

Table 346. DSPI_SDR field descriptions

Field	Description
0–31 SER_DATA [30:31]	Serialized Data The SER_DATA field contains the signal states of the Parallel Input signals.

25.3.2.12 DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

The DSPI_ASDR provides means for host software to write the data to be serialized. When the TXSS bit in the DSPI_DSICR is set, the data in the DSPI_ASDR is the source of the DSI frames. Writes to the DSPI_ASDR take effect on the next frame boundary.

Address: DSPI_BASE + 0xC4

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

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

Figure 520. DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

Table 347. DSPI_ASDR field descriptions

Field	Descriptions
0–31 ASER_DATA [0:31]	Alternate Serialized Data The ASER_DATA field holds the alternate data to be serialized.

25.3.2.13 DSPI DSI Transmit Comparison Register (DSPI_COMPR)

The DSPI_COMPR holds a copy of the last transmitted DSI data. The DSPI_COMPR is read-only. DSI data is transferred to this register as it is loaded into the TX Shift Register.

Address: DSPI_BASE + 0xC8

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

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

Figure 521. DSPI DSI Transmit Comparison Register (DSPI_COMPR)

Table 348. DSPI_COMPR field descriptions

Field	Description
0–31 COMP_DATA[0:31]	Compare Data The COMP_DATA field holds the last serialized DSI data.

25.3.2.14 DSPI DSI Deserialization Data Register (DSPI_DDR)

The DSPI_DDR holds the signal states for the Parallel Output signals. The DSPI_DDR is read-only and host software can read data from incoming DSI frames.

Address: DSPI_BASE + 0xCC

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

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

Figure 522. DSPI Deserialization Data Register (DSPI_DDR)

Table 349. DSPI_DDR field descriptions

Field	Descriptions
0–31 DESER_DATA[0:31]	Deserialized Data The DESER_DATA field holds deserialized data which is presented as signal states to the Parallel Output signals.

25.3.2.15 DSPI DSI Configuration Register 1 (DSPI_DSICR1)

The DSI Configuration Register 1 selects various attributes associated with TSB Configuration. The user must not write to the DSPI_DSICR1 while the DSPI is in the Running state. If TSBC bit is cleared the register value is ignored.

Address: DSPI_BASE + 0xD0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	TSBCNT					0	0	0	0	0	0	DSE1	DSE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TRGPRD								DPCS1_7	DPCS1_6	DPCS1_5	DPCS1_4	DPCS1_3	DPCS1_2	DPCS1_1	DPCS1_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 523. DSPI DSI Configuration Register 1 (DSPI_DSICR1)

Table 350. DSPI_DSICR1 field descriptions

Field	Description
0–2	Reserved, should be cleared.
3–7 TSBCNT[0:4]	Timed Serial Bus Operation Count When TSBC is set, TSBCNT defines the length of the data frame. TSBCNT field valid value is from 3 to 31. The TSBCNT field selects number of data bits to be shifted out during a transfer in TSB mode. The number of data bits in the data frame is one more than the value in the TSBCNT field.
8–13	Reserved, should be cleared.
14 DSE1	Data Select Enable1 When TBSC bit is set, the DSE1 bit controls insertion of the zero bit (Data Select) in the middle of the data frame. The insertion bit position is defined by FMSZ field of DSPI_CTARn register, selected by the DSPI_DSICR[DSICTAS] field. The TSBCNT field value must be greater than the FMSZ field value plus one for proper operation of the DSE1 bit. 1: Zero bit is inserted at the middle of the data frame. Total number of bits in the data frame is increased by 1. 0: No Zero bit inserted in the middle of the data frame.
15 DSE0	Data Select Enable0 The DSE0 bit controls insertion of the zero bit (Data Select) in the beginning of the DSI frame. 1: Zero bit is inserted at the beginning of the data frame. Total number of bits in the data frame is increased by 1. 0: No Zero bit inserted in the beginning of the frame

Table 350. DSPI_DSICR1 field descriptions (continued)

Field	Description
16–23 TRGPRD	Internal Trigger Period for the ITSB mode When TRG bit is clear and TSB & ITSB bits are set this field determines the trigger period for the internal trigger in number of baud rate clock cycles. The trigger period should be equal to Trigger Period = 32 bits (DSI frame max size) + 2 Selection Bits (max) + t_{PF} 0 - 255 - Number of baud rate clock cycles to be used as trigger period. The trigger period will be +1 the programmed value. See *Section 25.4.10: Interleaved TSB (ITSB) Mode* for mode details.
24–31 DPCS1_*x*	DSI Peripheral Chip Select 0–7 These bits define the PCSs to assert for the second part of the DSI frame when operating in TSB configuration with dual receiver. The DPCS1 bits select which of the PCS signals to assert during the second part of the DSI frame. The DPCS1 bits only control the assertions of the PCS signals in TSB mode. 0: Negate PCS[x] 1: Assert PCS[x]

25.4 Functional description

The Deserial Serial Peripheral Interface (DSPI) block supports full-duplex, synchronous serial communications between MCUs and peripheral devices. The DSPI can also be used to reduce the number of pins required for I/O by serializing and deserializing up to 32 Parallel Input/Output signals. All communications are done with SPI-like protocol.

The DSPI has three configurations:

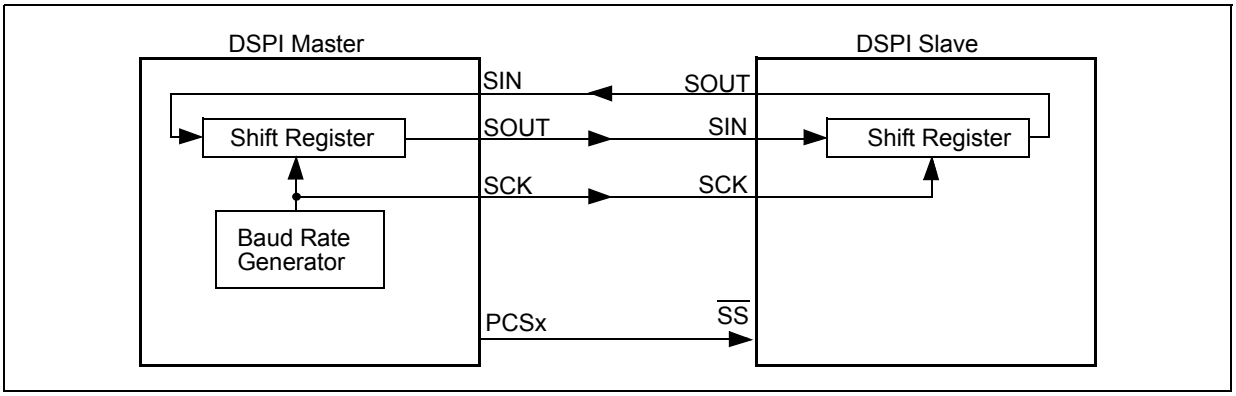
- SPI Configuration in which the DSPI operates as a basic SPI or a queued SPI.
- DSI Configuration in which the DSPI serializes and deserializes Parallel Input/Output signals or bits from memory mapped register.
- CSI Configuration in which the DSPI combines the functionality of the SPI and DSI configurations.

The DCONF field in the DSPI Module Configuration Register (DSPI_MCR) determines the DSPI Configuration. See *Table 331* for the DSPI configuration values.

The DSPI_CTAR0–DSPI_CTAR7 registers hold clock and transfer attributes. The SPI configuration allows to select which CTAR to use on a frame by frame basis by setting a field in the SPI command. The DSI configuration statically selects which CTAR to use. In CSI Configuration priority logic determines if SPI data or DSI data is transferred and dictates what CTAR is used for the data transfer. See *Section 25.3.2.3: DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR0–DSPI_CTAR7)* for information on the fields of the DSPI_CTAR registers.

Typical master to slave connections are shown in the *Figure 524*. 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 524. SPI and DSI 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 25.4.6: Transfer Formats*. The transfer rate and delay settings are described in *Section 25.4.5: DSPI Baud Rate and Clock Delay Generation*.

25.4.1 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.

25.4.2 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 25.4.2.4: Transmit First In First Out (TX FIFO) Buffering Mechanism* and *Section 25.4.2.5: Receive First In First Out (RX FIFO) Buffering Mechanism*. The interrupt and DMA request conditions are described in *Section 25.4.13: 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.

25.4.2.1 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 25.3.2.6: DSPI PUSH TX FIFO Register (DSPI_PUSHR)* for details on the SPI command fields. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the Serial Out (SOUT) pin. In SPI master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

25.4.2.2 Slave Mode

In SPI slave mode the DSPI responds to transfers initiated by a SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for successful communication with a SPI master. The SPI slave mode transfer attributes are set in the DSPI_CTAR0. The data is shifted out with MSB first.

25.4.2.3 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.

25.4.2.4 Transmit First In First Out (TX FIFO) Buffering Mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds from one to sixteen words, each consisting of a command field and a data field. The number of entries in the TX FIFO is SoC specific. SPI commands and data are added to the TX FIFO by writing to the 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 rolls over after reaching the maximum.

25.4.2.4.1 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 25.4.13.2: 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.

25.4.2.4.2 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 25.4.13.4: Transmit FIFO Underflow Interrupt Request* for details.

25.4.2.5 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 rolls over after reaching the maximum.

25.4.2.5.1 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.

25.4.2.5.2 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.

25.4.3 Deserial Serial Interface (DSI) Configuration

The DSI Configuration supports pin count reduction by serializing Parallel Input signals or register bits and shifting them out in a SPI-like protocol. The timing and transfer protocol is described in *Section 25.4.6: Transfer Formats*. The received serial frames are converted to a parallel form (deserialized) and placed on the Parallel Output signals or in the DSPI_DDR. The various features of the DSI Configuration are set in the DSPI DSI Configuration Register (DSPI_DSICR).

The DSI frames can be from four to 32 bits. With Multiple Transfer Operation (MTO) the DSPI supports serial chaining of DSPI blocks within an SoC to create DSI frames up to 64 bits, consisting of concatenated bits from multiple DSPIs. The DSPI also supports parallel chaining allowing several DSPIs and off-chip SPI devices to share the same Serial Communications Clock (SCK) and Peripheral Chip Select (PCS) signals. See *Section 25.4.3.6: Multiple Transfer Operation (MTO)* for details on the serial and parallel chaining support.

25.4.3.1 DSI Master Mode

In DSI master mode the DSPI initiates and controls the DSI transfers. The DSI master has four different conditions that can initiate a transfer:

- Continuous
- Change in data
- Trigger signal
- Trigger signal combined with a change in data

The four transfer initiation conditions are described in *Section 25.4.3.5: DSI Transfer Initiation Control*. Transfer attributes are set during initialization. The DSICTAS field in the DSPI_DSICR determines which of the DSPI_CTAR registers will control the transfer attributes.

25.4.3.2 Slave Mode

In DSI slave mode the DSPI responds to transfers initiated by a SPI or DSI bus master. In this mode the DSPI does not initiate DSI transfers. Certain transfer attributes such as clock polarity and phase must be set for successful communication with a DSI master. The DSI slave mode Transfer attributes are set in the DSPI_CTAR1. The data is shifted out with MSB first.

If the CID bit in the DSPI_DSICR is set and the data in the DSPI_COMPR differs from the selected source of the serialized data, the slave DSPI will assert the MTRIG signal. If the slave's HT signal is asserted and the TRRE is set, the slave DSPI asserts MTRIG. These features are included to support chaining of several DSPI. Details about the MTRIG signal is found in *Section 25.4.3.6: Multiple Transfer Operation (MTO)*.

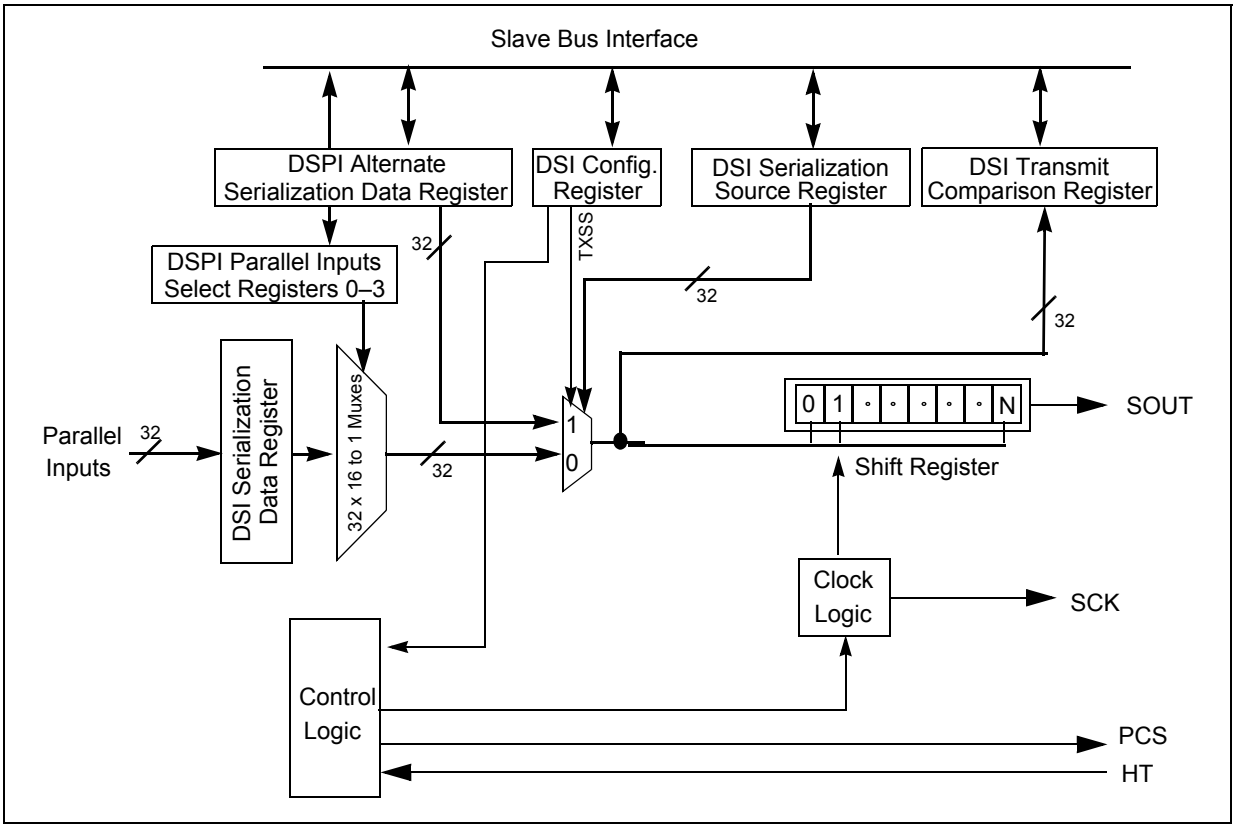
25.4.3.3 DSI Serialization

In the DSI Configuration from four to sixteen bits can be serialized using two different sources. The TXSS bit in the DSPI_DSICR selects between the DSPI DSI Serialization Data Register (DSPI_SDR) and the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR) as the source of the serialized data. The DSPI_SDR holds the latest Parallel Input signal values which is sampled at every rising edge of the system clock. The DSPI_ASDR is written by host software and used as an alternate source of serialized data.

DSPI_SSR provides additional way to create the frame for transmission. Each bit from this register is OR'd with the TXSS bit and controls individual transmitted bit source. This way, the transmitted frame can have any combination of the DSPI_SDR and DSPI_ASDR bits. This feature allows control SPI based devices, requiring control and data fields in the frame. Control field may come from DSPI_ASDR, set by the SoC CPU, while data field can be generated by SoC peripheral modules, like PWM timers.

A copy of the last 32-bit DSI frame shifted out of the Shift Register is stored in the DSPI DSI Transmit Comparison Register (DSPI_COMPR). This register provides added visibility for debugging and it serves as a reference for transfer initiation control. *Figure 525* shows the DSI Serialization logic.

Figure 525. DSI serialization diagram



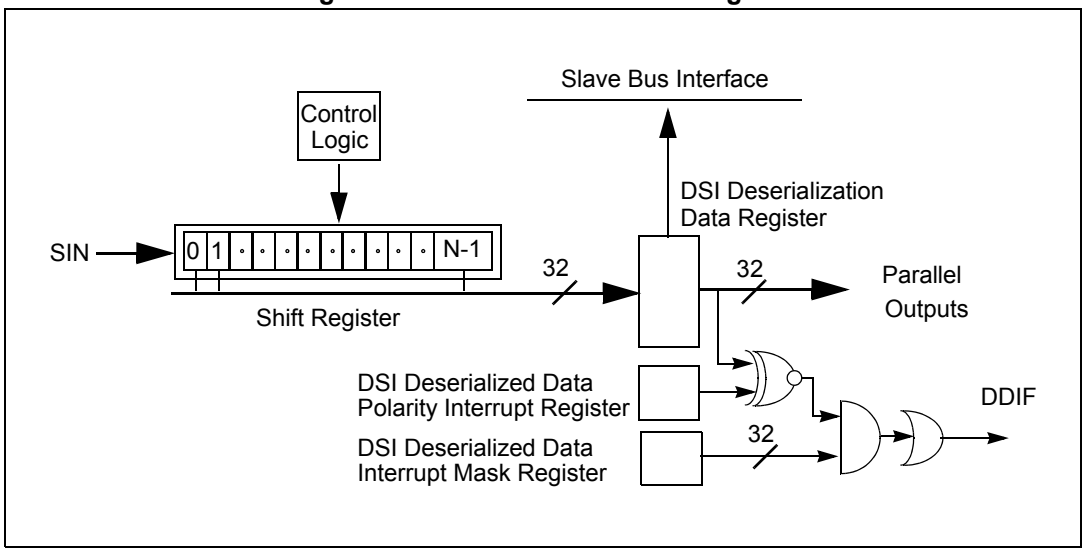
25.4.3.4 DSI deserialization

When all bits in a DSI frame have been shifted in, the frame is copied to the DSPI DSI Deserialization Data Register (DSPI_DDR). This register presents the deserialized data as Parallel Output signal values. The DSPI_DDR is memory mapped to allow host software to read the deserialized data directly.

The received data is bit-wise compared to the value of the DSI Deserialized Data Polarity Interrupt Register, bit-wise AND'ed with DSI Deserialized Interrupt Mask Register and the results OR'ed to produce DDIF flag in the DSPI_SR. Which in turn can cause DDI interrupt request if the DDIF_RE bit of DSPI_RSER is set and/or stop DSI frame transmissions if the DMS bit of the DSPI_DSICR is set.

Figure 526 shows the DSI Deserialization logic.

Figure 526. DSI deserialization diagram



25.4.3.5 DSI Transfer Initiation Control

Data transfers for a master DSPI in DSI configuration are initiated by a condition. The transfer initiation conditions are selected by the TRRE and CID bits in the DSPI_DSICR. *Table 351* lists the four transfer initiation conditions.

Table 351. DSI data transfer initiation control

DSPI_DSICR bits		Transfer initiation control
TRRE	CID	
0	0	Continuous
0	1	Change in Data
1	0	Triggered
1	1	Triggered or Change in Data

25.4.3.5.1 Continuous Control

For Continuous Control a new DSI frame shifts out when the previous transfer cycle has completed and the Delay after Transfer (t_{DT}) has elapsed.

25.4.3.5.2 Change In Data Control

For Change in Data Control a transfer is initiated when the data to be serialized has changed since the transfer of the last DSI frame. A copy of the previously transferred DSI data is stored in the DSPI_COMPR. When the data, selected for the transfer from the DSPI_SDR and DSPI_ASDR registers is different from the data in the DSPI_COMPR a new DSI frame is transmitted. The MTRIG output signal is asserted every time a change in data is detected.

25.4.3.5.3 Triggered Control

For Triggered Control initiation of a transfer is controlled by the Hardware Trigger signal (HT). The TPOL bit in the DSPI_DSICR selects the active edge of HT. For HT to have any affect, the TRRE bit in the DSPI_DSICR must be set.

25.4.3.5.4 Triggered or Change In Data Control

For Triggered or Change in Data Control initiation of a transfer is controlled by the HT signal or by the detection of a change in data to be serialized.

25.4.3.6 Multiple Transfer Operation (MTO)

In DSI Configuration the MTO feature allows for multiple DSPIs within an SoC to be chained together in a parallel or serial configuration. The parallel chaining allows multiple DSPIs internal to an SoC and multiple SPI devices external to an SoC to share SCK and PCS signals thereby saving the SoC pins. The serial chaining allows bits from multiple DSPIs to be concatenated into a single DSI frame. MTO is enabled by setting the MTOE bit in the DSPI_DSICR.

In parallel and serial chaining there is one bus master and multiple bus slaves. The bus master initiates and controls the transfers, but the DSPI slaves generate trigger signals for the bus DSPI master when an internal condition in the slave warrants a transfer. The DSPI slaves also propagate triggers from other slaves to the master. When a DSPI slave detects a trigger signal on its HT input, the slave generates a trigger signal on the $\overline{\text{MTRIG}}$ output.

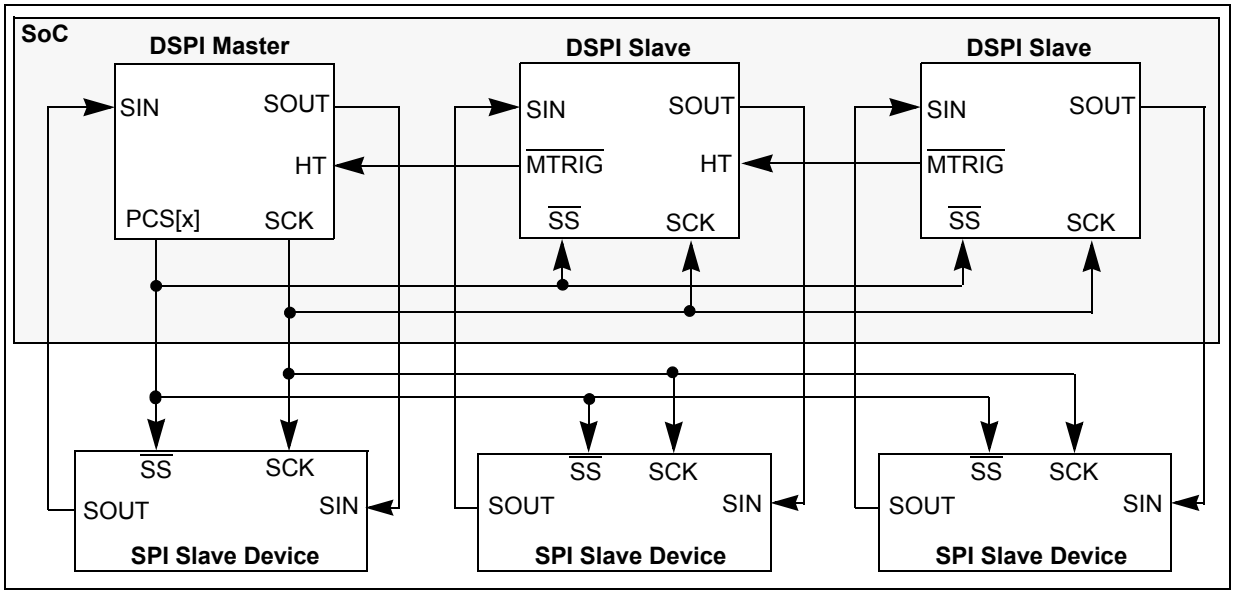
Serial and parallel chaining require multiplexing of signals external to the DSPI.

Note: TSB operation is not available in MTO mode. TSBC and MTOE bits of DSPI_DSICR should not be set simultaneously.

25.4.3.6.1 Parallel chaining

Parallel chaining allows multiple DSPIs internal to an SoC and multiple SPI/DSI devices external to an SoC to share common SCK and PCS signals thereby saving pins. Two pins are saved per pair of DSPI/SPI. *Figure 527* shows an example of how the blocks can be connected in an SoC.

Figure 527. Example of Parallel Chaining of DSPIs



The DSPI master controls and initiates all transfer, but the DSPI slaves have a trigger output signal MTRIG that indicates to the master DSPI to start a transfer. When the DSPI slave has a change in its data to be serialized, it generates a pulse on the MTRIG signal to the master DSPI, which initiates the transfer. When a DSPI slave has its HT signal asserted it also generates a pulse on its MTRIG signal thereby propagating trigger signals from other DSPI slaves to the DSPI master.

To enable MTRIG signals generation the MTO has to be enabled for the DSPI slaves. The chained DSPI modules also should be in DSI configuration. The DSPI_DSICR[MTOCNT] field must be written with the same number of bits to be transferred for each slave DSPI. The MTOCNT value must also match the FMSZ field of the DSPI_CTAR1 in the DSPI slaves and selected DSPI_CTAR for the master DSPI module.

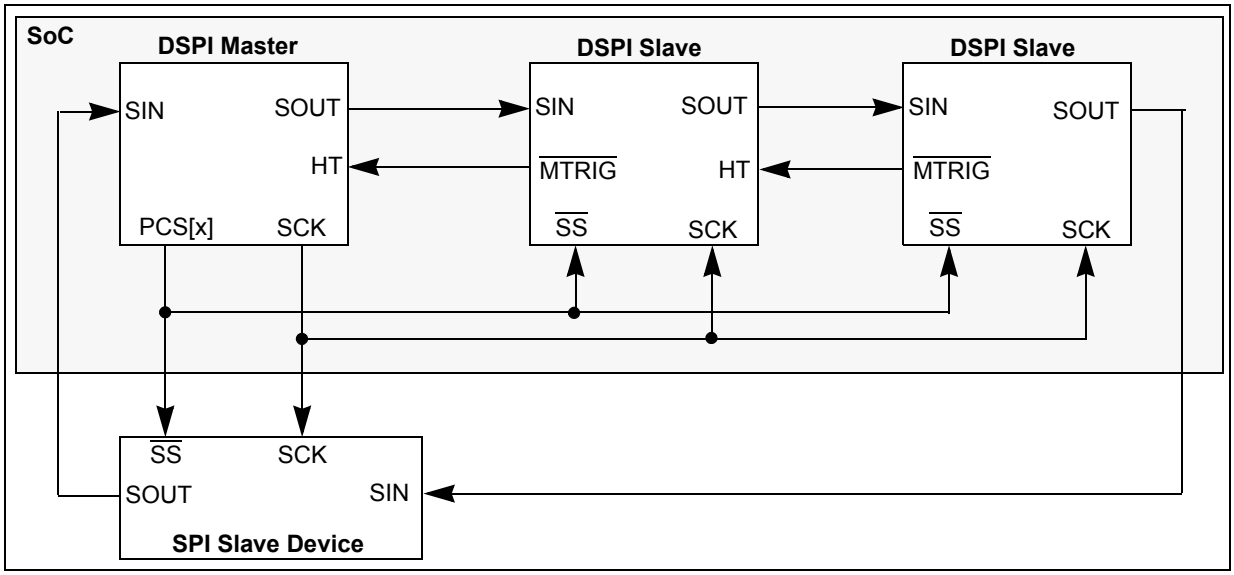
Note: When using the DSPI in DSI mode, with MTO enabled and clock phase set to leading edge capture (DSPIx_CTARn[CPHA]=0), the first bit shifted out of the master DSPI into the slave DSPI is read as "1", regardless of the actual value. To account for this behavior, the following options are recommended:

- *Select CPHA=1 (following edge capture), if suitable for external slave devices.*
- *Set the first bit of the transferred data to "1", or ignore the first bit.*
- *Externally connect master SOUT to SIN of the first slave, rather than connecting via internal signals. This requires setting SIU_DISR_SINSELx bits of the first slave DSPI to "00" and configuring the first slave's SIN pin and master SOUT pin as DSPI SIN and DSPI SOUT, respectively.*

25.4.3.6.2 Serial Chaining

The serial chaining allows transfers of DSI frames of up to a total of 64 bits, using transfers of smaller DSI frames concatenated together by multiple DSPIs. *Figure 528* shows an example of how the blocks can be connected in an SoC.

Figure 528. Example of Serial Chaining of DSPIs



The SOUT of the DSPI master is connected to the SIN of the first DSPI slave. The SOUT of the first DSPI slave is connected to the SIN input of the second slave and so on. The SOUT of the last DSPI slave is connected to the SIN of the external SPI slave. The SOUT of the external SPI slave is connected to the SIN of the DSPI master.

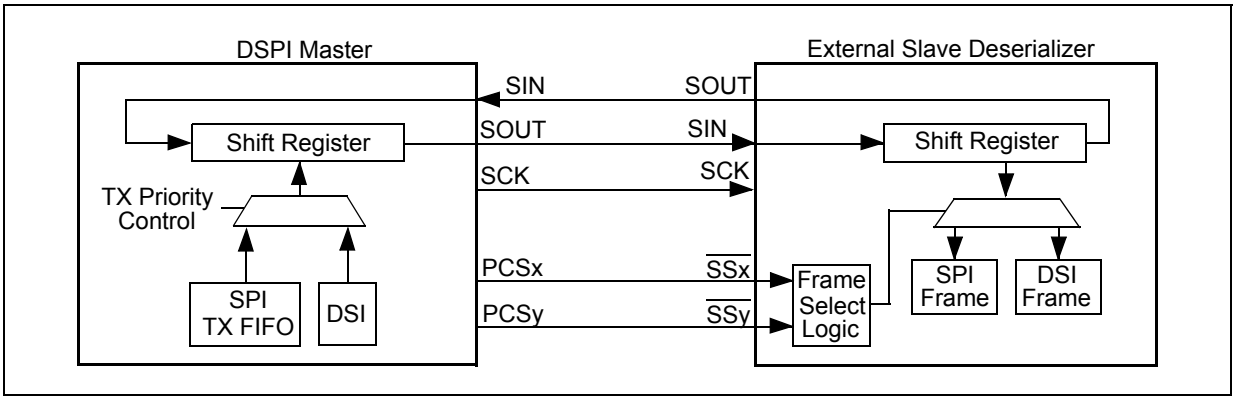
The DSPI master controls and initiates all transfers, but the slave DSPIs use the trigger output signal MTRIG to indicate to the DSPI master that a trigger condition has occurred. When a DSPI slave has a change in data to be serialized it asserts the MTRIG signal to the DSPI master which initiates the transfer. When a DSPI slave has its HT signal asserted it will assert its MTRIG signal thereby propagating trigger signals from other DSPI slaves to the DSPI master.

The DSPI_DSICR[MTOE] bit must be set in master and all slave devices. The DSPI_DSICR[MTOCNT] field for the master and all slave devices must be written with the same value. The value must equal the sum of all FMSZ fields in the selected DSPI_CTAR registers for the DSPI master and all DSPI slaves. For example if one 16-bit DSI frame is created by concatenating eight bits from the DSPI master, and four bits from each of the DSPI slaves in *Figure 528*, the DSPI master frame size must be set to eight in the FMSZ field, and the DSPI slaves frame size must be set to four. The largest DSI frame supported by the MTOCNT field is 64 bits. Any number of DSPIs can be connected together to concatenate DSI frames, as long as each DSPI transfers a minimum of 4 bits and a maximum of 32 bits and the total size of the concatenated frame is less than 65 bits long.

25.4.4 Combined Serial Interface (CSI) Configuration

The CSI Configuration of the DSPI is used to support SPI and DSI functions on a frame by frame basis. CSI Configuration allows interleaving of DSI data frames from the Parallel Input signals with SPI commands and data from the TX FIFO. The data returned from the bus slave is either used to drive the Parallel Output signals or it is stored in the RX FIFO. The CSI Configuration allows serialized data and configuration or diagnostic data to be transferred to a slave device using only one serial link. The DSPI is in CSI Configuration when the DCONF field in the DSPI_MCR is 0b10. *Figure 529* shows an example of how a DSPI can be used with a deserializing peripheral that supports SPI control for control and diagnostic frames.

Figure 529. Example of System using DSPI in CSI Configuration



In CSI Configuration the DSPI transfers DSI data based on DSI Transfer Initiation Control. When there are SPI commands in the TX FIFO, the SPI data has priority over the DSI frames. When the TX FIFO is empty, DSI transfer resumes.

Two peripheral chip select signals indicate whether DSI data or SPI data is transmitted. The user must configure the DSPI so that the two CTAR registers associated with DSI data and SPI data assert different peripheral chip select signals denoted in the figure as PCSx and PCSy. The CSI Configuration is only supported in master mode.

Data returned from the external slave while a DSI frame is transferred is placed on the Parallel Output signals. Data returned from the external slave while a SPI frame is transferred is moved to the RX FIFO. The TX FIFO and RX FIFO are fully functional in CSI mode.

25.4.4.1 CSI Serialization

Serialization in the CSI configuration is similar to serialization in DSI Configuration. The transfer attributes for SPI frames are determined by the DSPI_CTAR selected by the CTAS field in the SPI command halfword. The transfer attributes for the DSI frames are determined by the DSPI_CTAR selected by the DSICTAS field in the DSPI_DSICR.

The Parallel Inputs signal states are latched into the DSPI DSI Serialization Data Register (DSPI_SDR) on the rising edge of every system clock and serialized based on the transfer initiation control settings in the DSPI_DSICR. When SPI frames are written to the TX FIFO they have priority over DSI data from the DSPI_SDR and are transferred at the next frame boundary. A copy of the most recently transferred DSI frame is stored in the DSPI_COMPR. The Transfer Priority Logic selects the source of the serialized data and asserts the appropriate PCS signal.

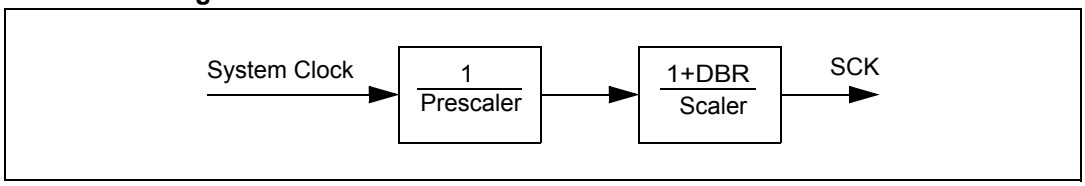
25.4.4.2 CSI Deserialization

The deserialized frames in CSI Configuration goes into the DSPI_DDR or the RX FIFO based on the transfer priority logic. When DSI frames are transferred the returned frames are deserialized and latched into the DSPI_DDR. When SPI frames are transferred the returned frames are deserialized and written to the RX FIFO.

25.4.5 DSPI Baud Rate and Clock Delay Generation

The SCK frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option for doubling the baud rate. *Figure 530* shows conceptually how the SCK signal is generated.

Figure 530. Communications Clock Prescalers and Scalers



25.4.5.1 Baud Rate Generator

The Baud Rate is the frequency of the Serial Communication Clock (SCK). The system clock is divided by a prescaler (PBR) and scaler (BR) to produce SCK with the possibility of halving the scaler division. The DBR, PBR and BR fields in the DSPI_CTAR registers select the frequency of SCK by the formula in the BR field description. *Table 352* shows an example of how to compute the baud rate.

Table 352. 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

25.4.5.2 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 532* 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. *Table 353* shows an example of how to compute the PCS to SCK delay.

Table 353. 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

PCSCSK and CSSCK fields have no effect in TSB configuration.

25.4.5.3 After SCK Delay (t_{ASC})

The After SCK Delay is the length of time between the last edge of SCK and the negation of PCS. See *Figure 532* and *Figure 533* for illustrations of the After SCK delay. The PASC and ASC fields in the DSPI_CTAR*x* registers select the After SCK Delay by the formula in the ASC field description. *Table 354* shows an example of how to compute the After SCK Delay.

Table 354. After SCK Delay computation example

f_{sys}	PASC	Prescaler	ASC	Scaler	After SCK delay
100 MHz	0b01	3	0b0100	32	0.96 µs

PCASC and ASC fields have no effect in TSB configuration.

25.4.5.4 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 532* for an illustration of the Delay after Transfer. The PDT and DT fields in the DSPI_CTAR*x* registers select the Delay after Transfer by the formula in the DT field description. *Table 355* shows an example of how to compute the Delay after Transfer.

Table 355. Delay after Transfer computation example

f_{sys}	PDT	Prescaler	DT	Scaler	Delay after transfer
100 MHz	0b01	3	0b1110	32768	0.98 ms

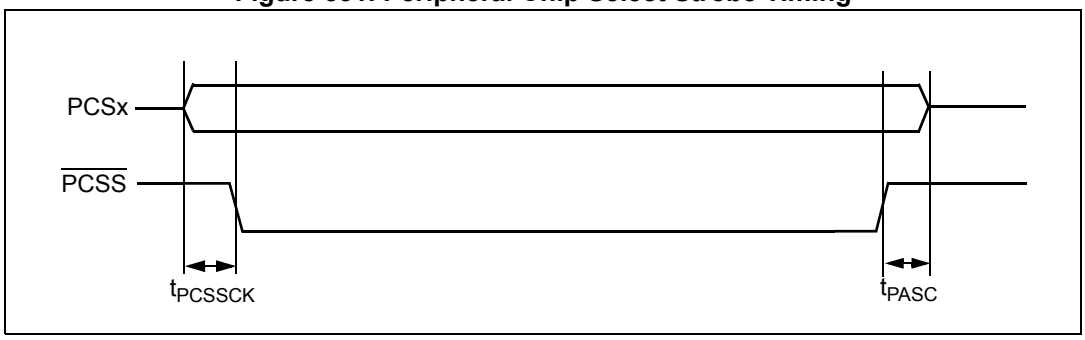
When in non-continuous clock mode the t_{DT} delay is configured according *Equation 14* . When in continuous clock mode and TSB is not enabled the delay is fixed at 1 SCK period.

In TSB mode the Delay after Transfer is equal to a number formed by concatenation of PDT and DT fields plus 1 of the SCK clock periods. See detailed information on *Section 25.4.9: Timed Serial Bus (TSB)*.

25.4.5.5 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 531* shows the timing of the $\overline{PCSS}$ signal relative to PCS signals.

Figure 531. 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 16

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

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 17

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC$$

Table 356 shows an example of how to compute the t_{pcssck} delay.

Table 356. Peripheral Chip Select Strobe Assert computation example

f_{sys}	PCSSCK	Prescaler	Delay before Transfer
100 MHz	0b11	7	70.0 ns

Table 357 shows an example of how to compute the t_{pasc} delay.

Table 357. Peripheral Chip Select Strobe Negate computation example

f_{sys}	PASC	Prescaler	Delay after Transfer
100 MHz	0b11	7	70.0 ns

The $\overline{PCSS}$ signal is not supported when Continuous Serial Communication SCK or TSB mode are enabled.

25.4.6 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 (SPI) or DSPI_CTAR1 (DSI) select the polarity and phase of the serial clock. Even though the bus slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master and the slave devices to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified Transfer format with CPHA = 0
- Modified Transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later

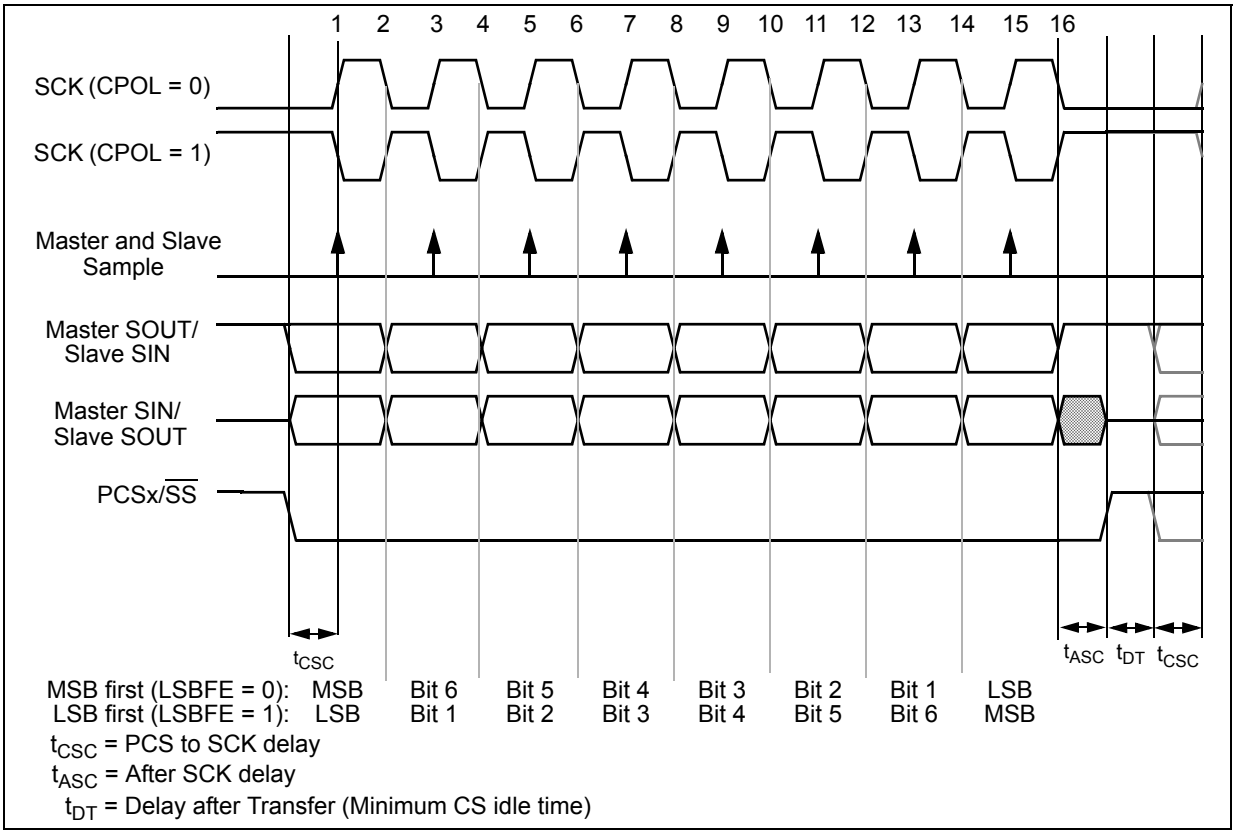
than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI_MCR selects between Classic SPI Format and Modified Transfer Format.

In the SPI and DSI Configurations, the DSPI provides the option of keeping the PCS signals asserted between frames. See *Section 25.4.6.5: Continuous Selection Format* for details.

25.4.6.1 Classic SPI Transfer Format (CPHA = 0)

The transfer format shown in *Figure 532* 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 532. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

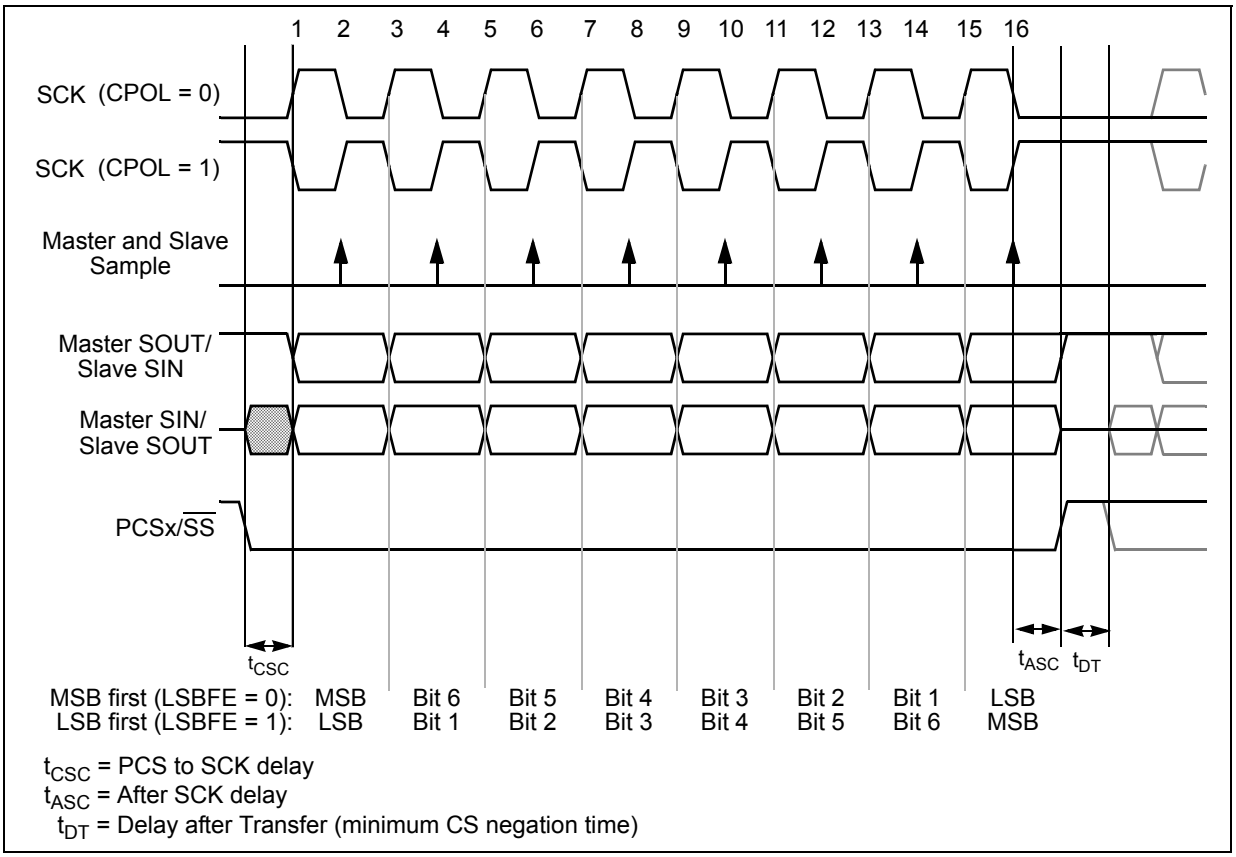


The master initiates the transfer by placing its first data bit on the SOUT pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT pin. After the t_{CSC} delay elapses, the master outputs the first edge of SCK. 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.

25.4.6.2 Classic SPI Transfer Format (CPHA = 1)

This transfer format shown in *Figure 533* 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

Figure 533. 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.

25.4.6.3 Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 0)

In this Modified Transfer Format both the master and the slave sample later in the SCK period than in Classic SPI mode to allow tolerate more delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

The master and the slave place data on the SOUT pins at the assertion of the 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. *Table 331* 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.

The 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 534 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 534. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/4$)

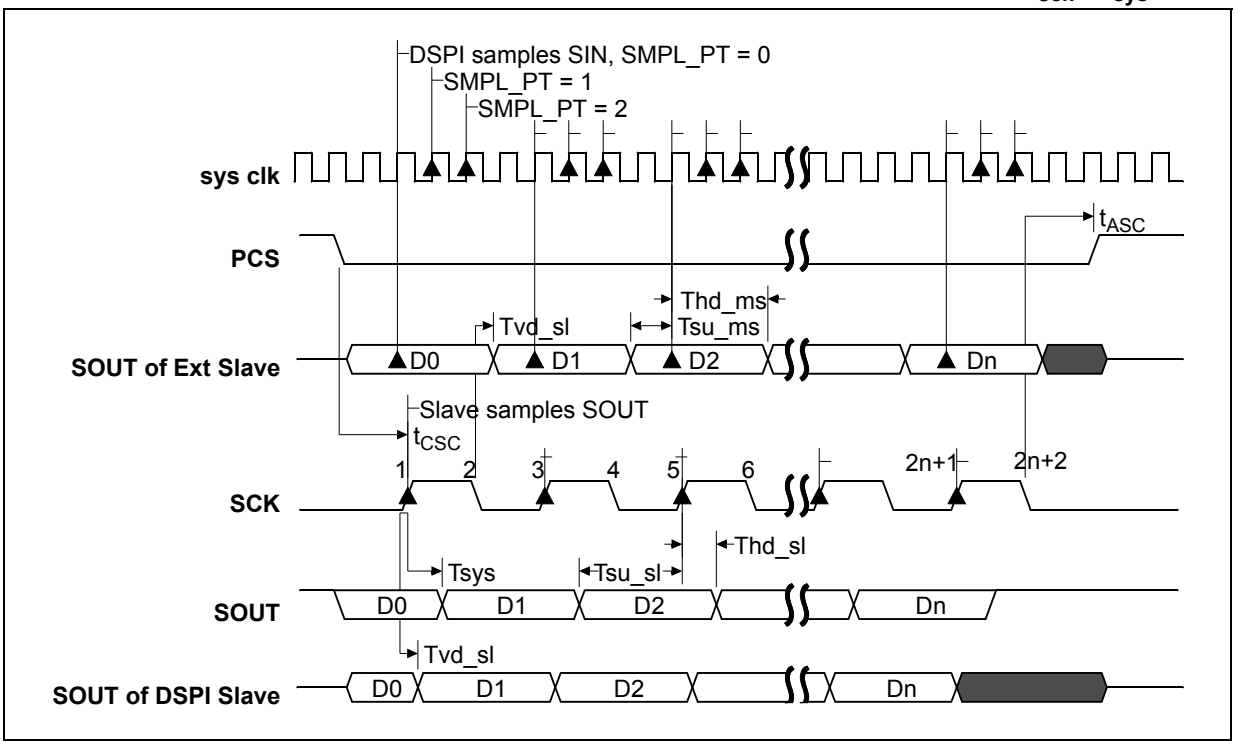


Figure 535. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/2$)

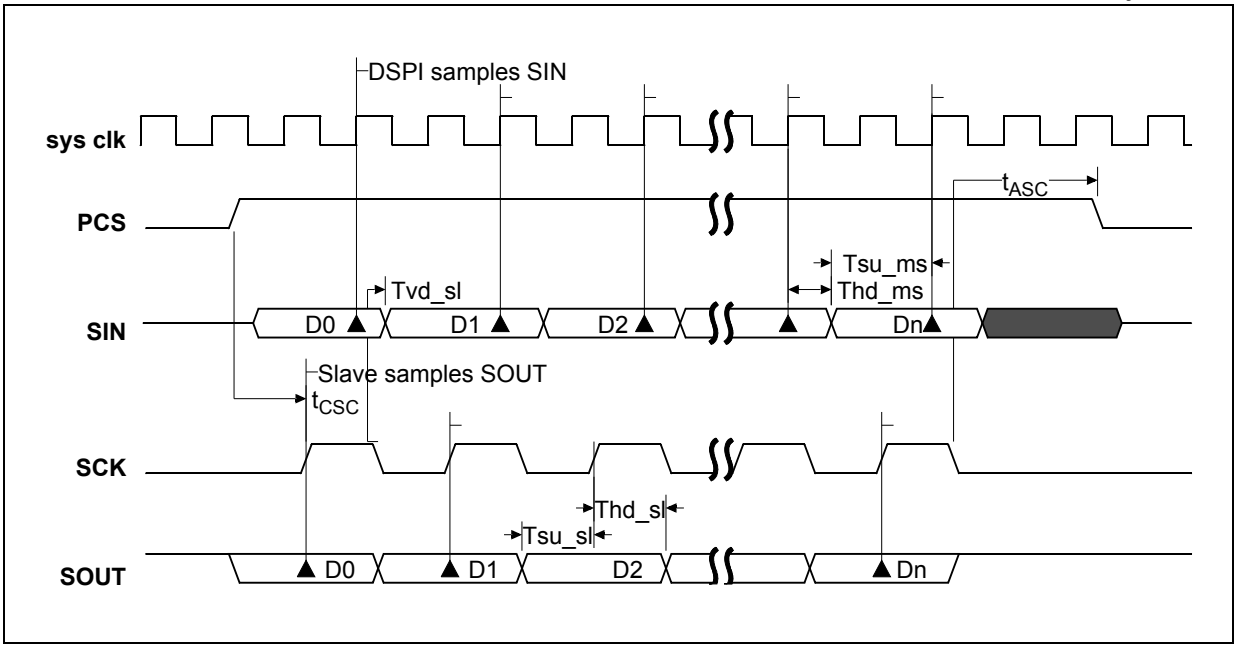
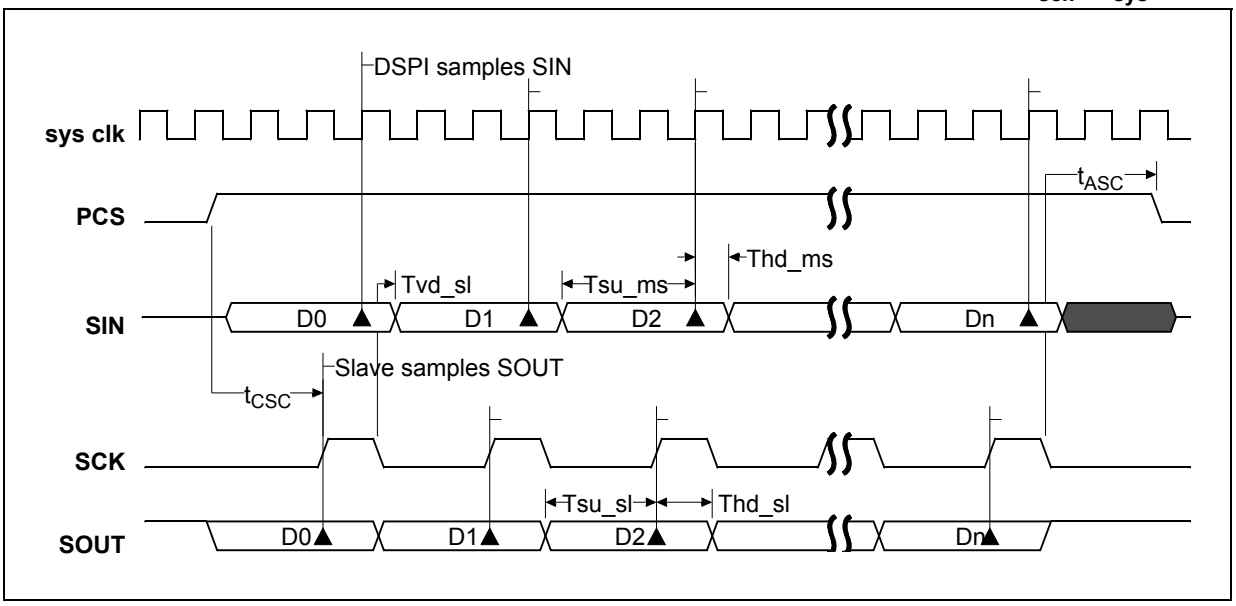


Figure 536. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/3$)



25.4.6.4 Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 1)

Figure 537, *Figure 538* and *Figure 539* 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 537. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/2$)

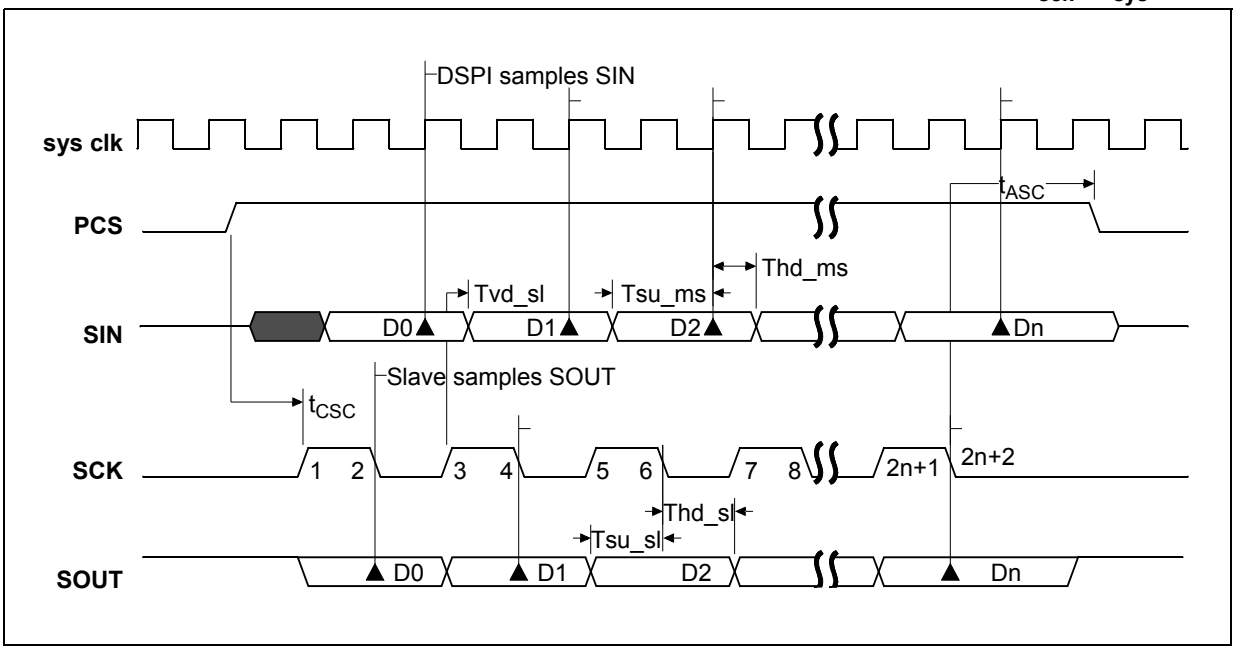


Figure 538. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/3$)

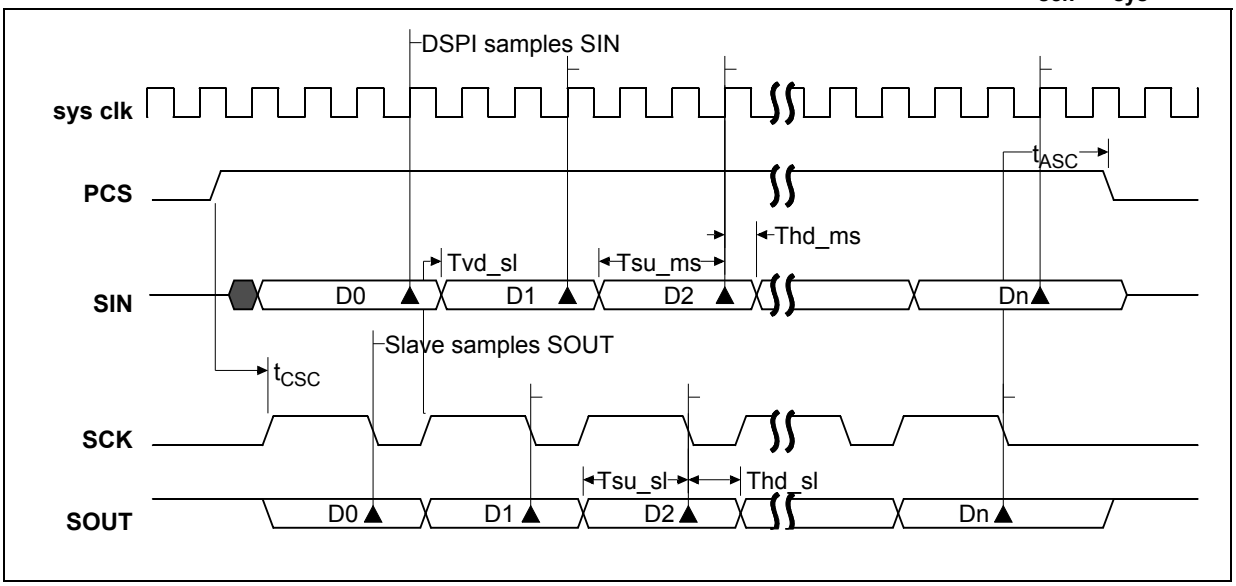
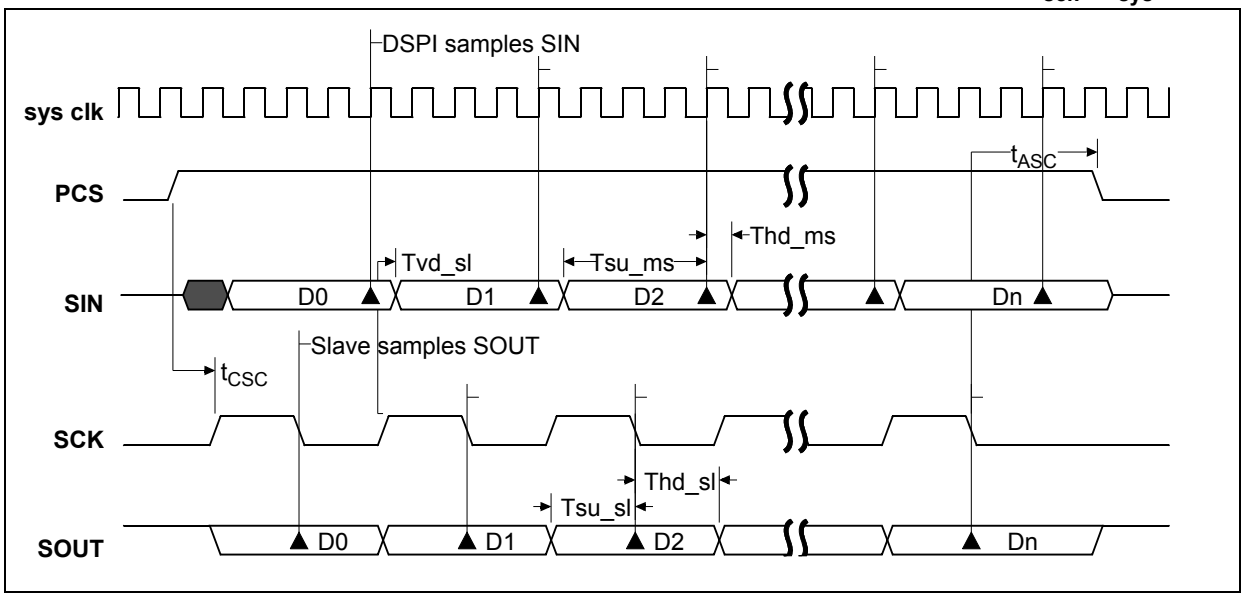


Figure 539. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/4$)

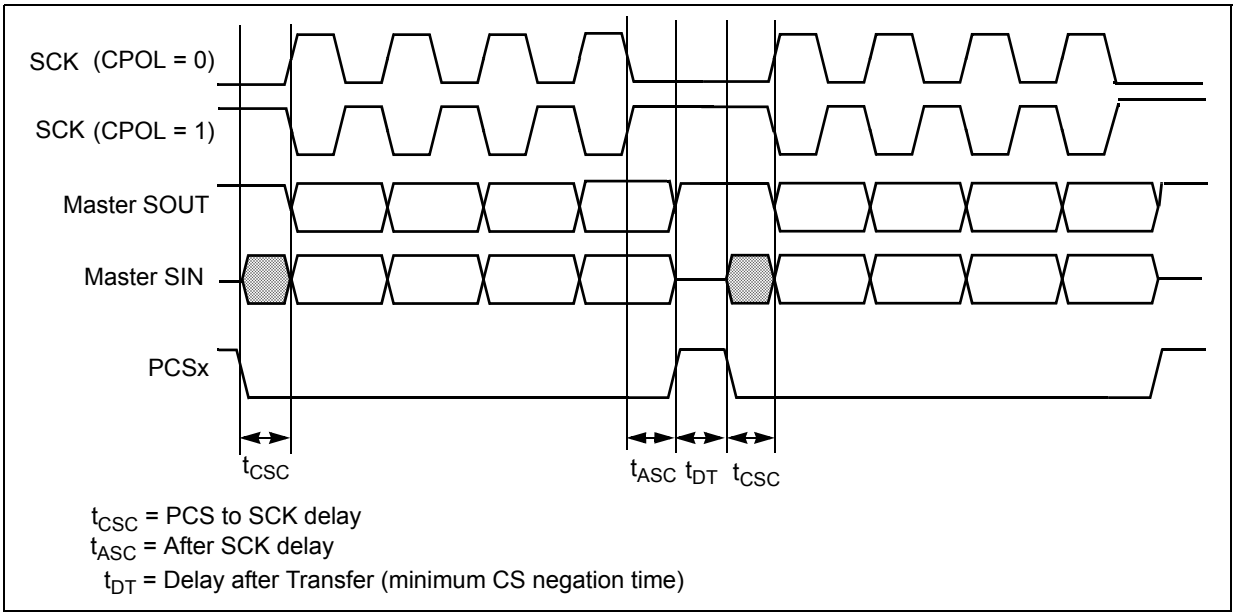


25.4.6.5 Continuous Selection Format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The Continuous Selection Format provides the flexibility to handle both cases. The Continuous Selection Format is enabled for the SPI Configuration by setting the CONT bit in the SPI command. Continuous Selection is enabled for the DSI Configuration by setting the DCONT bit in the DSPI_DSICR. The behavior of the PCS signals in the two configurations is identical so only SPI Configuration will be described.

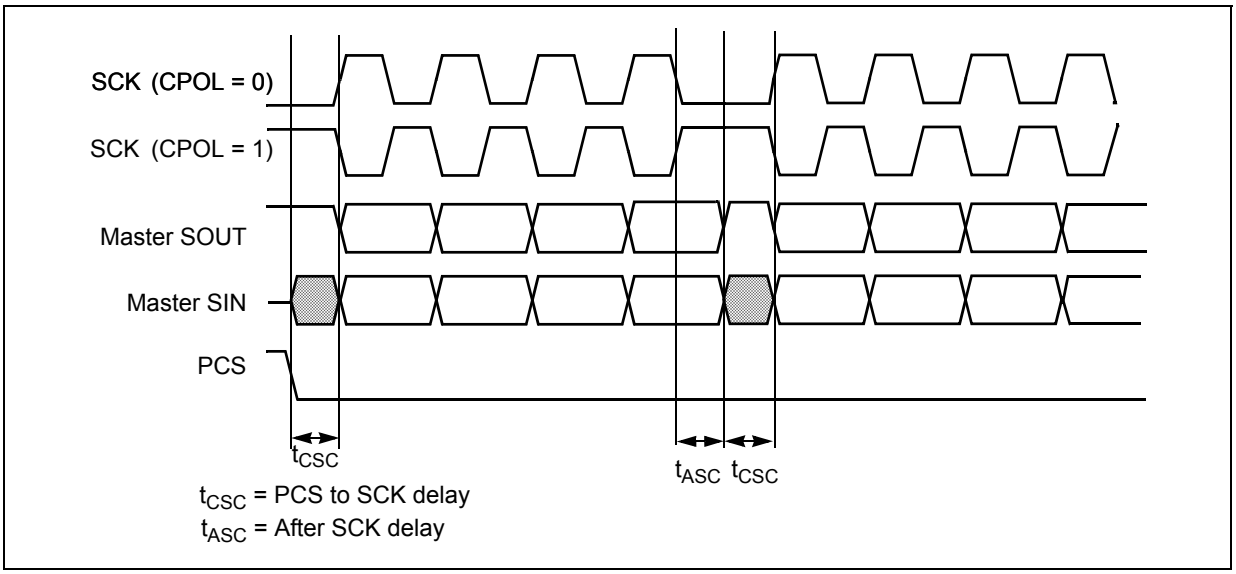
When the CONT bit = 0, the DSPI drives the asserted Chip Select signals to their idle states in between frames. The idle states of the Chip Select signals are selected by the PCSISn bits in the DSPI_MCR. *Figure 540* shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 0.

Figure 540. Example of Non-Continuous Format (CPHA = 1, CONT = 0)



When the CONT bit = 1, the PCS signal remains asserted for the duration of the two transfers. The Delay between Transfers (t_{DT}) is not inserted between the transfers. *Figure 541* shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 1.

Figure 541. Example of Continuous Transfer (CPHA = 1, CONT = 1)



When using DSPI with continuous selection format, 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.
- when transmitting multiple frames in this mode, the user software must ensure that the last frame has the CONT bit de-asserted (in master mode) and the user software must

provide sufficient frames in the Tx_FIFO to be sent out (in slave mode) and the master de-asserts the PCSx at end of transmission of last frame.

Note: During continuous selection mode, to avoid generation of erroneous data, do not change the DSPIx_CTAR values between frames in the following conditions:

- *If DSPIx_CTARn[CPHA=1] AND DSPIx_MCR[CONT_SCKE]=0, do not change DSPIx_CTARn[CPOL, CPHA, PCSSCK or PBR] between frames.*
- *If DSPIx_CTARn[CPHA]=0 OR DSPIx_MCR[CONT_SCKE]=1, do not change any bit field of DSPIx_CTARn, except [PBR], between frames.*

Note: It is mandatory to fill the TXFIFO with the number of entries that will be concatenated under one PCS assertion for both master and slave before the TXFIFO becomes empty. For example, while transmitting in master mode, it should be ensured that the last entry in the TXFIFO, after which TXFIFO becomes empty, must have the CONT bit in command frame as deasserted (that is, CONT bit = 0). While operating in slave mode, it should be ensured that when the last-entry in the TXFIFO is completely transmitted (that is, the corresponding TCF flag is asserted and TXFIFO is empty) the slave should be deselected for any further serial communication, else an underflow error occurs.

25.4.6.6 Fast Continuous Selection Format

The Fast Continuous Selection Format is same as the Continuous Selection Format described in *Section 25.4.6.5: Continuous Selection Format* except that the inter command delays i.e. t_{ASC} and t_{CSC} can be masked out and are not inserted by the hardware.

Note: The Fast Continuous Selection Format is available in the SPI configuration only and when Continuous Serial Communication Clock mode is disabled. Masking of delays is not allowed in DSI and CSI configurations or if the transfer is non-continuous.

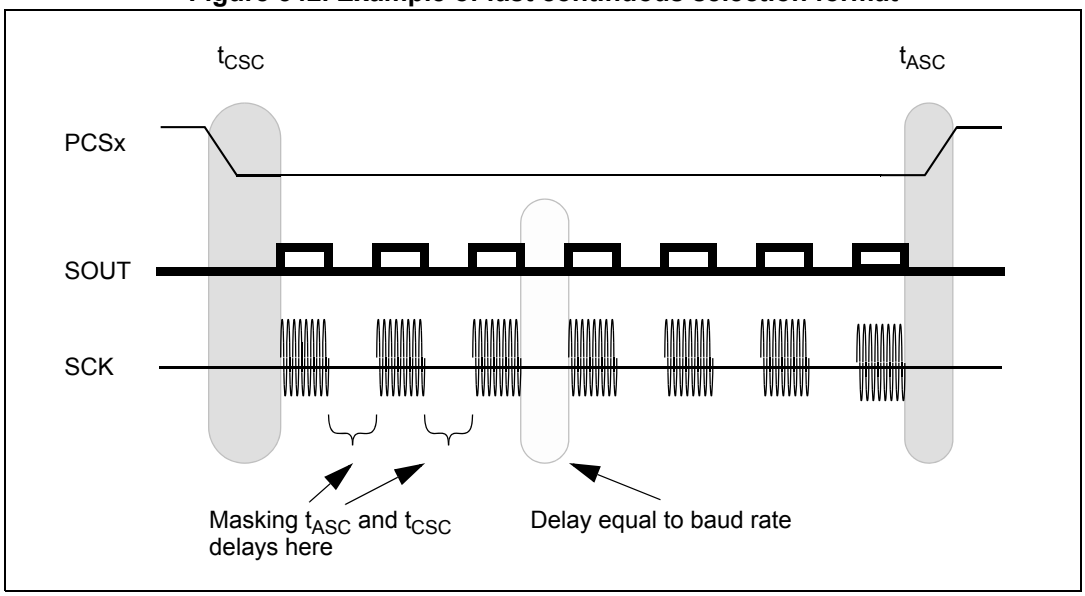
The Fast Continuous Selection Format is enabled by writing ‘1’ into bit 2 of the DSPI_MCR. When this bit is asserted, the bits 25:24 of the DSPI_PUSHR perform the function of mask bits for the transmit frame. These bits individually mask the t_{ASC} and t_{CSC} delays as programmed by the user software. A normal Continuous Selection Format has these two delays for each frame that is transmitted with the CONT bit asserted. In order to avoid these delays and speed of the transfer process, the software can simply mask these delays while programming the command in the DSPI_PUSHR.

While masking the delays the software must follow the following masking rules, else, correct operation is not guaranteed.

- Mask t_{ASC} bit masks the “After SCK” delay for the current frame
- Mask t_{CSC} bit masks the “PCS to SCK” delay for the next frame
- “After SCK” or ASC delay must not be masked when the current frame is the last frame in the continuous selection format.
- The “PCS to SCK” delay for the first frame in the continuous selection format cannot be masked.
- Masking of only t_{ASC} is not allowed. If t_{ASC} is masked then t_{CSC} must be masked too.
- Masking of both t_{ASC} and t_{CSC} delays is allowed. In this case, the delay between two frames is equal to the baud rate set by the user software.
- Masking of only t_{CSC} is allowed. In this case, the delay between two frames is equal to the t_{ASC} time and thus the user software must ensure that the t_{ASC} time is greater than the baud rate.
- The user software must not mask these delays if the continuous selection format is not used and DSPI_MCR[2] is asserted.
- Rules applicable to the Continuous Selection Format are applicable here too.

Figure 542 shows the timing for a Fast Continuous Selection Format transfer. Here seven frames are transferred with both ASC & CSC delays masked except for the last frame that terminated the transfer. The last frame has ASC delay at its end.

Figure 542. Example of fast continuous selection format



In case any chip select is to be changed, then the fast continuous selection format should be terminated and then the chips selects should change and appropriate delays must be introduced.

25.4.7 Continuous Serial Communications Clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting the CONT_SCKE bit in the DSPI_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Clearing CPHA is ignored if the CONT_SCKE bit is set. Continuous SCK is supported for Modified Transfer Format.

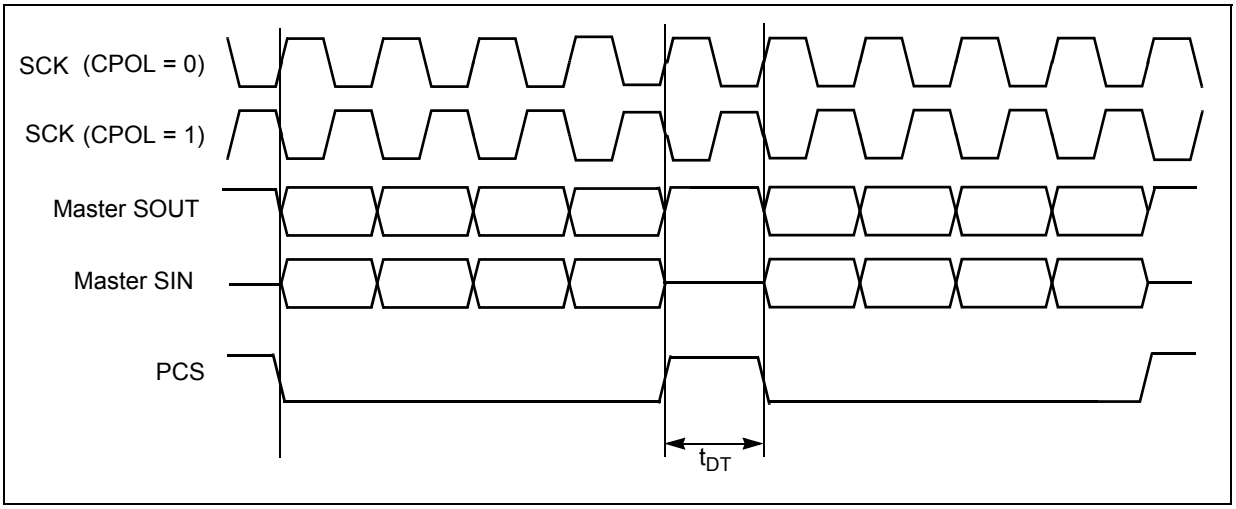
Clock and transfer attributes for the Continuous SCK mode are set according to the following rules:

- The TX FIFO must be cleared before initiating any SPI configuration transfer.
- When the DSPI is in SPI configuration, CTAR0 is used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame must be CTAR0.
- When the DSPI is in DSI configuration, the CTAR specified by the DSICTAS field is used at all times.
- When the DSPI is in CSI configuration, the CTAR selected by the DSICTAS field is used initially.
  - At the start of a SPI frame transfer, the CTAR specified by the CTAS value for the frame is used.
  - At the start of a DSI frame transfer, the CTAR specified by the DSICTAS field is used.
- In all configurations, the currently selected CTAR remains in use until the start of a frame with a different CTAR specified, or the Continuous SCK mode is terminated.

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. When TSB configuration is enabled the t_{DT} is programmable from 1 to 65 SCK cycles. *Figure 543* shows timing diagram for Continuous SCK format with Continuous Selection disabled.

Figure 543. Continuous SCK Timing Diagram (CONT = 0)



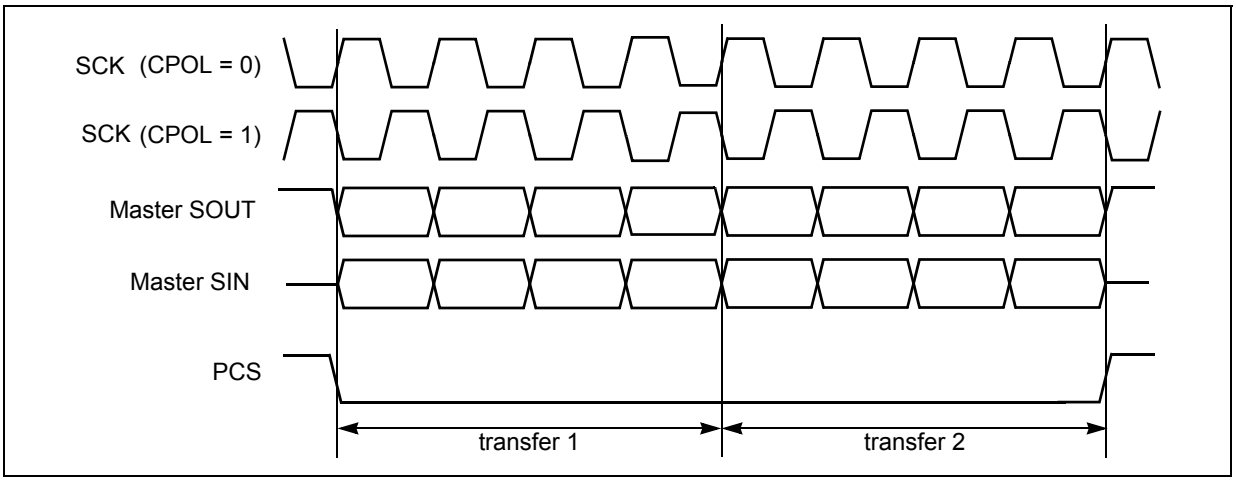
If the CONT bit in the TX FIFO entry is set or the DCONT in the DSPI_DSICR is set, PCS remains asserted between the transfers. Under certain conditions, SCK can continue with

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 25.4.1: Start and Stop of DSPI Transfers*)
- Continuous SCK with CONT bit set and entering Stop mode or Module Disable mode

Figure 544 shows timing diagram for Continuous SCK format with Continuous Selection enabled.

Figure 544. Continuous SCK Timing Diagram (CONT = 1)



25.4.8 Slave mode operation constraints

Slave mode logic shift register is buffered. This allows data streaming operation, when the DSPI is permanently selected and data is shifted in with a constant rate.

The transmit data is transferred at second SCK clock edge of the each frame to the shift register if the $\overline{SS}$ signal is asserted and any time when transmit data is ready and $\overline{SS}$ signal is negated.

Received data is transferred to the receive buffer at last SCK edge of each frame, defined by frame size programmed to the CTAR0/1 register. Then the data from the buffer is transferred to the RXFIFO or DDR.

If the $\overline{SS}$ negates before that last SCK edge, the data from shift register is lost.

This buffering scheme allows to operate slave clock with higher frequency than the system frequency. The clocks relationship is defined by *Equation 18* .

FrameSize is the value of the CTAR0/1[FMSZ] field plus one.

Equation 18

$$f_{SCK} < f_{SYS} \times FrameSize / 3$$

In slave mode, the DSPI performance is limited by synchronization and propagation through combinational logic to the serial data output. Performance limitations for various slave configurations are summarized in *Table 358*.

Table 358. DSPI slave mode system constraints

CPHA	System constraint(1)
0	$t_{CSC} > 2 \times T_{SYS} + t_{sl_vd} + t_{ms_su}$
1	$t_{CSC} > 2 \times T_{SYS}$

1. Timing parameter definitions are:
T_{SYS} = system clock period
t_{CSC} = delay from PCS asserted to first serial clock edge
t_{sl_vd}= SOUT data valid time in slave mode
t_{ms_su} = SPI master data setup time

These limitations must be taken into account at a system level with the SPI transfer timing set to allow for worst case performance of the SPI slave (including I/O and board level propagation delays). To ensure correct operation, the master SPI device must be configured so that t_{CSC} are greater the requirements shown in the *Table 358* and the SCK clock frequency should comply to the *Equation 18* .

25.4.9 Timed Serial Bus (TSB)

The DSPI can be programmed in Timed Serial Bus configuration by setting the TSBC bit in the DSPI_DSICR. See *Section 25.3.2.10: DSPI DSI Configuration Register (DSPI_DSICR)* for details.

TSB configuration provides the Micro Second Channel (MSC) downstream channel support.

The MSC upstream channel is not supported by the DSPI, but can be supported by any available Serial Communication Controller (SCI or UART) in the SoC.

To work in TSB mode the DSPI must be in master mode and in DSI (DCONF = 0b01) or CSI (DCONF = 0b10) configuration. Both Continuous and Non Continuous Serial Communication Clock (controlled by the DSPI_MCR[CONT_SCKE] bit) are supported in the TSB mode.

Figure 545 shows the signals used in the TSB interface.

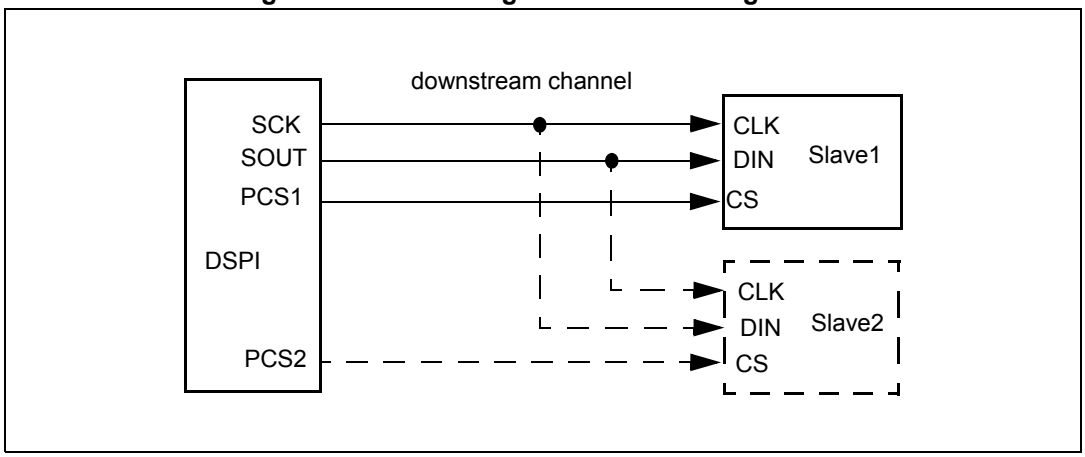
In the TSB configuration the DSPI is able to send from 4 to 34 bits MSC data frames (4 to 32 serialized data bits and up to 2 Data Selection zero bits). The serialized data bits source can be either:

- the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), written by the host software,
- Parallel Input pin states latched into the DSPI DSI Serialization Data Register (DSPI_SDR).

DSPI_DSICR TXSS bit or DSPI_SSR bits define the source of the data.

The Least Significant Bits of the DSPI_ASDR or DSPI_SDR registers are selected to be serialized if the data frame is set to less than 32 bits.

Figure 545. DSPI usage in the TSB configuration



The PCS signals are driven together with SOUT. The t_{CSC} and t_{ASC} delays are not available. Delay after Transfer (DT) is set in SCK clock periods as a binary number formed by concatenation of the DSPI_CTARn PDT and DT fields plus one, allowing to set DT from 1 to 64 serial clock periods. DT field provides least significant bits and PDT field provides most significant bits of the Delay after Transfer.

Figure 546. TSB Downstream Frames

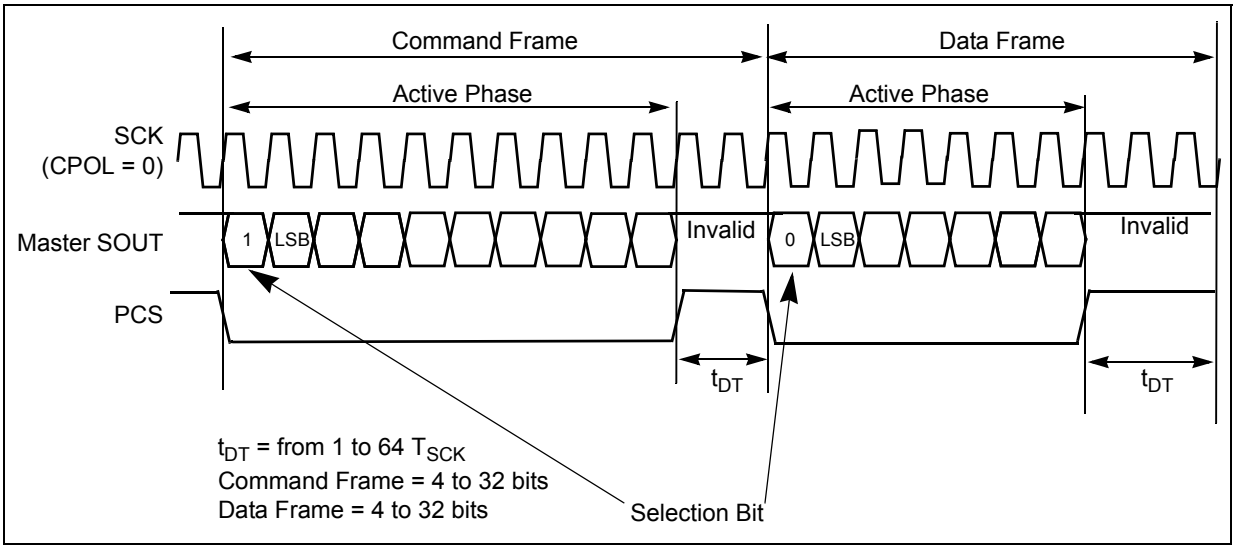


Figure 546 shows the two types of MSC downstream frames: command frame and data frame.

The first transmitted bit, called the selection bit, determines the frame type:

- The selection bit “0” indicates a data frame
- The selection bit “1” indicates a command frame

Data frame may contain up to 2 selection bits to support two external slave devices, (so called dual receiver configuration) or no selection bits at all.

The command frame can be written by software, through SPI TX FIFO, using one or two FIFO entries with help of the CONT bit. The data frame consists of up to 32 bits from the

SDR or ASDR registers and up to two zero selection bits. Number of data bits in the data frame is defined by the DSCICR1[TSBCNT] field.

The selection bit of the MSC command frames (1) can be implemented by software.

The selection bits in the data frames are enabled by DSPI_DSICR1 DSE0 and DSE1 bits. Each DSEn bit set increases the data frame size by one bit.

To comply with MSC specification, set DSPI_CTARn[LSBFE] to transmit least significant bit first.

Regardless the LSBFE bit setting, the Data Frame Selection Bits, if enabled, are always transmitted first, before corresponded data subframes.

25.4.9.1 MSC Dual Receiver Support with PCS Switchover

When in TSB mode it is possible to switch the set of PCS signals that are driven during the first part of the frame to a different set of PCS signals during the second part of the frame. The bit, at which this switchover occurs, is defined by FMSZ field of the DSPI_CTARn register, selected by DSICTAS field of the DSICR.

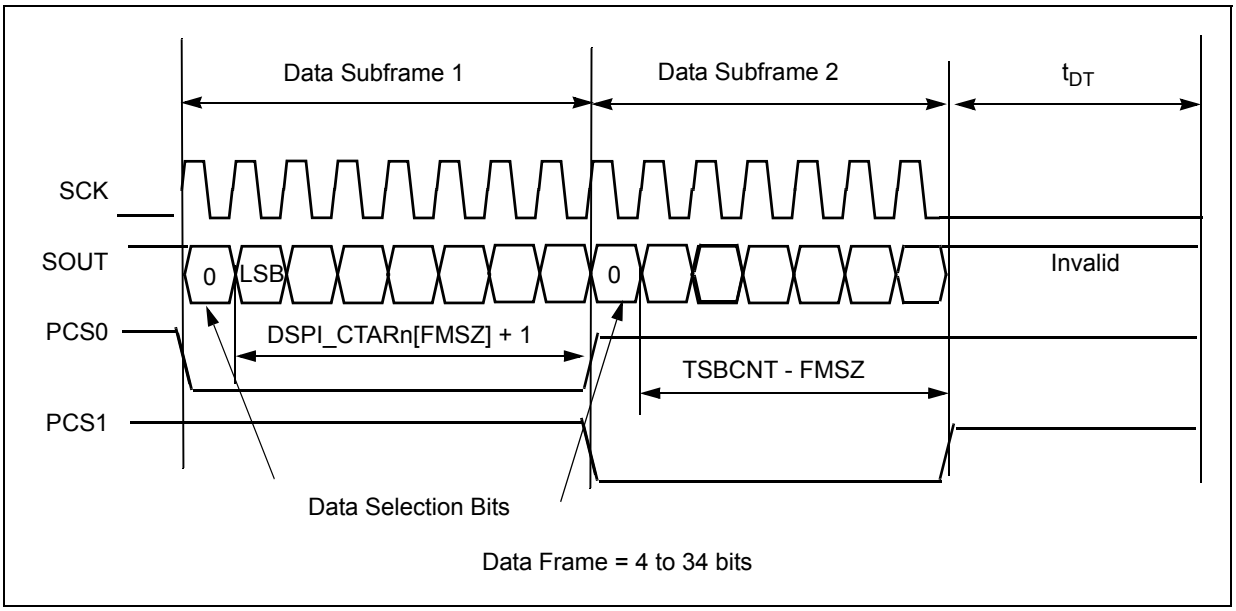
Number of the bits, not including the Data Selection Bit, in the first part of the frame is equal to value of the FMSZ field plus one. During this part of the frame the PCS signal levels are controlled by DSPI_DSICR DPCSn bits, after that by DSPI_DSICR1 DPCS1_n bits.

The PCS switchover occurs at driving edge of the SCK clock output.

The second Data Selection Bit is inserted after the PCS switchover if enabled.

Data Frame with PCS switchover is shown in *Figure 547*.

Figure 547. TSB Data Frame Format for MSC Dual Receiver Operation



25.4.10 Interleaved TSB (ITSB) Mode

This mode is similar to the TSB is all aspects except for the differences mentioned in *Table 359*. In the Interleaved TSB or ITSB mode, frame transmission is governed by the following rules:

- Transmission of frames is done on a periodic trigger that can be generated internal to the DSPI or externally. The trigger period is calculated as follows:
  - Trigger Period = 32 bits (max. data frame size) + 2 selections (max. bits in data frame) + t_{PT}

    t_{PT} is the passive frame time or dead time inserted to have constant trigger period
  - The user software shall configure the trigger period and trigger source.
- If the previously transmitted frame **was** a SPI frame **or** the TX_FIFO **is** empty, the frame transmitted on subsequent trigger will always be a DSI frame.
- If the previously transmitted frame **was not** a SPI frame **and** the TX_FIFO **is not** empty, the frame transmitted on subsequent trigger will always be a SPI frame.

This mode does not change the way DSI & SPI frame transmissions are done except that ITSB mode interleaves the source (SPI or DSI) in such a way to prevent back to back SPI frames. All features of SPI & DSI mode can be used except for the Continuous Frame Selection format (i.e. CONT bit in DSPI_PUSHR and DCONT bit in DSPI_DSICR cannot be asserted).

25.4.11 Differences between TSB and ITSB modes

Table 359 lists the differences between the TSB and ITSB modes.

Table 359. Differences between TSB and ITSB modes

Mode feature	Mode	
	TSB	ITSB
Priority	Priority available to frames from SPI. Back to back SPI frames are sent out.	No priority. Messages are interleaved such that there are no back-to-back SPI frames
Selection bit	Selection bit inserted for DSI frames only. Selection bit for SPI frames is inserted by software	Selection bit for both DSI & SPI frames inserted automatically. Encoding of selection bit same as TSB mode (0 = DSI; 1 = SPI)
Triggered transmission	Occasionally triggered transmission i.e. when trigger is received via hardware trigger input	Periodically triggered transmission i.e. frames transmission continuously on every trigger.
Frame gap	Gap between frames is configurable. Time between frames is t_{DT} which is configurable from 1 to 64 baud rate clock cycles	Gap between frames is inserted automatically as the whole operation is based on a periodic trigger. If a frame has less than 32-bits to be transmitted, dead or passive time is automatically inserted by waiting for the next trigger pulse. Time t_{DT} is not used in this mode as it can be made part of the trigger period (as passive time).

However, in both TSB and ITSB modes, separate frame completion interrupts will be available to indicate frame transmission completion from SPI and DSI. MSC dual receiver support with PCS switchover is also supported in ITSB mode. See *Section 25.4.9.1: MSC Dual Receiver Support with PCS Switchover* for details on this feature.

The ITSB mode is suitable to implement the Downstream Channel for the MSC Bus because of the above features.

25.4.11.1 Configuring DSPI for ITSB mode

To initialize the DSPI for the ITSB mode of operation, it is recommended to proceed as follows:

1. The DSPI should be put in HALT mode (by asserting the DSPI_MCR[HALT] bit) before programming the registers.
2. Set the TSB, ITSB bits in DSPI_DSICR. The TRRE and CID bits in this register should be cleared. DCONT should not use used in ITSB mode.
3. Depending on the source of trigger, set the values in the DSPI_DSICR[TRG] bit. Value = 0 implies internal trigger for which the trigger period is programmed in DSPI_DSICR1[TRG_PRD] field. Value = 1 implies external trigger for which is generated outside DSPI.
4. Enable frame completion interrupts for either any frame completion (i.e. TCF) or separate frame complete (i.e. DSITCF and SPITCF) as required by application
5. When remainder configuration of DSPI is complete, clear the HALT bit in DSPI_MCR to allow DSPI to start operations.

Once configured and DSPI is brought out of HALT, the ITSB mode shall start sequencing the frames from either DSI or SPI as per the rules mentioned in *Section 25.4.10: Interleaved TSB (ITSB) Mode*.

25.4.11.2 Using ITSB mode for MSC Downstream Channel

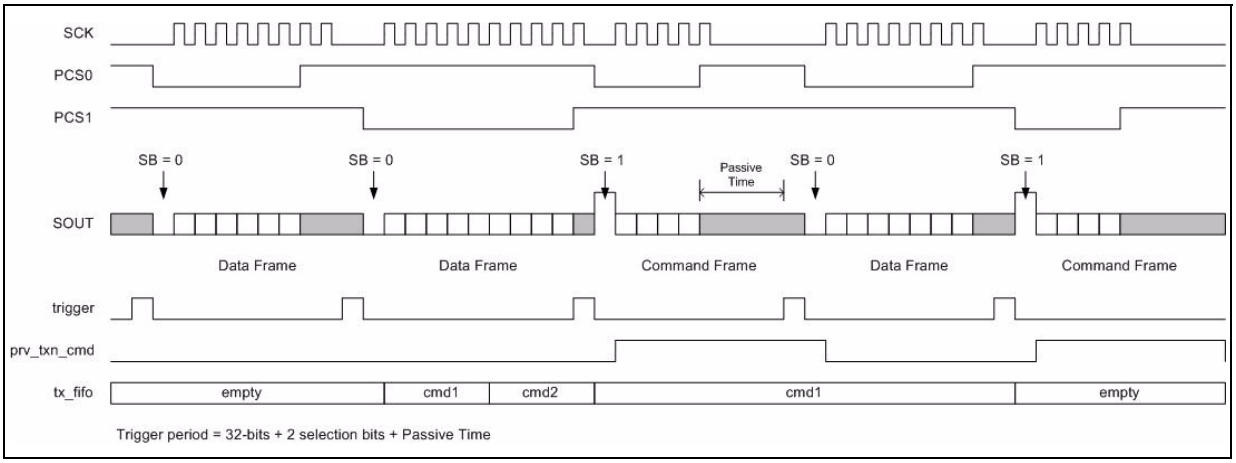
MSC Downstream Channel use two types of frames — command frame and data frames — and the operation requires that:

- Data frames can be sent back to back
- Back to back command frames are not a legal scenario
- If command frame has less bits than data frame, a gap is inserted to ensure next data frame starts at known reference time
- In dual MSC ASIC, some bits of one 32-bit data frame goes to first ASIC and remainder bits to second ASIC
- In dual MSC ASIC, if a command frame is sent to one ASIC, it must not change the timing of the next data frame

The ITSB mode can help implement the MSC Downstream Channel by allowing the command frames to be sent via the SPI interface and data frames to be sent via the DSI interface. Once the DSPI has been configured to work in ITSB mode (as explained in *Section 25.4.11.1: Configuring DSPI for ITSB mode*), the user software can program the DSI & SPI sides to send data and command frames respectively. The programming of frames for transmission via DSI or SPI is same as explained in *Section 25.4.4: Combined Serial Interface (CSI) Configuration*.

Figure 548 shows a sample MSC Downstream transmission using the ITSB mode.

Figure 548. Sample MSC downstream transmission using ITSB mode



Brief description of operation:

- At the first trigger, since there are no command frames, a data frame is transmitted
  - To send a command frame at the start of operation, the user software must program a command frame in the TX_FIFO after enabling DSPI (i.e. HALT = 0) and before the first trigger period expires.
- The second trigger sets off the transmission of another data frame.
- Between second and third trigger, two command frames are pushed into the Tx FIFO.
- At third trigger, a command frame is transmitted as the TX_FIFO is not empty.
  - Since only 16 bits or less are transmitted, no transmission happens till next trigger. This is marked as the “Passive Time”. The passive time is added in all frames where less than 32 bits are transmitted.
- At fourth trigger, since the previous transmission was a command frame, a data frame is transmitted.
- At fifth trigger, since one command frame is pending in Tx FIFO and last transmitted frame was not a command frame, a command frame is transmitted.
- At sixth trigger (not shown), the process continues.

25.4.12 Parity Generation and Check

The DSPI module can generate and check parity in the serial frame. The parity bit replaces the last transmitted bit in the frame. The parity is calculated for all transmitted data bits in frame, not including the last, would be transmitted, data bit. The parity generation/control is done on frame basis. The registers fields, setting frame size defines the total number of bits in the frame, including the parity bit. Thus, to transmit/receive the same number of data bits with parity check, increase the frame size by one versus the same data size frame without the parity check.

Parity can be selected as odd or even. Parity Errors in the received frame set Parity Error flags in the Status register. The Parity Error Interrupt Requests are generated if enabled. The DSPI module can be programmed to stop SPI or/and DSI frame transmission in case of a frame reception with parity error.

25.4.12.1 Parity for SPI Frames

When the DSPI is in the master mode the parity generation is controlled by PE and PP bits of the TX FIFO entries (DSPI_PUSHR). Setting the PE bit enables parity generation for transmitted SPI frames and parity check for received frames. PP bit defines polarity of the parity bit.

When continuous PCS selection is used to transmit SPI data, two parity generation scenarios are available:

- Generate/check parity for the whole frame
- Generate/check parity for each subframe separately.

To generate/check parity for the whole frame set PE bit only in the last command/TX FIFO entry, forming this frame (with the DSPI_PUSHR).

To generate/check parity for each subframe set PE bit in each command/TX FIFO entry, forming this frame.

If the parity error occurs for received SPI frame, the DSPI_SR[SPEF] bit is set. If DSPI_MCR[PES] bit is set, the DSPI stops SPI frames transmission. To resume SPI operation clear the DSPI_SR[SPEF] or the DSPI_MCR[PES] bits.

In slave mode the parity is controlled by the PE and PP bits of the DSPI_CTAR0 register similar to the master mode parity generation without continuous PCS selection.

25.4.12.2 Parity for DSI Frames

Parity generation is controlled by PE and PP bits of the DSPI_DSICR similar to the SPI frames. The parity is calculated and checked for each DSI frame. (DSPI_DSICR[DCONT] bit has no effect on parity generation.)

If the parity error occurs for received DSI frame, the DSPI_SR[DPEF] bit is set. If DSPI_DSICR[PES] bit is set, the DSPI stops DSI frames transmission. To resume DSI operation clear the DSPI_SR[DPEF] or the DSPI_DSICR[PES] bits.

25.4.13 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 360* lists these conditions.

Table 360. Interrupt and DMA Request Conditions

Condition	Flag	Interrupt	DMA
End of Queue (EOQ)	EOQF	X	
TX FIFO Fill	TFFF	X	X
Transfer Complete	TCF	X	
TX FIFO Underflow	TFUF	X	
RX FIFO Drain	RFDF	X	X
RX FIFO Overflow	RFOF	X	
SPI Parity Error	SPEF	X	
DSI Parity Error	DPEF	X	
DSI Deserialized Data Match	DDIF	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.

25.4.13.1 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.

25.4.13.2 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.

25.4.13.3 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.

25.4.13.4 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.

25.4.13.5 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.

25.4.13.6 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.

25.4.13.7 SPI Frame Parity Error Interrupt Request

The SPI Frame Parity Error Flag indicates that a SPI frame with parity error had been received. The SPEF_RE bit in the DSPI_RSER must be set for the interrupt request to be generated.

25.4.13.8 DSI Frame Parity Error Interrupt Request

The DSI Frame Parity Error Flag indicates that a DSI frame with parity error has been received. The DPEF_RE bit in the DSPI_RSER must be set for the interrupt request to be generated.

25.4.13.9 Deserialized Data Match Interrupt Request

The Deserialized Data Match Flag (DDIF) indicates that a DSI frame with data matches DSPI_DPIR data, masked with DSPI_DIMR, had been received. The DDIF_RE bit in the DSPI_RSER must be set for the interrupt request to be generated.

25.4.14 Power saving features

The DSPI supports two power-saving strategies:

- External Stop mode
- Module Disable mode - Clock gating of non-memory mapped logic

25.4.14.1 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.

25.4.14.2 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. The module disable mode can also be initiated by hardware. A power management block can initiate the module disable mode by asserting the DOZE mode signal while the DOZE bit in the DSPI_MCR is set.

When the MDIS bit is set or the DOZE mode signal is asserted while the DOZE 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.

25.5 Initialization/Application information

25.5.1 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.

25.5.2 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].

25.5.3 Baud rate settings

Table 361 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 361. 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

25.5.4 Delay settings

Table 362 shows the values for the Delay after Transfer (t_{DT}) and CS to SCK Delay (t_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI_CTAR registers. The values calculated assume a 100 MHz system frequency.

This table does not apply for TSB Continuous mode.

Table 362. 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

25.5.5 DSPI Compatibility with the QSPI of the MPC500 MCUs

Table 362 shows the translation of commands written to the TX FIFO command halfword with commands written to the Command Ram of the MPC500 family MCUs QSPI. The table illustrates how to configure the DSPI_CTAR registers to match the default cases for the possible combinations of the MPC500 Family QSPI Control Bits in its Command RAM. The defaults for QSPI are based on a system clock of 40MHz. All delay variables below will generate the same delay, or as close a possible, from the DSPI 100MHz system clock that an QSPI would generate from its 40MHz system clock. For other system clock frequencies, the customer can recompute the values using *Section 25.5.4: Delay settings*.

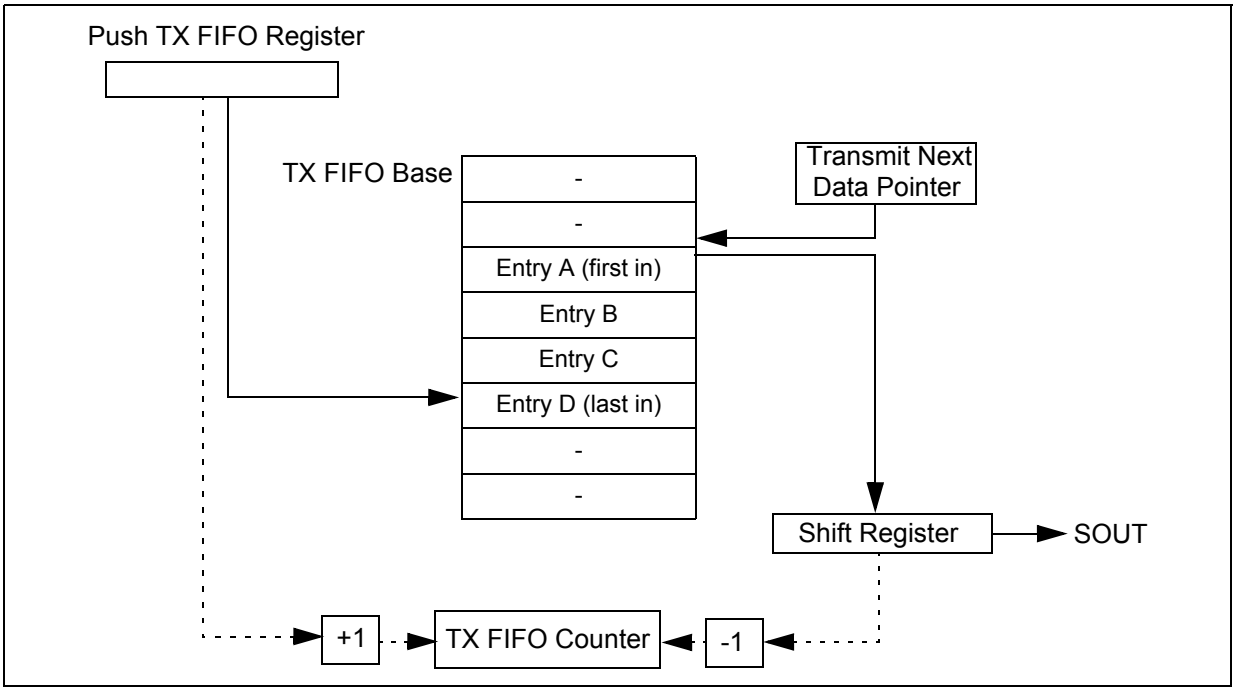
- For BITSE = 0 → 8 bits per transfer
- For DT = 0 → 0.425 µs delay: For this value, the closest value in the DSPI is 0.480 µs
- For DSCK = 0 → 1/2 SCK period: For this value, the value for the DSPI is 20 ns

25.5.6 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 549* 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 25.4.2.4: Transmit First In First Out (TX FIFO) Buffering Mechanism and *Section 25.4.2.5: Receive First In First Out (RX FIFO) Buffering Mechanism* for details on the FIFO operation.

Figure 549. TX FIFO Pointers and Counter



25.5.6.1 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 19

$$\text{First-in Entry Address} = \text{TX FIFO Base} + (4 \times \text{TXNXTPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

Equation 20

$$\text{Last-in Entry address} = \text{TX FIFO Base} + 4 \times (\text{TXCTR} + \text{TXNXTPTR} - 1) \bmod (\text{TXFIFOdepth})$$

TX FIFO Base - Base address of TX FIFO
TXCTR - TX FIFO Counter
TXNXTPTR - Transmit Next Pointer
TX FIFO Depth - Transmit FIFO depth, implementation specific

25.5.6.2 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 21

$$\text{First-in Entry Address} = \text{RX FIFO Base} + (4 \times \text{POPNXTPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

Equation 22

$$\text{Last-in Entry address} = \text{RX FIFO Base} + 4 \times (\text{RXCTR} + \text{POPNXTPTR} - 1) \bmod (\text{RXFIFOdepth})$$

RX FIFO Base - Base address of RX FIFO

RXCTR - RX FIFO counter

POPNXTPTR - Pop Next Pointer

RX FIFO Depth - Receive FIFO depth, implementation specific

26 Enhanced Serial Communication Interface (FMPLL)

26.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

26.1.1 Device-specific features

Members of the SPC563Mxx family of devices have two Enhanced Serial Communication Interface (eSCI) blocks:

- eSCI_A base address is 0xFFFB_0000
- eSCI_B base address is 0xFFFB_4000

Note: The above base addresses are referenced as ESCI_BASE in the register documentation in this chapter.

The eSCI signals on SPC563Mxx devices follow a different naming convention than the signals names used in this chapter.

Table 363. SPC563Mxx eSCI signal names

Signal name	Description
SCI_A_RX	eSCI A Receive
SCI_A_TX	eSCI A Transmit
SCI_B_RX	eSCI B Receive
SCI_B_TX	eSCI B Transmit

26.2 Introduction

The FMPLL block is an enhanced SCI block with a LIN interface layer and DMA support.

26.2.1 Bibliography

- LIN Specification Package Revision 1.3; December 12, 2002
- LIN Specification Package Revision 2.0; September 23, 2003

26.2.2 Acronyms and abbreviations

Table 364 contains acronyms and abbreviations used in this document.

Table 364. Acronyms and abbreviations

Term	Description
eSCI	Enhanced SCI block with LIN support and DMA support
SCI	Serial Communications Interface
LIN	Local Interconnect Network - A protocol for low-cost automobile networks
LIN FSM	LIN Finite State Machine - The control logic of the LIN hardware

Table 364. Acronyms and abbreviations (continued)

Term	Description
MCLK	Module Clock, defined in *Section 26.5.3.1: Module clock*
TCLK	Transmitter Clock, defined in *Section 26.5.3.2: Transmitter clock*
RCLK	Receiver Clock, defined in *Section 26.5.3.3: Receiver clock*
RSC	Receiver Sample Counter, defined in *Section 26.5.3.3: Receiver clock*

26.2.3 Glossary

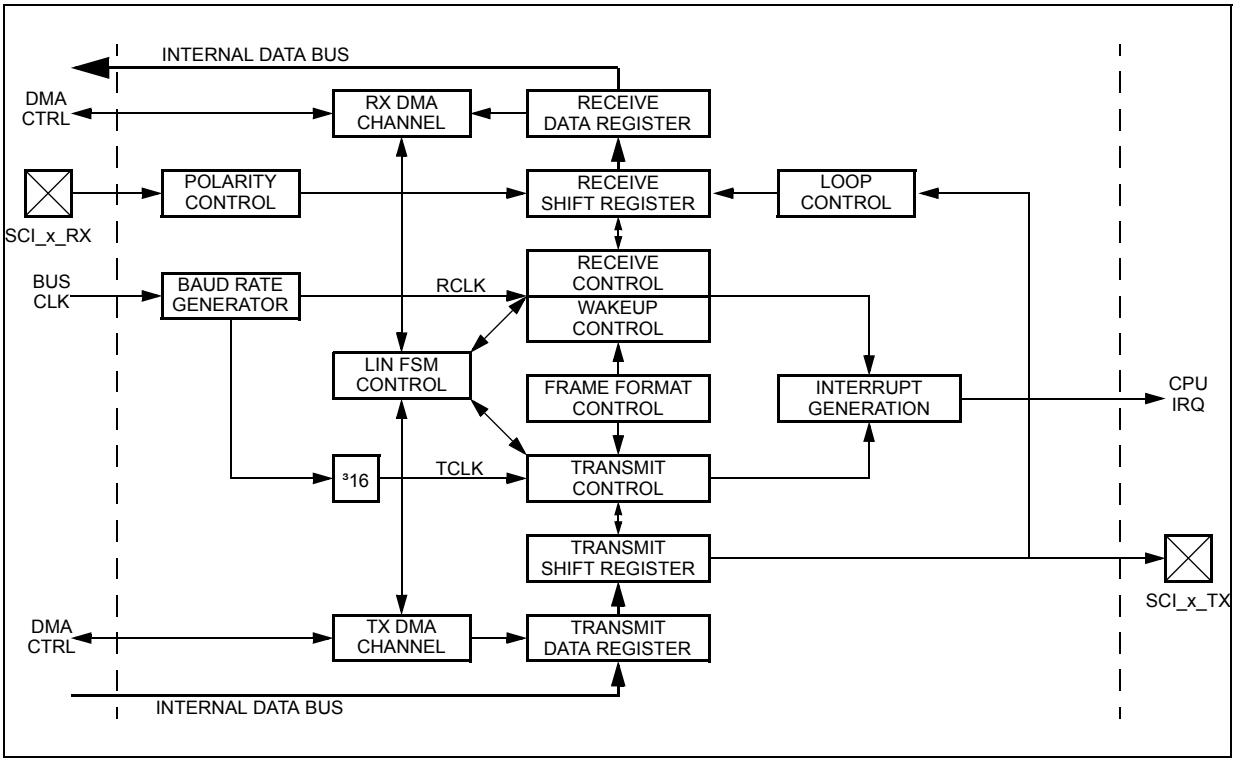
Table 365. Glossary

Term	Definition
Logic level one	The voltage that corresponds to Boolean true (1) state.
Logic level zero	The voltage that corresponds to Boolean false (0) state.
Set	To set a bit or bits means to establish logic level one on the bit or bits.
Clear	To clear a bit or bits means to establish logic level zero on the bit or bits.
Asserted	A signal that is asserted is in its active state. An active low signal changes from logic level one to logic level zero when asserted, and an active high signal changes from logic level zero to logic level one.
Preamble	The term preamble describes an idle character which is *transmitted* by the eSCI module.
Bit time	Duration of a single bit in a transmitted byte field or character, equivalent to the duration of one transmitter clock cycle defined in *Section 26.5.3.2: Transmitter clock*
frame	Entity that consists of the start bit followed by payload bits followed by one ore more stop bits
LIN byte field	Special instance of a frame
SCI frame	Special instance of a frame
LIN frame	Sequence of LIN byte fields

26.2.4 Overview

The eSCI block allows asynchronous serial communications with peripheral devices and other CPUs. It includes special support to interface to LIN slave devices.

Figure 550. eSCI block diagram



26.2.5 Features

The eSCI block includes these distinctive features:

- Full-duplex operation
- Standard mark/space non-return-to-zero (NRZ) format
- 13-bit baud rate selection
- Programmable frame, payload, and character format
- Support of 2 stop bits in receiver path
- Hardware parity generation and checking
  - Programmable even or odd parity
- Programmable polarity of SCI_x_RX pin
- Separately enabled transmitter and receiver
- 2 receiver wake-up methods:
  - Idle line wake-up
  - Address mark wake-up
- Interrupt-driven operation with 8 flags:
  - Transmitter empty
  - Transmission complete
  - Receiver full
  - Idle receiver input
  - Receiver overrun
  - Noise error
  - Framing error
  - Parity error
- Receiver framing error detection
- 1/16 bit-time noise detection
- 2-channel DMA interface
- LIN support
  - LIN Master Node functionality (master and slave task)
  - Compatible with LIN slaves from revisions 1.x and 2.0 of the LIN standard
  - Detection of Bit Errors, Physical Bus Errors and Checksum Errors
  - All status bit can generate maskable interrupts
  - Application layer CRC support
  - Programmable CRC polynom
  - Double Stop Flag insertion after Bit Errors
  - Detection and generation of wakeup characters
  - Programmable wakeup delimiter time
  - Programmable slave timeout
  - Configurable to include header bits in checksum
  - LIN DMA interface

26.2.6 Modes of operation

This section describes the basic operational power modes of the eSCI module.

26.2.6.1 Disabled mode

The Module Disable Bit MDIS in the SCI Control Register 2 can be used to turn off the eSCI. This will prevent the eSCI core to be clocked, and thus save power.

26.2.6.2 Run mode

In this mode, the eSCI is fully operational. The eSCI has two major run modes, the SCI mode and the LIN mode. The availability of certain register bits and fields depends on the selected major run mode.

26.3 External signal description

The eSCI module is connected to two external pins.

26.3.1 Detailed signal descriptions

26.3.1.1 eSCI transmit pin (SCI_x_TX)

This pin serves as transmit data output of eSCIx.

26.3.1.2 eSCI receive pin (SCI_x_RX)

This pin serves as receive data input of eSCIx.

26.4 Memory map and register definition

This section provides the memory map and a detailed description of the memory mapped registers.

26.4.1 Memory map

Table 366. Block memory map

Offset	Register	Access	Location
0x00	SCI control register 1 (SCI_CR1)	Read/Write	*on page 941*
0x04	SCI control register 2 (SCI_CR2)	Read/Write	*on page 944*
0x06	*SCI data register (SCI_DR)*	Read/Write	*on page 946*
0x08	*SR register (SCI_SR)*	Read/Write	*on page 947*
0x0C	*LIN control register (SCI_LCR)*	Write	*on page 949*
0x10	*LIN transmit register (SCI_LTR)*	Write	*on page 951*
0x14	*LIN receive register (SCI_LRR)*	Read	*on page 953*
0x18	*LIN polynomial register (SCI_LPR)*	Read/Write	*on page 954*

Table 367 provides a key for register figures and tables.

Table 367. Register conventions

Convention	Description
	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
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 values	
0	Resets to zero
1	Resets to one

26.4.2 Register descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Writes to a reserved register location do not have any effect and reads of these locations return a zero. Details of register bit and field function follow the register diagrams, in bit order.

26.4.2.1 SCI control register 1 (SCI_CR1)

This register provides baud rate control, interrupt enable bits and control bits for the transmitter and receiver.

The baud rate and clock generation is detailed in *Section 26.5.3: Baud rate and clock generation*.

Register address: ESCI Base + 0x0000

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LOOPS	0	RSRC	M	WAKE	ILT	PE	PT	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 551. SCI control register 1 (SCI_CR1)

Table 368. SCI_CR1 field descriptions

Field	Description
0–3	Reserved
3–15 SBR	SCI baud rate control value See *Section 26.5.3: Baud rate and clock generation* for details.
16 LOOPS	Loop Mode Select This control bit, together with the RSRC control bit, defines the receiver source mode. The mode coding is defined in *Table 369* and the modes are described in *Section 26.5.5.3.2: Receiver input mode selection*.
17	Reserved
18 RSRC	Receiver Source Control This control bit, together with the LOOPS control bit, defines the receiver source mode. The mode coding is defined in *Table 369* and the modes are described in *Section 26.5.5.3.2: Receiver input mode selection*.
19 M	Frame Format Mode This control bit, together with the M2 bit of the SCI_LPR register, controls the frame format used. The supported frame formats and the related settings are defines in *Section 26.5.2: Frame formats*.
20 WAKE	Receiver Wake-up Condition This control bit defines the wake-up condition for the receiver. The receiver wake-up is described in *Section 26.5.5.5: Multiprocessor communication*. 0 Idle line wake-up. 1 Address mark wake-up
21 ILT	Idle Line Type This control bit defines the type of idle line detection for the receiver wake-up. The two types are described in *Section 26.5.5.5.1: Idle-Line wakeup*. 0 Idle line detection starts after reception of a low bit. 1 Idle line detection starts after reception of the last stop bit.
22 PE	Parity Enable This control bit enables the parity bit generation and checking. The location of the parity bits is shown in *Section 26.5.2: Frame formats*. 0 Parity bit generation and checking disabled. 1 Parity bit generation and checking enabled.
23 PT	Parity Type This control bit defines whether even or odd parity has to be used. 0 Even parity (even number of ones in character clears the parity bit). 1 Odd parity (odd number of ones in character clears the parity bit).
24 TIE	Transmitter Interrupt Enable This bit controls the SIU_SR[TDRE] interrupt request generation. 0 TDRE interrupt request generation disabled. 1 TDRE interrupt request generation enabled.
25 TCIE	Transmission Complete Interrupt Enable This bit controls the SCI_SR[TC] interrupt request generation. 0 TC interrupt request generation disabled. 1 TC interrupt request generation enabled.

Table 368. SCI_CR1 field descriptions (continued)

Field	Description
26 RIE	Receiver Full Interrupt Enable This bit controls the SIU_SR[RDRF] interrupt request generation. 0 RDRF interrupt request generation disabled. 1 RDRF interrupt request generation enabled.
27 ILIE	Idle Line Interrupt Enable This bit controls the SIU_SR[IDLE] interrupt request generation. 0 IDLE interrupt request generation disabled. 1 IDLE interrupt request generation enabled.
28 TE	Transmitter Enable This control bit enables and disables the transmitter. The control features of the transmitter are described in *Section 26.5.5.2.1: Transmitter states and transitions*. 0 Transmitter disabled. 1 Transmitter enabled.
29 RE	Receiver Enable This control bit enables and disables the receiver. The control features of the receiver are described in *Section 26.5.5.3.1: Receiver states and transitions*. 0 Receiver disabled. 1 Receiver enabled.
30 RWU	Receiver Wake-Up Mode This bit controls and indicates the receiver wake-up mode, which is described in *Section 26.5.5.5: Multiprocessor communication*. 0 Normal receiver operation. 1 Receiver is in wake-up mode. **Note:** This bit should be set in SCI mode only.
31 SBK	Send Break Character This bit controls the transmission of break characters, which is described in *Section 26.5.5.2.7: Break character transmission*. 0 No break characters will be transmitted. 1 Break characters will be transmitted. **Note:** This bit should be set in SCI mode only.

Table 369. Receive source mode selection

LOOPS	RSCR	Receiver input mode
0	0	Dual Wire Mode
0	1	Reserved
1	0	Loop Mode
1	1	Single Wire Mode

26.4.2.2 SCI control register 2 (SCI_CR2)

Register address: ESCI Base + 0x0004

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MDIS	FBR	BSTP	IEBERR	RXDMA	TXDMA	BRK13	TXDIR	BESM13	SBSTP	RXPOL	PMSK	ORIE	NFIE	FEIE	PFIE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 552. SCI control register 2 (SCI_CR2)

Table 370. SCI_CR2 field descriptions

Field	Description
0 MDIS	Module Disabled Mode This bit controls the Module Mode of Operation, which is described in *Section 26.2.6: Modes of operation*. 0 Module is not in Disabled Mode. 1 Module is in Disabled Mode.
1 FBR	Fast Bit Error Detection This bit controls the Bit Error Detection mode. 0 Standard Bit error detection performed as described in *Section 26.5.6.5.3: Standard bit error detection*. 1 Fast Bit error detection performed as described in *Section 26.5.6.5.4: Fast bit error detection*. **Note:** This bit is used in LIN mode only.
2 BSTP	Bit Error or Physical Bus Error Stop This bit controls the transmit DMA requests generation in case of bit errors or physical bus errors. Bit errors are indicated by the BERR flag in the SR register (SCI_SR) and physical bus errors are indicated by the PBERR flag in the SR register (SCI_SR). 0 Transmit DMA requests generated regardless of bit errors or physical bus errors. 1 Transmit DMA requests are *not* generated if SCI_SR[BERR] flag or SCI_SR[PBERR] flag are set. **Note:** This bit is used in LIN mode only.
3 IEBERR	Bit Error Interrupt Enable This bit controls the BERR interrupt request generation. 0 BERR interrupt request generation disabled. 1 BERR interrupt request generation enabled.
4 RXDMA	Receive DMA Control This bit enables the receive DMA feature. 0 Receive DMA disabled. 1 Receive DMA enabled.
5 TXDMA	Transmit DMA Control This bit enables the transmit DMA feature. 0 Transmit DMA disabled. 1 Transmit DMA enabled.
6 BRK13	Break Character Length This bit is used to define the length of the break character to be transmitted. The settings are specified in *Section 26.5.2.2: Break character formats*.

Table 370. SCI_CR2 field descriptions (continued)

Field	Description
7 TXDIR	SCI_x_TX pin output enable This bit determines whether the SCI_x_TX pin is used as an output. 0 SCI_x_TX pin is not used as output. 1 SCI_x_TX pin is used as output. **Note:** This bit is used in Single Wire Mode only.
8 BESM13	Fast Bit Error Detection Sample Mode This bit defines the sample point for the Fast Bit Error Detection Mode. 0 Sample point is RS9. 1 Sample point is RS13. **Note:** This bit is used in LIN mode only.
9 SBSTP	Bit Error Stop Stops the SCI when a Bit Error is asserted. This allows to stop driving the LIN bus quickly after a Bit Error has been detected. The SCI won't start a new byte transmission until the time for the current byte has expired. 0 Transmission is not stopped on bit error. 1 Transmission is stopped on bit error. **Note:** This bit is used in LIN mode only.
10 RXPOL	SCI_x_RX Pin polarity This bit controls the polarity of the SCI_x_RX pin. See *Section 26.5.2.1.1: Inverted data frame formats*. 0 Normal Polarity. 1 Inverted Polarity.
11 PMSK	Parity Bit Masking This bit defines whether the received parity bit is presented in the related bit position in the SCI data register (SCI_DR). 0 The received parity bit is presented in the bit position related to the parity bit. 1 The value 0 is presented in the bit position related to the parity bit.
12 ORIE	Overrun Interrupt Enable This bit controls the SCI_SR[OR] interrupt request generation. 0 OR interrupt request generation disabled. 1 OR interrupt request generation enabled.
13 NFIE	Noise Interrupt Enable This bit controls the SCI_SR[NF] interrupt request generation. 0 NF interrupt request generation disabled. 1 NF interrupt request generation enabled.
14 FEIE	Frame Error Interrupt Enable This bit controls the SCI_SR[FE] interrupt request generation. 0 FE interrupt request generation disabled. 1 FE interrupt request generation enabled.
15 PFIE	Parity Error Interrupt Enable This bit controls the SCI_SR[PF] interrupt request generation. 0 PF interrupt request generation disabled. 1 PF interrupt request generation enabled.

26.4.2.3 SCI data register (SCI_DR)

This register is used to provide transmit data and retrieve received data in SCI mode. In LIN mode any write access to this register is ignored and any read access returns all 0. In case of data transmission this register is used to provide a part of the transmit data. In case of data reception this register provides a part of the received data and related error information.

If an application writes to this register, the internal commit flag iCMT, which is not visible to the application, is set to indicate that the register has been updated and ready to transmit new data.

If the application reads from this register, a signal is send to the internal receiver unit to indicate that the register has been read and is ready to receive new data. The read access does not change the content of the register.

Register address: ESCI Base + 0x0006

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R8	T8	ERR	0	R				D							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 553. SCI data register (SCI_DR)

Table 371. SCI_DR field descriptions

Field	Description
0 R8	Received Most Significant Bit The semantic of this bit depends on the frame format selected by SCI_LPR[M2], SCI_CR1[M], and SCI_CR1[PE]. [M2 = 0, M = 1, PE = 0]: value of received data bit 8 or address bit. [M2 = 0, M = 1, PE = 1]: value of received parity bit if SCI_CR2[PMSK] = 0, 0 otherwise. [M2 = 1, M = 0, PE = 1]: value of received parity bit if SCI_CR2[PMSK] = 0, 0 otherwise. [M2 = 1, M = 1, PE = 1]: value of received parity bit if SCI_CR2[PMSK] = 0, 0 otherwise. It is 0 for all other frame formats.
1 T8	Transmit Most Significant Bit The semantic of this bit depends on the frame format selected by SCI_LPR[M2], SCI_CR1[M], and SCI_CR1[PE]. [M2 = 0, M = 1, PE = 0]: value to be transmitted as data bit 8 or address bit. It is not used for all other frame formats.
2 ERR	Receive Error Bit This bit indicates the occurrence of the errors selected by the SCI_LPR register during the reception of the frame presented in SCI data register (SCI_DR). In case of an overrun error for subsequent frames this bit is set too. 0 None of the selected errors occurred. 1 At least one of the selected errors occurred.
3	Reserved

Table 371. SCI_DR field descriptions (continued)

Field	Description
4–7 R	Received Data Bits[11:8] The semantic of this field depends on the frame format selected by SCI_LPR[M2] and SCI_CR1[M]. [M2 = 1, M = 1]: value of the received data bits 11:8. (Rx = BITx). It is all 0 for all other frame formats.
8–15 D	Received or transmitted data

26.4.2.4 SR register (SCI_SR)

This register provides flags that indicate the occurrence of module events. All flags can be used to generate interrupt requests.

Register address: ESCI Base + 0x0008

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	0	0	0	BERR	0	0	TACT	RAF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c				w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXRDY	TXRDY	LWAKE	STO	PBERR	CERR	CKERR	FRC	0	0	0	0	0	0	UREQ	OVFL
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

= Unimplemented

Figure 554. Status register (SCI_SR)

Table 372. SCI_SR field descriptions

Field	Description
0 TDRE	Transmit Data Register Empty Flag This flag is set when the content of the SCI data register (SCI_DR) was transferred into internal shift register. **Note:** This flag is set in SCI mode only.
1 TC	Transmit Complete Flag This flag is set when a frame, break or idle character transmission has been completed and no data were written into SCI data register (SCI_DR) after the last setting of the TDRE flag and the SBK bit in Synthesizer Control Register (SYNCR) is 0. This flag is set in LIN mode, if the preamble was transmitted after the enabling of the transmitter.
2 RDRF	Receive Data Register Full Flag This flag is set when the payload data of a received frame was transferred into the SCI data register (SCI_DR). **Note:** This flag is set in SCI mode only.

Table 372. SCI_SR field descriptions (continued)

Field	Description
3 IDLE	Idle Line Flag This flag is set when an idle character was detected and the receiver is not in the wakeup state. **Note:** This flag is set in SCI mode only.
4 OR	Overrun Flag This flag is set when an overrun was detected as described in *Section 26.5.5.3.11: Receiver overrun*. **Note:** This flag is set in SCI mode only.
5 NF	Noise Flag This flag is set when the payload data of a received frame was transferred into the SCI data register (SCI_DR) or LIN receive register (SCI_LRR) and the receiver has detected noise during the reception of that frame, as described in *Section 26.5.5.3.13: Bit sampling*.
6 FE	Framing Error Flag This flag is set when the payload data of a received frame was transferred into the SCI data register (SCI_DR) or LIN receive register (SCI_LRR) and the receiver has detected a framing error during the reception of that frame, as described in *Section 26.5.5.3.18: Stop bit verification*.
7 PF	Parity Error Flag This flag is set when the payload data of a received frame was transferred into the SCI data register (SCI_DR) and the receiver has detected a parity error for the character, as described in *Section 26.5.5.4: Reception error reporting*. **Note:** This flag is set in SCI mode only.
8–10	Reserved
11 BERR	Bit Error Flag This flag is set when a bit error was detected as described in *Section 26.5.6.5.3: Standard bit error detection*. **Note:** This flag is set in LIN mode only.
12–13	Reserved
14 TACT	Transmitter Active The status bit is set as long as the transmission of a frame or special character is ongoing. 0 No transmission in progress. 1 Transmission in progress.
15 RAF	Receiver Active The bit will be set 3 receiver clock (RCLK) cycles after the successful qualification of a start bit. This bit will be cleared, when an idle character was detected. 0 No reception in progress. 1 Reception in progress.
16 RXRDY	Receive Data Ready Flag This flag is set when the payload data of a received frame was transferred into the LIN receive register (SCI_LRR).
17 TXRDY	Transmit Data Ready Flag This flag is set when the content of the LIN transmit register (SCI_LTR) was transferred into internal transmit shift register.
18 LWAKE	LIN Wakeup Received Flag This flag is set when a LIN Wakeup character was received, as described in *Section 26.5.6.6: LIN wakeup*.

Table 372. SCI_SR field descriptions (continued)

Field	Description
19 STO	Slave Timeout Flag This flag is set when a slave does not complete a frame within the specified maximum frame length which is defined in *Section 26.5.6.5: LIN error reporting*.
20 PBERR	Physical Bus Error Flag This flag is set when the receiver input remains unchanged for at least 31 RCLK clock cycles after the start of a byte transmission, as described in *Section 26.5.6.5: LIN error reporting*.
21 CERR	CRC Error Flag This flag is set when an incorrect CRC pattern was detected for a received LIN frame.
22 CKERR	Checksum Error Flag This flag is set when a checksum error was detected for a received LIN frame.
23 FRC	Frame Complete Flag This flag is set when a LIN frame was completely transmitted or received.
24–29	Reserved
30 UREQ	Unrequested Data Received Flag This flag is set when unrequested activity has been detected on the LIN bus, as described in *Section 26.5.6.5: LIN error reporting*. 0 No such event. 1 Unrequested Data received.
31 OVFL	Overflow Flag This flag is set when an overflow (described in *Section 26.5.6.5.8: Overflow detection*) was detected. 0 No such event 1 Overflow detected

26.4.2.5 LIN control register (SCI_LCR)

This register provides control bits to control and configure the LIN hardware.

Register address: ESCI Base + 0x000C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	LRES	WU	WUD0	WUD1	LDBG	DSF	PRTY	LIN	RXIE	TXIE	WUIE	STIE	PBIE	CIE	CKIE	FCIE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 555. LIN control register (SCI_LCR)

Table 373. SCI_LCR field descriptions

Field	Description
0 LRES	LIN FSM Resync This bit controls the state of the LIN protocol engine. 0 LIN protocol engine in normal mode. 1 LIN protocol engine hold in initial state.
1 WU	LIN Bus Wake-Up Trigger This bit is used to trigger the generation of a wake-up signal on the LIN bus, as described in *Section 26.5.6.6: LIN wakeup*. 0 Write has no effect. 1 Write triggers the generation of a wakeup signal.
2:3 WUD[0:1]	LIN Bus Wake-Up Delimiter Time This field determines how long the LIN protocol engine waits after the end of the transmitted wakeup signal, before starting the next LIN frame transmission. 00 4 bit times. 01 8 bit times. 10 32 bit times. 11 64 bit times.
4 LDBG	LIN FSM Debug Mode This bit controls the automatic reset of the LIN protocol engine after the detection of an LIN error or LIN Wake-up, as described in *Section 26.5.6.5: LIN error reporting* and *Section 26.5.6.6: LIN wakeup* 0 Automatic reset of LIN FSM enabled. 1 Automatic reset of LIN FSM disabled.
5 DSF	Double Stop Bits This bit controls the transmission of a second stop bit when a bit error has been detected in a byte field. 0 One stop bit transmitted in case of bit error. 1 Two stop bits transmitted in case of bit error.
6 PRTY	Parity Generation Control This bit controls the generation of the two parity bits in the LIN header. 0 Parity bits generation disabled. 1 Parity bits generation enabled.
7 LIN	LIN Mode Control This bit controls whether the device is in SCI or LIN Mode. 0 SCI Mode. 1 LIN Mode.
8 RXIE	Receive Data Ready Interrupt Enable This bit controls the SCI_SR[RXRDY] interrupt request generation. 0 RXRDY interrupt request generation disabled. 1 RXRDY interrupt request generation enabled.
9 TXIE	Transmit Data Ready Interrupt Enable This bit controls the SCI_SR[TXRDY] interrupt request generation. 0 TXRDY interrupt request generation disabled. 1 TXRDY interrupt request generation enabled.

Table 373. SCI_LCR field descriptions (continued)

Field	Description
10 WUIE	LIN Wakeup Received Interrupt Enable This bit controls the SCI_SR[LWAKE] interrupt request generation. 0 LWAKE interrupt request generation disabled. 1 LWAKE interrupt request generation enabled.
11 STIE	Slave Timeout Flag Interrupt Enable This bit controls the SCI_SR[STO] interrupt request generation. 0 STO interrupt request generation disabled. 1 STO interrupt request generation enabled.
12 PBIE	Physical Bus Error Interrupt Enable This bit controls the SCI_SR[PBERR] interrupt request generation. 0 PBERR interrupt request generation disabled. 1 PBERR interrupt request generation enabled.
13 CIE	CRC Error Interrupt Enable This bit controls the SCI_SR[CERR] interrupt request generation. 0 CERR interrupt request generation disabled. 1 CERR interrupt request generation enabled.
14 CKIE	Checksum Error Interrupt Enable This bit controls the SCI_SR[CKERR] interrupt request generation. 0 CKERR interrupt request generation disabled. 1 CKERR interrupt request generation enabled.
15 FCIE	Frame Complete Interrupt Enable This bit controls the SCI_SR[FRC] interrupt request generation. 0 FRC interrupt request generation disabled. 1 FRC interrupt request generation enabled.
16–21	Reserved
22 UQIE	Unrequested Data Received Interrupt Enable This bit controls the SCI_SR[UREQ] interrupt request generation. 0 UREQ interrupt request generation disabled. 1 UREQ interrupt request generation enabled.
23 OFIE	Overflow Interrupt Enable This bit controls the SCI_SR[OVFL] interrupt request generation. 0 OVFL interrupt request generation disabled. 1 OVFL interrupt request generation enabled.
24–31	Reserved

26.4.2.6 LIN transmit register (SCI_LTR)

This register provides control bits to control and configure the LIN hardware.

Register address: ESCI Base + 0x0010

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSM	CSE	CRC	TD	TO				DATA							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 556. LIN transmit register (SCI_LTR)

Table 374. SCI_LTR field descriptions

Field	Description
0–1 P	Identifier Parity This field provides the identifier parity which is used to create the protected identifier if the automatic identifier parity generation is disabled, i.e., the PRTY bit in LIN control register (SCI_LCR) is 0.
2-7 ID	Identifier This field is used for the identifier field in the protected identifier.
8–15 LEN	Frame Length This field defines the number of data bytes to be transmitted or received.
16 CSM	Checksum Model This bit controls the checksum calculation model used. 0 Classic Checksum Model (LIN 1.3). 1 Enhanced Checksum Model (LIN 2.0).
17 CSE	Checksum Enable This bit control the generation and checking of the checksum byte. 0 No generation and checking of checksum byte. 1 Generation and checking of checksum byte.
18 CRC	CRC Enable This bit controls the generation of checking standard or enhanced LIN frames, which are described in *Section 26.5.6.2: LIN frame formats* 0 Standard LIN frame generation and checking. 1 Enhanced LIN frame generation and checking.
19 TD	Transfer Direction This bit control the transfer direction of the data, crc, and checksum byte fields. 0 Data, CRC, and Checksum byte fields received, described in *Section 26.5.6.4: LIN RX frame generation*. 1 Data, CRC, and Checksum byte fields transmitted, described in *Section 26.5.6.3: LIN TX Frame generation*.

Table 374. SCI_LTR field descriptions (continued)

Field	Description
20–23 TO	Timeout Value The content of the field depends on the transfer direction. RX frame: Defines the time available for a complete RX frame transfer, as described in *Section 26.5.6.5.5: Slave timeout detection* TX frame: Must be set to 0.
24–31 DATA	Transmit Data Data bits for transmission.

26.4.2.7 LIN receive register (SCI_LRR)

This register provides control bits to control and configure the LIN hardware.

Register address: ESCI Base + 0x0014

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

= Unimplemented

Figure 557. LIN receive register (SCI_LRR)

Table 375. SCI_LRR field descriptions

Field	Description
0–31 D	Receive Data This field provides the data bytes of received LIN RX frames.

26.4.2.8 LIN polynomial register (SCI_LPR)

Register address: ESCI Base + 0x0018

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	SYNM	ERROE	ERFE	ERPE	M2	0	0	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

Figure 558. LIN polynomial register (SCI_LPR)

Table 376. SCI_LPR field descriptions

Field	Description
0–15 P	Polynomial bit x^n Used to define the LIN polynomial - standard is $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$ (the polynomial used for the CAN protocol).
16-18	Reserved
19 SYNM	Synchronization Mode This bit controls the synchronization mode of the receiver. The synchronization modes are described in *Section 26.5.5.3.14: Bit synchronization*. 0 Synchronization performed at falling start and data bit edge. 1 Synchronization performed at falling start bit edge only.
20 EROE	ERR flag overrun enable 0 SCI_DR[ERR] flag not affected by overrun detection. 1 SCI_DR[ERR] flag is set on overrun detection during frame reception.
21 ERFE	ERR flag frame error enable 0 SCI_DR[ERR] flag not affected by frame error detection. 1 SCI_DR[ERR] flag is set on frame error detection for the data provided in SCI_DR.
22 ERPE	ERR flag parity error enable 0 SCI_DR[ERR] flag not affected by parity error detection. 1 SCI_DR[ERR] flag is set on parity error detection for the data provided in SCI_DR.
23 M2	Frame Format Mode 2 This control bit together with the M bit of the Synthesizer Control Register (SYNCR) controls the frame format used. The supported frame formats and the related settings are defines in *Section 26.5.2: Frame formats*.
24–31	Reserved

26.5 Functional description

This section provides a complete functional description of the eSCI block, detailing the operation of the design from the end user perspective in a number of subsections.

26.5.1 Module control

The operational mode of the module is controlled by the MDIS bit in the SCI control register 2 (SCI_CR2). The module can transmit and receives data when it is enabled, i.e MDIS = 0.

26.5.2 Frame formats

The eSCI module uses the standard NRZ mark/space data format. The eSCI supports three basic frame types: data frames, break characters, and idle characters.

26.5.2.1 Data frame formats

Each data frame contains a character that is surrounded by a start bit, an optional parity or address bit, and one or two stop bits. The supported data frame formats for transmission and reception are specified in *Table 377*. The supported data frame formats for reception only are specified in *Table 378*.

Table 377. Supported data frame formats for RX and TX

Control				Frame content				
SCI_LPR	SCI_CR1			Start bits	Payload bits			Stop bits
M2	M	PE	WAKE		Character bits	Address bits(1)	Parity bits	
LIN byte fields (*Figure 559*)								
0	0	0	0	1	8	0	0	1
SCI frames (8 payload bits) (*Figure 560*)								
0	0	0	0	1	8	0	0	1
0	0	0	1	1	7	1	0	1
0	0	1	0	1	7	0	1	1
SCI frames (9 payload bits) (*Figure 561*)								
0	1	0	0	1	9	0	0	1
0	1	0	1	1	8	1	0	1
0	1	1	0	1	8	0	1	1

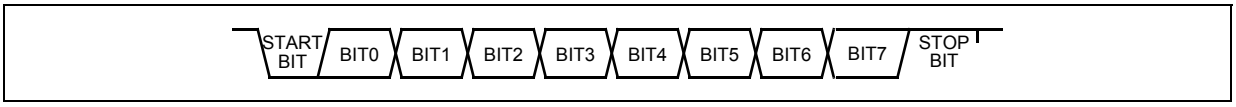
1. The address bit identifies the frame as an address character. See *Section 26.5.5.5: Multiprocessor communication*.

Table 378. Supported data frame formats for RX only

Control				Frame content				
SCI_LPR	SCI_CR1			Start bits	Payload bits			Stop bits
M2	M	PE	WAKE		Character bits	Address bits	Parity bits	
SCI frames (2 stop bits) (see *Figure 562*)								
1	0	1	0	1	8	0	1	2
1	1	1	0	1	12	0	1	2

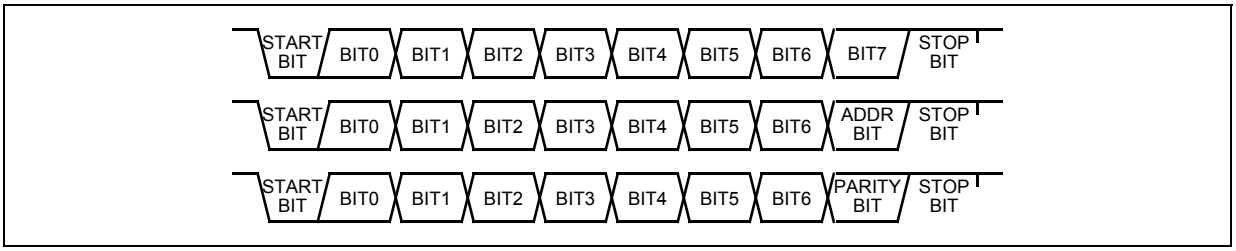
The structure of the LIN frames in normal polarity is shown in *Figure 559*.

Figure 559. LIN byte field format



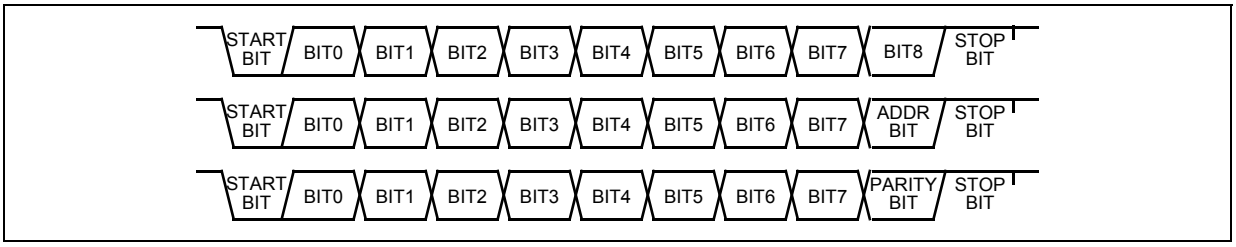
The structures of the supported SCI frame formats with 8 payload bits are shown in *Figure 560*.

Figure 560. SCI frame formats (8 payload bits)



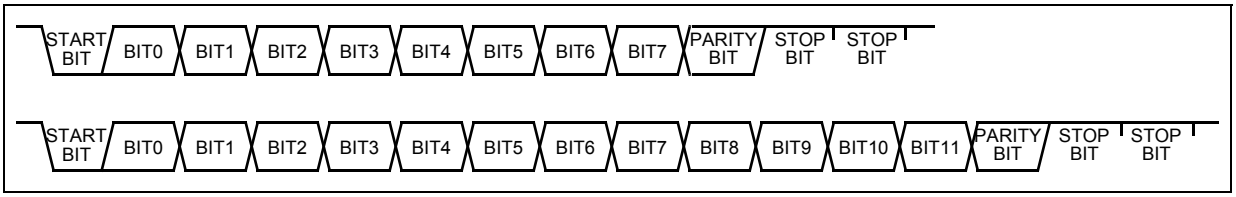
The structures of the supported SCI frame formats with 9 payload bits are shown in *Figure 561*.

Figure 561. SCI frame formats (9 payload bits)



The structures of the supported SCI frame formats with 2 stop bits in normal polarity are shown in *Figure 562*. This frame format is supported for reception only.

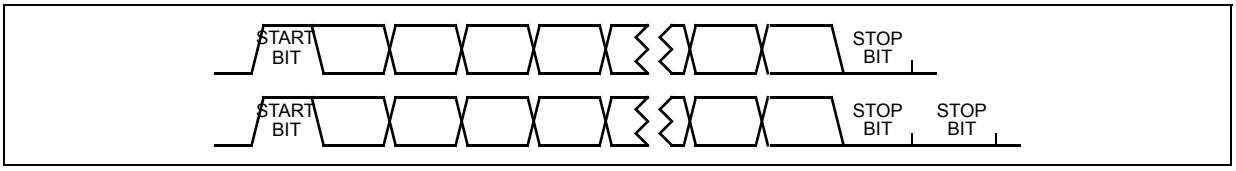
Figure 562. SCI frame formats (2 stop bits)



26.5.2.1.1 Inverted data frame formats

The structures of the supported data frame formats in inverted polarity are shown in *Figure 563*. These frame types are supported for reception only. The polarity of the SCI_x_RX pin is controlled by the RXPOL bit in the SCI control register 2 (SCI_CR2).

Figure 563. Inverted SCI frame formats



26.5.2.2 Break character formats

The supported break character formats are specified in *Table 379*.

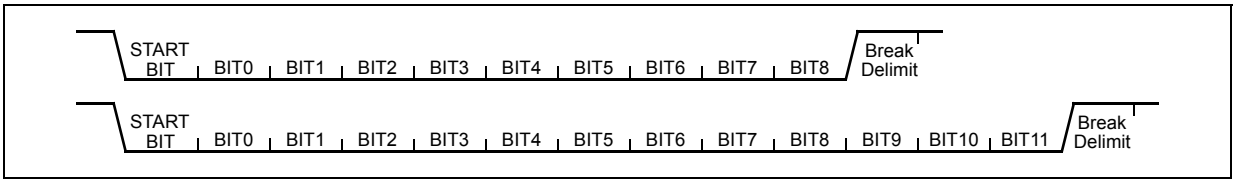
Table 379. Supported break character formats

Control[1]			Break character content		
SCI_LPR	SCI_CR1	SCI_CR2	Start bit	Character bits	Delimit bits
M2	M	BRK13			
LIN break symbol (see *Figure 564*)					
0	0	0	1	9	1
0	0	1	1	12	1
SCI break character (see *Figure 565*)					
0	0	0	1	9	0
0	0	1	1	12	0
0	1	0	1	10	0
0	1	1	1	13	0

1. All codings which are not listed are reserved and must not be used.

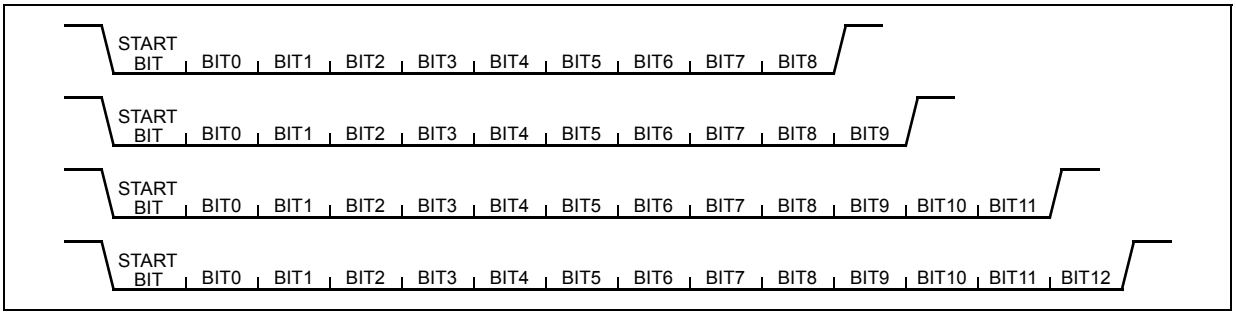
The structure and content of the LIN break symbols is shown in *Figure 564*.

Figure 564. LIN break symbol format



The structure and content of the SCI break characters is shown in *Figure 565*.

Figure 565. SCI break character formats



26.5.2.3 Idle character formats

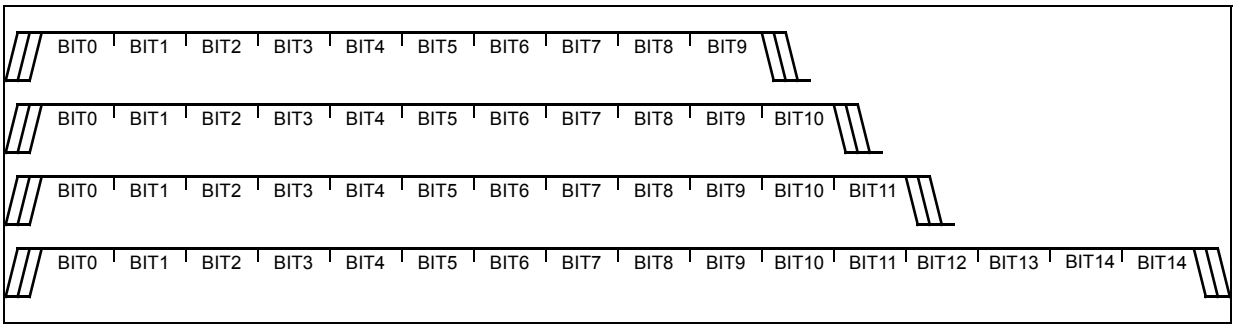
An idle character is a sequence of bits with the value 1. The supported idle character formats are specified in *Table 380*. The preamble has the same structure and content as an idle character.

Table 380. Supported idle character formats

Control		Idle character length
SCI_LPR	SCI_CR1	
M2	M	
SCI idle characters (see *Figure 566*)		
0	0	10
0	1	11
1	0	12
1	1	16

The structure and content of the SCI idle characters is shown in *Figure 566*.

Figure 566. SCI idle character formats



26.5.3 Baud rate and clock generation

A 13-bit modulus counter in the baud rate generator derives the baud rate for both the receiver and the transmitter. The value written to the SBR field in the Synthesizer Control Register (SYNCR) determines the module clock divisor. The baud rate clock is synchronized with the bus clock and drives the receiver. The baud rate clock divided by 16 drives the transmitter. The receiver has an acquisition rate of 16 samples per bit time.

The baud rate generator is enabled when the TE bit or RE bit in the Synthesizer Control Register (SYNCR) is set to 1 for the first time. The baud rate generator is disabled when SBR = 0.

Baud rate generation is subject to one source of error:

- Integer division of the module clock may not give the exact required target baud rate.

Figure 381 lists some examples of achieving target baud rates with a module clock frequency of MCLK = 10.2 MHz.

Table 381. Baud rates error example (MCLK = 10.2 MHz)

SBR[12:0]	RCLK (Hz)	TCLK (Hz)	Target baud rate	Error (%)
17	600,000.0	37,500.0	38400	2.3
33	309,090.9	19,318.2	19200	0.62
66	154,545.5	9659.1	9600	0.62
133	76,691.7	4793.2	4800	0.14
266	38,345.9	2396.6	2400	0.14
531	19,209.0	1200.6	1200	0.11
1062	9604.5	600.3	600	0.05
2125	4800.0	300.0	300	0.00
4250	2400.0	150.0	150	0.00
5795	1760.1	110.0	110	0.00

26.5.3.1 Module clock

The module clock MCLK is derived from the system bus clock. It has the same phase and frequency.

26.5.3.2 Transmitter clock

The transmitter clock TCLK is used to drive the data to the serial bus via the SCI_x_TX pin. It is derived from the system bus clock by the baud rate generator. The baud rate generator is controlled by the value of the SBR[12:0] field in the Synthesizer Control Register (SYNCR). The frequency of the transmitter clock is determined by *Equation 23* and defines the length of the transmitted bits, which is denoted as the *bit time*.

Equation 23

$$f_{TCLK} = \frac{f_{MCLK}}{16 \cdot SBR[12:0]}$$

26.5.3.3 Receiver clock

The receiver clock RCLK is used to sample the data received on the SCI_x_RX or SCI_x_TX pin. It is derived from the system bus clock by the baud rate generator. The baud rate generator is controlled by the value of the SBR[12:0] field in the Synthesizer Control Register (SYNCR). The frequency of the receiver sample clock is determined by *Equation 24*.

Equation 24

$$f_{RT} = \frac{f_{MCLK}}{SBR[12:0]}$$

The frequency of the receiver clock is 16 times the frequency of the transmitter clock, this each bit is sampled with 16 samples. Each of the 16 samples of a bit has a sample number assigned, which is defined by the receiver sample counter RSC. The n-th sample is denoted

by RSn. The receiver sample counter RSC is updated with each rising edge of the receiver clock RCLK.

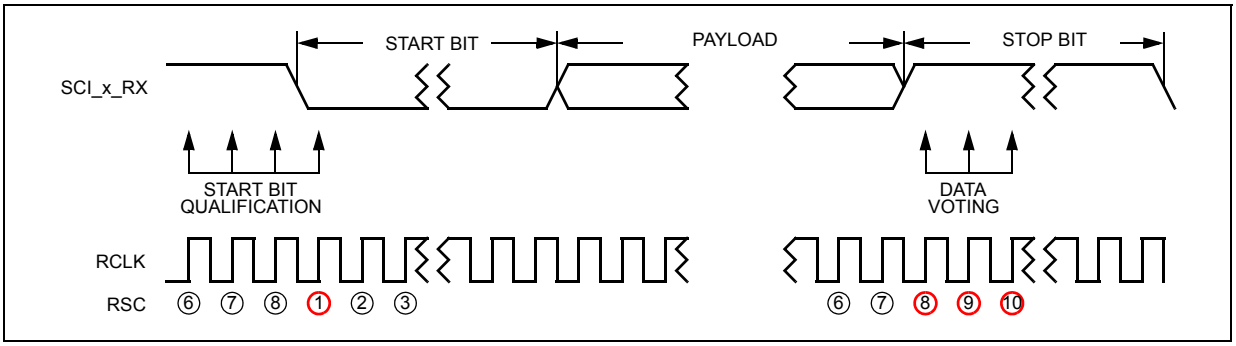
26.5.4 Baud rate tolerance

A transmitting device may be operating at a baud rate below or above the receiver baud rate. Accumulated bit time misalignment can cause one of the three stop bit data samples RS8, RS9, and RS10 to fall outside the actual stop bit. A noise error will occur if the stop bit sample RS8, RS9, and RS10 samples are not all the same logical value 1. A framing error will occur if the receiver clock is misaligned in such a way that the majority of the RS8, RS9, and RS10 stop bit samples are a logic zero.

26.5.4.1 Faster receiver tolerance

In this case the receiver has a higher baud rate than the transmitter, thus the stop bit sampling starts already in the last transmitted payload bit. To ensure the correct, noise and framing error free reception of the stop bit, the samples RS8, RS9, and RS10 must be located in the transmitted stop bit as shown in *Figure 567*.

Figure 567. Faster receiver



The maximum tolerance that ensures error free reception can be calculated with the assumption, that RS7 is sampled during the last transmitted payload bit and RS8 is sampled in the stop bit.

For an frame with *n* payload bits the transmitter starts the transmission of the stop bit

Equation 25

$$tx_{STOP} = (n + 1) \cdot 16 \cdot RT_{TR}$$

after the start of the transmission of the start bit.

For an frame with *n* payload bits the receiver samples the RS8 sample of the stop bit

Equation 26

$$rx_{STOP} = (n + 1) \cdot 16 \cdot RT_{RE} + 7 \cdot RT_{RE}$$

after the successful qualification of the start bit.

To ensure error free reception of the stop bit, the transmitter must start the transmission of the stop bit before the receiver samples RSC8.

Equation 27

$$tx_{STOP} < rx_{STOP}$$

The maximum percent difference between the receiver baud rate and the transmitter baud rate is:

Equation 28

$$\Delta baudrate \leq \left(\frac{rx_{STOP} - tx_{STOP}}{rx_{STOP}} \right) \times 100$$

The maximum percent differences for the supported frames is given in *Table 382*

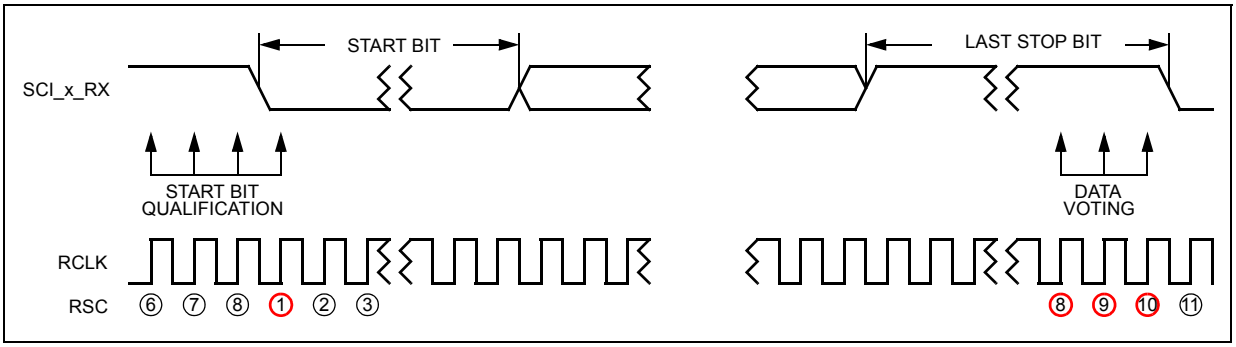
Table 382. Faster receiver maximum tolerance

Payload bits	Max baud rate difference	tx_{STOP}	rx_{STOP}
8	4.63%	144	151
9	4.19%	160	167
13	3.03%	224	231

26.5.4.2 Slower receiver tolerance

In this case the receiver has a slower baud rate than the transmitter, thus the stop bit sampling is still running while the next start bit is already transmitted. To ensure the correct, noise and framing error free reception of the stop bit, the samples RS8, RS9, and RS10 must be located in the transmitted stop bit as shown in *Figure 568*.

Figure 568. Slower receiver



The maximum tolerance that ensures error free reception can be calculated with the assumption, that RS11 is sampled in the transmitted start bit and RS10 is sampled in the last stop bit.

For an frame with *n* payload bits and *s* stop bits, the transmitter starts the transmission of the next start bit

Equation 29

$$tx_{START} = (n + s + 1) \cdot 16 \cdot RT_{TR}$$

after the start of the transmission of the previous start bit.

For an frame with *n* payload bits and s stop bits, the receiver samples the RS10 sample of the last stop bit

Equation 30

$$rx_{STOP} = (n + s) \cdot 16 \cdot RT_{RE} + 9 \cdot RT_{RE}$$

after the successful qualification of the start bit.

To ensure error free reception of the last stop bit, the transmitter must start the transmission of the start bit after the receiver samples RS10.

Equation 31

$$rx_{STOP} < tx_{START}$$

The maximum percent difference between the receiver baud rate and the transmitter baud rate is:

Equation 32

$$\Delta baudrate \leq \left(\frac{tx_{START} - rx_{STOP}}{tx_{START}} \right) \times 100$$

The maximum percent differences for the supported frames is given in *Table 383*

Table 383. Slower receiver maximum tolerance

Payload bits	Stop bits	Max baud rate difference	rx_{STOP}	tx_{START}
8	1	4.37%	153	160
9	1	3.97%	169	176
9	2	3.57%	185	196
13	2	2.73%	249	256

26.5.5 SCI mode

26.5.5.1 SCI mode configuration

The application must configure the following bits and fields in order to achieve correct SCI operation.

- enable *SCI* Mode
  LIN control register (SCI_LCR)[LIN]:= 0
- select *baud rate*
  Synthesizer Control Register (SYNCR)
- select *receiver input* mode
  Synthesizer Control Register (SYNCR)[LOOPS]
  Synthesizer Control Register (SYNCR)[RSRC]
- select *frame format*
  Synthesizer Control Register (SYNCR)[M]
  Synthesizer Control Register (SYNCR)[PE]
  Synthesizer Control Register (SYNCR)[WAKE]
  LIN polynomial register (SCI_LPR)[M2]
- select *parity type*
  Synthesizer Control Register (SYNCR)[PT]

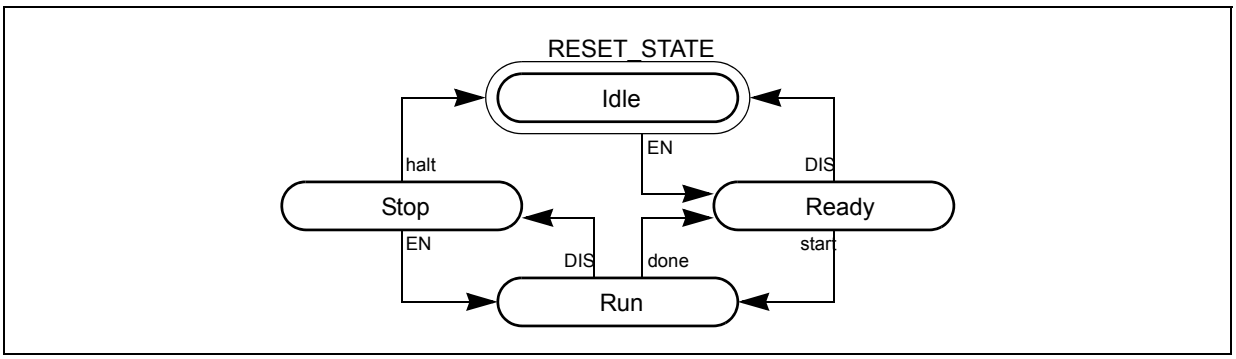
26.5.5.2 Transmitter

The transmitter supports the transmission of all frame types defined in *Table 377*, of all break characters defined in *Table 379*, and of all idle characters defined in *Table 380*.

26.5.5.2.1 Transmitter states and transitions

The transmitter has four basic states which are shown and described in *Table 384*. The state transitions that can triggered by the application commands are shown in *Table 385*. The state transitions that can triggered by the module are shown in *Table 386*. The state diagram of the transmitter is shown in *Figure 569*.

Figure 569. Transmitter state diagram



The current state of the transmitter can be determined by the control bit SCI_CR1[TE]and the status bit SCI_SR[TACT].

Table 384. Transmitter states

State	Indication		Description
	SCI_CR1[TE]	SCI_SR[TACT]	
Idle	0	0	Transmitter is *disabled* and *no* transmission is running
Ready	1	0	Transmitter is *enabled* and *no* transmission is running
Run	1	1	Transmitter is *enabled* and transmission is running
Stop	0	1	Transmitter is *disabled* and transmission is running

The application triggers a transition described in *Table 385* when it issues a command by writing to bit SCI_CR1[TE]. The transition is triggered only if the conditions are fulfilled. As a result of the transition the state of the transmitter is changed as shown in *Figure 569* and the action given in *Table 385* is executed.

Table 385. Transmitter application transitions

Transition	Command	Condition	Action	Description
EN	TE:= 1	TE = 0	iPRE:= 1	Transmitter is *enabled* by application command.
DIS	TE:= 0	TE = 1		Transmitter is *disabled* by application command

The module transition shown in *Table 386* are triggered when the described condition or event occurs. The send break bit SCI_CR1[SBK] is checked for the start condition. The internal commit bit iCMT, the transmitter active bit TACT in the SR register (SCI_SR), the TDRE, and the TC flag in the SR register (SCI_SR) are changed as a action result of the transition.

Table 386. Transmitter module transitions

Transition	Condition	Action	Description
start	(State = Ready) and (SBK = 1 or iPRE = 1 or iCMT = 1)	TACT:=1	Start of transmission of data frame or special character when data are available or character transmission request is pending.
done	State = Run and last stop bit transmitted	TACT:=0 TC:= (SBK & iPRE & TC)	Finished transmission of data frame or special character and transmitter still enabled. Transmission is complete if no transmit request is pending.
halt	State = Stop and last stop bit transmitted	TACT:=0 TC:=1 iCMT:=0	Finished transmission of data frame or special character and transmitter was disabled.

26.5.5.2.2 Frame and character transmission

The transmitter starts the transmission of a data frame or special character when the condition for the *start* transition as described in *Table 386* is fulfilled. There are three source for data or character transmission. The priority among these source are specified in *Table 387*. All three sources can be available at one point in time.

Table 387. Transmit source priority

Priority	Indication	Transmission source
(highest) 0	SCI_CR1[SBK] = 1	Break character
1	iPRE = 1	Preamble
(lowest) 2	iCMT = 1	SCI data register (SCI_DR) frame

26.5.5.2.3 Application controlled SCI data frame transmission

The transmission of a data frame is started when the transmitter is in its Ready state and only the commit bit iCMT is set.

As the first step, the content of the SCI data register (SCI_DR) is transferred into the internal transmitter shift register. When this transfer is finished, the internal commit bit iCMT is cleared and the transmit data register empty flag TDRE in the SR register (SCI_SR) is set. If the transmit interrupt enable bit SCR_CR1[TIE] is also set, the TDRE flag generates a transmitter interrupt request.

The transmitter shift register then shifts a frame out through the SCI_x_TX output signal, which is prefaced with a start bit and appended with the parity bit, if configured, and the configured number of stop bits.

When the last stop bit has been transmitted and the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the SR register (SCI_SR) is set.

If the application has disabled the transmitter while the frame is transmitted and stop bit has been transmitted, the transmitter goes into the Idle state via the *halt* transition. The transfer complete flag TC in the SR register (SCI_SR) is set and the internal commit bit iCMT is cleared.

26.5.5.2.4 DMA controlled SCI data frame transmission

In this mode, the eSCI module handles the generation of Data Frames internally.

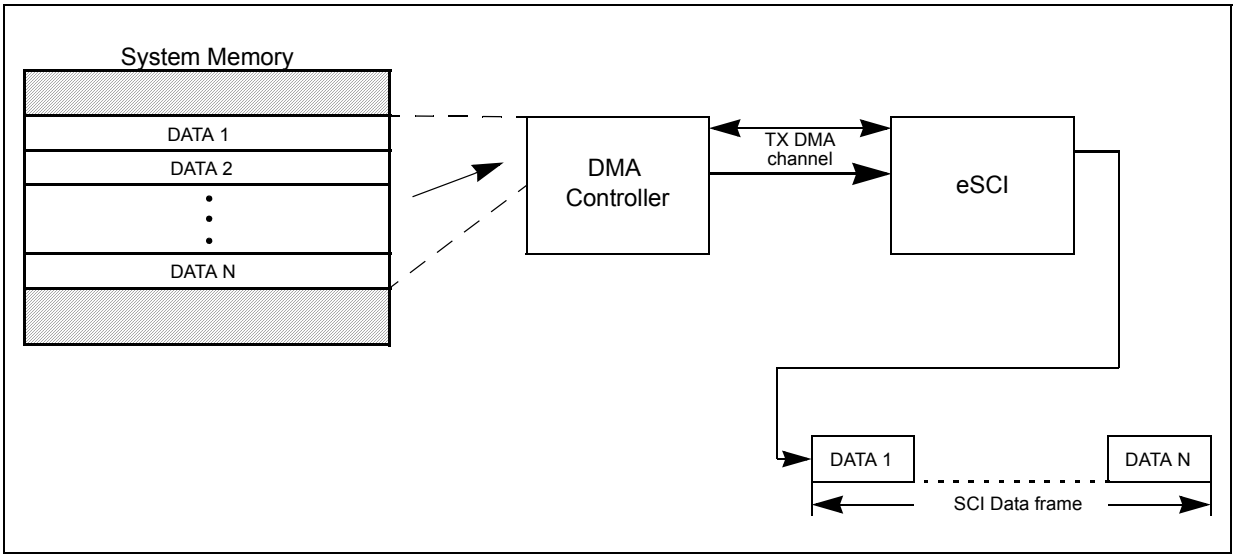
When new data required for transmission, the module generates the transmit DMA request and the DMA controller delivers the required data via write accesses to the SCI data register (SCI_DR). The write access to the data portion of the SCI_DR register triggers the transmission of the data. The write access to the high byte triggers no internal operation.

The application request the eSCI module to enter this mode by setting bit SCI_CR2[TXDMA]. From this point in time, the module starts the generation of DMA requests and frame transmission. Before entering this mode, the application should perform the following actions:

1. Configure the module for SCI mode.
2. Enable the transmitter by setting SCI_CR1[TE] to 1.
3. Set up the DMA controller channel and provide frame data in system memory

A block diagram which presents an overview of the DMA Controlled Date Frame Transmission is shown in *Figure 570*.

Figure 570. DMA controlled SCI data frame generation



26.5.5.2.5 Parity generation

The eSCI module generates the parity bit in transmitted data frame when the parity enable bit SCI_CR1[PE] is set. The parity type bit SCI_CR1[PT] defines whether the odd or even parity is generated.

26.5.5.2.6 Preamble transmission

The transmission of a preamble is started when the transmitter is in Ready state, the internal iPRE bit, which is not visible to the application, is set, and SCI_CR1[SBK] is cleared.

After the transmission of the stop bit and if the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the SR register (SCI_SR) is set.

If the application has disabled the transmitter while the preamble is transmitted and if the stop bit has been transmitted, the transmitter goes into the Idle state via the halt transition. The transfer complete flag TC in the SR register (SCI_SR) is set and the internal commit bit iCMT is cleared.

26.5.5.2.7 Break character transmission

The transmission of a break character is started when the transmitter is in Ready state and the send break character bit SCI_CR1[SBK] is set. After the transmission of the break character and if the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition and restarts the transmission. As long as SBK bit remains set, the transmitter continues to send break characters.

When the application has cleared the SBK bit or has disabled the transmitter, the transmitter continues to transmit the current break character and after it has finished the transmission of this break character it transmits a stop bit. The stop bit at the end of a break character sequence guarantees the recognition of the start bit of the next data frame.

After the transmission of the stop bit and if the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition. If no frame or character

transmit request is pending, the transfer complete flag TC in the SR register (SCI_SR) is set.

If the application has disabled the transmitter while the break character is transmitted and if the stop bit has been transmitted, the transmitter goes into the Idle state via the *halt* transition. The transfer complete flag TC in the SR register (SCI_SR) is set and the internal commit bit iCMT is cleared.

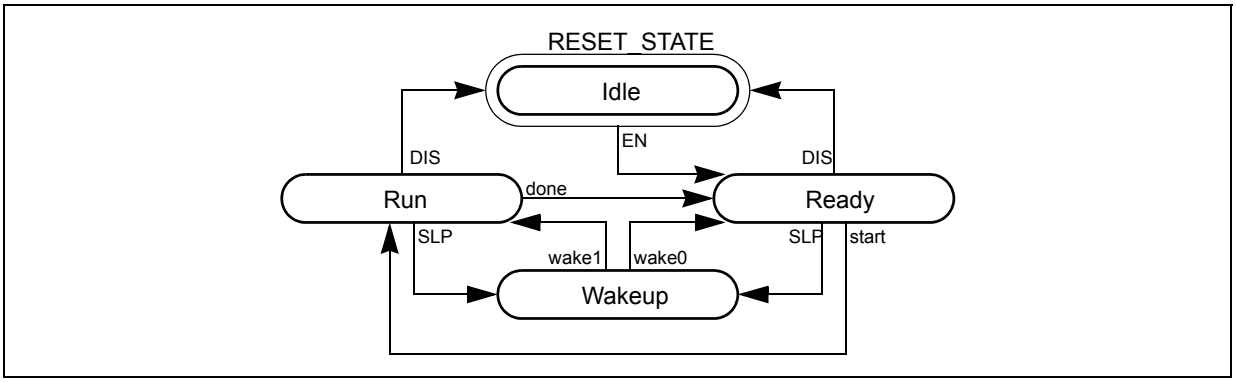
26.5.5.3 Receiver

The receiver supports the reception of all data frame types defined in *Table 377* and *Table 378*, of all break characters defined in *Table 379*, and of all idle characters defined in *Table 380*.

26.5.5.3.1 Receiver states and transitions

The receiver has four basic states which are shown and described in *Table 385*. The state transitions that can triggered by the application commands are shown in *Table 385*. The state transitions that can triggered by the module are shown in *Table 386*. The state diagram of the transmitter is shown in *Figure 569*.

Figure 571. Receiver state diagram



The current state of the receiver can be determined by the SCI_CR1 bits RE and RWU and the status bit SCI_SR[RAF].

Table 388. Receiver states

State	Indication			Description
	RE	RAF	RWU	
Idle	0	0	0	Receiver is *disabled* and *no* reception is running
Ready	1	0	0	Receiver is *enabled* and *no* reception is running
Run	1	1	0	Receiver is *enabled* and reception is running
Wakeup	1	—	1	Receiver is in wakeup mode

The application triggers a transition described in *Table 389* when it issues a command by writing to bit SCI_CR1[RE]. The transition is triggered only if the conditions are fulfilled. As a result of the transition the state of the receiver is changed as shown in *Figure 571* and the action given in *Table 389* is executed.

Table 389. Receiver application transition

Transition	Command	Condition	Action	Description
EN	RE:=1	RE = 0		Receiver is *enabled* by application command.
DIS	RE:=0	RE = 1		Receiver is *disabled* by application command
SLP	RWU:=1	RE = 1		Receiver is set into wakeup mode

The module transition shown in *Table 390* are triggered when the described event occurs.

Table 390. Receiver module transition

Transition	Condition	Action	Description
start	(State = Ready,Run) and (start bit received)	RAF:=1	Start of reception of data frame or break character
done	(State = Run) and (idle character received)	RAF:=0	Idle Character received
wake0	(State = Wakeup) and (idle character received)	RWU:= 0	Wakeup Idle Character received
wake1	(State = Wakeup) and (address frame received)	RWU:= 0	Wakeup address frame received

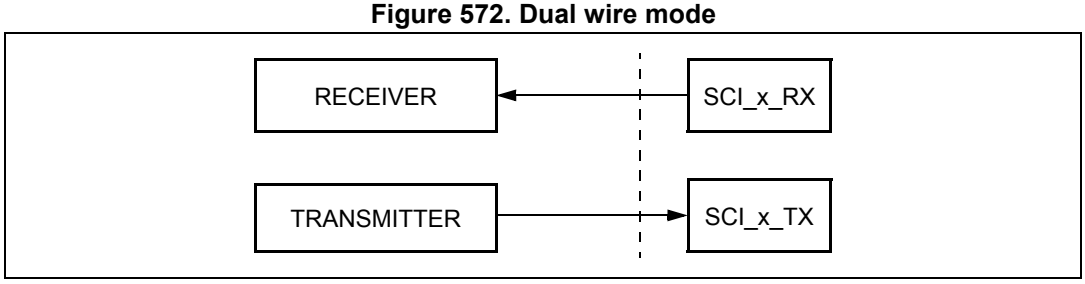
26.5.5.3.2 Receiver input mode selection

This section describes the three receiver input modes supported by the eSCI module. The modes are selected by the SCI_CR1 control bits LOOPS and RSRC.

26.5.5.3.3 Dual wire mode

In Dual Wire Mode, the eSCI uses the SCI_x_TX pin for transmitting and the SCI_x_RX pin for data receiving.

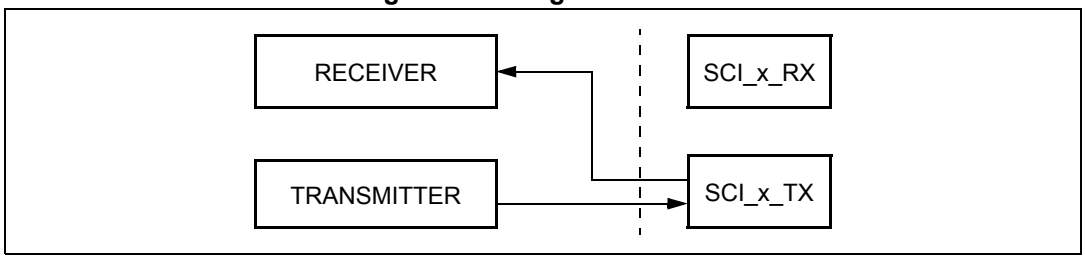
Figure 572. Dual wire mode



26.5.5.3.4 Single wire mode

In Single Wire Mode, the SCI_x_RX pin is disconnected from the eSCI module and the SCI_x_TX pin is used for both receiving and transmitting.

Figure 573. Single wire mode

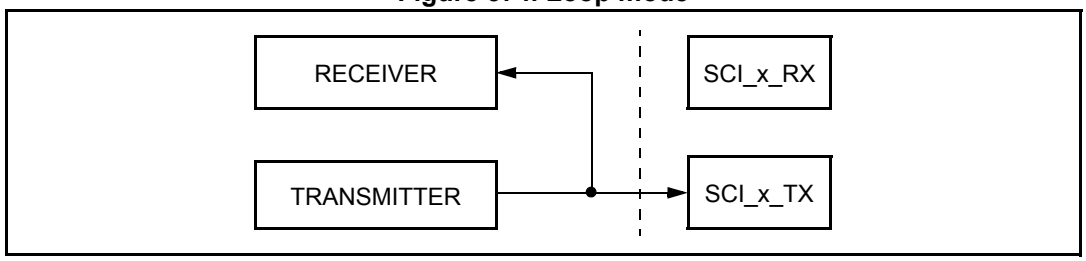


The TXDIR bit (SCI_CR2[1]) determines whether the SCI_x_TX pin is going to be used as an input (TXDIR = 0) or an output (TXDIR = 1) in this mode of operation.

26.5.5.3.5 Loop mode

In Loop Mode, the input of the receiver is driven by the output of the transmitter. The SCI_x_RX pin is disconnected from the eSCI module.

Figure 574. Loop mode



26.5.5.3.6 Frame and character reception

The receiver is started when it is in Ready or Wakeup state and on the selected receiver input (see *Section 26.5.5.3.2: Receiver input mode selection*) an active signal is sampled. The receiver enters the Run or Wakeup state. The received bits are recovered by the bit sampling described in *Section 26.5.5.3.13: Bit sampling*. During the reception, the received bits are shifted into the internal shift register.

26.5.5.3.7 Break character detection

The receiver does not provide any means to detect the reception of a break character. Instead, break characters are processed as data frames. Due to the received 0 at the stop bit location, the reception of a break character causes at least a framing error. The error reporting is performed as described in *Section 26.5.5.4: Reception error reporting*.

26.5.5.3.8 Idle character detection

The start point of the idle character detection is controlled by the idle line type bit SCI_CR1[ILT].

If SCI_CR1[ILT] is 0, the idle character detection starts always immediately after the reception of a bit with the value 0. In this mode, a data frame with a payload section of all ones will be erroneously detected as an idle character.

If SCI_CR1[ILT] is 1, the idle character detection starts after the reception of the last stop bit.

26.5.5.3.9 Application controlled SCI data frame reception

This section describes the reception process when the receiver is in the Run state.

When the required number of frame bits have been received, the payload bits of the received frame are transferred into SCI data register (SCI_DR) if the RDRF flag is 0. The receive data register full flag RDRF in SR register (SCI_SR) is set. If the receive interrupt enable bit SCI_SR[RIE] is set, the RDRF interrupt request is generated.

If an idle character has been detected, the IDLE flag in the SR register (SCI_SR) is set. If the idle line interrupt enable bit SCI_CR1[ILIE] is set, the IDLE interrupt request is generated.

If any of the receiver errors described in *Section 26.5.5.4: Reception error reporting* have been occurred, that corresponding flags will be set.

If the application disabled the receiver by clearing the receiver enable bit SCI_SR[RE], the current frame is discarded and no flags will be updated.

26.5.5.3.10 DMA controlled SCI data frames reception

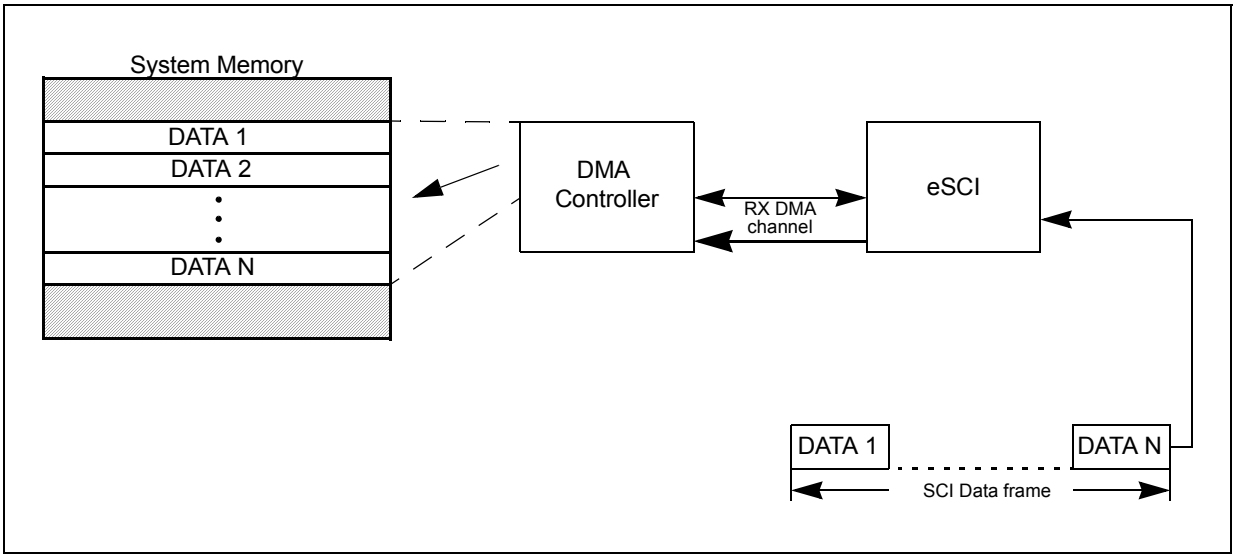
In this mode, the eSCI module controls the reception of SCI Data frames automatically and utilizes the connected DMA channels. A block diagram which presents an overview of the DMA Controlled SCI Data Frame reception is shown in *Figure 575*. The RX DMA channel is used to transfer the received frame data into the memory.

When new data was received, the module generates the receive DMA request and the DMA controller retrieves the provided data from the SCI data register (SCI_DR). The read access from the data portion of the SCI_DR signals the end of the DMA cycle for the current data and triggers the reception of new data. The read access from the high byte of the SCI_DR triggers no internal action.

The application requests the eSCI module to enter this mode by setting bit SCI_CR2[RXDMA]. From this point in time, the module starts the generation of DMA requests and frame transmission and reception. Before entering this mode, the application should perform the following actions:

1. Configure the module for SCI mode.
2. Enable the receiver by setting SCI_CR1[RE] to 1.
3. Set up the DMA controller channel.

Figure 575. DMA controlled SCI data frame reception



26.5.5.3.11 Receiver overrun

When the module attempts to transfer the payload data of a received frame into the SCI data register (SCI_DR) but neither the application nor the DMA controller has read the data portion (low byte) of the SCI_DR since its last update, the overrun flag OR in the SR register (SCI_SR) is set. The data contained in SCI_DR are not changed, but the received data are lost.

26.5.5.3.12 Wake-up frame reception

This section describes the reception process when the receiver is in the Wakeup state.

When the required number of frame bits have been received, the payload bits of the received frame are transferred into SCI data register (SCI_DR) if the RDRF flag is 0.

If the *address-mark* wake-up mode is selected and the received frame has the address bit set, the receive data register full flag RDRF in SR register (SCI_SR) is set. If the receive interrupt enable bit SCI_SR[RIE] is set, the RDRF interrupt request is generated. The RWU bit is cleared, and the receiver enters the Run state via the wake1 transition.

If the *idle line* wake-up mode is selected and the receiver has detected an idle character, The RWU bit is cleared, and the receiver enters the Ready state via the wake0 transition.

If any of the receiver errors described in *Section 26.5.5.4: Reception error reporting* have occurred, the corresponding flags will be set.

26.5.5.3.13 Bit sampling

The receiver samples the selected receiver input (see *Section 26.5.5.3.2: Receiver input mode selection*) with the receiver clock RCLK. The sampling for start bit detection is shown in *Figure 576*, the sampling for data and stop bit reception is shown in *Figure 577*. The samples indicated by dashed arrows are not used by the receiver. The received data bits are transferred into the internal shift register after the data strobing. If noise or framing errors were detected, this is flagged as described in *Section 26.5.5.4: Reception error reporting*.

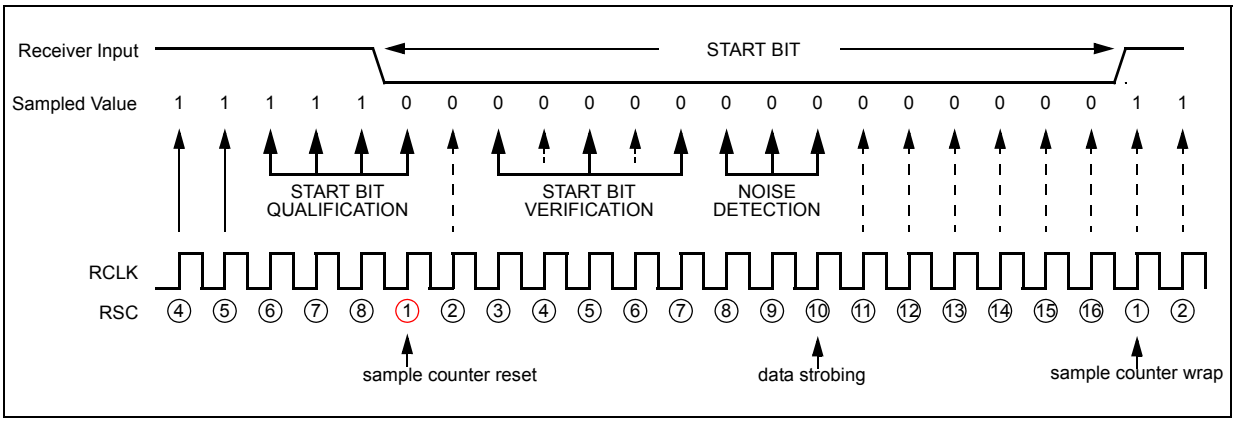
26.5.5.3.14 Bit synchronization

To adjust for baud rate mismatch, a synchronization of the cyclic sample counter RSC is performed during start bit reception as described in *Section 26.5.5.3.15: Start bit sampling*.

Additionally, the synchronization of the cyclic sample counter RSC can be configured to be performed during data bit reception as described in *Section 26.5.5.3.17: Data bit synchronization*.

26.5.5.3.15 Start bit sampling

Figure 576. Start bit sampling and strobing



26.5.5.3.15.1 Start bit qualification

To adjust for baud rate mismatch, the cyclic sample counter RSC is re-synchronized by reset after successful start bit qualification. A start bit is successfully qualified, if no reception is ongoing and three consecutive high samples are followed immediately by a low sample.

26.5.5.3.15.2 Start bit verification

After the successful start bit qualification the receiver starts to verify the start bit by a two out of three samples majority voting.

A start bit is verified if at least two out of the three sample RSC3, RSC5, and RSC5 are sampled low. Noise is detected when exactly one out of the three samples is high. The results of the start bit verification is summarized in *Table 391*.

Table 391. Start bit verification result

[RS3, RS5, RS7]	Start bit verified	Noise detected
000	Yes	No
001	Yes	Yes
010	Yes	Yes
100	Yes	Yes
011	No	No
101	No	No

Table 391. Start bit verification (continued)result (continued)

[RS3, RS5, RS7]	Start bit verified	Noise detected
110	No	No
111	No	No

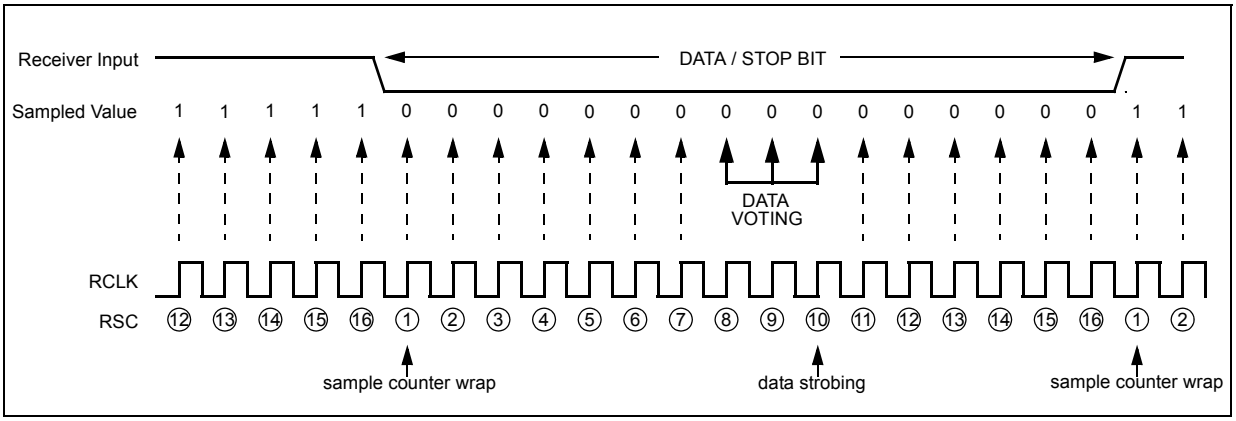
If the start bit verification was *not successful*, the receiver resumes the start bit qualification. If the start bit verification was *successful*, the receiver continues sampling to perform noise detection on the samples at RS8, RS9, and RS10. The results of the start bit noise detection is summarized in *Table 392*.

Table 392. Start bit noise detection

[RS8, RS9, RS10]	Noise detected
000	No
001	Yes
010	Yes
100	Yes
011	Yes
101	Yes
110	Yes
111	Yes

26.5.5.3.16 Data bit sampling

Figure 577. Data and stop bit sampling and strobing



To determine the value of a data bit and to detect noise, a two out of three majority voting is performed on the samples RS8, RS9, and RS10. *Table 393* summarizes the results of the data bit sample. The receiver detects the number of data bit according to the selected frame format.

Table 393. Data bit sampling

[RS8, RS9, RS10]	Data bit value	Noise detected
000	0	No
001	0	Yes
010	0	Yes
100	0	Yes
011	1	Yes
101	1	Yes
110	1	Yes
111	1	No

26.5.5.3.17 Data bit synchronization

To adjust for baud rate mismatch during the reception of data bits, the cyclic sample counter RSC can be configured to be synchronized on falling edges during data bit reception. This kind of synchronization is performed only if the synchronization mode bit SCI_LPR[SYNM] is 0.

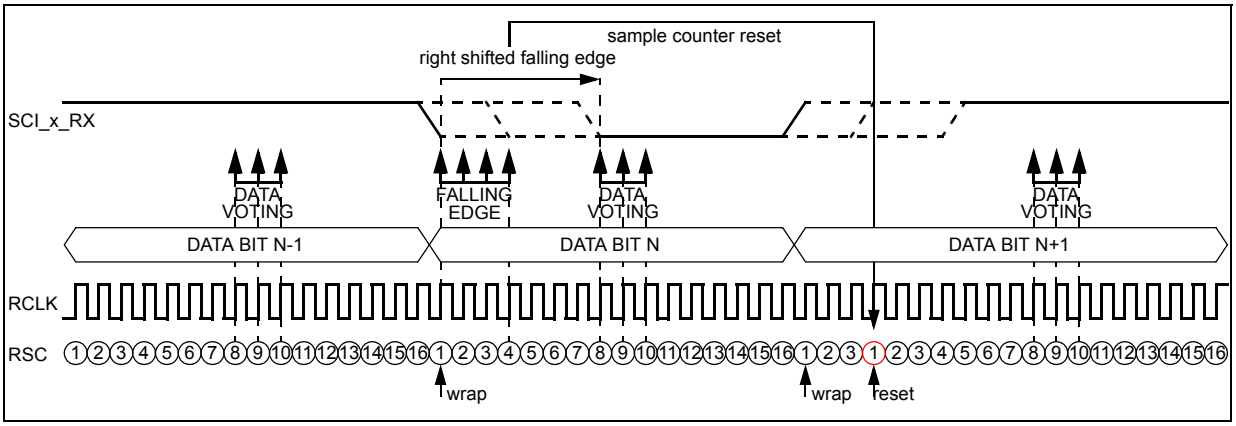
26.5.5.3.17.1 Data bit synchronization (right shifted edges)

This kind of sample counter synchronization happens if the transmitter is slower than the receiver. The reset behavior of the sample counter is shown in *Figure 578*. The sample counter reset condition is:

1. The data bit N-1 is sampled as 1, and
2. the data bit N is sampled as 0, and
3. a falling edge consisting of three consecutive 1-samples and a following 0-sample is detected, and
4. the 0-sample of the falling edge is received at data bit N sample j, with 1 <= j <= 8.

If the condition is fulfilled, the sample counter is reset 16 RCLK cycles after the 0 of the falling edge condition was received. The bit counter is not increased.

Figure 578. Data bit synchronization (right shifted edges)



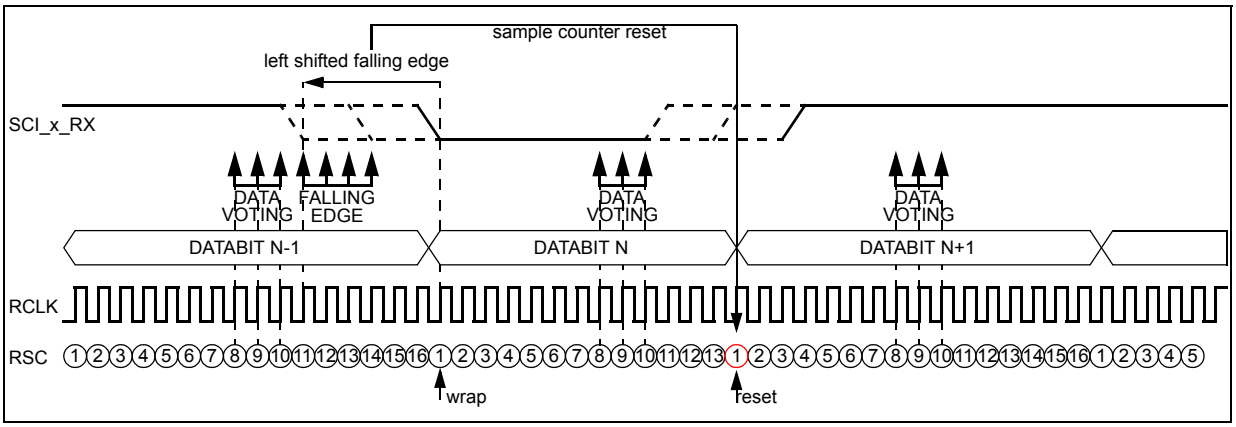
26.5.5.3.17.2 Data bit synchronization (left shifted edges)

This kind of sample counter synchronization happens if the transmitter is faster than the receiver. The reset behavior of the sample counter is shown in *Figure 579*. The sample counter reset condition is:

1. The data bit N-1 is sampled as 1, and
2. The data bit N is sampled as 0, and
3. A falling edge consisting of three consecutive 1-samples and a following 0-sample is detected, and
4. The 0-sample of the falling edge is received at data bit N sample j, with 11 <= j <= 16.

If the condition is fulfilled, the sample counter is reset 16 RCLK cycles after the 0-sample of the falling edge condition was received. The bit counter is increased by 1.

Figure 579. Data bit synchronization (left shifted edges)



If the 0-sample of the falling edge condition is received at sample 9 or 10, no sample counter synchronization is performed.

26.5.5.3.18 Stop bit verification

The reception of a valid stop bit is verified if at least two out of the sample RS8, RS9, and RS10 are sampled high. If this is not that case, a framing error is detected. Noise is detected if not all of the samples are of the same value. The results of the stop bit verification are summarized in *Table 394*.

Table 394. Stop bit verification

[RS8, RS9, RS10]	Stop bit verified	Framing error detected	Noise detected
000	No	Yes	No
001	No	Yes	Yes
010	No	Yes	Yes
100	No	Yes	Yes
011	Yes	No	Yes
101	Yes	No	Yes
110	Yes	No	Yes
111	Yes	No	No

26.5.5.3.19 Parity checking

The eSCI module calculates the parity of a received character and checks is versus the received parity bit in the received data frame when the parity enable bit SCI_CR1[PE] is set. The parity type bit SCI_CR1[PT] defines whether to check for odd or even parity is generated. If an parity error is detected, this is reported as described in *Section 26.5.5.4: Reception error reporting*.

26.5.5.4 Reception error reporting

The receiver can detect four error types: parity errors, framing errors, noise errors, and the overrun error.

The receiver reports the errors detected during frame reception at the end of the reception of the last stop bit of a frame. For error reporting the receiver utilizes the OR, NF, FE, and PF flags in the SR register (SCI_SR).

If the receiver has detected an overrun as described in *Section 26.5.5.3.11: Receiver overrun*, only the OR flag is set. All other error flags are not updated.

If the receiver has *not* detected an overrun and has detected noise as described in *Section 26.5.5.3.13: Bit sampling* the NF flag is set.

If the receiver has *not* detected an overrun and has detected a framing error as described in *Section 26.5.5.3.13: Bit sampling*, the FE flag is set.

If the receiver has *not* detected an overrun and has detected a parity error as described in *Section 26.5.5.3.19: Parity checking*, the PF flag is set.

26.5.5.5 Multiprocessor communication

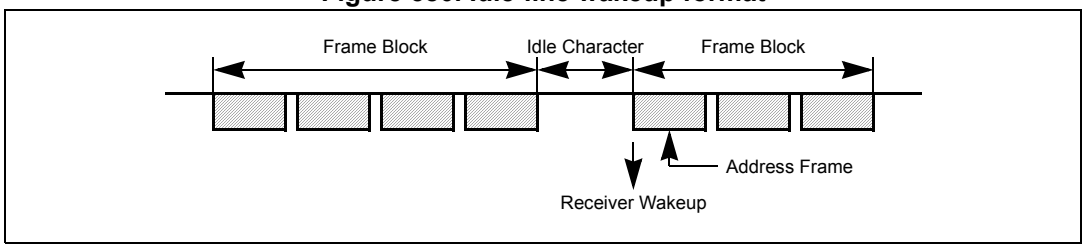
The multiprocessor communication allows one processor to send blocks of frames to other processors on the same serial link. To avoid the received data interrupt for frames not intended for the processor, the eSCI receiver can be put into the Wakeup state. If the receiver is in the Wakeup state, the eSCI will still load the received data into the SCI data register (SCI_DR), but will not set the RDRF flag and consequently no request the RDRF interrupt.

The receiver leaves the Wakeup state and clears bit SCI_CR1[RWU] when the wakeup pattern configured by SCI_CR1[WAKE] is received. The eSCI module supports two types of wakeup patterns: the idle-line wakeup pattern and the address-mark wakeup pattern.

26.5.5.5.1 Idle-Line wakeup

The idle-line wakeup mode is selected when SCI_CR1[WAKE] is 0. In this mode, the receiver leaves the wakeup state, when an idle character is detected as described in *Section 26.5.5.3.8: Idle character detection*. The next received frame is the address frame that contains address information which can be evaluated by the application. If the application decides not to receive the frame block, it can set SCI_CR1[RWU] and return the receiver to the wakeup state.

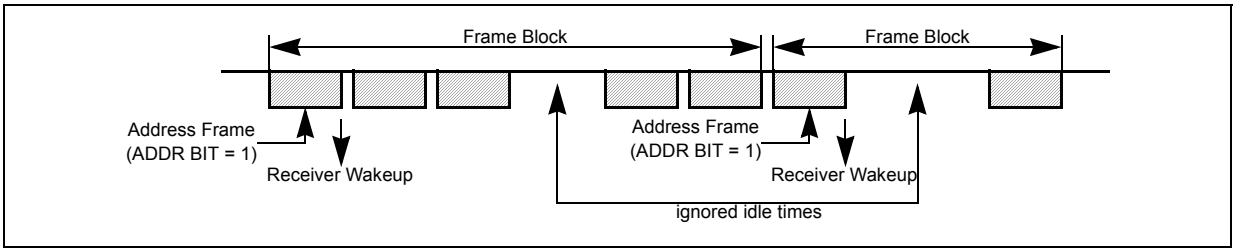
Figure 580. Idle-line wakeup format



26.5.5.5.2 Address-mark wakeup

The address-mark wakeup mode is selected when bit SCI_CR1[WAKE] is 1. If SCI_CR1[WAKE] is set, the address bit is added to the frame format. In this mode, the receiver leaves the wakeup state, when a data frame with the address bit value of 1 was received. This frame is the address frame and contains address information which can be evaluated by the application. If the application decides not to receive the frame block, it can set SCI_CR1[RWU] and return the receiver to the wakeup state. All data frames that belong to the frame block must have the address bit cleared.

Figure 581. Address-mark wakeup format



26.5.6 LIN mode

The eSCI provides support for the LIN protocol. It can be used to automate most tasks of a LIN master. In conjunction with the DMA interface it is possible to transmit entire LIN frames and sequences of LIN frames as well as to receive data from LIN slaves without application intervention. There is no special support for LIN slave mode.

26.5.6.1 LIN mode configuration

The application must configure the following bits and fields in order to achieve correct LIN operation. The configuration of bits and fields not mentioned in this section depend on the connected LIN slaves and the current application.

- enable *LIN* Mode
  - SCI_LCR[LIN]:= 1
- select *SCI_x_RX* pin as receiver input
  - SCI_CR1[LOOPS]:= 0
  - SCI_CR1[RSRC]:= 0
- select *LIN byte fields* as used frame format
  - SCI_CR1[M]:= 0
  - SCI_CR1[PE]:= 0
  - SCI_CR1[WAKE]:= 0
  - SCI_LPR[M2]:= 0
- select *break character length* of 13 bit as required by LIN 2.0
  - SCI_CR2[BRK13]:= 1
- select both transmitter and receiver *reset on bit error* detection
  - SCI_LCR[LDBG]:= 0
- select transmission stop on *bit error* detection
  - SCI_CR2[SBSTP]:= 1
- select transmission DMA stop on *bit error* detection
  - SCI_CR2[BSTP]:= 1
- enable both *transmitter* and *receiver*
  - SCI_CR1[TE]:= 1
  - SCI_CR1[RE]:= 1

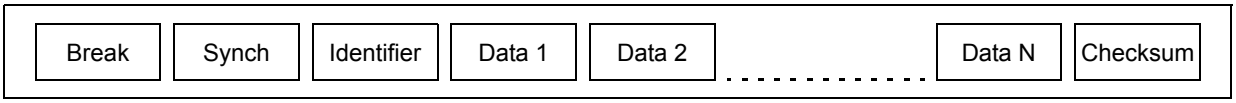
26.5.6.2 LIN frame formats

The term LIN frame refers to a sequence of LIN byte fields preceded by a break character, both are described in *Section 26.5.2: Frame formats*. The eSCI module allows to generate LIN frames for LIN slaves of LIN standards 1.3 and 2.0.

26.5.6.2.1 Standard LIN frames

A standard LIN frame, shown in *Figure 582* consists of a break character, a sync field, an ID field, zero or more data fields, and a checksum field. The data fields and the checksum field are generated by the LIN master for TX LIN frames and generated by the LIN slave for RX LIN frames. The header fields will always be generated by the LIN master.

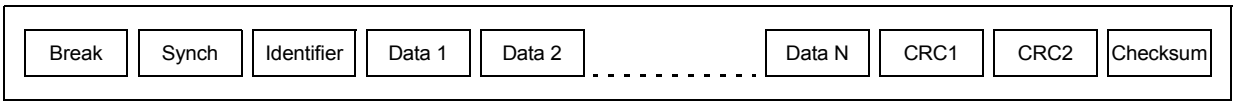
Figure 582. Standard LIN frame format



26.5.6.2.2 CRC enhanced LIN frames

The CRC Enhanced LIN frames shown in *Figure 583* contain two additional CRC byte fields. These fields are located between the last data field and the Checksum field. The value of the CRC is calculated on the same byte fields as the Checksum is calculated on. The polynomial used for the CRC calculation is defined by the LIN polynomial register (SCI_LPR). The eSCI module generates the CRC fields for TX frames and checks the CRC fields for RX frames if bit SCI_LTR[CRC] was written with a value of 1.

Figure 583. CRC Enhanced LIN frame format



The CRC Enhanced LIN frames are not part of the LIN standard.

26.5.6.3 LIN TX Frame generation

The eSCI module supports two modes of LIN TX Frame generation. In the application controlled mode, the application provides the required frame configuration and frame data by subsequent write accesses to the LIN transmit register (SCI_LTR). In the DMA generation mode, the DMA controller provides the required frame configuration and frame data in response to DMA requests generated by the eSCI module.

26.5.6.3.1 Application controlled LIN TX frame generation

In this mode, the application requests and controls the generation of an LIN TX Frame by subsequent write access to the LIN transmit register (SCI_LTR). To determine when to write to the SCI_LTR, the application can use the TXRDY flag in the SR register (SCI_SR). If this flag is set, the application can write to the SCI_LTR and should clear the TXRDY flag immediately.

The first byte written to the SCI_LTR contains the Identifier and Identifier Parity fields. The second byte written defines the number of data bytes to be transmitted. The third write access defines the CRC and checksum generation. The TD bit has to set to 1 to invoke the TX frame generation. The TO field bits must be set to 0.

After the third byte is written the generation of a LIN TX frame starts. Firstly, a break field is transmitted, then the synch field and the protected identifier field.

All subsequent write accesses provide data bytes, and a data byte field will be transmitted as soon as the data are available. After the last data byte, defined by the value written to the LEN field, is sent out, the configured CRC and checksum fields will be send out.

After the transmission of the last byte field of the frame, the write counter for the SCI_LTR is reset and the TXRDY flag is set.

26.5.6.3.2 DMA controlled LIN TX frame generation

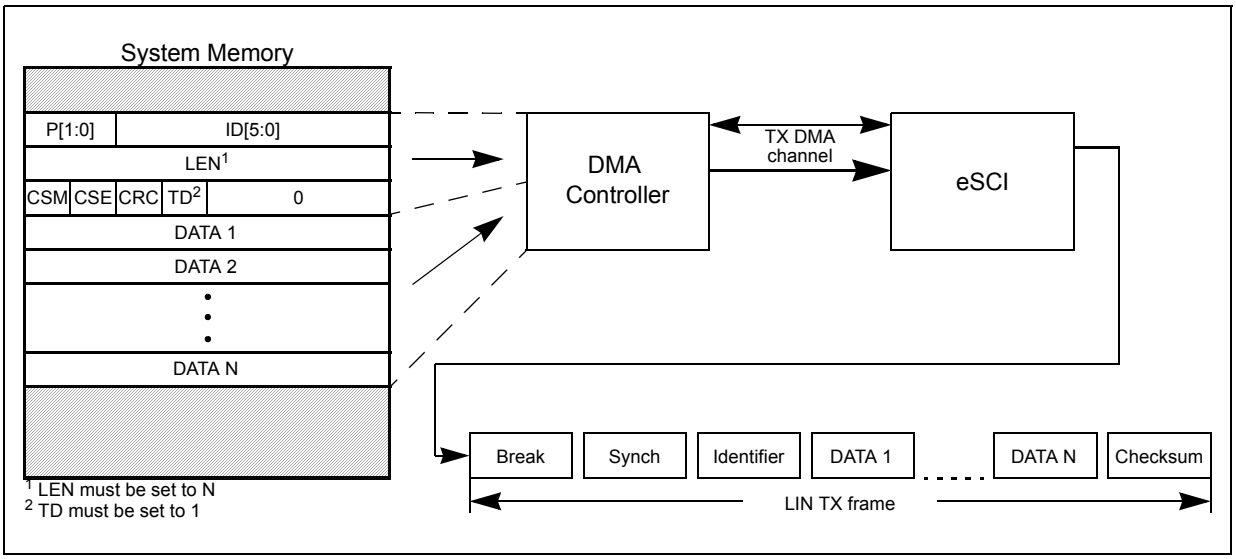
In this mode, the eSCI module handles the generation of an LIN TX Frame internally. When new data required for transmission, the module generates the transmit DMA request and

the DMA controller delivers the required data. The application requests the eSCI module to enter this mode by setting the TXDMA bit in the SCI control register 2 (SCI_CR2). From this point in time, the module starts the generation of DMA requests and frame transmission. Before entering this mode, the application should perform the following actions:

1. Configure the module for LIN mode.
2. Enable the transmitter by setting bit SCI_CR1[TE] to 1.
3. Reset the internal transmitter controller using bit SCI_LCR[LRES].
4. Set up the DMA controller channel and provide frame data in system memory.

A block diagram which presents an overview of the DMA Controlled LIN TX Frame is shown in *Figure 584*. The content of the fields in the memory is the same as described in LIN transmit register (SCI_LTR).

Figure 584. DMA controlled LIN TX frame generation



26.5.6.4 LIN RX frame generation

The eSCI module supports two modes of LIN RX Frame generation and reception. In the application mode, the application provides the required data by subsequent write and read accesses to and from the LIN transmit register (SCI_LTR). In the DMA mode, the DMA controller provides the required frame configuration data in response to DMA requests generated by the eSCI module and transfers the received frame data to the memory in response to DMA requests generated by the eSCI module.

26.5.6.4.1 Application controlled LIN RX frames generation

In this mode, the application provides the frame header and frame control data for the LIN RX frame by subsequent write access to the LIN transmit register (SCI_LTR). To determine when to write to the SCI_LTR, the application can use the TXRDY flag in the SR register (SCI_SR). If this flag is set, the application can write to the SCI_LTR and should clear the TXRDY flag immediately.

The first byte written to the SCI_LTR contains the Identifier and Identifier Parity fields. The second byte written defines the number of data bytes requested from the LIN slave. The third write access defines the CRC and checksum checking. The TD bit has to be set to 0 to

invoke the RX frame generation. The TO field defines the upper part of the timeout value. The fourth byte written defines the lower part of the timeout value.

After the fourth byte is written the generation of a LIN RX frame starts. Firstly, a break field is transmitted, then the synch field and the protected identifier field. After the transmission of the protected identifier, the eSCI module starts to receive the frame data transmitted by the LIN slave. When the module has received a complete byte field, the received data is transferred into the LIN receive register (SCI_LRR) and the receive data ready flag RXRDY in the SR register (SCI_SR) is set.

The application can retrieve the received data by subsequent read access from SCI_LRR after checking the RXRDY flag. The application should clear the RXRDY flag immediately after reading the SCI_LRR.

After the reception of the configured number of data from the slave, the module starts the reception of the configured CRC and Checksum byte fields. These data are not transferred into the SCI_LRR. The CRC and Checksum checking is performed internally. In case of errors, they will be reported as described in *Section 26.5.6.5: LIN error reporting*.

26.5.6.4.2 DMA controlled LIN RX frames generation

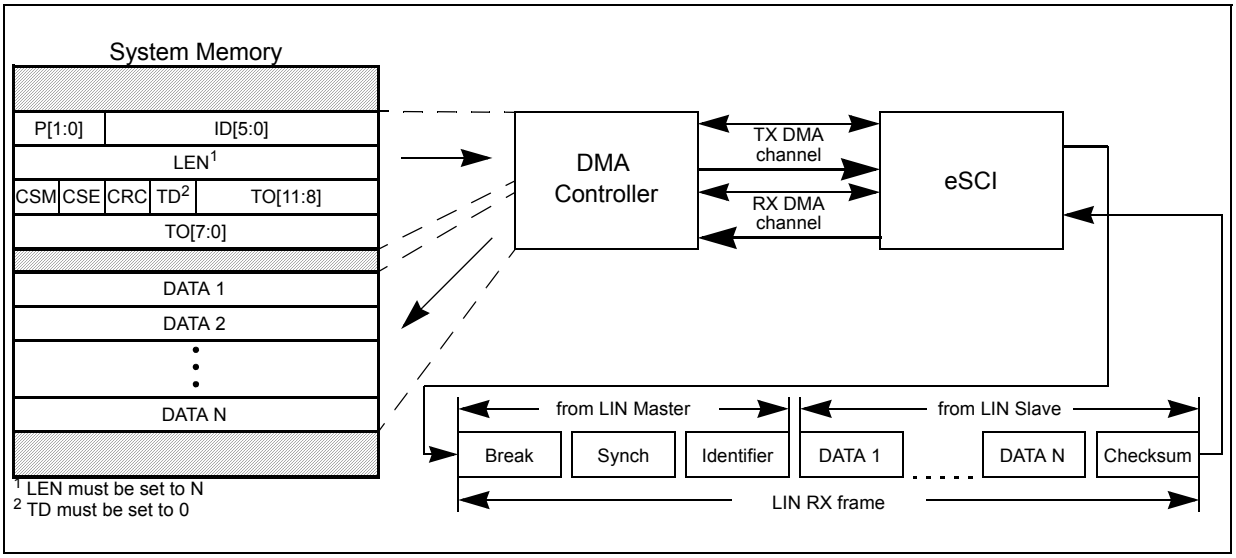
In this mode, the eSCI module controls the generation of LIN RX frame header and the reception of the frame data automatically and utilizes the two connected DMA channels. A block diagram which presents an overview of the DMA Controlled LIN RX Frame generation and reception is shown in *Figure 584*. The content of the header fields in the memory is the same as described in LIN transmit register (SCI_LTR). The TX DMA channel is used to fetch the LIN RX frame header and control information. The RX DMA channel is used to transfer the received frame data into the memory.

When new data required for transmission, the module generates the transmit DMA request and the DMA controller delivers the required data. When new data was received, the module generates the receive DMA request and the DMA controller retrieves the provided data.

The application request the eSCI module to enter this mode by setting the RXDMA bit in the SCI control register 2 (SCI_CR2). From this point in time, the module start the generation of DMA requests and frame transmission and reception. Before entering this mode, the application should perform the following actions:

1. Configure the module for LIN mode.
2. Enable transmitter and receiver by setting TE and RE in Synthesizer Control Register (SYNCR) to 1.
3. Reset the internal transmitter controller using bit SCI_LCR[LRES].
4. Setup the two DMA controller channels and provide frame header data in system memory.

Figure 585. DMA controlled LIN RX frame generation and reception



26.5.6.5 LIN error reporting

This section describes error checking and the signaling of detected errors in LIN mode.

26.5.6.5.1 Physical bus error detection

If the receiver input is sampled 0 for at least 31 sample clock cycles after the start of the transmission of a LIN frame, the physical bus error flag PBERR in the SR register (SCI_SR) will be set. If the LIN debug mode bit LDBG in the LIN control register (SCI_LCR) is not set, the transmission is aborted and the transmitter is reset.

26.5.6.5.2 Unrequested activity detection

If an unrequested byte is received (i.e. a byte which is not part of an RX frame) which is not recognized as a wakeup or break character, the bit error flag BERR in the SR register (SCI_SR) is set. In addition the RXRDY flag will also be set, the SCI_LRR register must be read before normal operations can proceed.

26.5.6.5.3 Standard bit error detection

The standard bit error detection is performed on each byte field transmission.

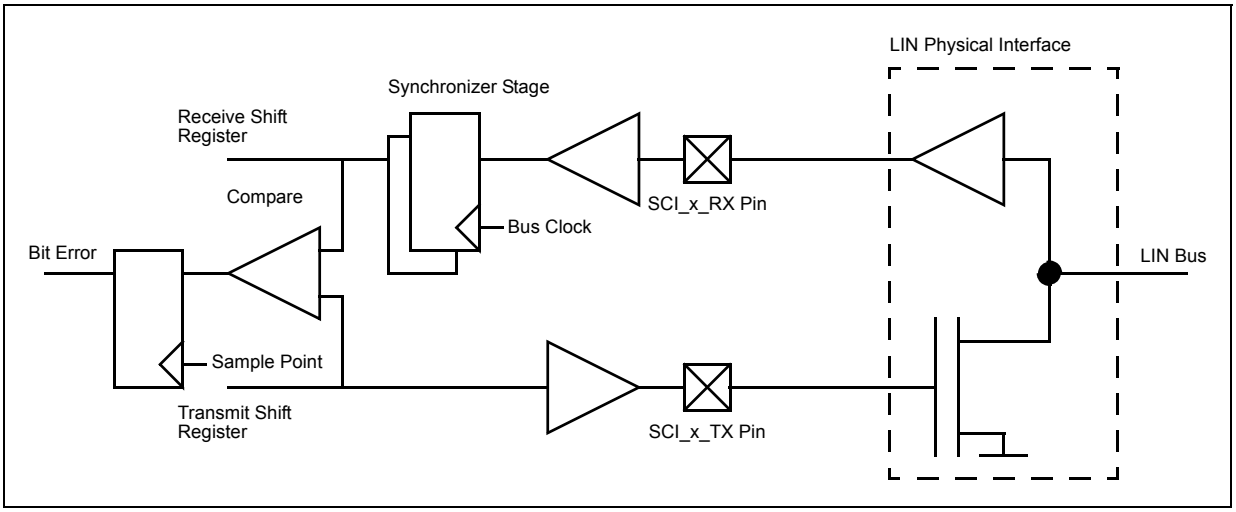
During the transmission of the frame header and frame data, the receiver is running and receives the signal values on the serial bus. After the complete transmission of a byte field, the eSCI compares the data that was transmitted and the data that was received. If they do not match, the bit error flag BERR in the SR register (SCI_SR) is set.

If the LIN debug mode bit LDBG in the LIN control register (SCI_LCR) is not set, the transmission is aborted and the transmitter is reset.

26.5.6.5.4 Fast bit error detection

Fast Bit Error Detection has been designed to allow flagging of LIN bit errors while they occur, rather than flagging them after a byte transmission has completed (see *Figure 586*).

Figure 586. Fast bit error detection on a LIN bus

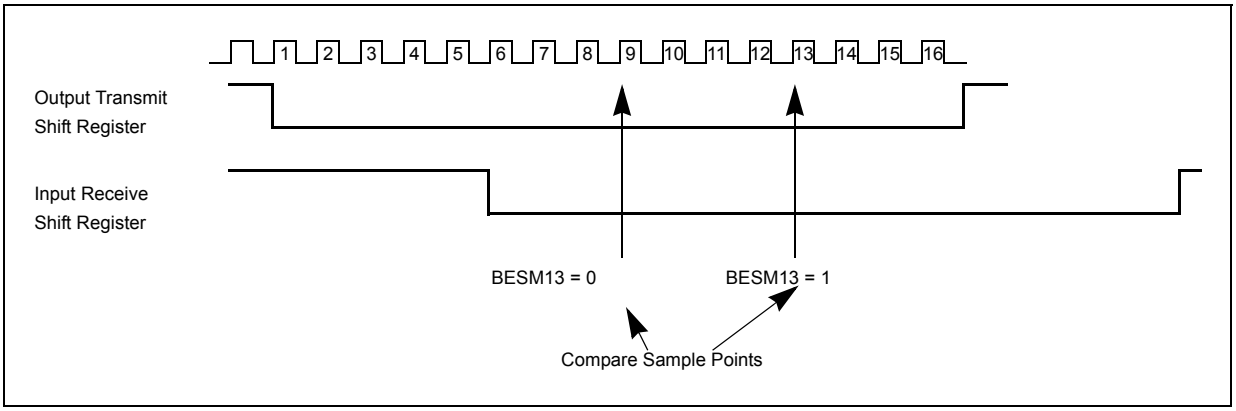


If fast bit error detection bit SCI_CR2[FBR] is set the eSCI will compare the transmitted and the received data stream while the transmitter is active (not idle). Once a mismatch between the transmitted data and the received data is detected the following actions are performed:

- The bit error flag BERR will be set
- if SBSTP is 0 the remainder of the byte will be transmitted normally
- if SBSTP is 1 the remaining bits in the byte after the error bit are transmitted as 1s (idle)

To adjust to different bus loads the sample point at which the incoming bit is compared to the one which was transmitted can be selected with the BESM13 bit (see *Figure 587*). If set the comparison will be performed with sample RS13, otherwise with RS9 (also see *Section 26.5.5.3.13: Bit sampling*).

Figure 587. Timing diagram fast bit error detection



Note: To calculate the exact position of the sample point with regard to the RX pin, the delays through the pads and the two Bus Clock cycle delay through the input synchronizer also needs to be taken into account.

26.5.6.5.5 Slave timeout detection

A slave timeout is detected, when the LIN slave has not transmitted all requested data bytes of an LIN RX frame within the amount of timeout specified in the TO[11:0] field in the LIN RX

frame control data. The timeout period starts with the transmission start of the LIN break character and is specified in multiples of the bit time.

If the last data byte is not received within the specified amount of time, the STO flag in the SR register (SCI_SR) will be set.

According to LIN 1.3 the timeout value TO has to be set to

Equation 33

$$TO = ((10 \cdot LEN + 45)) \times 1.4 \qquad (LEN \text{ is the number of data bytes in the frame})$$

If the LIN debug mode bit SCI_LCR[LDBG] is not set, the transmission is aborted and the transmitter and receiver are reset.

26.5.6.5.6 Checksum error detection

If the checksum enable bit SCI_LTR[CSE] was set, the checksum checking is performed based on the received checksum byte. The checksum mode is selected by the bit SCI_LTR[CSM]. If the value received in the checksum bytes did not match the calculated checksum, the checksum error flag CKERR in the SR register (SCI_SR) will be set.

26.5.6.5.7 CRC error detection

The CRC checking is performed on the two received CRC bytes CRC1 and CRC2 if the CRC Enhanced LIN frame format was selected by bit SCI_LTR[CRC]. If the value received in the two CRC bytes did not match the calculated CRC pattern, the CRC error flag CERR in the SR register (SCI_SR) will be set.

26.5.6.5.8 Overflow detection

When the receiver has received the next byte field, which should be transferred into the LIN receive register (SCI_LRR), but neither the application nor the RX DMA channel have read data from this register since the last update, the received data overflow flag OVFL in the SR register (SCI_SR) will be set. In this case the content of the SCI_LRR is not changed. The data received most recently are lost.

26.5.6.6 LIN wakeup

The section describes the LIN wakeup behavior of the eSCI module.

26.5.6.6.1 LIN wakeup generation

The eSCI module can cause the LIN bus to exit the sleep mode by sending a break character. The application triggers the transmission of a break character by writing 1 to the LIN bus wakeup trigger WU in the LIN control register (SCI_LCR). After the end of transmission of this break character the transmitter will neither set the TXRDY flag in the SR register (SCI_SR) nor start the transmission of frame data until the wakeup delimiter period has been expired. The wakeup delimiter period is defined by the field SCI_LCR[WUD].

To generate a valid wakeup character according to LIN 2.0, the eSCI first needs to be programmed to a baud rate lower than 32k baud, then WU can be set. Should the application require a higher baud rate, then this rate can be set once the wakeup character has been transmitted.

26.5.6.6.2 LIN wakeup reception

If the eSCI receives a valid wakeup condition on the selected receiver input the LIN wakeup flag LWAKE in the SR register (SCI_SR) will be set. If the LIN debug mode bit SCI_LCR[LDBG] is not set, the transmitter and receiver will be reset. Since each valid wakeup condition violates the byte field structure the frame error flag FE in the SCI_SR will be set, too.

The eSCI detects the following conditions as valid wakeup conditions:

- Reception of a break signal
- Reception of a LIN 1.x wakeup character (80h, 00h or C0h)
- Reception of a LIN 2.0 wakeup character (low pulse of 250 ms to 5 ms).

  To detect LIN 2.0 wakeup characters, the baud rate must to be set to 32k down to 1.6k baud.

26.5.7 Interrupts

This section describes the interrupt sources and interrupt request generation.

26.5.7.1 Interrupt flags and enables

All interrupt sources, interrupt flags, and interrupt enable bits are listed in *Table 395*.

Table 395. eSCI interrupt flags and enable bits

Interrupt source	Interrupt flag	Interrupt enable
Transmitter	SCI_SR[TDRE]	SCI_CR1[TIE]
Transmitter	SCI_SR[TC]	SCI_CR1[TCIE]
Receiver	SCI_SR[RDRF]	SCI_CR1[RIE]
Receiver	SCI_SR[IDLE]	SCI_CR1[ILIE]
Receiver	SCI_SR[OR]	SCI_CR2[ORIE]
Receiver	SCI_SR[NF]	SCI_CR2[NFIE]
Receiver	SCI_SR[FE]	SCI_CR2[FEIE]
Receiver	SCI_SR[PF]	SCI_CR2[PFIE]
LIN	SCI_SR[BERR]	SCI_CR2[BERRIE]
LIN	SCI_SR[RXRDY]	SCI_LCR[RXIE]
LIN	SCI_SR[TXRDY]	SCI_LCR[TXIE]
LIN	SCI_SR[LWAKE]	SCI_LCR[WUIE]
LIN	SCI_SR[STO]	SCI_LCR[STIE]
LIN	SCI_SR[PBERR]	SCI_LCR[PBIE]
LIN	SCI_SR[CERR]	SCI_LCR[CIE]
LIN	SCI_SR[CKERR]	SCI_LCR[CKIE]

Table 395. eSCI interrupt flags and enable bits (continued)

Interrupt source	Interrupt flag	Interrupt enable
LIN	SCI_SR[FRC]	SCI_LCR[FCIE]
LIN	SCI_SR[UREQ]	SCI_LCR[URIE]
LIN	SCI_SR[OVFL]	SCI_LCR[OFIE]

26.5.7.2 Interrupt request generation

The eSCI module provides one hardware interrupt request signal to the systems interrupt controller. This interrupt request signal is asserted if and only if at least one of the interrupt flags and the corresponding interrupt enables are set to 1. Otherwise the interrupt line is deasserted.

26.6 Application information

26.6.1 SCI data frames separated by preamble

To separate SCI data frame with preambles with minimum idle line time, use this sequence between messages:

1. Write to SCI data register (SCI_DR)
   – This sets the internal iCMT bit which requests the data transmission.
2. Wait until SCI_SR[TDRE] is set
   – This indicates the start of transmission; the iCMT bit was cleared.
3. Clear and subsequently set SCI_CR1[TE]
   – This sets the internal iPRE bit which requests the preamble transmission.
4. Write to SCI data register (SCI_DR)
   – This sets the internal iCMT bit which requests the data transmission.

The priority scheme of the transmitter which is described in *Table 387* ensures, that the preamble is transmitted before the data frame.

27 FlexCAN Module

27.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

27.1.1 Device-specific features

The device has two Controller Area Network (FlexCAN) blocks.

- FlexCAN A contains an embedded memory capable of storing 64 MBs.
- FlexCAN C contains an embedded memory capable of storing 32 message buffers (MB); the memory area corresponding to MBs 32–63 and Rx Individual Filters 32–63 are considered reserved space

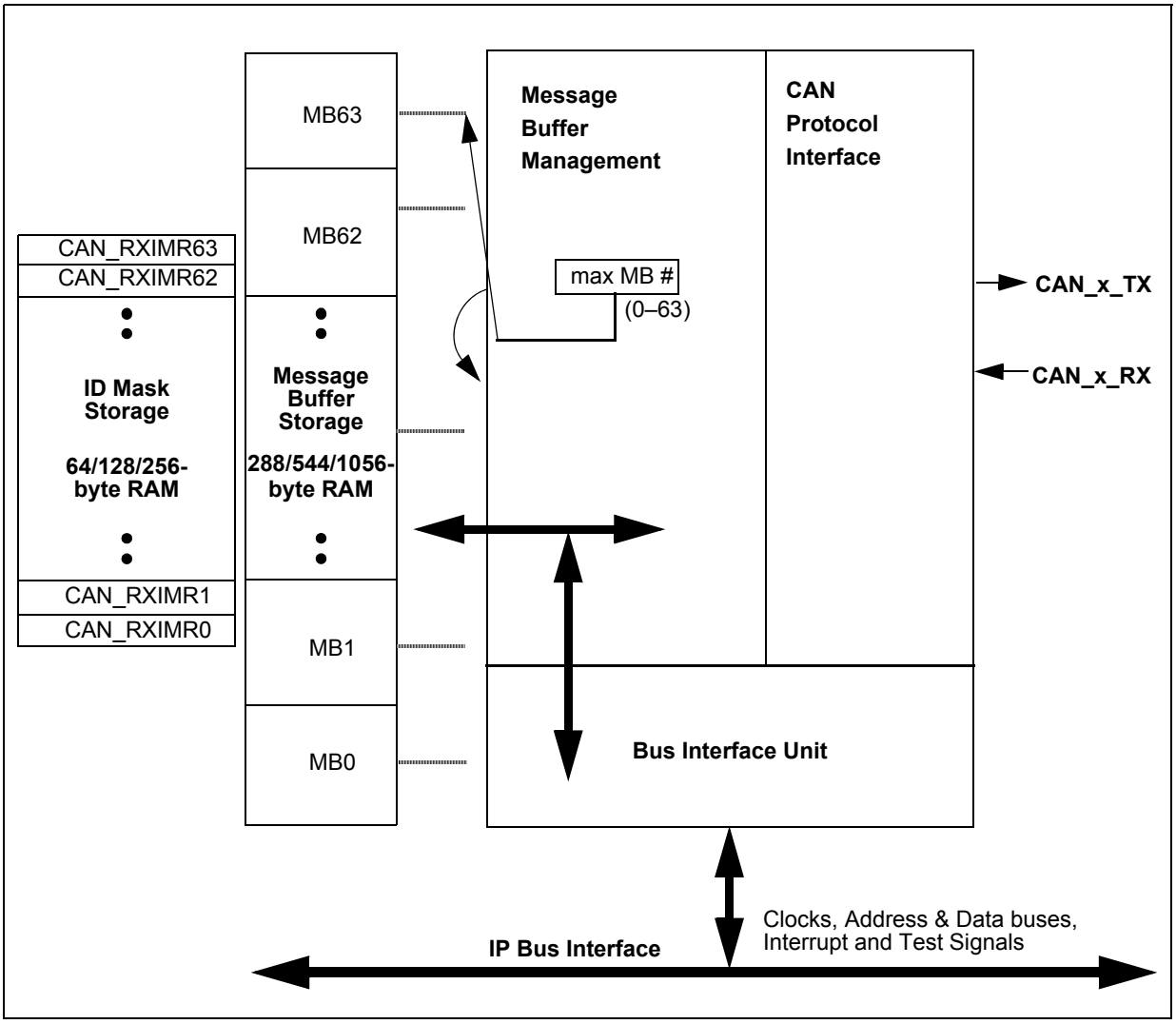
Each FlexCAN module has the following specific features implemented:

- Although the FlexCAN module provides a differentiation between Supervisor and User access types, all accesses will be always considered of the Supervisor type. As a consequence, CAN_MCR[SUPV] has no effect on the module behavior.
- The clock source to the CAN Protocol Interface can be either the system clock or a direct feed from the oscillator pin EXTAL. The clock source is selected by CAN_CR[CLKSRC].
- After reset, the modules are enabled (CAN_MCR[MDIS] is ‘0’) and in freeze mode (bit FRZ and HALT in CAN_MCR are set). As a consequence, CAN_MCR[FRZACK] is set and CAN_MCR[MDISACK] is negated.
- The Individual Filters per Message Buffer feature is implemented.
- The memory to store the individual filters is implemented.
- Rx and Tx warning interrupts when error counter reaches 96.
- Full featured Rx FIFO (capacity for 6 frames and internal pointer handling) is implemented.
- All 4 FlexCAN functional modes are supported: Normal, Freeze, Listen-Only and Loop-Back.
- Only 2 power modes are supported, the Disable and Stop Mode. Doze Mode is not supported.
- The wake-up feature is implemented and the CAN_x_RX input has an embedded low pass filter to reject noise interfering with wake-up.

27.2 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 588*, which describes the main subblocks implemented in the FlexCAN module, including two embedded memories, one for storing message buffers (MB) and another one for storing Rx Individual Mask Registers. Support for up to 64 message buffers is provided (see *Section 27.1: Information specific to this device* for the actual size implemented on the MCU). The functions of the submodules are described in subsequent sections.

Figure 588. FlexCAN block diagram



27.2.1 Overview

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B, which supports both standard and extended message frames. A flexible number of message buffers (16, 32 or 64) is also supported. The message buffers are stored in an embedded RAM dedicated to the FlexCAN module. Please refer to *Section 27.1: Information specific to this device* for the actual number of message buffer configured in the MCU.

The CAN Protocol Interface (CPI) submodule manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The Message Buffer Management (MBM) submodule handles message buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The Bus Interface Unit (BIU) submodule

controls the access to and from the internal interface bus, in order to establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs and test signals are accessed through the Bus Interface Unit.

27.2.2 FlexCAN module features

The FlexCAN module includes these distinctive features:

- Full implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Zero to eight bytes data length
  - Programmable bit rate up to 1 Mbit/s
  - Content-related addressing
- Flexible message buffers (up to 64) of zero to eight bytes data length
- Each MB configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask Registers per message buffer
- Includes either 1056 bytes (64 MBs), 544 bytes (32 MBs) or 288 bytes (16 MBs) of RAM used for MB storage
- Includes either 256 bytes (64 MBs), 128 bytes (32 MBs) or 64 bytes (16 MBs) of RAM used for individual Rx Mask Registers
- Full featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard or 32 partial (8 bits) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN version
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- Unused MB and Rx Mask Register space can be used as general purpose RAM space
- Listen only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number or highest priority
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes, with programmable wake up on bus activity

Note: The individual Rx Mask per Message Buffer feature may not be available in low-cost MCUs. Please consult Section 27.1: Information specific to this device to find out if this feature is supported.

27.2.3 Modes of operation

The FlexCAN module has four functional modes: Normal Mode (User and Supervisor), Freeze Mode, Listen-Only Mode and Loop-Back Mode. There are also three low power modes: Disable Mode, Doze Mode and Stop Mode.

- Normal Mode (User or Supervisor)—In Normal Mode, the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN Protocol functions are enabled. User and Supervisor Modes differ in the access to some restricted control registers.
- Freeze Mode—This mode is enabled when CAN_MCR[FRZ] is asserted. If enabled, Freeze Mode is entered when CAN_MCR[HALT] is set or when Debug Mode is requested at MCU level. In this mode, no transmission or reception of frames is done and synchronicity to the CAN bus is lost. See *Section 27.5.9.1: Freeze Mode* for more information.
- Listen-Only Mode—The module enters this mode when CAN_CR[LOM] is asserted. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.
- Loop-Back Mode—The module enters this mode when CAN_CR[LPB] 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 CAN_x_RX input pin is ignored and the CAN_x_TX output goes to the recessive state (logic ‘1’). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.
- Module Disable Mode—This low power mode is entered when CAN_MCR[MDIS] is asserted by the CPU. When the FlexCAN module is disabled, the module sends a request to disable the clocks to the CAN Protocol Interface and Message Buffer Management submodules. Exit from this mode is done by negating CAN_MCR[MDIS]. See *Section 27.5.9.2: Module Disable Mode* for more information.
- Doze Mode—This low power mode is entered when the DOZE bit in CAN_MCR is asserted and Doze Mode is requested at MCU level. When in Doze Mode, the module shuts down the clocks to the CAN Protocol Interface and the Message Buffer Management submodules. Exit from this mode happens when the DOZE bit in CAN_MCR is negated, when the MCU is removed from Doze Mode, or when activity is detected on the CAN bus and the Self Wake Up mechanism is enabled. See *Section 27.5.9.3: Doze Mode* for more information.
- 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 and the Self Wake Up mechanism is enabled. See *Section 27.5.9.4: Stop Mode* for more information.

27.3 External signal description

27.3.1 Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in *Table 396* and described in more detail in the next subsections.

Table 396. FlexCAN signals

Signal name(1)	Direction	Description
CAN_x_RX	Input	CAN receive pin
CAN_x_TX	Output	CAN transmit pin

1. The actual MCU pins may have different names. Please consult *Chapter 3: Signal Descriptions* for the actual signal names.

27.3.2 Signal descriptions

27.3.2.1 CAN_x_RX

This pin is the receive pin from the CAN bus transceiver. Dominant state is represented by logic level ‘0’. Recessive state is represented by logic level ‘1’.

27.3.2.2 CAN_x_TX

This pin is the transmit pin to the CAN bus transceiver. Dominant state is represented by logic level ‘0’. Recessive state is represented by logic level ‘1’.

27.4 Memory map/register definition

This section describes the registers and data structures in the FlexCAN module. The base address of the module depends on the particular memory map of the MCU. The addresses presented here are relative to the base address.

The address space occupied by FlexCAN has 96 bytes for registers starting at the module base address, followed by MB storage space in embedded RAM starting at address 0x0060, and an extra ID Mask storage space in a separate embedded RAM starting at address 0x0880.

27.4.1 FlexCAN memory mapping

The complete memory map for a FlexCAN module with 64 MBs capability is shown in *Table 397*. 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 CAN_MCR[SUPV]. These registers are identified as S/U in the Access column of *Table 397*.

The CAN_IFRH and CAN_IMRH registers are considered reserved space when FlexCAN is configured with 16 or 32 MBs. The Rx Global Mask (CAN_RXGMASK), Rx Buffer 14 Mask (CAN_RX14MASK) and the Rx Buffer 15 Mask (CAN_RX15MASK) registers are provided for backwards compatibility, and are not used when CAN_MCR[MBFEN] is asserted.

The address ranges 0x0060–0x047F and 0x0880–0x097F are occupied by two separate embedded memories. These two ranges are completely occupied by RAM (1056 and 256

bytes, respectively) only when FlexCAN is configured with 64 MBs. When it is configured with 16 MBs, the memory sizes are 288 and 64 bytes, so the address ranges 0x0180–0x047F and 0x08C0–0x097F are considered reserved space. When it is configured with 32 MBs, the memory sizes are 544 and 128 bytes, so the address ranges 0x0280–0x047F and 0x0900–0x097F are considered reserved space. Furthermore, if CAN_MCR[MBFEN] is negated, then the whole Rx Individual Mask Registers address range (0x0880–0x097F) is considered reserved space.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult Section 27.1: Information specific to this device to find out if this feature is supported. If not supported, the address range 0x0880–0x097F is considered reserved space, independent of CAN_MCR[MBFEN]'s value.

Table 397. Module memory map

Address	Use	Access type	Affected by hard reset	Affected by soft reset	Location
Base + 0x0000	Module Configuration (CAN_MCR)	S	Yes	Yes	on page 998
Base + 0x0004	Control Register (CAN_CR)	S/U	Yes	No	on page 1003
Base + 0x0008	Free Running Timer (CAN_TIMER)	S/U	Yes	Yes	on page 1006
Base + 0x000C	Reserved	—	—	—	—
Base + 0x0010	Rx Global Mask (CAN_RXGMASK)	S/U	Yes	No	on page 1007
Base + 0x0014	Rx Buffer 14 Mask (CAN_RX14MASK)	S/U	Yes	No	on page 1008
Base + 0x0018	Rx Buffer 15 Mask (CAN_RX15MASK)	S/U	Yes	No	on page 1008
Base + 0x001C	Error Counter Register (CAN_ECR)	S/U	Yes	Yes	on page 1008
Base + 0x0020	Error and Status Register (CAN_ESR)	S/U	Yes	Yes	on page 1010
Base + 0x0024	Interrupt Masks 2 (CAN_IMRH)	S/U	Yes	Yes	on page 1013
Base + 0x0028	Interrupt Masks 1 (CAN_IMRL)	S/U	Yes	Yes	on page 1013
Base + 0x002C	Interrupt Flags 2 (CAN_IFRH)	S/U	Yes	Yes	on page 1014
Base + 0x0030	Interrupt Flags 1 (CAN_IFRL)	S/U	Yes	Yes	on page 1014
Base + 0x0034–0x005C	Reserved	—	—	—	—
Base + 0x0060–0x007F	Reserved	—	—	—	—
Base + 0x0080–0x017F	Message Buffers MB0–MB15	S/U	No	No	

Table 397. Module memory map (continued)

Address	Use	Access type	Affected by hard reset	Affected by soft reset	Location
Base + 0x0180–0x027F	Message Buffers MB16–MB31	S/U	No	No	
Base + 0x0280–0x047F	Message Buffers MB32–MB63	S/U	No	No	
Base + 0x0480–0x087F	Reserved	—	—	—	—
Base + 0x0880–0x08BF	Rx Individual Mask Registers CAN_RXIMR0–CAN_RXIMR15	S/U	No	No	*on page 1016*
Base + 0x08C0–0x08FF	Rx Individual Mask Registers CAN_RXIMR16–CAN_RXIMR31	S/U	No	No	*on page 1016*
Base + 0x0900–0x097F	Rx Individual Mask Registers CAN_RXIMR32–CAN_RXIMR63	S/U	No	No	*on page 1016*

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 398*. *Table 398* shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

Table 398. Message buffer MB0 memory mapping

Address offset	MB field
0x80	Control and Status (C/S)
0x84	Identifier Field
0x88–0x8F	Data Field 0 – Data Field 7 (1 byte each)

27.4.2 Message buffer structure

The message buffer structure used by the FlexCAN module is represented in *Figure 589*. 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 589. Message buffer structure

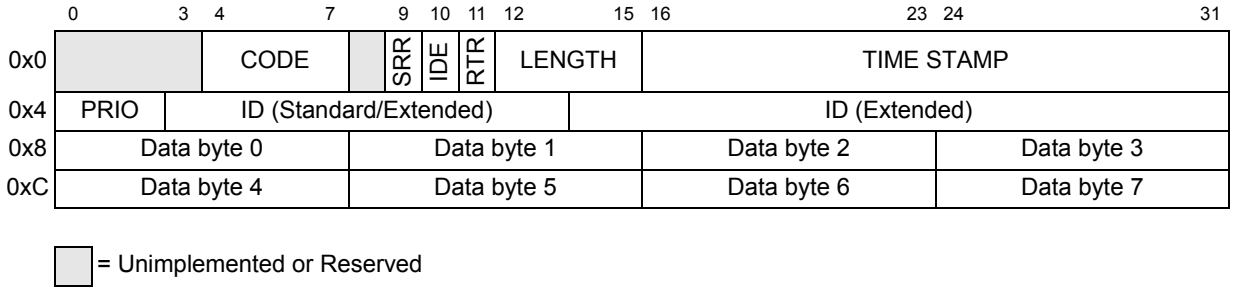


Table 399. Message buffer field descriptions

Field	Description
CODE (Rx Buffer)	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 400* and *Table 401*. See *Section 27.5: Functional description* for additional information.
CODE (Tx Buffer)	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 400* and *Table 401*. See *Section 27.5: Functional description* for additional information.
SRR	Substitute Remote Request Fixed recessive bit, used only in extended format. It must be set to '1' by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 1 Recessive value is compulsory for transmission in Extended Format frames 0 Dominant is not a valid value for transmission in Extended Format frames
IDE	ID Extended Bit This bit identifies whether the frame format is standard or extended. 1 Frame format is extended 0 Frame format is standard
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 589*). 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 CAN_MCR[LPRIO_EN] is set and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See *Section 27.5.3: 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.

Table 400. Message buffer code for Rx buffers

Rx code BEFORE Rx new frame	Description	Rx code AFTER Rx new frame	Comment
0000	INACTIVE: MB is not active.	—	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to *Section 27.5.5: 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 27.5.5: 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 401*), the BUSY bit should be ignored upon read, except when CAN_MCR[AEN] is set.

Table 401. 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 CAN_MCR[AEN] is asserted. MB does not participate in the arbitration process.

Table 401. Message buffer code for Tx buffers (continued)

RTR	Initial Tx code	Code after successful transmission	Description
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to ‘1110’ to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to ‘1010’ to restart the process again.
0	1110	1010	This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame will be transmitted unconditionally once and then the code will automatically return to ‘1010’. The CPU can also write this code with the same effect.

27.4.3 Rx FIFO structure

When CAN_MCR[FEN] is set, the memory area from 0x80 to 0xFC (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. *Figure 590* 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 590* shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the CAN_MCR. Note that all elements of the table must have the same format. See *Section 27.5.7: Rx FIFO* for more information.

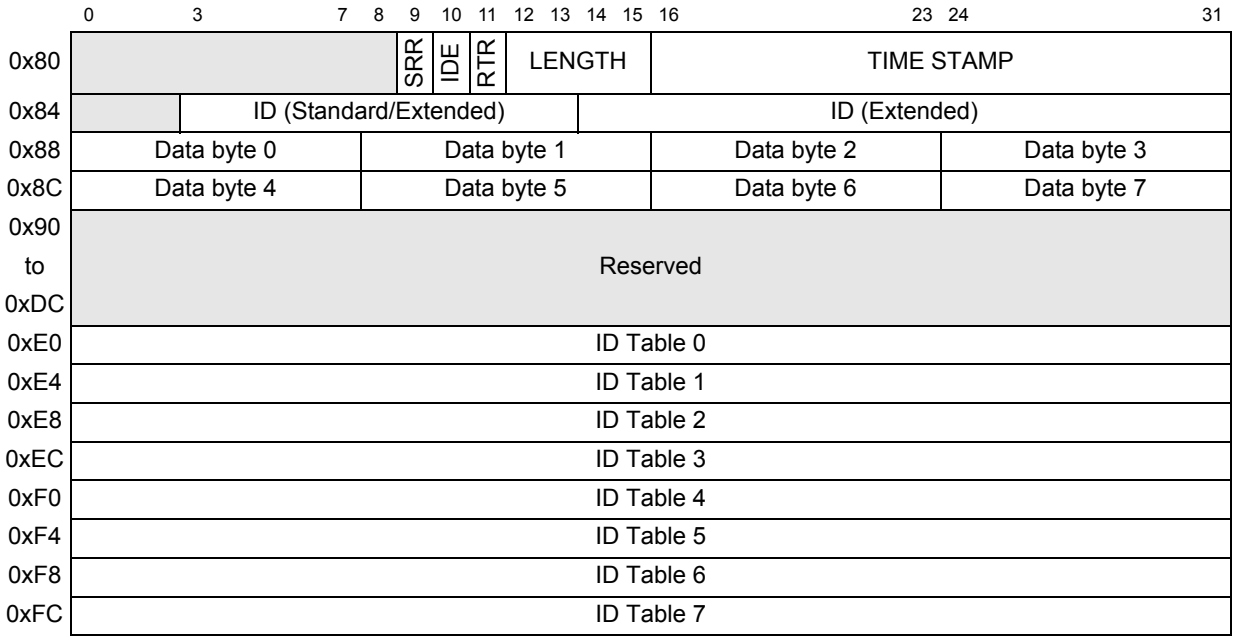


Figure 590. Rx FIFO structure

Figure 591. ID Table 0–7

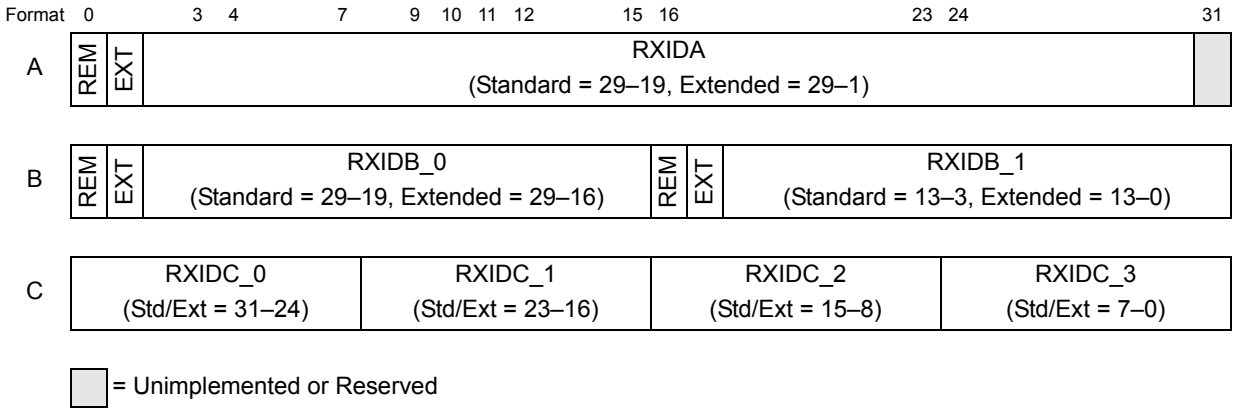


Table 402. Rx FIFO field descriptions

Field	Description
REM	Remote Frame This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID. 1 Remote Frames can be accepted and data frames are rejected 0 Remote Frames are rejected and data frames can be accepted

Table 402. Rx FIFO field descriptions (continued)

Field	Description
EXT	Extended Frame Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 1 Extended frames can be accepted and standard frames are rejected 0 Extended frames are rejected and standard frames can be accepted
RXIDA	Rx Frame Identifier (Format A) Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification. In the extended frame format, all bits are used.
RXIDB_0, RXIDB_1	Rx Frame Identifier (Format B) Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (3 to 13) are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.
RXIDC_0, RXIDC_1, RXIDC_2, RXIDC_3	Rx Frame Identifier (Format C) Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

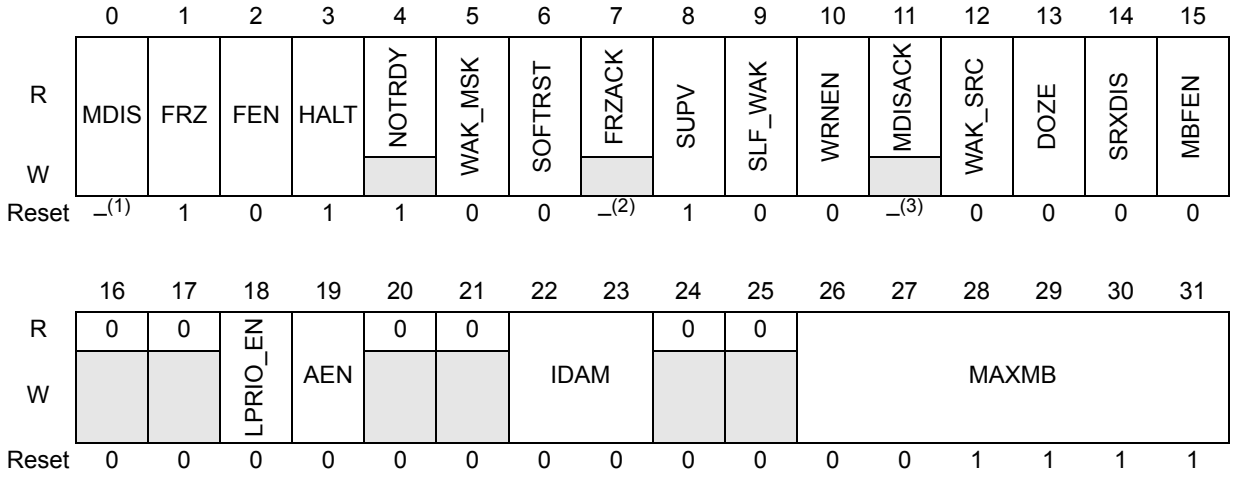
27.4.4 Register descriptions

The FlexCAN registers are described in this section in ascending address order.

27.4.4.1 Module Configuration Register (CAN_MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in Freeze Mode.

Figure 592. Module Configuration Register (CAN_MCR)



= Unimplemented or Reserved

1. Reset value of this bit is different on various platforms. Consult *Section 27.1: Information specific to this device* to determine its value.
2. Different on various platforms, but it is always the opposite of the MDIS reset value.
3. Different on various platforms, but it is always the same as the MDIS reset value.

Table 403. CAN_MCR field descriptions

Field	Description
0 MDIS	Module Disable This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clocks to the CAN Protocol Interface and Message Buffer Management submodules. This is the only bit in CAN_MCR not affected by soft reset. See *Section 27.5.9.2: Module Disable Mode* for more information. 1 Disable the FlexCAN module 0 Enable the FlexCAN module
1 FRZ	Freeze Enable The FRZ bit specifies the FlexCAN behavior when the HALT bit in the CAN_MCR is set or when Debug Mode is requested at MCU level. When FRZ is asserted, FlexCAN is enabled to enter Freeze Mode. Negation of this bit field causes FlexCAN to exit from Freeze Mode. 1 Enabled to enter Freeze Mode 0 Not enabled to enter Freeze Mode
2 FEN	FIFO Enable This bit controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0 to 7 cannot be used for normal reception and transmission because the corresponding memory region (0x80–0xFF) is used by the FIFO engine. See *Section 27.4.3: Rx FIFO structure* and *Section 27.5.7: Rx FIFO* for more information. 1 FIFO enabled 0 FIFO not enabled

Table 403. CAN_MCR field descriptions (continued)

Field	Description
3 HALT	Halt FlexCAN Assertion of this bit puts the FlexCAN module into Freeze Mode. The CPU should clear it after initializing the message buffers and Control Register. No reception or transmission is performed by FlexCAN before this bit is cleared. While in Freeze Mode, the CPU has write access to the Error Counter Register, that is otherwise read-only. Freeze Mode can not be entered while FlexCAN is in any of the low power modes. See *Section 27.5.9.1: Freeze Mode* for more information. 1 Enters Freeze Mode if the FRZ bit is asserted. 0 No Freeze Mode request.
4 NOTRDY	FlexCAN Not Ready This read-only bit indicates that FlexCAN is either in Disable Mode, Doze Mode, Stop Mode or Freeze Mode. It is negated once FlexCAN has exited these modes. 1 FlexCAN module is either in Disable Mode, Doze Mode, Stop Mode or Freeze Mode 0 FlexCAN module is either in Normal Mode, Listen-Only Mode or Loop-Back Mode
5 WAK_MSK	Wake Up Interrupt Mask This bit enables the Wake Up Interrupt generation. 1 Wake Up Interrupt is enabled 0 Wake Up Interrupt is disabled
6 SOFTRST	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: CAN_MCR (except the MDIS bit), CAN_TIMER, CAN_ECR, CAN_ESR, CAN_IMRL, CAN_IMRH, CAN_IFRL, CAN_IFRH. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected: – CAN_CR – CAN_RXIMR0–CAN_RXIMR63 – CAN_RXGMASK, CAN_RX14MASK, CAN_RX15MASK – all message buffers The SOFTRST bit can be asserted directly by the CPU when it writes to the CAN_MCR, but it is also asserted when global soft reset is requested at MCU level. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFTRST 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 397* 0 No reset request
7 FRZACK	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 FRZACK 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 FRZACK bit will only be set when the low power mode is exited. See *Section 27.5.9.1: Freeze Mode* for more information. 1 FlexCAN in Freeze Mode, prescaler stopped 0 FlexCAN not in Freeze Mode, prescaler running

Table 403. CAN_MCR field descriptions (continued)

Field	Description
8 SUPV	Supervisor Mode This bit configures some of the FlexCAN registers to be either in Supervisor or Unrestricted memory space. The registers affected by this bit are marked as S/U in the Access Type column of *Table 397*. Reset value of this bit is '1', so the affected registers start with Supervisor access restrictions. 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
9 SLF_WAK	Self Wake Up This bit enables the Self Wake Up feature when FlexCAN is in Doze Mode or Stop Mode. If this bit had been asserted by the time FlexCAN entered Doze Mode or Stop Mode, then FlexCAN will look for a recessive to dominant transition on the bus during these modes. If a transition from recessive to dominant is detected during Doze Mode, FlexCAN resumes its clocks and, if enabled to do so, generates a Wake Up interrupt to the CPU. If a transition from recessive to dominant is detected during Stop Mode, then FlexCAN generates, if enabled to do so, a Wake Up interrupt to the CPU so that it can resume the clocks globally. This bit can not be written while the module is in Doze Mode or Stop Mode. 1 FlexCAN Self Wake Up feature is enabled 0 FlexCAN Self Wake Up feature is disabled
10 WRNEN	Warning Interrupt Enable When asserted, this bit enables the generation of the TWRNINT and RWRNINT flags in the Error and Status Register (CAN_ESR). If WRNEN is negated, the TWRNINT and RWRNINT flags will always be zero, independent of the values of the error counters, and no warning interrupt will ever be generated. 1 TWRNINT and RWRNINT bits are set when the respective error counter transition from <96 to ≥ 96. 0 TWRNINT and RWRNINT bits are zero, independent of the values in the error counters.
11 MDISACK	Low Power Mode Acknowledge This read-only bit indicates that FlexCAN is either in Disable Mode, Doze 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 MDISACK bit to know when FlexCAN has actually entered low power mode. See *Section 27.5.9.2: Module Disable Mode*, *Section 27.5.9.3: Doze Mode*, and *Section 27.5.9.4: Stop Mode* for more information. 1 FlexCAN is either in Disable Mode, Doze Mode or Stop mode 0 FlexCAN not in any of the low power modes
12 WAK_SRC	Wake Up Source This bit defines whether the integrated low-pass filter is applied to protect the CAN_x_RX input from spurious wake up. See *Section 27.5.9.3: Doze Mode* and *Section 27.5.9.4: Stop Mode* for more information. 1 FlexCAN uses the filtered Rx input to detect recessive to dominant edges on the CAN bus 0 FlexCAN uses the unfiltered Rx input to detect recessive to dominant edges on the CAN bus. **Note:** The integrated low-pass filter may not be available in all MCUs. In case it is not available, the unfiltered input is always used for wake up purposes, and this bit has no effect on the FlexCAN operation. Please refer to *Section 27.1: Information specific to this device*.
13 DOZE	Doze Mode Enable This bit defines whether FlexCAN is allowed to enter low power mode when Doze Mode is requested at MCU level. This bit is automatically reset when FlexCAN wakes up from Doze Mode upon detecting activity on the CAN bus (self wake-up enabled). 1 FlexCAN is enabled to enter low power mode when Doze Mode is requested 0 FlexCAN is not enabled to enter low power mode when Doze Mode is requested

Table 403. CAN_MCR field descriptions (continued)

Field	Description
14 SRXDIS	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. 1 Self reception disabled 0 Self reception enabled
15 MBFEN	Message Buffer Enable (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 CAN_RXGMASK, CAN_RX14MASK and CAN_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. 1 Individual Rx masking and queue feature are enabled. 0 Individual Rx masking and queue feature are disabled.
16–17	Reserved
18 LPRIO_EN	Local Priority Enable This bit is provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It 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. 1 Local Priority enabled 0 Local Priority disabled
19 AEN	Abort Enable This bit is supplied for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification. 1 Abort enabled 0 Abort disabled
20–21	Reserved
22–23 IDAM	ID Acceptance Mode This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in *Table 404*. Note that all elements of the table are configured at the same time by this field (they are all the same format). See *Section 27.4.3: Rx FIFO structure*.
24–25	Reserved

Table 403. CAN_MCR field descriptions (continued)

Field	Description
26–31 MAXMB	Maximum Number of Message Buffers This 6-bit field defines the maximum number of message buffers that will take part in the matching and arbitration processes. The reset value (0x0F) is equivalent to 16 MB configuration. This field should be changed only while the module is in Freeze Mode. Maximum MBs in use = MAXMB + 1. **Note:** MAXMB must be programmed with a value smaller or equal to the number of available message buffers, otherwise FlexCAN can transmit and receive wrong messages.

Table 404. IDAM coding

IDAM	Format	Explanation
00	A	One full ID (standard or extended) per filter element.
01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.
10	C	Four partial 8-bit IDs (standard or extended) per filter element.
11	D	All frames rejected.

27.4.4.2 Control Register (CAN_CR)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, Loop Back Mode, Listen Only Mode, Bus Off recovery behavior and interrupt enabling (Bus-Off, Error, Warning). It also determines the Division Factor for the clock prescaler. Most of the fields in this register should only be changed while the module is in Disable Mode or in Freeze Mode. Exceptions are the BOFFMSK, ERRMSK, TWRNMSK, RWRNMSK and BOFFREC bits, that can be accessed at any time.

Figure 593. Control Register (CAN_CR)

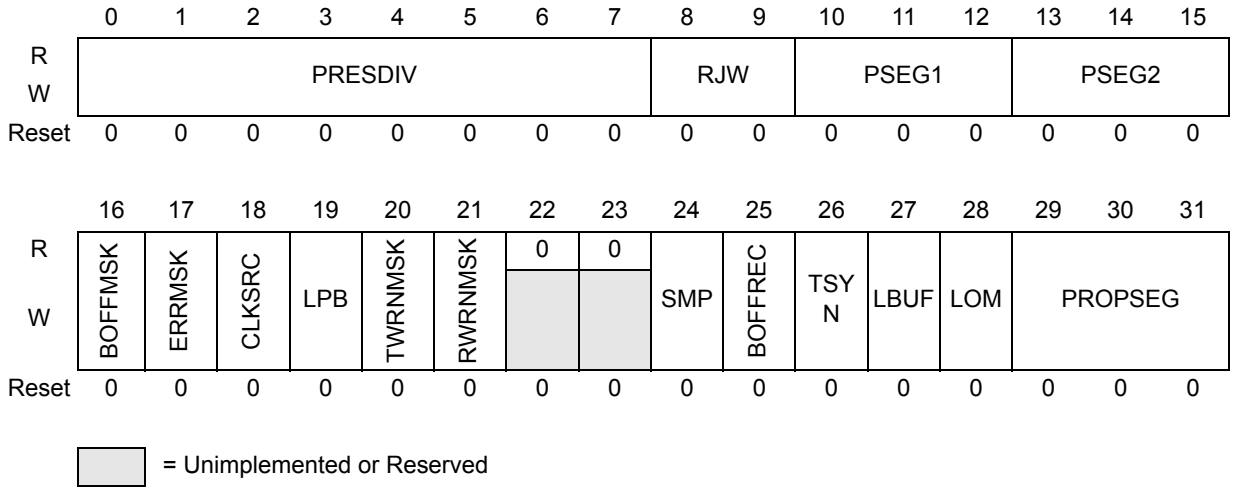


Table 405. CAN_CR field descriptions

Field	Description
0–7 PRESDIV	Prescaler Division Factor This 8-bit field defines the ratio between the CPI clock frequency and the Serial Clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The Maximum value of this register is 0xFF, that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information refer to *Section 27.5.8.4: Protocol timing*. Sclock frequency = CPI clock frequency / (PRESDIV + 1)
8–9 RJW	Resync Jump Width This 2-bit field defines the maximum number of time quanta(1) that a bit time can be changed by one re-synchronization. The valid programmable values are 0–3. Resync Jump Width = RJW + 1.
10–12 PSEG1	Phase Segment 1 This 3-bit field defines the length of Phase Buffer Segment 1 in the bit time. The valid programmable values are 0–7. Phase Buffer Segment 1 = (PSEG1 + 1) x Time-Quanta.
13–15 PSEG2	Phase Segment 2 This 3-bit field defines the length of Phase Buffer Segment 2 in the bit time. The valid programmable values are 1–7. Phase Buffer Segment 2 = (PSEG2 + 1) x Time-Quanta.
16 BOFFMSK	Bus Off Mask This bit provides a mask for the Bus Off Interrupt. 1 Bus Off interrupt enabled 0 Bus Off interrupt disabled
17 ERRMSK	Error Mask This bit provides a mask for the Error Interrupt. 1 Error interrupt enabled 0 Error interrupt disabled
18 CLKSRC	CAN Engine Clock Source This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the peripheral clock (driven by the PLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit should only be changed while the module is in Disable Mode. See *Section 27.5.8.4: Protocol timing* for more information. 1 The CAN engine clock source is the bus clock 0 The CAN engine clock source is the oscillator clock **Note:** This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, this bit has no effect on the module operation. Please refer to *Section 27.1: Information specific to this device*.

Table 405. CAN_CR field descriptions (continued)

Field	Description
19 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 CAN_x_RX input pin is ignored and the CAN_x_TX 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. 1 Loop Back enabled 0 Loop Back disabled
20 TWRNMSK	Tx Warning Interrupt Mask This bit provides a mask for the Tx Warning Interrupt associated with the TWRNINT flag in the Error and Status Register (CAN_ESR). This bit has no effect if CAN_MCR[WRNEN] is negated and it is read as zero when WRNEN is negated. 1 Tx Warning Interrupt enabled 0 Tx Warning Interrupt disabled
21 RWRNMSK	Rx Warning Interrupt Mask This bit provides a mask for the Rx Warning Interrupt associated with the RWRNINT flag in the Error and Status Register (CAN_ESR). This bit has no effect if CAN_MCR[WRNEN] is negated and it is read as zero when WRNEN is negated. 1 Rx Warning Interrupt enabled 0 Rx Warning Interrupt disabled
22–23	Reserved
24 SMP	Sampling Mode This bit defines the sampling mode of CAN bits at the Rx input. 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 Time-Quantum = one Sclock period.
25 BOFFREC	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 BOFFREC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN will resynchronize to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFFREC bit can be re-asserted again during Bus Off, but it will only be effective the next time the module enters Bus Off. If BOFFREC 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

Table 405. CAN_CR field descriptions (continued)

Field	Description
26 TSYN	Timer Sync Mode This bit enables a mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special “SYNC” message (that is, global network time). If the FEN bit in CAN_MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. 1 Timer Sync feature enabled 0 Timer Sync feature disabled
27 LBUF	Lowest Buffer Transmitted First This bit defines the ordering mechanism for message buffer transmission. When asserted, CAN_MCR[LPRIO_EN] does not affect the priority arbitration. 1 Lowest number buffer is transmitted first 0 Buffer with highest priority is transmitted first
28 LOM	Listen-Only Mode This bit configures FlexCAN to operate in Listen Only Mode. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode. 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. 1 FlexCAN module operates in Listen Only Mode 0 Listen Only Mode is deactivated
29–31 PROPSEG	Propagation Segment This 3-bit field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. Propagation Segment Time = (PROPSEG + 1) * Time-Quanta.

1. One time quantum is equal to the Sclock period.

27.4.4.3 Free Running Timer (CAN_TIMER)

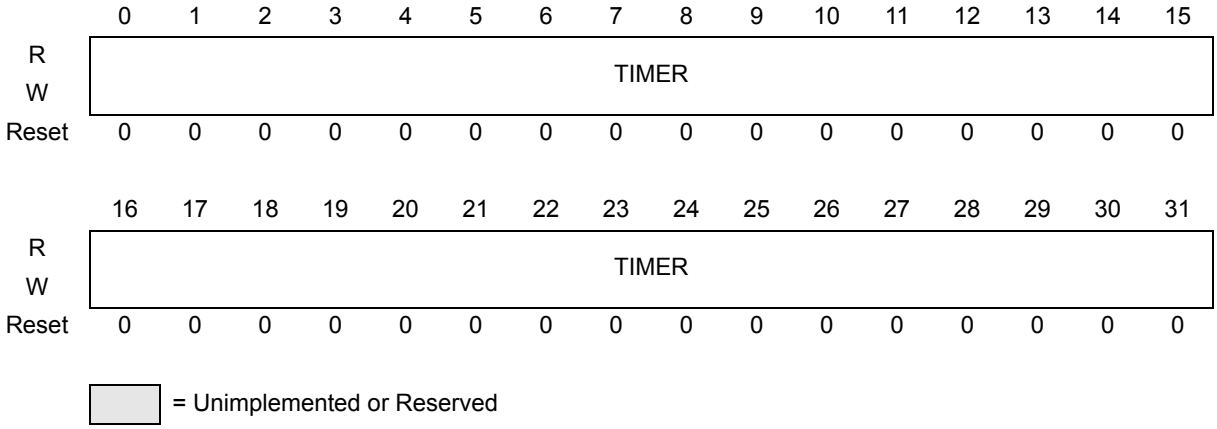
This register represents a 16-bit free running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Figure 594. Free Running Timer (CAN_TIMER)



27.4.4.4 Rx Global Mask (CAN_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 CAN_MCR[MBFEN] causes the CAN_RXGMASK Register to have no effect on the module operation. For MCUs not supporting individual masks per MB, this register is always effective.

CAN_RXGMASK is used as acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When CAN_MCR[FEN] is set (FIFO enabled), the CAN_RXGMASK also applies to all elements of the ID filter table, except elements 6–7, which have individual masks.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Note: If the FIFO is enabled and individual masks are not enabled (CAN_MCR[MBFEN] = 0) it is impossible to use the Global masks for FIFO and MB at same time. If FIFO is enabled, no MBs can be configured as RX.

Figure 595. Rx Global Mask Register (CAN_RXGMASK)

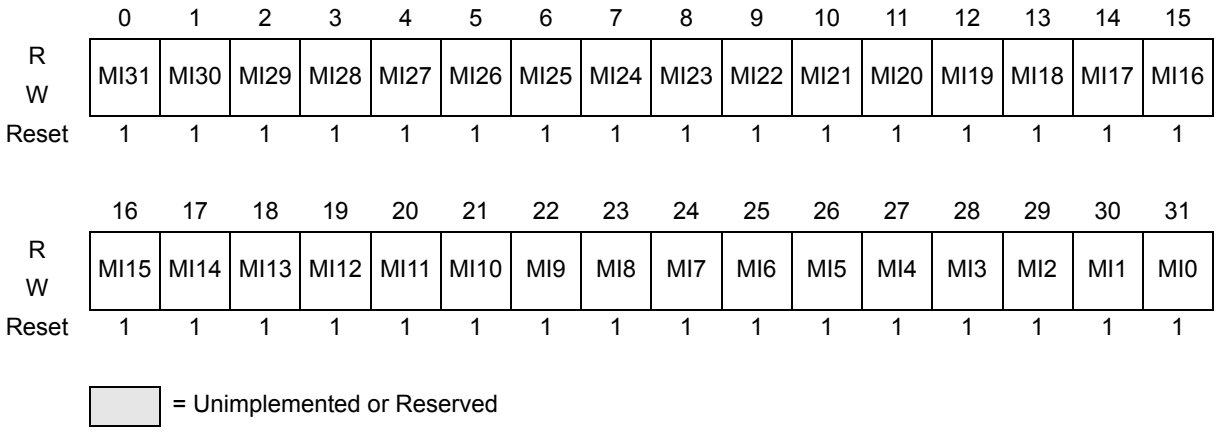


Table 406. CAN_RXGMASK field descriptions

Field	Description
MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 1 The corresponding bit in the filter is checked against the one received 0 The corresponding bit in the filter is “don’t care”

27.4.4.5 Rx 14 Mask (CAN_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 CAN_MCR[MBFEN] causes the CAN_RX14MASK Register to have no effect on the module operation.

CAN_RX14MASK is used as acceptance mask for the Identifier in Message Buffer 14. When CAN_MCR[FEN] 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. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address offset: 0x14
- Reset value: 0xFFFF_FFFF

Note: It is impossible to use the Rx 14 mask for FIFO and MB14 at same time. If FIFO is enabled and CAN_MCR[MBFEN] = 0, MB14 cannot be configured as RX.

27.4.4.6 Rx 15 Mask (CAN_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 CAN_MCR[MBFEN] causes the CAN_RX15MASK Register to have no effect on the module operation.

When CAN_MCR[MBFEN] is negated, CAN_RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When CAN_MCR[FEN] 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. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address offset: 0x18
- Reset value: 0xFFFF_FFFF

Note: It is impossible to use the Rx 15 mask for FIFO and MB15 at same time. If FIFO is enabled and CAN_MCR[MBFEN] = 0, MB15 cannot be configured as RX.

27.4.4.7 Error Counter Register (CAN_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 FLTCONF field in the Error and Status Register (CAN_ESR) 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 FLTCONF field in the CAN_ESR is updated to reflect ‘Error Active’ state.
- If the value of Tx_Err_Counter increases to be greater than 255, the FLTCONF field in the CAN_ESR 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 FLTCONF field in the CAN_ESR 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 CAN_ESR[ACKERR]). After the transition to ‘Error Passive’ state, the Tx_Err_Counter does not increment anymore by acknowledge errors. Therefore the device never goes to the ‘Bus Off’ state.
- If the Rx_Err_Counter increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to ‘Error Active’ state.

Figure 596. Error Counter Register (CAN_ECR)

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

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

= Unimplemented or Reserved

27.4.4.8 Error and Status Register (CAN_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 bits16–23. Bits 22–27 are status bits.

Most bits in this register are read only, except TWRNINT, RWRNINT, BOFFINT, WAK_INT and ERRINT, that are interrupt flags that can be cleared by writing ‘1’ to them (writing ‘0’ has no effect). See *Section 27.5.10: Interrupts* for more details.

Figure 597. Error and Status Register (CAN_ESR)

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BIT1ERR	BIT0ERR	ACKERR	CRCERR	FRMERR	STFERR	TXWRN	RXWRN	IDLE	TXRX	FLTCONF		0	BOFFINT	ERRINT	WAK_INT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Table 407. CAN_ESR field descriptions

Field	Description
0–13	Reserved

Table 407. CAN_ESR field descriptions (continued)

Field	Description
14 TWRNINT	Tx Warning Interrupt Flag If CAN_MCR[WRNEN] is asserted, the TWRNINT bit is set when the TXWRN flag transition from '0' to '1', meaning that the Tx error counter reached 96. If the corresponding mask bit (CAN_CR[TWRNMSK]) 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
15 RWRNINT	Rx Warning Interrupt Flag If CAN_MCR[WRNEN] is asserted, the RWRNINT bit is set when the RXWRN flag transition from '0' to '1', meaning that the Rx error counters reached 96. If the corresponding mask bit (CAN_CR[RWRNMSK]) 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
16 BIT1ERR	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.
17 BIT0ERR	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
18 ACKERR	Acknowledge Error This bit indicates that an Acknowledge Error has been detected by the transmitter node, that is, a dominant bit has not been detected during the ACK slot. 1 An ACK error occurred since last read of this register 0 No such occurrence
19 CRCERR	Cyclic Redundancy Check Error This bit indicates that a CRC Error has been detected by the receiver node, that is, the calculated CRC is different from the received. 1 A CRC error occurred since last read of this register 0 No such occurrence
20 FRMERR	Form Error This bit indicates that a Form Error has been detected by the receiver node, that is, a fixed-form bit field contains at least one illegal bit. 1 A Form Error occurred since last read of this register 0 No such occurrence
21 STFERR	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.

Table 407. CAN_ESR field descriptions (continued)

Field	Description
22 TXWRN	TX Error Counter This bit indicates when repetitive errors are occurring during message transmission. 1 TX_Err_Counter ≥ 96 0 No such occurrence
23 RXWRN	Rx Error Counter This bit indicates when repetitive errors are occurring during message reception. 1 Rx_Err_Counter ≥ 96 0 No such occurrence
24 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
25 TXRX	Current FlexCAN status (transmitting/receiving) This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 1 FlexCAN is transmitting a message (IDLE = 0) 0 FlexCAN is receiving a message (IDLE = 0)
26–27 FLTCONF	Fault Confinement State This 2-bit field indicates the Confinement State of the FlexCAN module, as shown in *Table 402*. If CAN_CR[LOM] is asserted, the FLTCONF field will indicate “Error Passive”. Since the Control Register is not affected by soft reset, the FLTCONF field will not be affected by soft reset if CAN_CR[LOM] is asserted. 00 Error Active 01 Error Passive 1X Bus Off
28	Reserved
29 BOFFINT	‘Bus Off’ Interrupt This bit is set when FlexCAN enters ‘Bus Off’ state. If the corresponding mask bit (CAN_CR[BOFFMSK]) 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
30 ERRINT	Error Interrupt This bit indicates that at least one of the Error Bits (bits 16-21) is set. If the corresponding mask bit (CAN_CR[ERRMSK]) 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
31 WAK_INT	Wake-Up Interrupt When FlexCAN is in Doze Mode or Stop Mode and a recessive to dominant transition is detected on the CAN bus and if CAN_MCR[WAK_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 a recessive to dominant transition received on the CAN bus when the FlexCAN module is in Doze Mode or Stop Mode 0 No such occurrence

27.4.4.9 Interrupt Masks 2 Register (CAN_IMRH)

This register allows any number of a range of 32 message buffer interrupts to be enabled or disabled. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (that is, when the corresponding CAN_IFRH register bit is set).

Figure 598. Interrupt Masks 2 Register (CAN_IMRH)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63M	BUF 62M	BUF 61M	BUF 60M	BUF 59M	BUF 58M	BUF 57M	BUF 56M	BUF 55M	BUF 54M	BUF 53M	BUF 52M	BUF 51M	BUF 50M	BUF 49M	BUF 48M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 47M	BUF 46M	BUF 45M	BUF 44M	BUF 43M	BUF 42M	BUF 41M	BUF 40M	BUF 39M	BUF 38M	BUF 37M	BUF 36M	BUF 35M	BUF 34M	BUF 33M	BUF 32M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 408. CAN_IMRH field descriptions

Field	Description
BUF63M–BUF32M	Buffer MB_i Mask Each bit enables or disables the respective FlexCAN message buffer (MB32 to MB63) Interrupt. 1 The corresponding buffer Interrupt is enabled 0 The corresponding buffer Interrupt is disabled

Note: Setting or clearing a bit in the CAN_IMRH register can assert or negate an interrupt request, if the corresponding CAN_IFRH register bit is set.

27.4.4.10 Interrupt Masks 1 Register (CAN_IMRL)

This register allows to enable or disable any number of a range of 32 message buffer interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (that is, when the corresponding CAN_IFRL register bit is set).

Figure 599. Interrupt Masks 1 Register (CAN_IMRL)

	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 09M	BUF 08M	BUF 07M	BUF 06M	BUF 05M	BUF 04M	BUF 03M	BUF 02M	BUF 01M	BUF 00M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 409. CAN_IMRL field descriptions

Field	Description
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 CAN_IMRL Register can assert or negate an interrupt request, if the corresponding CAN_IFRL register bit is set.

27.4.4.11 Interrupt Flags 2 Register (CAN_IFRH)

This register defines the flags for 32 message buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding CAN_IFRH bit. If the corresponding CAN_IMRH register 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 CAN_MCR[AEN] is set (Abort enabled), while the CAN_IFRH register bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

Figure 600. Interrupt Flags 2 Register (CAN_IFRH)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63I	BUF 62I	BUF 61I	BUF 60I	BUF 59I	BUF 58I	BUF 57I	BUF 56I	BUF 55I	BUF 54I	BUF 53I	BUF 52I	BUF 51I	BUF 50I	BUF 49I	BUF 48I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 47I	BUF 46I	BUF 45I	BUF 44I	BUF 43I	BUF 42I	BUF 41I	BUF 40I	BUF 39I	BUF 38I	BUF 37I	BUF 36I	BUF 35I	BUF 34I	BUF 33I	BUF 32I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 410. CAN_IFRH field descriptions

Field	Description
BUF32I–BUF63I	Buffer MB_i Interrupt Each bit flags the respective FlexCAN message buffer (MB32 to MB63) interrupt. 1 The corresponding buffer has successfully completed transmission or reception 0 No such occurrence

27.4.4.12 Interrupt Flags 1 Register (CAN_IFRL)

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 CAN_IFRL bit. If the corresponding CAN_IMRL register 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 CAN_MCR[AEN] is set (Abort enabled), while the CAN_IFRL register bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

When CAN_MCR[FEN] is set (FIFO enabled), the function of the 8 least significant interrupt flags (BUF7I - BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Figure 601. Interrupt Flags 1 Register (CAN_IFRL)

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15I	BUF 14I	BUF 13I	BUF 12I	BUF 11I	BUF 10I	BUF 09I	BUF 08I	BUF 07I	BUF 06I	BUF 05I	BUF 04I	BUF 03I	BUF 02I	BUF 01I	BUF 00I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 411. CAN_IFRL field descriptions

Field	Description
BUF31I–BUF8I	Buffer MB_i Interrupt Each bit flags the respective FlexCAN message buffer (MB8 to MB31) interrupt. 1 The corresponding MB has successfully completed transmission or reception 0 No such occurrence
BUF7I	Buffer MB7 Interrupt or “FIFO Overflow” If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full). 1 MB7 completed transmission/reception or FIFO overflow 0 No such occurrence
BUF6I	Buffer MB6 Interrupt or “FIFO Warning” If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 5 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 1 MB6 completed transmission/reception or FIFO almost full 0 No such occurrence
BUF5I	Buffer MB5 Interrupt or “Frames available in FIFO” If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO. 1 MB5 completed transmission/reception or frames available in the FIFO 0 No such occurrence
BUF4I–BUF0I	Buffer MB_i Interrupt or “reserved” If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations. 1 Corresponding MB completed transmission/reception 0 No such occurrence

27.4.4.13 Rx Individual Mask Registers (CAN_RXIMR0–CAN_RXIMR63)

These registers are used as acceptance masks for ID filtering in Rx MBs and the FIFO. If the FIFO is not enabled, one mask register is provided for each available message buffer, providing ID masking capability on a per message buffer basis. When the FIFO is enabled (CAN_MCR[FEN] is set), the first eight Mask Registers apply to the eight elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze Mode. Out of Freeze Mode, write accesses are blocked and read accesses will return “all zeros”. Furthermore, if CAN_MCR[MBFEN] is negated, any read or write operation to these registers results in access error.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. If not supported, the CAN_RXGMASK, CAN_RX14MASK and CAN_RX15MASK registers are available, regardless of CAN_MCR[MBFEN]’s value.

Figure 602. Rx Individual Mask Registers (CAN_RXIMR0–CAN_RXIMR63)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0

Table 412. CAN_RXIMR0–CAN_RXIMR63 field descriptions

Field	Description
MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 1 The corresponding bit in the filter is checked against the one received 0 The corresponding bit in the filter is “don’t care”

27.5 Functional description

27.5.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed of a set of up to 64 message buffers (MB) that store configuration and control data, time stamp, message ID and data (see *Section 27.4.2: Message buffer structure*). The memory corresponding to the first eight 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 thirty-two 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 400*). Similarly, a Tx MB with a ‘1000’ or ‘1001’ code is also inactive (refer to *Table 401*). An MB not programmed with ‘0000’, ‘1000’ or ‘1001’ will be temporarily deactivated (will not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see *Section 27.5.6.2: Message buffer deactivation*).

27.5.2 Transmit process

In order to transmit a CAN frame, the CPU must prepare a message buffer for transmission by executing the following procedure:

1. If the message buffer is active (transmission pending), write ‘1000’ to the Code field to inactivate the message buffer. The deactivated message buffer can transmit without setting IFLAG and without updating the CODE field. (see *Section 27.5.6.2: Message buffer deactivation*).
2. Write the ID word.
3. Write the data bytes.
4. Write the Length, Control and Code fields of the Control and Status word to activate the MB.

Once the MB is activated in step 4, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit. The new Code field after transmission depends on the code that was used to activate the MB in step 4 (see *Table 400* and *Table 401* in *Section 27.4.2: Message buffer structure*). When the Abort feature is enabled (CAN_MCR[AEN] is asserted), after the Interrupt Flag is asserted for a MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to update it until the Interrupt Flag be negated by CPU. It means that the CPU must clear the corresponding CAN_IFRL or CAN_IFRH register before starting to prepare this MB for a new transmission or reception.

27.5.3 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[v] or the lowest MB number or the

v. Actually, if CAN_CR[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 CAN_CR[LBUF] and CAN_MCR[LPRIO_EN]. 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 CAN_CR[LBUF] is asserted, CAN_MCR[LPRIO_EN] has no effect and the lowest number buffer is transmitted first. When CAN_CR[LBUF] and CAN_MCR[LPRIO_EN] are both negated, the MB with the lowest ID is transmitted first but. If CAN_CR[LBUF] is negated and CAN_MCR[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 CAN_MCR[AEN] 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.

27.5.4 Receive process

To be able to receive CAN frames into the mailbox MBs, the CPU must prepare one or more message buffers for reception by executing the following steps:

1. If the MB has a pending transmission, write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the CAN_IFRL or CAN_IFRH register to check if the transmission was aborted (see *Section 27.5.6.1: Transmission abort mechanism*). If backwards compatibility is desired (CAN_MCR[AEN] negated), just write ‘1000’ to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification (see *Section 27.5.6.2: 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.
2. Write the ID word
3. Write ‘0100’ to the Code field of the Control and Status word to activate the MB

Once the MB is activated in step 3, 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:

1. The value of the Free Running Timer is written into the Time Stamp field
2. The received ID, Data (8 bytes at most) and Length fields are stored
3. The Code field in the Control and Status word is updated (see *Table 400* and *Table 401* in *Section 27.4.2: Message buffer structure*)
4. A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (mandatory – activates an internal lock for this buffer)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Read the Free Running Timer (optional – releases the internal lock)

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the MB until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see *Section 27.5.6: Data coherence*).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in the corresponding CAN_IFRL or CAN_IFRH register 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 400*. If the CPU tries to work around this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary: **never do polling by reading directly the C/S word of the MBs. Instead, read the corresponding CAN_IFRL or CAN_IFRH register.**

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 CAN_MCR[SRXDIS] is not asserted. If CAN_MCR[SRXDIS] is asserted, FlexCAN will not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal will be generated due to the frame reception.

To be able to receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during Freeze Mode (see *Section 27.5.7: Rx FIFO*). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

27.5.5 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 and CAN_MCR[MBFEN] = 1, 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 27.5.6.3: 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 400* and *Table 401*). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see *Section 27.5.6.3: 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 CAN_MCR[MBFEN] 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 CAN_MCR[MBFEN] 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 27.4.4.13: Rx Individual Mask Registers (CAN_RXIMR0–CAN_RXIMR63)*. During the matching algorithm, if a mask bit is asserted,

then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is “don’t care”. Please note that the Individual Mask Registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if CAN_MCR[MBFEN] is asserted and while the module is in Freeze Mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, CAN_RX14MASK and CAN_RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when CAN_MCR[MBFEN] is negated. To use the global masks, if FIFO is enabled, no MBs can be configured as RX.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. If not supported, the CAN_RXGMASK, CAN_RX14MASK and CAN_RX15MASK registers are available, regardless of CAN_MCR[MBFEN]’s value.

27.5.6 Data coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in *Section 27.5.2: Transmit process* and *Section 27.5.4: 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.

27.5.6.1 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 CAN_MCR[AEN].

In order to abort a transmission, the CPU must write a specific abort code (1001) to the Code field of the Control and Status word. When the abort mechanism is enabled, the active MBs configured as transmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that was already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into Freeze Mode

If none of conditions above are reached, the MB is transmitted correctly, the interrupt flag is set in the corresponding CAN_IFRL or CAN_IFRH 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 corresponding CAN_IFRL or CAN_IFRH register 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 CAN_MCR[AEN] 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 CAN_IFRL or CAN_IFRH register to check if the frame was transmitted or it is being currently transmitted. If the corresponding CAN_IFRL or CAN_IFRH register is set, the frame was transmitted. If the corresponding CAN_IFRL or CAN_IFRH register 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 into its CODE field. Therefore, the TxMB cannot be aborted or deactivated until it completes a transmission by winning the CAN bus arbitration.

27.5.6.2 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 27.5.6.1: Transmission abort mechanism* should be used.

27.5.6.3 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[w] (‘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 (CAN_ESR).

While the message is being moved-in from the SMB to the MB, the BUSY bit on the Code field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

Note: If the BUSY bit is asserted or if the MB is empty, then reading the Control and Status word does not lock the MB.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the MB.

27.5.7 Rx FIFO

The receive-only FIFO is enabled by asserting CAN_MCR[FEN]. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80–0xFF) is now reserved for use of the FIFO engine (see *Section 27.4.3: Rx FIFO structure*). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a message buffer structure at the beginning of the memory.

The FIFO can store up to 6 frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80 with the

w. In previous FlexCAN versions, reading the C/S word locked the message buffer even if it was EMPTY. In current FlexCAN versions, this behavior is maintained when CAN_MCR[MBFEN] is negated”.

next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. A warning interrupt is also generated when 5 frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of eight 32-bit registers that can be configured to one of the following formats (see also *Section 27.4.3: Rx FIFO structure*):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

Note: A chosen format is applied to all eight registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight Individual Mask Registers (CAN_RXIMR0–CAN_RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the CAN_RXIMR, starting from RXIM8, continue to affect the regular MBs, starting from MB8. If CAN_MCR[MBFEN] is negated (or if the CAN_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 CAN_RX14MASK, element 7 is affected by CAN_RX15MASK and the other elements (0 to 5) are affected by CAN_RXGMASK.

27.5.7.1 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. *Table 413* table shows recommended actions depending on FEN and BCC settings.

Table 413. Recommended FEN and BCC settings

Case	MCR[FEN] RxFIFO	MCR[BCC] Rx Individual Mask	Notes
Case 1	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: Do not use RXGMASK, RX14MASK, and RX15MASK in this case, leave the masks in their reset state.

Table 413. Recommended FEN and BCC settings (continued)

Case	MCR[FEN] RxFIFO	MCR[BCC] Rx Individual Mask	Notes
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, 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. Thus, RXGMASK, RX14MASK, and RX15MASK are not used. Particularly, when MCR[FEN] = 0, RxFIFO is disabled; RXGMASK, RX14MASK, and RX15MASK do not affect filtering. Individual masks are used.

27.5.8 CAN protocol related features

27.5.8.1 Remote frames

Remote frame is a special kind of frame. The user can program a MB to be a Request Remote Frame by writing the MB as Transmit with the RTR bit set to '1'. After the Remote Request frame is transmitted successfully, the MB becomes a Receive Message Buffer, with the same ID as before.

When a Remote Request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the Code field '1010'. If there is a matching ID, then this MB frame will be transmitted. Note that if the matching MB has the RTR bit set, then FlexCAN will transmit a Remote Frame as a response.

A received Remote Request Frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a Remote Request Frame was received and matched an MB, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

If the Rx FIFO is enabled (CAN_MCR[FEN] is set), FlexCAN will not generate an automatic response for Remote Request Frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it will be stored in the FIFO and presented to the CPU. Note that for filtering formats A and B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID).

27.5.8.2 Overload frames

FlexCAN does transmit overload frames due to detection of following conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of Intermission
- Detection of a dominant bit at the 7th bit (last) of End of Frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of Error Frame Delimiter or Overload Frame Delimiter

27.5.8.3 Time stamp

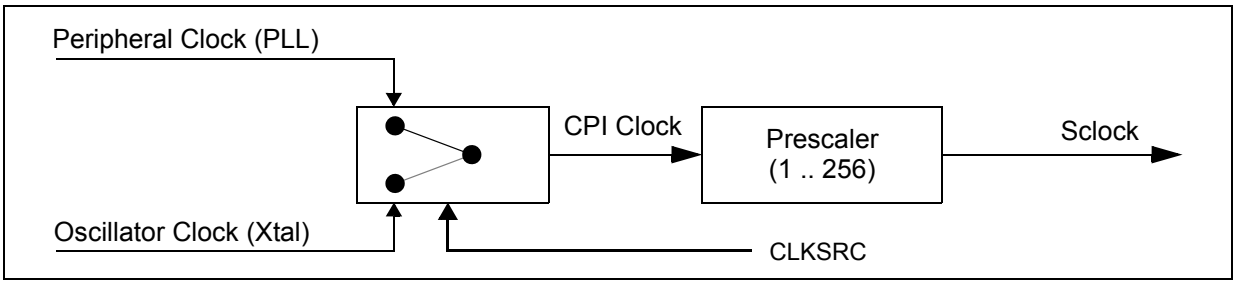
The value of the Free Running Timer is sampled at the beginning of the Identifier field on the CAN bus, and is stored at the end of “move-in” in the TIME STAMP field, providing network behavior with respect to time.

Note that the Free Running Timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in *Section 27.4.4.2: Control Register (CAN_CR)*.

27.5.8.4 Protocol timing

Figure 603 shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) submodule. The clock source bit (CAN_CR[CLKSRC]) defines whether the internal clock is connected to the output of a crystal oscillator (oscillator clock) or to the peripheral clock (generally from a PLL). In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (CAN_MCR[MDIS] is set).

Figure 603. CAN engine clocking scheme



The crystal oscillator clock should be selected whenever a tight tolerance (up to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks.

Note: This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, CAN_CR[CLKSRC] has no effect on the module operation.

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW. See *Section 27.4.4.2: Control Register (CAN_CR)*.

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.

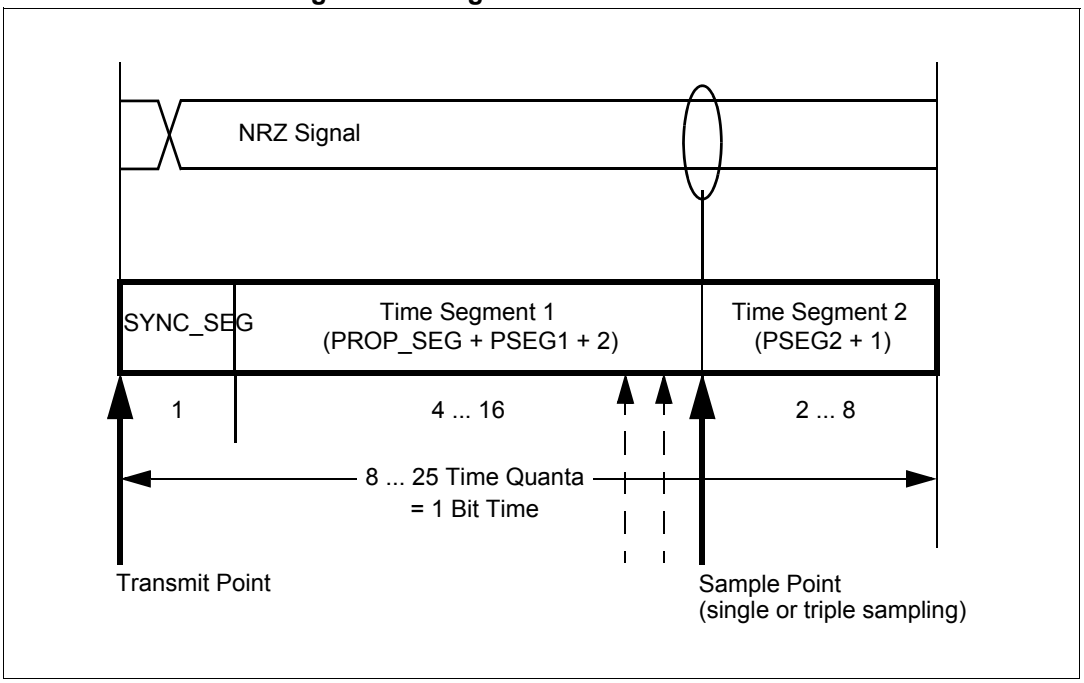
$$f_{Tq} = \frac{f_{CANCLK}}{(Prescaler\ \ Value)}$$

A bit time is subdivided into three segments[x] (reference *Figure 604* and *Table 414*):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section
- Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CAN_CR 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 CAN_CR (plus 1) to be 2 to 8 time quanta long

$$Bit\ \ Rate = \frac{f_{Tq}}{(number\ \ of\ \ Time\ \ Quanta)}$$

Figure 604. Segments within the bit time



x. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

Table 414. 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 415 gives an overview of the CAN compliant segment settings and the related parameter values.

Table 415. CAN standard compliant bit time segment settings

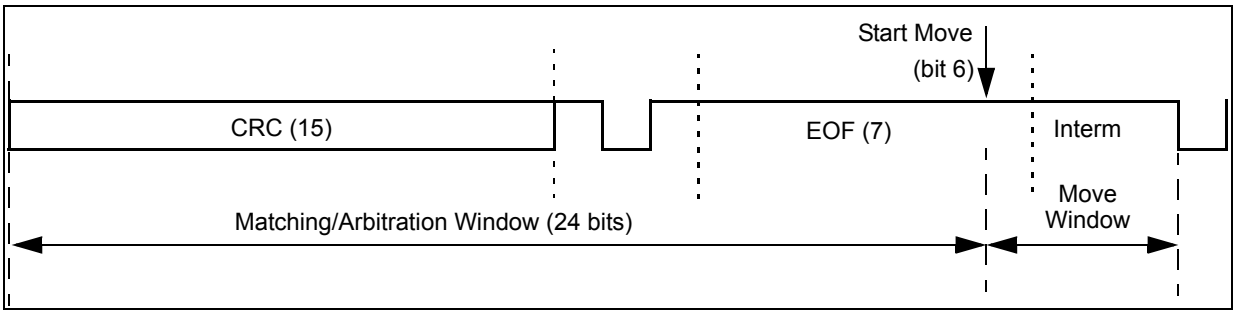
Time Segment 1	Time segment 2	Resynchronization jump width
5...10	2	1 .. 2
4...11	3	1 .. 3
5...12	4	1 .. 4
6...13	5	1 .. 4
7...14	6	1 .. 4
8...15	7	1 .. 4
9...16	8	1 .. 4

Note: It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module.

27.5.8.5 Arbitration and matching timing

During normal transmission or reception of frames, the arbitration, matching, move-in and move-out processes are executed during certain time windows inside the CAN frame, as shown in *Figure 605*.

Figure 605. 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 415*
- The peripheral clock frequency can not be smaller than the oscillator clock frequency, that is, the PLL can not be programmed to divide down the oscillator clock
- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in *Table 416*

Table 416. 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 416* can be achieved by choosing a high enough peripheral clock frequency when compared to the oscillator clock frequency, or by adjusting one or more of the bit timing parameters (PRESDIV, PROPSEG, PSEG1, PSEG2). As an example, taking the case of 64 MBs, if the oscillator and peripheral clock frequencies are equal and the CAN bit timing is programmed to have 8 time quanta per bit, then the prescaler factor (PRESDIV + 1) should be at least 2. For prescaler factor equal to one and CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

27.5.9 Modes of operation details

27.5.9.1 Freeze Mode

This mode is entered by asserting CAN_MCR[HALT] or when the MCU is put into Debug Mode. In both cases it is also necessary that CAN_MCR[FRZ] is asserted and the module is not in any of the low power modes (Disable, Doze, 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 NOTRDY and FRZACK bits in CAN_MCR

After requesting Freeze Mode, the user must wait for CAN_MCR[FRZACK] to be asserted 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 CAN_MCR[FRZ]
- The MCU is removed from Debug Mode and/or CAN_MCR[HALT] is negated

Once out of Freeze Mode, FlexCAN tries to resynchronize to the CAN bus by waiting for 11 consecutive recessive bits.

27.5.9.2 Module Disable Mode

This low power mode is entered when CAN_MCR[MDIS] is asserted by the CPU. When the FlexCAN module is disabled during Freeze Mode, the module sends a request to disable the clocks to the CPI and MBM submodules, sets CAN_MCR[MDISACK] and negates CAN_MCR[FRZACK]. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM submodules
- Sets the NOTRDY and MDISACK bits in CAN_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 CAN_MCR[MDIS], which will resume the clocks and negate CAN_MCR[MDISACK].

27.5.9.3 Doze Mode

This is a system low power mode in which the CPU bus is kept alive and a global Doze Mode request is sent to all peripherals asking them to enter low power mode. When Doze Mode is globally requested, the DOZE bit in CAN_MCR needs to have been asserted previously for Doze Mode to be triggered. If Doze Mode is triggered during Freeze Mode, FlexCAN shuts down the clocks to the CPI and MBM submodules, sets CAN_MCR[MDISACK] and negates CAN_MCR[FRZACK]. If Doze Mode is triggered 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
- Shuts down the clocks to the CPI and MBM submodules
- Sets the NOTRDY and MDISACK bits in CAN_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 can not be accessed in Doze Mode.

Exiting Doze Mode is done in one of the following ways:

- CPU removing the Doze Mode request
- CPU negating the DOZE bit of the CAN_MCR
- Self Wake mechanism

In the Self Wake mechanism, if CAN_MCR[SLF_WAK] was set at the time FlexCAN entered Doze Mode, then upon detection of a recessive to dominant transition on the CAN bus, FlexCAN negates the DOZE bit and resumes its clocks. It also sets CAN_ESR[WAK_INT] and, if enabled by CAN_MCR[WAK_MSK], generates a Wake Up interrupt to the CPU.

FlexCAN will then wait for 11 consecutive recessive bits to synchronize to the CAN bus. As a consequence, it will not receive the frame that woke it up. *Table 417* details the effect of SLF_WAK and WAK_MSK upon wake-up from Doze Mode.

Table 417. Wake-up from doze mode

SLF_WAK	WAK_MSK	FlexCAN clocks enabled	Wake-up interrupt generated
0	0	No	No
0	1	No	No
1	0	Yes	No
1	1	Yes	Yes

The sensitivity to CAN bus activity can be modified by applying a low-pass filter function to the CAN_x_RX input line while in Doze Mode. See the WAK_SRC bit in *Section 27.4.4.1: Module Configuration Register (CAN_MCR)*. This feature can be used to protect FlexCAN from waking up due to short glitches on the CAN bus lines. Such glitches can result from electromagnetic interference within noisy environments.

Note: Not all MCUs are equipped with the low-pass filter.

27.5.9.4 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 CAN_MCR[MDISACK], negates CAN_MCR[FRZACK] 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 NOTRDY and MDISACK bits in CAN_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
- CPU resuming the clocks and Stop Mode request as a result of the Self Wake mechanism

In the Self Wake mechanism, if CAN_MCR[SLF_WAK] was set at the time FlexCAN entered Stop Mode, then upon detection of a recessive to dominant transition on the CAN bus, FlexCAN sets CAN_ESR[WAK_INT] and, if enabled by CAN_MCR[WAK_MSK], generates a Wake Up interrupt to the CPU. Upon receiving the interrupt, the CPU should resume the clocks and remove the Stop Mode request. FlexCAN will then wait for 11 consecutive recessive bits to synchronize to the CAN bus. As a consequence, it will not receive the frame that woke it up. *Table 418* details the effect of SLF_WAK and WAK_MSK upon wake-up from Stop Mode. Note that wake-up from Stop Mode only works when both bits are asserted.

Table 418. Wake-up from stop mode

SLF_WAK	WAK_MSK	MCU clocks enabled	Wake-up interrupt generated
0	0	No	No
0	1	No	No
1	0	No	No
1	1	Yes	Yes

The sensitivity to CAN bus activity can be modified by applying a low-pass filter function to the CAN_x_RX input line while in Stop Mode. See the WAK_SRC bit in *Section 27.4.4.1: Module Configuration Register (CAN_MCR)*. This feature can be used to protect FlexCAN from waking up due to short glitches on the CAN bus lines. Such glitches can result from electromagnetic interference within noisy environments.

Note: Not all MCUs are equipped with the low-pass filter.

27.5.10 Interrupts

The module can generate up to 70 interrupt sources (64 interrupts due to message buffers and 6 interrupts due to Ored interrupts from MBs, Bus Off, Error, Tx Warning, Rx Warning and Wake Up). 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 bit in the CAN_IMRL or CAN_IMRH register 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 corresponding CAN_IFRL or CAN_IFRH register. 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 (CAN_MCR[FEN] is set), the interrupts corresponding to MBs 0 to 7 have a different behavior. Bit 7 of the CAN_IFRL becomes the “FIFO Overflow” flag; bit 6 becomes the FIFO Warning flag, bit 5 becomes the “Frames Available in FIFO flag” and bits 4–0 are unused. See *Section 27.4.4.12: Interrupt Flags 1 Register (CAN_IFRL)* 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 corresponding CAN_IFRL or CAN_IFRH register to determine which MB caused the interrupt.

The other 5 interrupt sources (Bus Off, Error, Tx Warning, Rx Warning and Wake Up) generate interrupts like the MB ones, and can be read from the Error and Status Register (CAN_ESR). The Bus Off, Error, Tx Warning and Rx Warning interrupt mask bits are located in the Control Register, and the Wake-Up interrupt mask bit is located in the CAN_MCR.

27.5.11 Bus interface

The CPU access to FlexCAN registers are subject to the following rules:

- Read and write access to supervisor registers in User Mode results in access error.
- Read and write access to unimplemented or reserved address space also results in access error. Any access to unimplemented MB or Rx Individual Mask Register locations results in access error. Any access to the Rx Individual Mask Register space when CAN_MCR[MBFEN] is negated results in access error.
- If MAXMB is programmed with a value smaller than the available number of MBs, then the unused memory space can be used as general purpose RAM space. Note that the Rx Individual Mask Registers can only be accessed in Freeze Mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 64 MBs and MAXMB is programmed with zero. The maximum number of MBs in this case becomes one. The MB memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one MB. This leaves us with the available space from 0x0090 to 0x047F. The available memory in the Mask Registers space would be from 0x0884 to 0x097F.

Note: Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

27.6 Initialization/application information

This section provide instructions for initializing the FlexCAN module.

27.6.1 FlexCAN initialization sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- MCU level soft reset, which resets some of the memory mapped registers synchronously (refer to *Table 397* to see what registers are affected by soft reset)
- Bit CAN_MCR[SOFTRST], 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. CAN_MCR[SOFTRST] 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 (CAN_CR[CLKSRC]) should be selected while the module is in Disable Mode. After the clock source is selected and the module is enabled (CAN_MCR[MDIS] negated), FlexCAN automatically goes to Freeze Mode. In Freeze Mode, FlexCAN is un-synchronized to the CAN bus, the HALT and FRZ bits in CAN_MCR are set, the internal state machines are disabled and the FRZACK and NOTRDY bits in the CAN_MCR are set. The Tx pin is in recessive state and FlexCAN does not initiate any transmission or reception of CAN frames. Note that the message buffers and the Rx Individual Mask Registers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization it is required that FlexCAN is put into Freeze Mode (see *Section 27.5.9.1: Freeze Mode*). The following is a generic initialization sequence applicable to the FlexCAN module:

- Initialize the Module Configuration Register (CAN_MCR)
  - Enable the individual filtering per MB and reception queue features by setting CAN_MCR[MBFEN]
  - Enable the warning interrupts by setting CAN_MCR[WRNEN]
  - If required, disable frame self reception by setting CAN_MCR[SRXDIS]
  - Enable the FIFO by setting CAN_MCR[FEN]
  - Enable the abort mechanism by setting CAN_MCR[AEN]
  - Enable the local priority feature by setting CAN_MCR[LPRIO_EN]
- Initialize the Control Register (CAN_CR)
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
  - Determine the bit rate by programming the PRESDIV field
  - Determine the internal arbitration mode (CAN_CR[LBUF])
- 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 corresponding CAN_IMRL or CAN_IMRH register (for all MB interrupts), in CAN_CR (for Bus Off and Error interrupts) and in CAN_MCR for Wake-Up interrupt
- Negate CAN_MCR[HALT]

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

27.6.2 FlexCAN addressing and RAM size configurations

Three possible RAM configurations can be implemented within the FlexCAN module:

- For 16 MBs: 288 bytes for MB memory and 64 bytes for Individual Mask Registers
- For 32 MBs: 544 bytes for MB memory and 128 bytes for Individual Mask Registers
- For 64 MBs: 1056 bytes for MB memory and 256 bytes for Individual Mask Registers

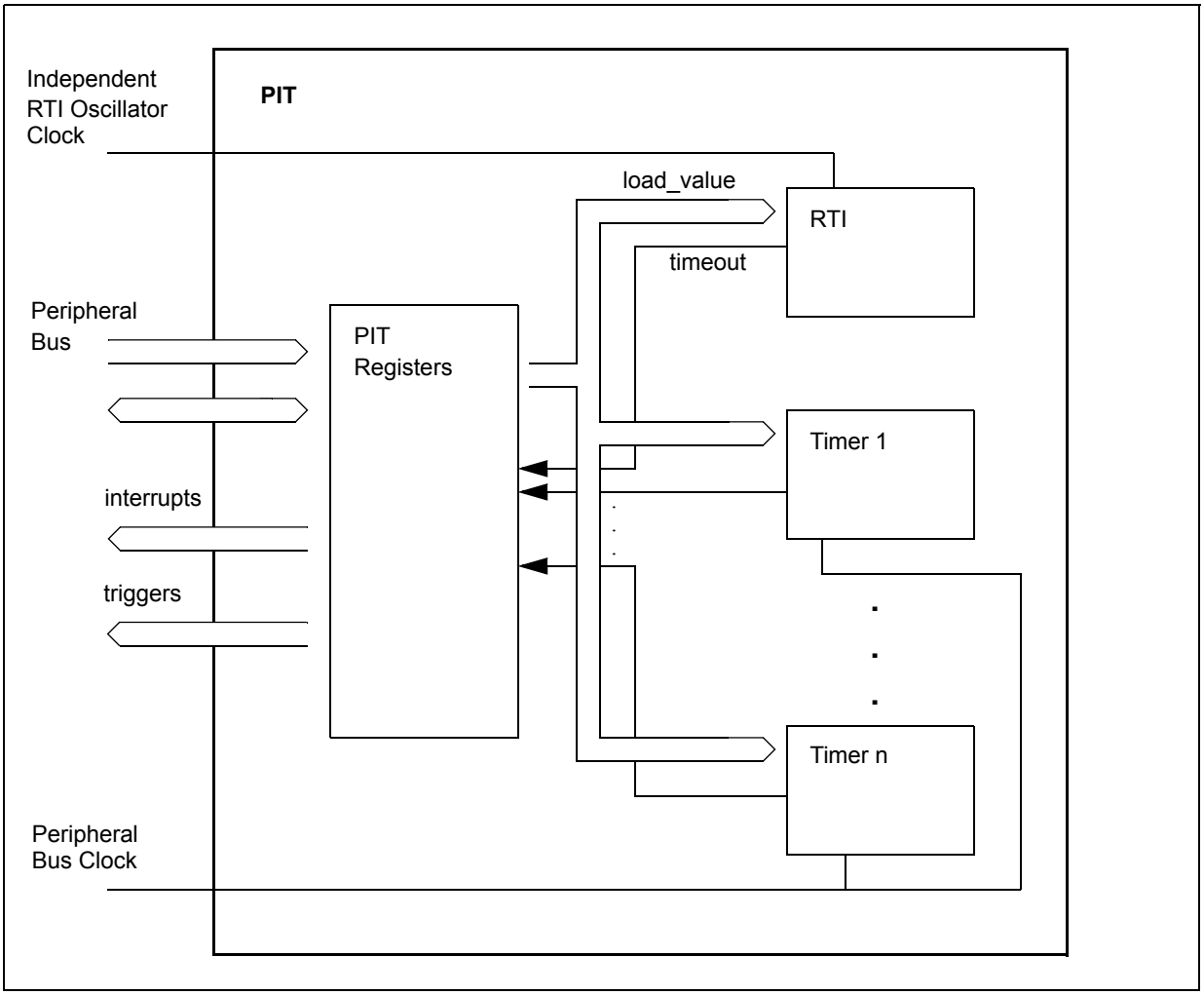
In each configuration the user can program the maximum number of MBs that will take part in the matching and arbitration processes using the MAXMB field in the CAN_MCR. For 16 MB configuration, MAXMB can be any number between 0–15. For 32 MB configuration, MAXMB can be any number between 0–31. For 64 MB configuration, MAXMB can be any number between 0–63.

28 Periodic Interrupt Timer (PIT_RTI)

28.1 Introduction

The PIT_RTI block implements several timers which can be used for DMA triggering, general purpose interrupts and system wakeup. *Figure 606* shows the PIT_RTI block diagram.

Figure 606. Block diagram of PIT_RTI



28.1.1 Overview

The PIT_RTI block on this chip provides:

- An array of timers that can be used to raise interrupts and trigger DMA channels
- A dedicated Real-Time Interrupt Timer (RTI), which runs on a separate clock and can be used for system wakeup

28.1.2 Features

The main features of this block are:

- One RTI (Real-Time Interrupt) timer to wake up the CPU in stop mode
- Timers can generate DMA trigger pulses
- Timers can generate interrupts
- All interrupts are maskable
- RTI interrupt can be raised, even when the bus clock is switched off
- Power saving with a separate input clock for the RTI timer (all other timers share one common core clock)
- Independent timeout periods for each timer

28.2 Modes of operation

This subsection describes briefly all operating modes supported by the PIT.

- Run Mode—All functional parts of the PIT are running during normal Run mode.
- Stop Mode—The RTI can continue to run in Stop mode.

28.3 Signal description

The PIT module has no external pins.

28.4 Memory map and register description

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

28.4.1 Memory map

Table 419 gives an overview of all PIT_RTI registers.

Table 419. PIT_RTI memory map

Address offset	Use	Access	Location
0x000	PIT Module Control Register (PITMCR)	R/W	*on page 1037*
0x004–0x0EC	Reserved	R	—
0x0F0–0x0FC	RTI Channel	(1)	
0x100–0x10C	Timer Channel 0	(1)	
0x110–0x11C	Timer Channel 1	(1)	
0x120–0x12C	Timer Channel 2	(1)	
0x130–0x13C	Timer Channel 3	(1)	

1. See *Table 420*.

Table 420. Timer Channel n / RTI Channel

Address offset	Use	Access	Location
Channel + 0x00	Timer Load Value Register n (LDVALn)	R/W	*on page 1038*
Channel + 0x04	Current Timer Value Register n (CVALn)	R	*on page 1039*
Channel + 0x08	Timer Control Register n (TCTRLn)	R/W	*on page 1039*
Channel + 0x0C	Timer Flag Register n (TFLGn)	R/W	*on page 1040*

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 have no effect.

28.4.2 Register descriptions

This section describes in address order all the PIT_RTI registers and their individual bits.

Note: The RTI registers should be programmed only when the RTI clock is running.

28.4.2.1 PIT Module Control Register (PITMCR)

This register controls whether the timer clocks should be enabled and whether the timers should run in debug mode.

Offset 0x000 Access: Read/Write

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

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

Figure 607. PIT Module Control Registers (PITMCR)

Table 421. PITMCR field descriptions

Field	Description
MDIS_RTI	Module Disable—RTI section This is used to disable the RTI timer. This bit should be enabled before any RTI setup is done. 0: Clock for RTI is enabled (default) 1: Clock for RTI disabled

Table 421. PITMCR field descriptions (continued)

Field	Description
MDIS	Module Disable—PIT section This is used to disable the standard timers. The RTI timer is not affected by this bit. This bit should be enabled before any other setup is done. 0: Clock for PIT Timers is enabled (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.

28.4.2.2 Timer Load Value Register *n* (LDVAL*n*)

These registers select the timeout period for the timer interrupts. In the case of the RTI, it will take several cycles until this value is synchronized into the RTI clock domain. For all other timers the value change is visible immediately. The synchronization mechanism allows 0 wait states in this case.

Offset channel_base + 0x00 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TSV31	TSV30	TSV29	TSV28	TSV27	TSV26	TSV25	TSV24	TSV23	TSV22	TSV21	TSV20	TSV19	TSV18	TSV17	TSV16
Reset	0	0	0	0	0	0	0	0	0	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	TSV15	TSV14	TSV13	TSV12	TSV11	TSV10	TSV9	TSV8	TSV7	TSV6	TSV5	TSV4	TSV3	TSV2	TSV1	TSV0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 608. Timer Load Value Register (LDVAL)

Table 422. 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 613*). **Note:** For the RTI, the timer should not be set to a value lower than 32 cycles, otherwise interrupts may be lost, as it takes several cycles to clear the RTI interrupt. For the other timers, this limit does not apply, however there will be practical limits, since the processor will require several cycles to service an interrupt.

28.4.2.3 Current Timer Value Register *n* (CVAL*n*)

These registers indicate the current timer position. In the case of the RTI, this will show a value which is several cycles old, since it originates from a potentially different clock domain.

Offset channel_base + 0x04 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TVL31	TVL30	TVL29	TVL28	TVL27	TVL26	TVL25	TVL24	TVL23	TVL22	TVL21	TVL20	TVL19	TVL18	TVL17	TVL16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TVL15	TVL14	TVL13	TVL12	TVL11	TVL10	TVL9	TVL8	TVL7	TVL6	TVL5	TVL4	TVL3	TVL2	TVL1	TVL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 609. Current Timer Value Register (CVAL)

Table 423. 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 607*).

28.4.2.4 Timer Control Register *n* (TCTRL*n*)

These registers contain the control bits for each timer.

Offset channel_base + 0x08 Access: Read/Write

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

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

Figure 610. Timer Control Register (TCTRL)

Table 424. TCTRL field descriptions

Field	Description
TIE	Timer Interrupt Enable Bit 0: Interrupt requests from Timer x are disabled 1: Interrupt will be requested whenever TIF is set When an interrupt is pending (TIF set), enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TIF flag must be cleared first.
TEN	Timer Enable Bit 0: Timer will be disabled 1: Timer will be active

28.4.2.5 Timer Flag Register *n* (TFLG*n*)

These registers hold the PIT interrupt flags.

Offset channel_base + 0x0C Access: Read/Write

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

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

Figure 611. Timer Flag Register (TFLG)

Table 425. 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: Timeout has not yet occurred 1: Timeout has occurred

28.5 Functional description

28.5.1 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. Additionally the RTI timer can be used to wakeup the processor.

28.5.1.1 Timers / RTI

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. Since in the case of the RTI, clearing the interrupt crosses clock domains, a minimum load value of 32 should be maintained.

If desired, the current counter value of the timer can be read via the CVAL registers. The value of the RTI counter can be delayed considerably, as it is synchronized to the bus clock from the RTI clock domain.

The counter period can be restarted, by first disabling, then enabling the timer with the TEN bit (see *Figure 612*).

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 613*).

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 614*).

Figure 612. Stopping and starting a timer

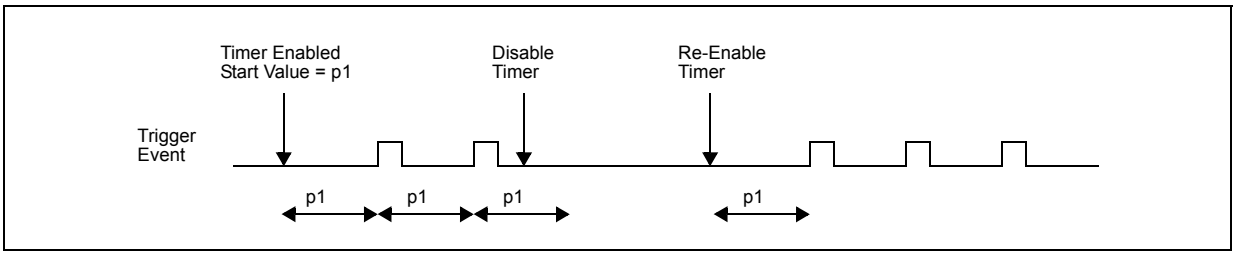


Figure 613. Modifying running timer period

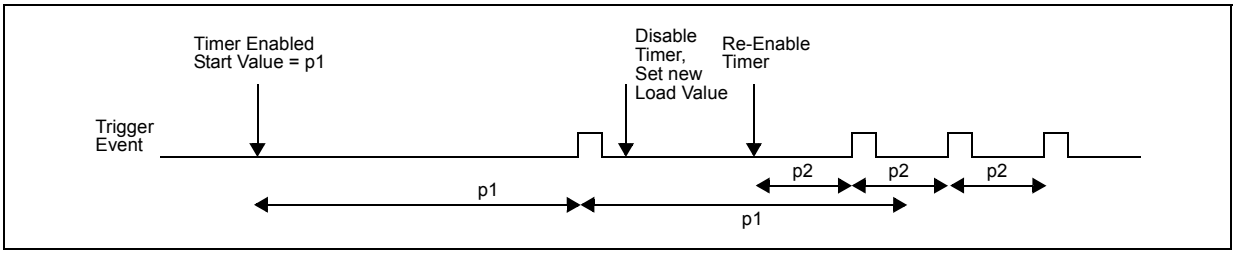
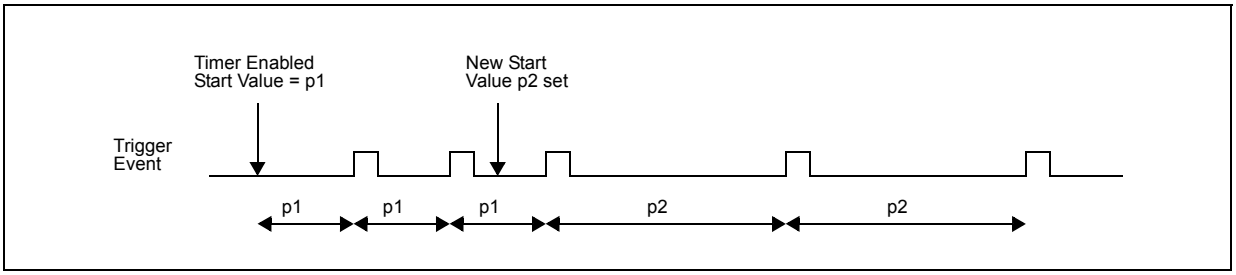


Figure 614. Dynamically setting a new load value



28.5.1.2 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.

28.5.2 Interrupts

All of the timers support interrupt generation. The RTI is typically used for system wakeup, but can be used for interrupt generation as well. Refer to the section “Functional description” in *Chapter 13: Interrupt Controller (INTC)* for related vector addresses and priorities.

Timer interrupts can be disabled by setting the TIE bits to zero. The timer interrupt flags (TIF) are set to 1 when a timeout occurs on the associated timer, and are cleared to 0 by writing a 1 to that TIF bit.

The PIT_RTI generates a real -time interrupt when the selected interrupt time period elapses. The RTI interrupt is disabled locally by setting the TIE bit to zero. The real-time interrupt flag (TIF) is set to 1 when a timeout occurs, and is cleared by writing a 1 to the TIF bit. (The flag will be set regardless whether the interrupt is enabled.)

The RTI can be used for periodic wakeup from a low power mode. It can also be used to generate a general purpose interrupt.

28.6 Initialization and application information

28.6.1 Example configuration

In the example configuration:

- PIT clock frequency = 50 MHz
- RTI clock frequency = 10 MHz
- RTI shall be set up to create a wakeup interrupt every 500 ms
- 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.

The 50 MHz clock frequency equates to a clock period of 20 ns and the 10 MHz frequency equates to a clock period of 100 ns. Therefore the RTI timer needs to trigger every 500 ms/100 ns = 5000000 cycles. 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 RTI LDVAL will be written with 0x004C_4B3F, LDVAL1 with 0x0003_E7FF and LDVAL3 with 0x0016_E35F.

To generate the wakeup interrupt, the interrupt line must be enabled by writing a 1 to the RTI TIE bit in the TCTRL register. To start the RTI, the TEN bit in the RTI TCTRL register must also be set.

The interrupt for Timer 1 is enabled by setting TIE in the TCTRL1 register. The timer is started by writing a 1 to bit TEN in the TCTRL1 register.

Timer 3 shall be used only for triggering. Therefore Timer 3 is started by writing a 1 to bit TEN in the TCTRL3 register, bit TIE stays at 0.

The following example code matches the described setup:

```
// turn on PIT
PIT_CTRL = 0x00;

// RTI
PIT_RTI_LDVAL = 0x004C4B3F; // setup RTI for 5000000 cycles
PIT_RTI_TCTRL = PIT_TIE; // let RTI generate interrupts
PIT_RTI_TCTRL |= PIT_TEN; // start RTI

// Timer 1
PIT_LDVAL1 = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_TCTRL1 = TIE; // enable Timer 1 interrupts
PIT_TCTRL1 |= TEN; // start timer 1

// Timer 3
PIT_LDVAL3 = 0x0016E35F; // setup timer 3for 1500000 cycles
PIT_TCTRL3 = TEN; // start timer 3
```

29 Power Management Controller (PMC)

29.1 Overview

Internally, SPC563Mxx devices have four supply voltages, nominally 5 V, 3.3 V, 1.2 V and 1.0 V V_{STBY}. Externally only a 5 V supply is required. The other voltages are supplied by internal regulators. All supply voltages have voltage monitors and both the V_{DD} regulator and all monitors are adjustable. The PMC controls the internal voltage supplies, with the exception of V_{STBY}, which has it's own regulator. Additionally the PMC controls the low voltage inhibit (LVI) circuits and power-on reset (POR) functions.

Note: Although SPC563Mxx devices have features intended for use in low-power applications, reduced power modes are achieved by clock gating. It is not possible to switch off the power to any on-chip module. Reduced power consumption is achieved by turning off the system clock to the modules. See Chapter 5: Operating modes for details.

Table 426. PMC ADC monitor channels

ADC channel	ADC	Description
144	ADC0	Buffered bandgap voltage
145	ADC0	Reference voltage for the 1.2 V low voltage inhibit (LVI)
146	ADC0	Reference voltage for the 1.2 V regulator controller
147	ADC0	Voltage divider tap for LVD monitoring voltage for the 3.3 V low voltage inhibit (LVI)(1)
162	ADC0	50% of VDDEH1B
163	ADC0	50% of VDDEH1B
164	ADC0	50% of VDDEH4B
165	ADC0	50% of VDDEH6A
166	ADC0	50% of VDDEH6B
167	ADC0	50% of VDDEH7
180	ADC0	Voltage divider tap for LVD monitoring voltage for 5.0 V low voltage inhibit (LVI)(1)
181	ADC0	Voltage divider tap for LVD monitoring voltage for the LVI3p3_h6 sampling reference
182	ADC0	Reference for the 3.3V supply regulator
183	—	Not used
194	ADC1	Not used
195	ADC1	Not available
196	ADC1	VRC33—3.3 V supply of the device
197	ADC1	VRC33—3.3 V supply of the device
198	ADC1	VDD12—1.2 V supply of the device
199	ADC1	50% of VDDEH1A

1. The voltage on this channel should be compared to the buffered bandgap voltage to determine the actual low voltage trip point.

The power management controller contains circuitry to generate the internal 3.3 V supply and to control the regulation of 1.2 V supply with external npn ballast transistor. It also contains low voltage inhibit (LVI) and power-on reset (POR) circuits for the 1.2 V supply, the 3.3 V supply, the 3.3 V/5 V supply of the closest I/O segment (VDDEH) and the 5 V supply of the regulators (VDDREG). There is no requirement for special power up or down sequencing. VDDREG can be tied to VSS to bypass the 3.3 V and 1.2 V internal regulators.

Regulators and power supply LVI control blocks use a precision bandgap voltage reference.

A low impedance buffered version of the absolute and curvature corrected bandgap voltage reference is available to be measured using a ADC dedicated channel. The PMC block has the following operating modes:

- Normal mode
- VDDREG supply grounded (See *Section 29.4: Functional description* for details)

29.1.1 Block diagram

Figure 615. Power management controller diagram

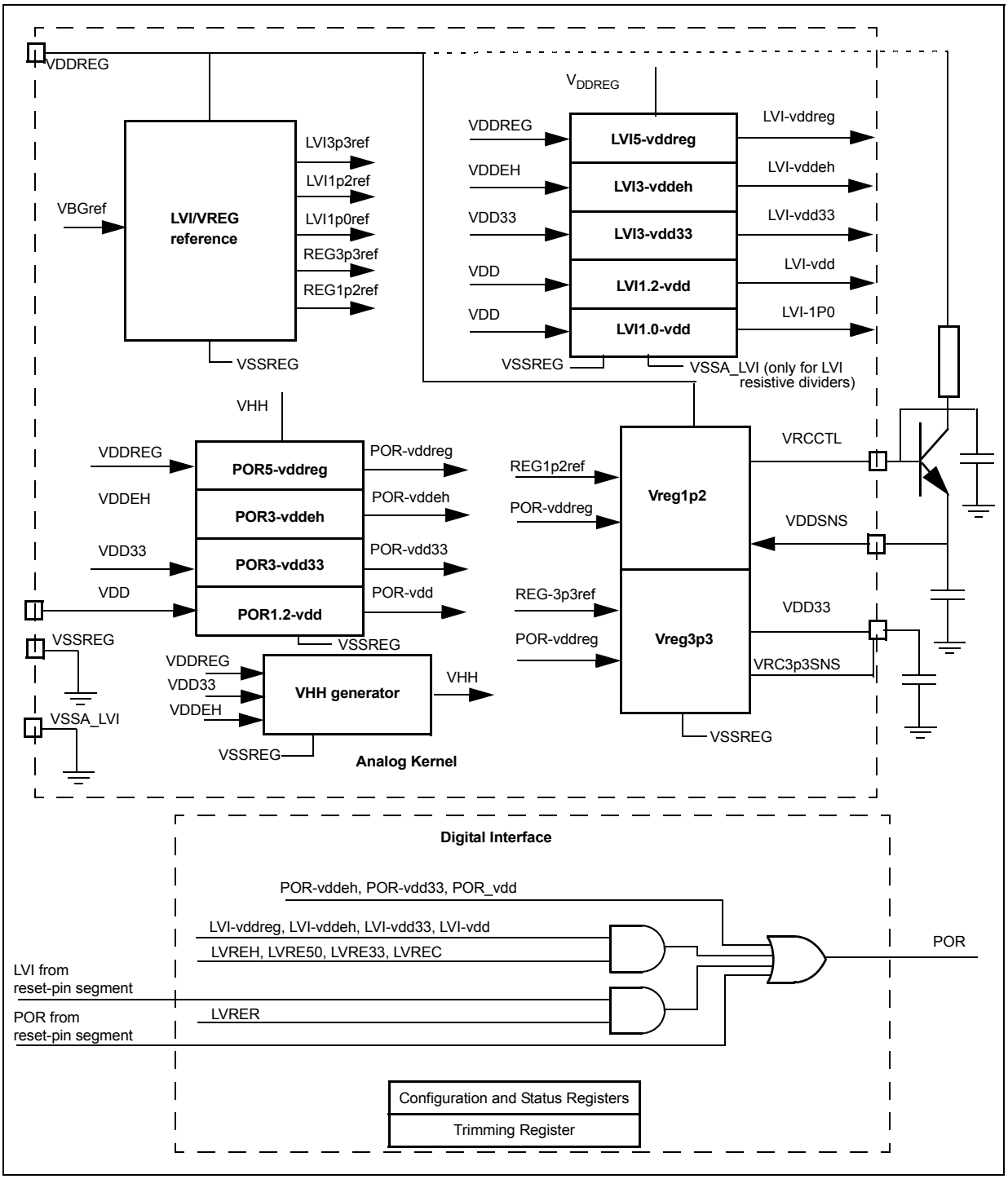
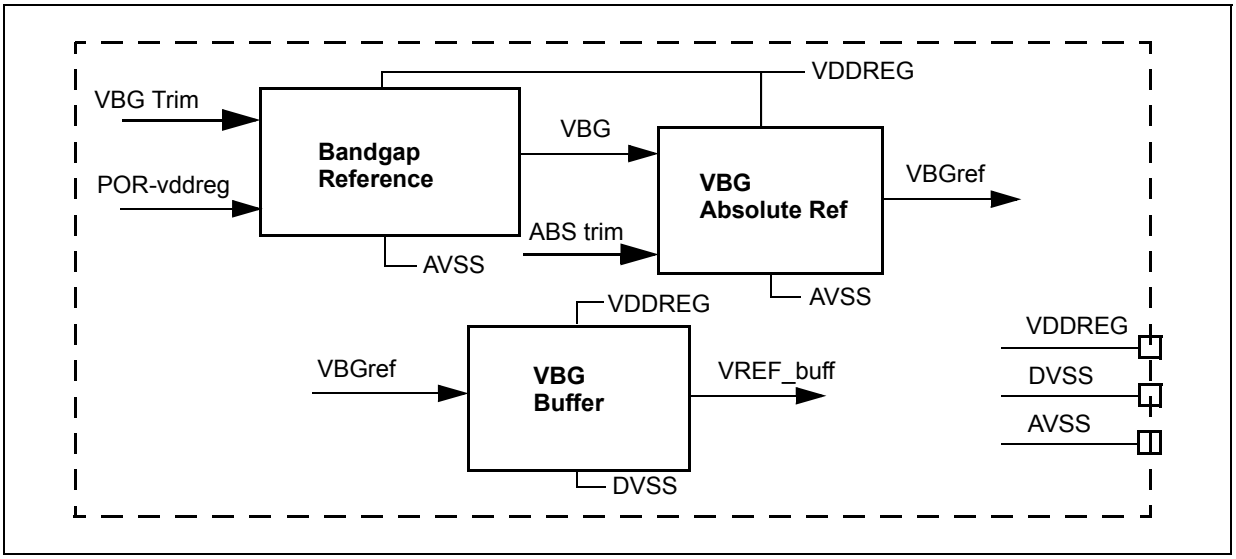


Figure 616. Bandgap reference block diagram



29.2 External signal description

Table 427 provides an overview of the PMC supply signals.

Table 427. Power management controller external signals (maximum ratings)

Name	Type	Voltage	Description
VDDREG	Supply	4.5–5.5 V	Power supply for the voltage regulator
VDDEH	Supply	2.7–5.5 V	Power supply of the closest I/O segment
VDD33(1)	Supply	3.3–3.6 V	3.3 V bypass capacitor or 3.3 V external power supply
VDD	Supply	1.2–1.32 V	1.2 V supply from external ballast transistor
VSSREG	Ground	—	Ground supply for digital core and PMC
VRCCTL	Output	—	Regulator drive for external npn base

1. This table represents the maximum variation of the supply when internal regulators are used. For external power supply requirements check PMC electrical specifications in the PMC operating conditions and external regulators supply voltage table in the device datasheet.

Note: The VDD33 supply on this device is routed to the VRC33 pin.

29.2.1 Detailed signal descriptions

The following paragraph provides the descriptions of external signals coming into and going out of the power management control unit.

29.2.1.1 VDDREG

Quiet 5 V supply for the voltage regulator and LVI block. It must have an external decoupling capacitor with a value of 4.7 µF–20 µF. Regulators and LVI can be turned off by grounding VDDREG. In this case external regulation and low voltage control must be supplied.

29.2.1.2 VDDEH

Power supply input (5 V or 3.3 V nominal), taken from one of the pad ring I/O segment which is near the voltage regulator.

29.2.1.3 VDD33

When the internal 3.3 V voltage regulator is enabled, this pin must be connected to an external bypass capacitor of 600 nF–2 μF with low ESR (max 50 mΩ). If the voltage regulator is not powered or the regulator is disabled (pin VDDREG to ground or NVUSRO[V33EN] = 0), this pin must be connected to an external 3.3 V supply.

29.2.1.4 VDD

This is the 1.2 V supply coming from the emitter of an external NPN ballast transistor, whose base current is supplied by VRCCTL. If the internal voltage regulator controller is not powered (pin VDDREG tied to ground) or the external ballast transistor is not present, the VDD pin must be connected to an external 1.2 V power supply.

For maximum transient performance, the recommended bypass capacitor for each pin that supplies the digital core is 2.2 μF–6 μF with very low ESR (max 50 mΩ). A ceramic capacitor is also desirable, with 100 nF capacitance. Moreover, a 1 μF to 2 μF cap might be connected to the base of the external bipolar.

29.2.1.5 VRCCTL

1.2 V regulator output that drives the base of the external NPN transistor.

29.3 Memory map/register definition

Table 428 shows the PMC memory map. The PMC memory map has three registers for configuring, monitoring, and trimming the LVI monitors.

Table 428. Power management controller memory map

Address	Register	Access	Reset value	Location
PMC_BASE + 0x0000	PMC_MCR — Module Configuration Register	R/W	0x98000000	*on page 1048*
PMC_BASE + 0x0004	PMC_TRIMR — Trimming Register	R/W	0x00000000	*on page 1050*
PMC_BASE + 0x0008	PMC_SR — Status Register	R/W	0x03000000 or 0x06000000	*on page 1053*

29.3.1 Module Configuration Register (PMC_MCR)

The configuration register contains configuration and reset and interrupt enable bits for the LVI monitors.

Please note that in the SPC563Mxx devices the LVI reset is equivalent to a POR. After an LVI reset the PORS bit is set in the SIU_RSR and the PMC_SR is reset—no LVI event information is retained. To enable application to report an LVI event, disable the reset using the appropriate field, i.e., LVRER or LVREH, and enable the interrupt for the LVI using its interrupt enable field, e.g., LVIER. You must make sure that the ISR will be finished before the voltage sinks below its functional specification and this may require an increase in the LVI level.

The software system reset and the POR/LVI reset are handled differently by the device. See *Section 4.5: Reset source descriptions* for more details. After a software system reset different status flags are set in the SIU_RSR.

Note: LVI resets and interrupts are only enabled when the voltage regulator is enabled (VDDREG = 5 V). If the user grounds VDDREG (VDDREG = 0 V) and supplies the voltages externally (1.2 V and 3.3 V), it is also necessary to provide the LVI monitoring externally.

Offset: PMC_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LVRER	LVREH	LVRE50	LVRE33	LVREC	0	0	0	LVIER	LVIEH	LVIE50	LVIE33	LVIC	0	0	TLK
W																
Reset	1	0	0	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 617. Module Configuration Register (PMC_MCR)

Table 429. PMC_MCR field descriptions

Field	Description
0 LVRER	Reset-pin-supply LVI reset enable This bit defines whether an LVI assertion on the supply of the I/O segment that contains the reset pin will generate system reset or not. 0 No reset. LVI assertion on the supply of the I/O segment that contains the reset pin does not cause system reset. 1 Reset. LVI assertion on the supply of the I/O segment that contains the reset pin causes system reset.
1 LVREH	VDDEH LVI reset enable This bit defines whether an LVI assertion on the VDDEH supply will generate system reset or not. 0 No reset. LVI assertion on the VDDEH supply does not cause system reset. 1 Reset. LVI assertion on the VDDEH supply causes system reset.
2 LVRE50	5 V LVI reset enable This bit defines whether an LVI assertion on the 5 V supply of the voltage regulator (VDDREG) will generate system reset or not. 0 No reset. LVI assertion on the 5 V supply of the voltage regulator does not cause system reset. 1 Reset. LVI assertion on the 5 V supply of the voltage regulator causes system reset.
3 LVRE33	3.3 V LVI reset enable This bit defines whether an LVI assertion on the 3.3 V supply will generate system reset or not. 0 No reset. LVI assertion on the 3.3 V supply does not cause system reset. 1 Reset. LVI assertion on the 3.3 V supply causes system reset.
4 LVREC	1.2 V LVI reset enable This bit defines whether an LVI assertion on the 1.2 V supply will generate system reset or not. 0 No reset. LVI assertion on the 1.2 V supply does not cause system reset. 1 Reset. LVI assertion on the 1.2 V supply causes system reset.
5-7	Reserved

Table 429. PMC_MCR field descriptions (continued)

Field	Description
8 LVIER	Reset-pin-supply LVI enable This bit enables the generation of the LVI interrupt request when the supply of the I/O segment that contains the reset pin falls below the corresponding LVI threshold. The LVI interrupt is independent from LVI reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. LVI interrupt request is disabled. 1 Enabled. LVI interrupt request is enabled.
9 LVIEH	VDDEH LVI enable This bit enables the generation of the LVI interrupt request when the VDDEH supply falls below the corresponding LVI threshold. The LVI interrupt is independent from LVI reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. LVI interrupt request is disabled. 1 Enabled. LVI interrupt request is enabled.
10 LVIE50	5 V LVI enable This bit enables the generation of the LVI interrupt request when the 5 V supply of the voltage regulator (VDDREG) falls below the corresponding LVI threshold. The LVI interrupt is independent from LVI reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. LVI interrupt request is disabled. 1 Enabled. LVI interrupt request is enabled.
11 LVIE33	3.3 V LVI enable This bit enables the generation of the LVI interrupt request when the 3.3 V power supply gets below the corresponding LVI threshold. The LVI interrupt is independent from LVI reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. LVI interrupt request is disabled. 1 Enabled. LVI interrupt request is enabled.
12 LVIC	1.2 V LVI enable This bit enables the generation of the LVI interrupt request when the 1.2 V power supply gets below the corresponding LVI threshold. The LVI interrupt is independent from LVI reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. LVI interrupt request is disabled. 1 Enabled. LVI interrupt request is enabled.
13–14	Reserved, should be cleared.
15 TLK	Trimming lock This is a set-only bit that comes out of reset negated, and can be asserted one time after reset to lock the trimming register. Once asserted, it cannot be negated anymore. When TLK is asserted, the Trimming Register becomes read-only and cannot be changed until the next reset. 0 Trimming register can be written 1 Trimming register is read-only
16-31	Reserved

29.3.2 Trimming Register (PMC_TRIMR)

The trimming register enables the user to fine tune the voltage of the regulators and the LVI thresholds. It can only be written when the TLK bit of the PMC_MCR is negated. Once TLK has been asserted, this register becomes read-only until the next system reset.

Offset: PMC_BASE + 0x0004 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VDD33TRIM				LVD33TRIM				VDDCTRIM				LVDCTRIM			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 618. Trimming Register (PMC_TRIMR)

Table 430. PMC_TRIMR field descriptions

Field	Description
0–11	Reserved, should be cleared.
12–15 LVDREGTRIM	LVI 5 V trimming This field is used to fine tune the voltage threshold of the 5 V rising LVI, which is used to monitor the VDDREG supply. Nominal configuration: 0111 4.43 V 0110 4.41 V 0101 4.39 V 0100 4.37 V 0011 4.35 V 0010 4.33 V 0001 4.31 V 0000 4.29 V (default) 1111 4.27 V 1110 4.25 V 1101 4.23 V 1100 4.21 V 1011 4.19 V 1010 4.17 V 1001 4.15 V 1000 4.13 V

Table 430. PMC_TRIMR field descriptions (continued)

Field	Description
16–19 VDD33TRIM	VREG 3.3 V trimming This field is used to fine tune the voltage of the 3.3 V regulator. Nominal configuration: 0111 3.60 V 0110 3.57 V 0101 3.54 V 0100 3.51 V 0011 3.48 V 0010 3.45 V 0001 3.42 V 0000 3.39 V (default) 1111 3.36 V 1110 3.33 V 1101 3.30 V 1100 3.27 V 1011 3.24 V 1010 3.21 V 1001 3.18 V 1000 3.15 V
20–23 LVD33TRIM	LVI 3.3 V trimming This field is used to fine tune the voltage threshold of the 3.3 V rising LVI, which is used to monitor the 3.3 V regulated output and the VDDEH supply. Nominal configuration: 0111 3.23 V 0110 3.21 V 0101 3.19 V 0100 3.17 V 0011 3.15 V 0010 3.13 V 0001 3.11 V 0000 3.09 V (default) 1111 3.07 V 1110 3.05 V 1101 3.03 V 1100 3.01 V 1011 2.99 V 1010 2.97 V 1001 2.95 V 1000 2.93 V **Note:** The recommended value for the LVD33TRIM is 0b0011. In the cut 1 device, the register should be written by software.

Table 430. PMC_TRIMR field descriptions (continued)

Field	Description
24–27 VDDCTRIM	VREG 1.2 V trimming This field is used to fine tune the voltage of the 1.2 V regulator. Nominal configuration: 0111 1.42 V 0110 1.40 V 0101 1.38 V 0100 1.36 V 0011 1.34 V 0010 1.32 V 0001 1.30 V 0000 1.28 V (default) 1111 1.26 V 1110 1.24 V 1101 1.22 V 1100 1.20 V 1011 1.18 V 1010 1.16 V 1001 1.14 V 1000 1.12 V
28–31 LVDCTRIM	LVI 1.2 V trimming This field is used to fine tune the voltage threshold of the 1.2 V rising LVI. Nominal configuration: 0111 1.30 V 0110 1.28 V 0101 1.26 V 0100 1.24 V 0011 1.22 V 0010 1.20 V 0001 1.18 V 0000 1.16 V (default) 1111 1.14 V 1110 1.12 V 1101 1.10 V 1100 1.08 V 1011 1.06 V 1010 1.04 V 1001 1.02 V 1000 1.0 V

29.3.3 Status Register (PMC_SR)

The status register contains interrupt flag bits for the LVI monitors.

Offset: PMC_BASE + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LVFVSTBY	BGRDY	BGTS	0	0	0	0	0	0	0	V33DIS
W														LVFCSTBY		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	LVFR	LVFH	LVF50	LVF33	LVFC	0	0	0
W	LVFCR	LVFCH	LVFC50	LVFC33	LVFCC											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 619. Status Register (PMC_SR)

Table 431. PMC_SR field descriptions

Field	Description
0–4	Reserved
5 LVFVSTBY	Brown out flag This bit indicates that a brown out condition was detected on the RAM standby regulator switch. 0No brown out detected 1 Brown out detected
6 BGRDY	Bandgap Status 1 This read-only bit gets asserted when the bandgap circuit has finished its startup procedure during power-up. The LVIs are disabled (output negated) while BGRDY is negated. 0 Bandgap not ready. LVIs disabled. 1 Bandgap ready. LVIs enabled.
7 BGTS	Bandgap Status 2 This read-only bit stores bandgap temperature status information. 0 Temperature out of range. Above 160 °C. 1 Temperature in range. Below 160 °C.
8-12	Reserved
13 LVFCSTBY	Standby-RAM-supply LVF clear This write-only bit is used to clear the low-voltage flag reported by the RAM standby regulator switch. Writing 1 to this bit informs the RAM standby regulator switch to clear LVFVSTBY. Writing 0 has no effect. Reading this bit always returns 0. 0No effect 1Clears LVFVSTBY
14	Reserved

Table 431. PMC_SR field descriptions (continued)

Field	Description
15 V33DIS	3.3 V Internal Regulator status bit 0 Enabled. Vreg3p3 ON. 1 Disabled. Vreg3p3 OFF. **Note:** V33DIS is the logically negated value of NVUSRO[V33EN]. See *Figure 621* and *Table 432* for details.
16 LVFCR	Reset-pin-supply LVI clear This write-only bit is used to clear the LVI interrupt flag associated with the supply of the I/O segment that contains the reset pin. Writing 1 to this bit clears the LVFR flag. Writing 0 has no effect. Reading this bit always return 0. 0 No effect 1 Clears the LVFR flag
17 LVFCH	VDDEH LVI clear This write-only bit is used to clear the LVI interrupt flag associated with the VDDEH supply. Writing 1 to this bit clears the LVFH flag. Writing 0 has no effect. Reading this bit always return 0. 0 No effect 1 Clears the LVFH flag
18 LVFC50	5 V LVI clear This write-only bit is used to clear the LVI interrupt flag associated with the 5 V voltage regulator supply (VDDREG). Writing 1 to this bit clears the LVF50 flag. Writing 0 has no effect. Reading this bit always return 0. 0 No effect 1 Clears the LVF50 flag
19 LVFC33	3.3 V LVI clear This write-only bit is used to clear the LVI interrupt flag associated with the 3.3 V supply. Writing 1 to this bit clears the LVF33 flag. Writing 0 has no effect. Reading this bit always return 0. 0 No effect 1 Clears the LVF33 flag
20 LVFCC	1.2 V LVI clear This write-only bit is used to clear the LVI interrupt flag associated with the 1.2 V supply. Writing 1 to this bit clears the LVFC flag. Writing 0 has no effect. Reading this bit always return 0. 0 No effect 1 Clears the LVFC flag
21-23	Reserved
24 LVFR	Reset-pin-supply LVI flag This read-only bit is the LVI interrupt flag associated with the supply of the I/O segment that contains the reset pin. It is asserted when the supply falls below the corresponding LVI threshold, and can be cleared by the CPU by writing 1 to the LVFCR bit. If the LVIER bit is also asserted, an LVI interrupt is sent to the CPU. If LVRER is also asserted, a system reset will be generated, which will clear the LVFR flag and negate the interrupt request. 0 No occurrence 1 LVI occurrence detected on the supply of the I/O segment that contains the reset pin

Table 431. PMC_SR field descriptions (continued)

Field	Description
25 LVFH	VDDEH LVI flag This read-only bit is the LVI interrupt flag associated with the VDDEH supply. It is asserted when the supply falls below the corresponding LVI threshold, and can be cleared by the CPU by writing 1 to the LVFCH bit. If the LVIEH bit is also asserted, an LVI interrupt is sent to the CPU. If LVREH is also asserted, a system reset will be generated, which will clear the LVFH flag and negate the interrupt request. 0 No occurrence 1 LVI occurrence detected on the VDDEH supply
26 LVF50	5 V LVI flag This read-only bit is the LVI interrupt flag associated with the 5 V supply of the voltage regulator. It can be cleared by the CPU by writing 1 to the LVFC50 bit. If the LVIE50 bit is also asserted, an LVI interrupt is sent to the CPU. If LVRE50 is also asserted, a system reset will be generated, which will clear the LVF50 flag and negate the interrupt request. 0 No occurrence 1 LVI occurrence detected on the 5 V supply of the voltage regulator
27 LVF33	3.3 V LVI flag This read-only bit is the LVI interrupt flag associated with the 3.3 V supply. It is asserted when the 3.3 V supply falls below the corresponding LVI threshold, and can be cleared by the CPU by writing 1 to the LVFC33 bit. If the LVIE33 bit is also asserted, an LVI interrupt is sent to the CPU. If LVRE33 is also asserted, a system reset will be generated, which will clear the LVF33 flag and negate the interrupt request. 0 No occurrence 1 LVI occurrence detected on the 3.3 V supply
28 LVFC	1.2 V LVI flag This read-only bit is the LVI interrupt flag associated with the 1.2 V supply. It is asserted when the 1.2 V supply falls below the corresponding LVI threshold, and can be cleared by the CPU by writing 1 to the LVFCC bit. If the LVIC bit is also asserted, an LVI interrupt is sent to the CPU. If LVREC is also asserted, a system reset will be generated, which will clear the LVFC flag and negate the interrupt request. 0 No occurrence 1 LVI occurrence detected on the 1.2 V supply
29-31	Reserved

29.4 Functional description

The power management controller provides the base current for an external ballast transistor to generate the 1.2 V supply. It also contains a 3.3 V regulator and several POR / LVI voltage monitors for system supervision.

A 5 V loose tolerance detection circuit (POR 5 V) is used to determine if the supply voltage VDDREG is high enough to guarantee the startup of bandgap voltage reference. As a consequence 5 V and 3 V LVI's can safely startup with reliable output value. The device is kept in reset until 5 V LVI has been cleared, allowing for correct 3.3 V regulator, 1.2 V regulator and 1.2 V LVI startup. Internal 3.3 V and 1.2 V POR are included to provide minimum low voltage reset capability.

The voltage regulators can be disabled to support the connection of external power supplies to the 3.3 V and 1.2 V pins. It can be done in two ways:

- By grounding VDDREG
  Internal regulators are automatically disabled, external bipolar transistor is omitted. In this case, all LVI monitors are also disabled. Therefore, when using external power supplies and grounding VDDREG, the user has to provide external LVI monitoring.
- By keeping VDDREG powered
  External bipolar transistor is omitted, 3.3 V internal regulator is disabled after startup by clearing the V33EN bit in the NVUSR0 register (Non Volatile User Option register in the flash memory. See *Section 29.4.3: 3.3 V internal voltage regulator* for more details.). In this case there is no need of external LVI monitoring.

As PORs are powered by VHH their value is reliable also when VDDREG is grounded, as long as VDDEH or VDD33 are at the correct voltage.

29.4.1 Bandgap

The bandgap voltage of the PMC is capable of generating a reference voltage of 1.219 V. It is used as reference to generate all supply voltages, for this reason it is powered directly out of the 5 V domain to avoid startup racing conditions. Supply voltage range is from 4.5 V to 5.5 V. The bandgap shall work with decreased performance down to 4.0 V supply.

The bandgap is calibrated during factory test—see the PMC electrical characteristics table in the device datasheet for accuracy details. The calibration data is typically stored in flash memory and is automatically read from the flash every time the part is initiated through reset. The default LVI thresholds are set to a safe value that accommodates the full uncalibrated bandgap range, so that external voltages can be applied in the specified ranges—see the PMC electrical characteristics table in the device datasheet for details.

The bandgap has a monitor signal: bandgap_OK. This signal is low during startup and rises when bandgap reference has settled. The signal and POR value are available in the status register (PMC_SR) as bandgap status bits.

Bandgap is powered by the same supply of the voltage regulator. When voltage regulator supply is removed (to allow for external powering of 3.3 V and 1.2 V domains), bandgap voltage reference will also be disabled.

29.4.2 5 V LVI

A programmable low voltage monitor is connected internally to the 5 V supply that feeds the voltage regulator (VDDREG). The output of the LVI goes to logical 1 when the monitored voltage rises above the LVI rising trip point. In case the monitored voltage falls below the falling trip point, the LVI output goes to logical 0.

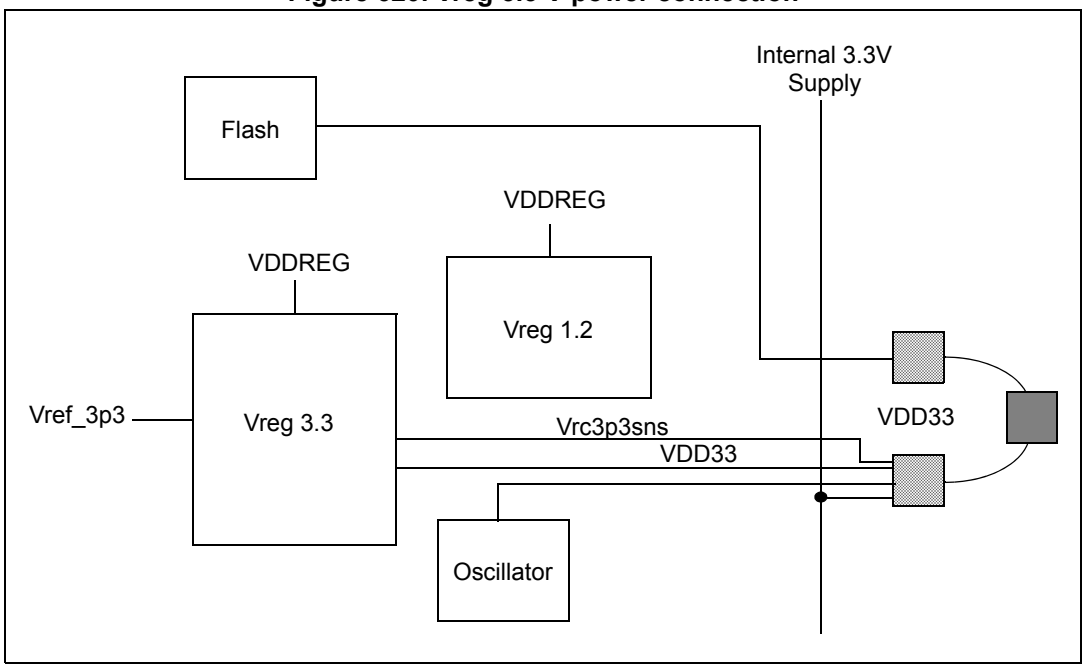
The assertion and negation voltages are adjustable via software by writing to the LVDREGTRIM field of the PMC_TRIMR, which selects one of the 16 voltages available. The reset value of the 4-bit register is ‘0000’, corresponding to the rising trip point voltage of 4.29 V. This is the typical default value, the real default value may vary from sample to sample according to process, temperature and voltage supply conditions, as detailed in the PMC electrical characteristics table in the device datasheet.

29.4.3 3.3 V internal voltage regulator

The 3.3 V internal voltage regulator supplies a total DC current of up to 80 mA or a maximum transient current peak up to 300 mA (if the external decoupling capacitor has the recommended value of 600 nF - 2 µF and ESR < 50 mΩ).

The regulator is powered by VDDREG and works in the range 4.5 V to 5.5 V, down to 4.0 V with lower current drive capabilities, and uses the bandgap voltage as absolute reference. The 3.3 V regulator output is connected to an I/O pad (VDD33) for external capacitance decoupling, then all blocks (except the flash) that need a 3.3 V supply are connected to the VDD33 pad. The 3.3 V flash power supply is taken at a pad next to the regulator I/O, which is double bonded together with the VDD33 pad.

Figure 620. Vreg 3.3 V power connection



The regulator output voltage is adjustable via software by writing to the VDD33TRIM field of the PMC_TRIMR, which selects one of the 16 voltages available. The reset value of the 4-bit register is '0000', corresponding to nominal voltage of 3.39 V. This is the typical default value, the real default value may vary from sample to sample according to process, temperature and voltage supply conditions, as detailed in the PMC electrical characteristics table in the device datasheet.

The 3.3 V supply is internally connected to a 5 V ADC channel such that the actual voltage may be read. Moreover, if an external 3.3 V voltage source will be used, the user can disable this regulator by setting the NVUSRO[V33EN] bit to logical 0. User might have both internal 3.3 V regulator and external 3.3 V voltage source at the same time at the cost of additional power consumption on external voltage source. It is recommended to disable internal 3.3 V regulator in this case.

Additional features of this regulator include a current limiter that protects a PMOS pass device from overload condition and soft start-up to avoid overshoot during power-on.

If an external 3.3 V supply will be used, this regulator can be disabled by setting NVUSRO[V33EN] bit to logical 0. The bit resides in the NVUSRO register in the flash memory shadow row. See *Figure 621* and *Table 432* for details.

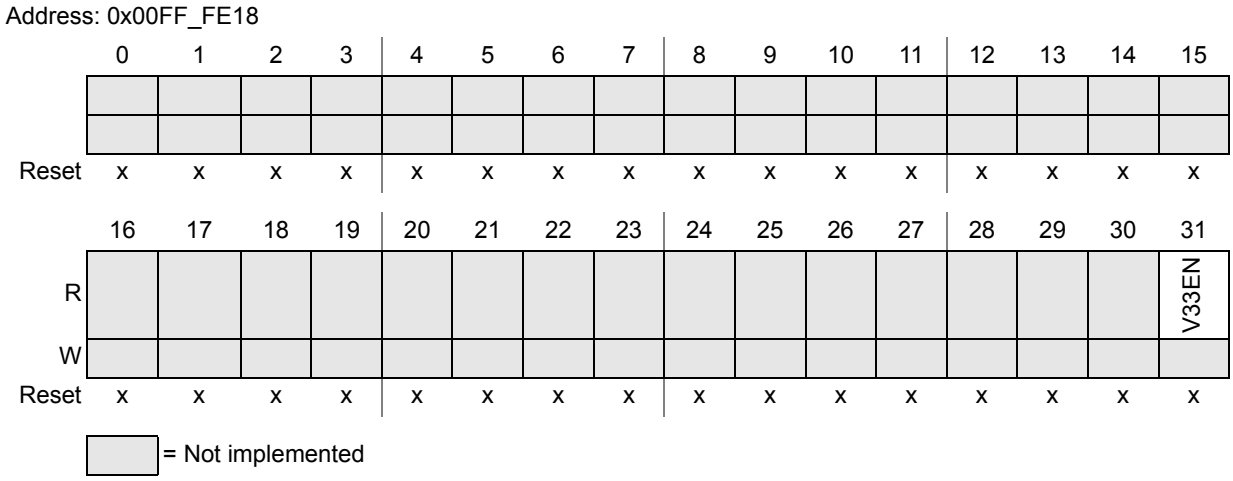


Figure 621. Non-Volatile User Options Register (NVUSRO) - Array0

Table 432. Non-Volatile User Options Register (NVUSRO) field descriptions

Field	Description
0-30	Reserved
31 V33EN	VREG33 enable bit. 1: 3.3 V regulator is enabled. 0: 3.3 V regulator is disabled. **Note:** This bit is masked until the 1.2 V achieves its POR trip point, keeping the 3p3 regulator on.

29.4.4 3.3 V LVI

The PMC contains two 3.3 V low voltage monitors. One is connected to the internal 3.3 V supply and the other is connected to VDDEH.

The output of the LVI goes to logical 1 when the monitored voltage rises above the rising trip point. In case the monitored voltage falls below the falling trip point, the LVI output goes to logical 0. The assertion and negation voltages are adjustable via software by writing to the LVD33TRIM field of the PMC_TRIMR, which selects one of the 16 voltages available. The reset value of the 4-bit register is ‘0000’, corresponding to rising trip point voltage of 3.09 V. This is the typical default value, the real default value may vary from sample to sample according to process, temperature and voltage supply conditions, as detailed in the PMC electrical characteristics table in the device datasheet.

The LVIs can be programmed to trigger power-on reset (enabled by default). If programmed to generate reset, the monitors are able to hold reset from 3.3 V POR until greater or equal to LVI threshold.

29.4.5 1.2 V voltage regulator controller

A voltage regulator controller is used to source current to the base of an external NPN transistor which operates as an emitter follower. The 1.2 V supply is produced at the emitter of this transistor with a maximum DC current of 400 mA (or maximum transient current of 1.2 A).

The regulator output voltage is adjustable using software, to permit the device to center the supply for maximum transient margin. This adjustment is achieved by writing to the VDDCTRIM field of the PMC_TRIMR, which selects one of the 16 voltages available. The reset value of the 4-bit register is ‘0000’, corresponding to a typical default voltage of 1.28 V.

The 1.2 V supply can be internally connected internally to a 5 V ADC channel such that the actual voltage may be read.

29.4.6 1.2 V LVI

A programmable low voltage monitor is connected to the 1.2 V supply. The output of the LVI goes to logical 1 when the monitored voltage rises above the rising trip point. In case the monitored voltage falls below the falling trip point, the LVI output goes to logical 0.

The assertion and negation voltages are adjustable via software by writing to the LVDCTRIM field of the PMC_TRIMR, which selects one of the 16 voltages available. The reset value of the 4-bit register is ‘0000’, corresponding to a rising trip point voltage of 1.16 V. This is the typical default value, the real default value may vary from sample to sample according to process, temperature and voltage supply conditions, as detailed in the PMC electrical characteristics table in the device datasheet.

The LVI 1.2 V can be programmed to trigger power-on reset (enabled by default). If programmed to generate reset, the monitor is able to hold reset from 1.2 V POR until greater or equal to LVI threshold.

29.4.7 Resets and interrupts

29.4.7.1 Power-on reset

Power-on reset (POR) circuits are present at the following power supplies:

- 3.3 V / 5 V supply of the closest I/O segment (VDDEH);
- 5 V supply of the PMC block and bandgap (VDDREG);
- 3.3 V regulated supply VDD33;
- 1.2 V regulated supply.

Power-on reset will assert when the voltage levels of the POR power supplies begin to rise. Each POR will negate when its power supply rises into the specified range. The behavior for each POR during power supply ramping is shown in *Figure 622*.

The dependence between POR and LVI is summarized in *Figure 623*. As shown, the LVI will reach a consistent state before the POR actually releases the reset, avoiding false startup condition. This is valid for each voltage supply monitored by POR/LVI.

The PORs for each power supply are not intended to indicate that the power supply has dropped below the specified voltage range for the device. The 1.2 V and the 3.3 V supplies are monitored, respectively, by the LVI 1.2 V and LVI 3.3 V circuits for this purpose.

Figure 622. POR rising and falling edges

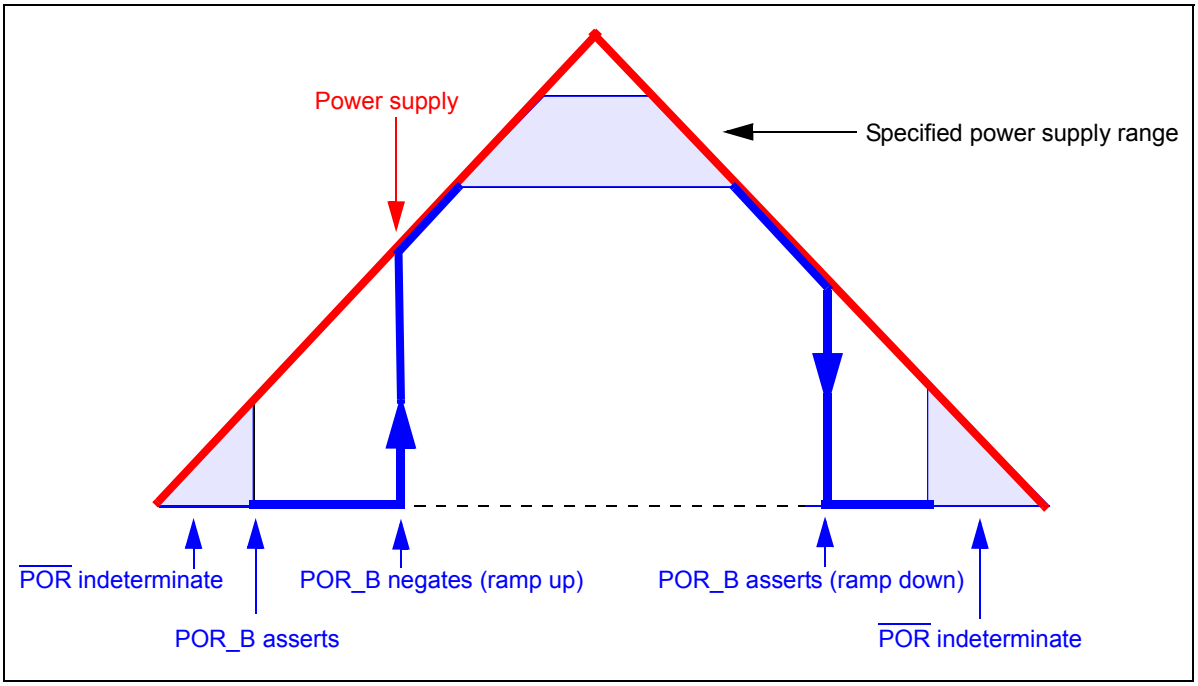
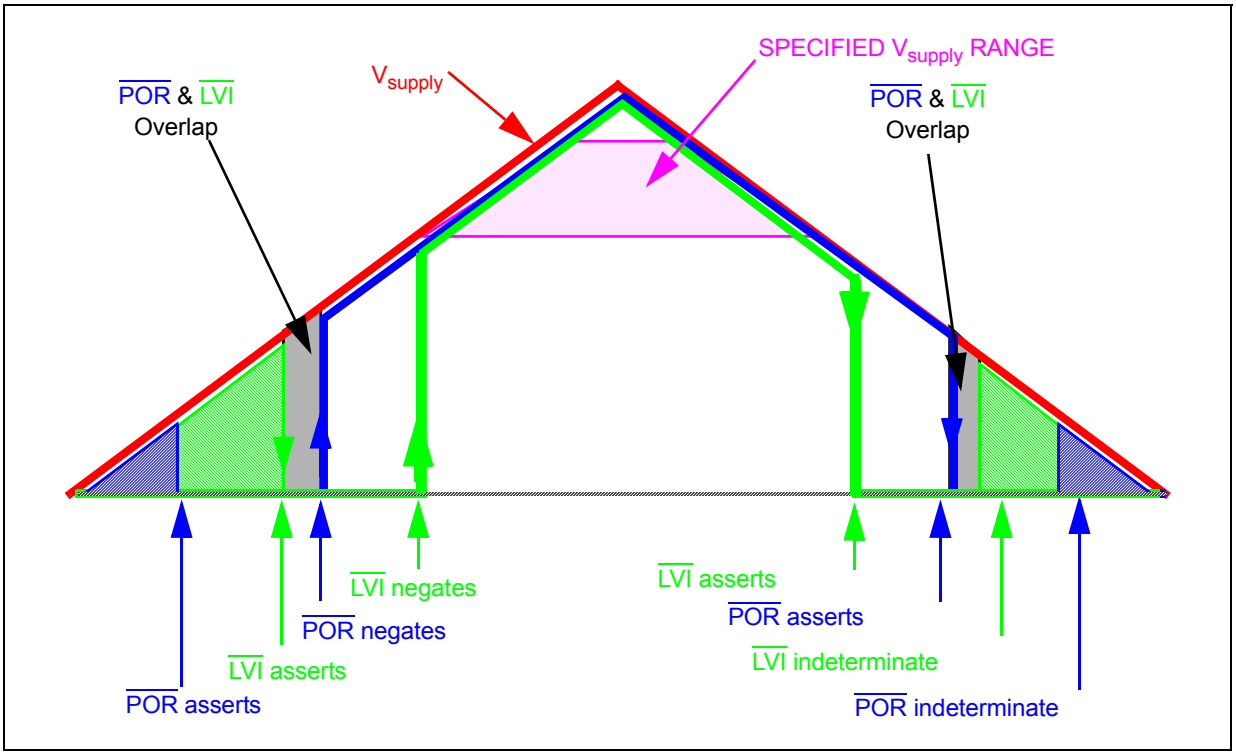


Figure 623. POR - LVI relative rising and falling edges



The LVI monitors can be configured to generate power-on reset by programming the LVRER, LVREH, LVRE50, LVRE33 and LVREC bits in the PMC_MCR.

The combination of POR and LVI sources within the PMC generates a single power-on reset output signal which can be distributed throughout the device.

The following sections discuss the various modules and functions that use the POR.

29.4.7.1.1 Clock control

The clock control divides the system clock to generate CLKOUT. Because CLKOUT toggles during system reset, one of the sources of reset for the dividers is POR.

29.4.7.1.2 SIU

The SIU uses POR in its reset controller state machine, synchronizers and $\overline{\text{RESET}}$ filter, SIU_RSR, SIU_RCSR, and SIU_CCR.

29.4.7.1.2.1 Reset controller state machine

Because the reset controller state machine is active during system reset, it is reset with POR.

29.4.7.1.2.2 Synchronizers and $\overline{\text{Reset}}$ filter

Signals affecting the reset controller's ability to negate system reset need to be synchronized with synchronizers reset by POR if they are asynchronous or come from clock domains other than the system clock.

A synchronized and filtered assertion of the $\overline{\text{RESET}}$ pin will hold the device in system reset. The synchronizers and filter are reset with POR so that $\overline{\text{RESET}}$ appears to be asserted while POR is asserted. Synchronizers for the watchdog, Nexus, checkstop, and JTAG sources of reset use POR to reset the synchronizers, but system reset can suffice because those sources are not analyzed until after system reset negates. The loss of clock source synchronizers also are reset with POR, but since loss of clock results in loss of lock, which needs and uses POR as reset, system reset can suffice for loss of clock.

The synchronization of WKPCFG and BOOTCFG uses POR. In the case of WKPCFG, the pin is applied during POR. In the case of BOOTCFG, the pin value is latched before the negation of system reset. Changes in BOOTCFG have no effect after the negation of system reset.

29.4.7.1.2.3 SIU_RSR

Because POR sets the PORS bit of the SIU_RSR to indicate that POR was the source of reset, the other source indicators, ERS, LLRS, LCRS, WDRS, CRS, and SSRS, are reset with POR. The WKPCFG and BOOTCFG bits are reset with POR so that the WKPCFG and BOOTCFG values can be latched at the negation of system reset. The RGF bit also is reset with POR. The SERF bit is reset with POR, but also is cleared during system reset. System reset can suffice to reset the SERF bit.

29.4.7.1.2.4 SIU_SRCR

The CRE bit of the SIU_SRCR is reset with POR. The SSR and SER bits are reset with POR, but system reset can suffice for both of them.

29.4.7.1.2.5 SIU_CCR

The MATCH and DISNEX bits of the SIU_CCR use POR.

29.4.7.1.3 Flash

The flash state machine is reset during POR. VFLASH must be at a high enough voltage to read the shadow row before system reset negates. RESET must be asserted when VFLASH is below the minimum specification. Since the synchronized and filtered RESET appears as asserted during POR and then must remain asserted until VFLASH is within specification, it is used to reset the flash state machine.

When the system reset is caused by other sources besides POR or RESET assertion, the flash state machine uses the system reset indication as an input, but not as a reset, to indicate that the state machine is to read the shadow row.

Note: The field LVD33TRIM of the register PMC_TRIMR must be programmed with '0011' at least, in order to enhance the LVI33 threshold by 60 mV and monitor the VDD33 Voltage in all the corners (voltage, process and temperature). This is to ensure that the voltage never becomes lower than 3.0 V in order to guarantee the operations on the flash.

29.4.7.1.4 FMPLL

The FMPLL analog hard block is held in power down state during POR to guarantee that it starts operating only when voltages are high enough to allow its operation.

The FMPLL programmer's model registers are reset with POR so that the FMPLL does not lose lock every time a system reset is asserted.

The Clock Quality Monitor (CQM) inside the FMPLL uses POR to initialize its registers and counters because it operates during system reset to detect when the crystal oscillator has stabilized.NPC

The NPC uses POR because it can be configured during system reset.

29.4.7.1.5 JTAGC

The assertion of POR is equivalent to the negation of the JCOMP pin.

29.4.7.1.6 e200z3

The e200z3 is reset with POR or system reset except in some portions of its Nexus interface, which only are reset with POR.

29.4.7.1.7 Padring

The padring 3-states all of the output pins, including CLKOUT, MCKO, and RSTOUT when the 1.2 V supply is too low to propagate internal signals, including the POR indication. When the POR indication can be propagated, the output pins, including CLKOUT, MCKO, and RSTOUT, also are 3-stated during POR.

During POR, the actual value of the pin cannot be read. Instead, the padring drives an input value. The actual value during POR is important for only two pins: WKPCFG and PLLREF. The values driven during POR for all other pins are irrelevant. For those pins, the input value is a 0.

29.4.7.1.7.1 WKPCFG

During POR, the direction of the weak pull is determined by the reset state of the WPS bit in the SIU_PCRs. For those pins whose WPS reset state is determined by the WKPCFG pin, the value of WKPCFG is treated as a 1 during POR to be consistent with the default pull up

for the WKPCFG pin. Therefore, those pins whose WPS reset state is determined by WKPCFG will have a pull up during POR.

29.4.7.1.7.2 PLLREF

The PLLREF pins selects whether crystal or external clock is used as clock source in bypass mode, which is the default mode out of POR. Furthermore, PLLREF selects whether the clock reference is monitored or not by the Clock Quality Monitor. If the reference is the crystal oscillator, it is monitored. If the reference is an external clock, it is not monitored.

The PLLREF value during POR is kept at logic level 0 to minimize the probability of a clock glitch in the more stringent case when PLLREF = 0, therefore the CQM will not monitor the reference clock and the internal POR will be released as soon as the voltages achieve the LVI thresholds. The clock glitch may occur when the POR is released near the external clock falling edge. Even if such a glitch happens, it will be still inside the POR pulse because all synchronous logic that use POR are supposed to synchronize the POR negation with a double-register.

29.4.7.2 Interrupts

The PMC generates one interrupt request signal for each LVI source: reset-pin-supply LVI, VDDEH LVI, 5 V LVI, 3.3 V LVI and 1.2 V LVI. The module also generates combined interrupt request signal which is asserted whenever any of the three individual interrupt request signals becomes asserted.

29.4.8 Soft-start (for 1.2 V and 3.3 V regulators)

A soft-start circuit has been implemented for 1.2 V and 3.3 V regulators. This circuit controls the reference voltage rise time to avoid abrupt ramp-up of these regulators.

29.5 Electrical characteristics

For electrical characteristics, please refer to the device datasheet.

30 JTAG Controller (JTAGC)

30.1 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.

30.1.1 Device-specific parameters

Table 433 shows the parameters and associated values for this device.

Table 433. Device-specific parameters

Parameter	Value
Length of the boundary scan chain path for the device	264
Number of auxiliary TAP controllers that share the TAP with the JTAGC via an ACCESS_AUX_TAP_x instruction (not including the JTAGC)	5
Size of the CENSOR_CTRL register (bits)	64

30.1.2 Device identification register parameters

Table 434 shows the fields and values for the device identification register on this device.

Table 434. Device identification register parameters

Field	Value
Part revision number (PRN)	Changes in each revision
Design center code (DC)	0x2B
Part identification number (PIN)	0x201
Manufacturer's identification code (MIC)	0x020

Table 435. Device JTAG ID

Device	JTAG ID	JTAG PIN
SPC563M54	0AE0_4041(1)	0x204
SPC563M60	0AE0_0041(1)	0x200
SPC563M64	0AE0_1041(1)	0x201

1. First 4 bits indicate device revision, e.g., SPC563M54 cut 2.0 devices have the JTAG ID 2AE0_4041.

30.1.3 JTAG instructions

CENSOR_CTRL is code 0b00111.

30.1.4 Unavailable instructions

The following instruction is not available on this device:

- TEST_LEAKAGE

30.1.5 Auxiliary TAP controller instructions

Table 436 lists the auxiliary TAP controller instructions (discussed in *Section 30.5.4: JTAGC block instructions*) available on this device.

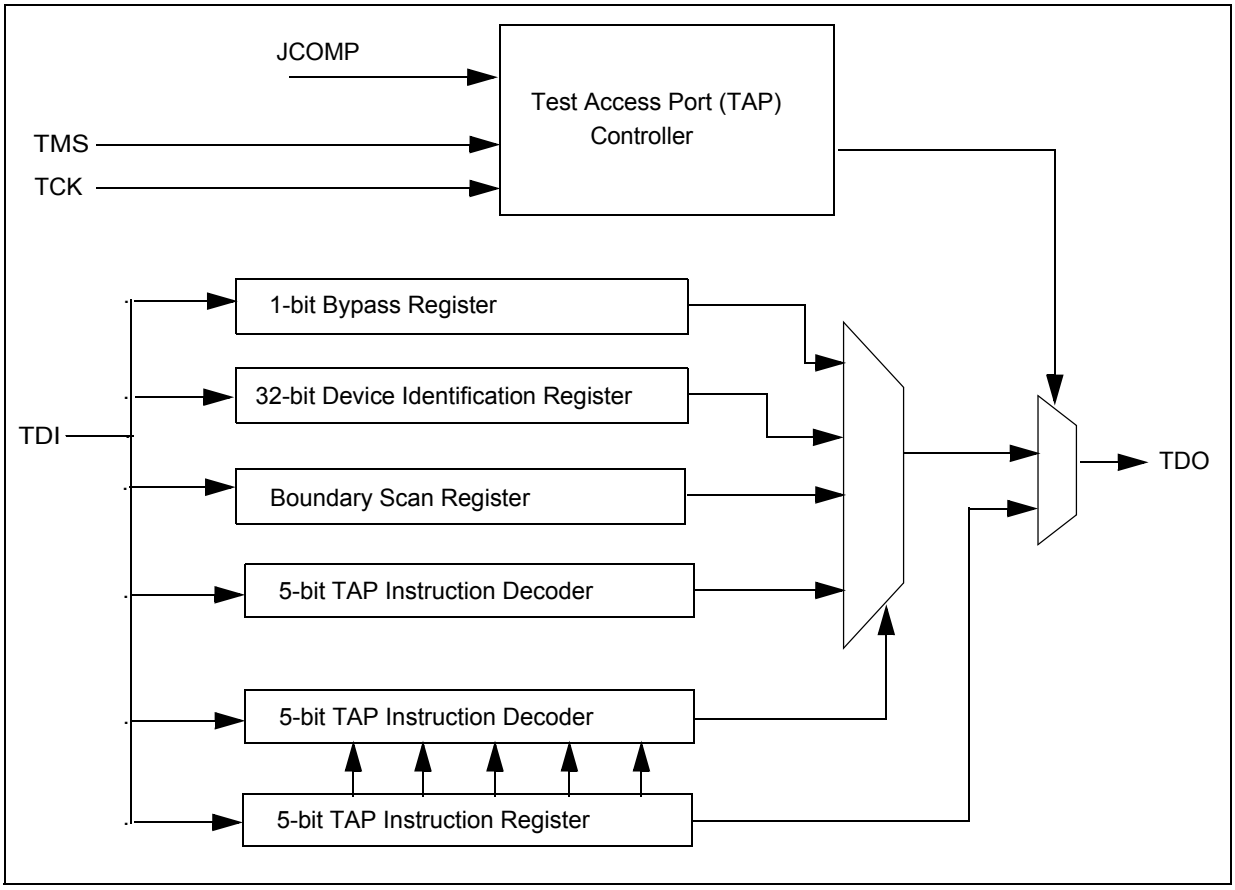
Table 436. Device-specific auxiliary TAP controller instructions

Instruction	Code[4:0]	Instruction summary
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller
ACCESS_AUX_TAP_ONCE	10001	Enables access to the e200z335 OnCE TAP controller
ACCESS_AUX_TAP_eTPUN3	10010	Enables access to the eTPU Nexus TAP controller

30.2 Introduction

Figure 624 is a block diagram of the JTAG Controller (JTAGC) block.

Figure 624. JTAG STL (IEEE 1149.1) block diagram



30.2.1 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.

30.2.2 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 440* for a list of supported instructions.
- Sharing of the TAP with other TAP controllers via ACCESS_AUX_TAP_x instructions.
- Test data registers: a bypass register, a boundary scan register, and a device identification register.
- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry.

30.2.3 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.

30.2.3.1 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

30.2.3.2 IEEE 1149.1-2001 defined test modes

The JTAGC block supports several IEEE 1149.1-2001 defined test modes. A test mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, HIGHZ, CLAMP, SAMPLE and SAMPLE/PRELOAD. Each instruction defines the set of data register(s) that may operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is enabled for serial access between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. The single-bit bypass

register shift stage is enabled for serial access between TDI and TDO when the HIGHZ, CLAMP or reserved instructions are active. The functionality of each test mode is explained in more detail in *Section 30.5.4: JTAGC block instructions*.

30.2.3.3 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.

30.3 External signal description

30.3.1 Overview

The JTAGC consists of five signals that connect to off chip development tools and allow access to test support functions. The JTAGC signals are outlined in *Table 437*.

Table 437. JTAG signal properties

Name	I/O	Function	Reset State	Pull(1)
TCK	Input	Test Clock	—	Down
TDI	Input	Test Data In	—	Up
TDO	Output	Test Data Out	High Z(2)	—
TMS	Input	Test Mode Select	—	Up
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.

30.3.2 Detailed signal descriptions

This section describes each of the signals listed in *Table 437* in more detail.

30.3.2.1 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.

30.3.2.2 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.

30.3.2.3 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 30.5.3: 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.

30.3.2.4 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.

30.3.2.5 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.

30.4 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.

30.4.1 Register descriptions

The JTAGC block registers are described in this section.

30.4.1.1 Instruction register

The JTAGC block uses a 5-bit instruction register as shown in *Figure 625*. 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 625. 5-bit Instruction Register

30.4.1.2 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, CLAMP, HIGHZ or reserve instructions are active. After entry into the Capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

30.4.1.3 Device Identification Register

The device identification register, shown in *Figure 626*, 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.

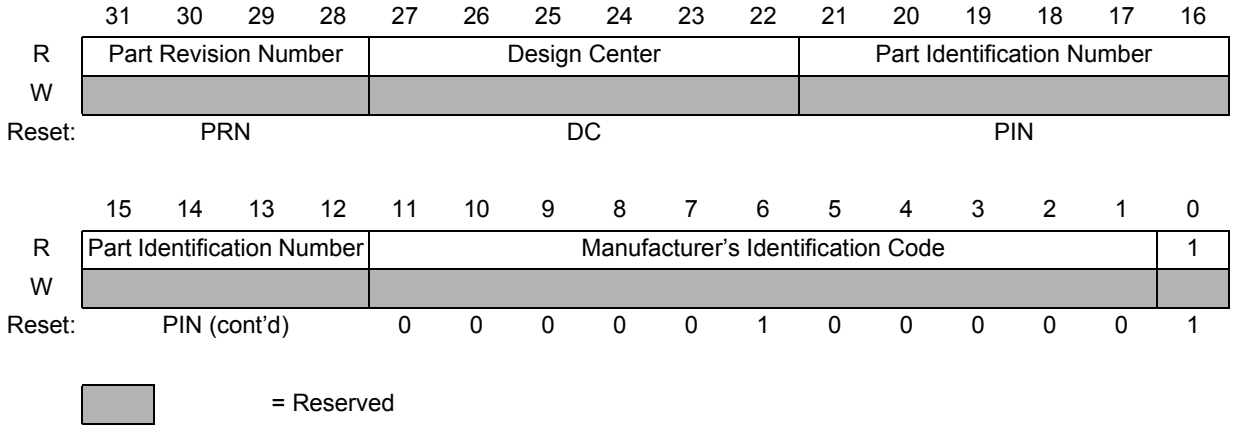


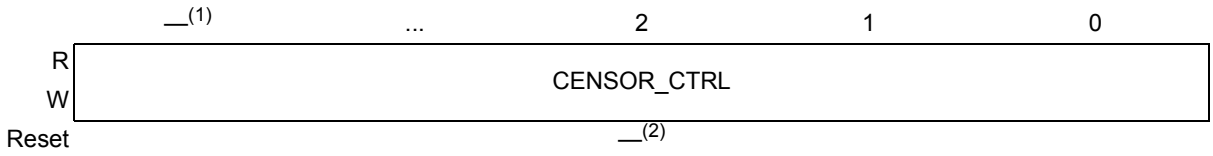
Figure 626. Device Identification Register

Table 438. Device Identification Register field descriptions

Field	Description
PRN	Part Revision Number Bits [31:28] contain the revision number of the part.
DC	Design Center Bits [27:22] indicate the design center.
PIN	Part Identification Number Bits [21:12] contain the part number of the device.
MIC	Manufacturer's Identification Code Bits [11:1] contain the JEDEC (Joint Electron Device Engineering Council) manufacturer's identification code. 0x020: STMicroelectronics
Bit [0]	IDCODE Register ID Bit [0] identifies this register as the device identification register and not the bypass register.

30.4.1.4 CENSOR_CTRL Register

The CENSOR_CTRL register is a 64-bit shift register path from TDI to TDO selected when the ENABLE_CENSOR_CTRL instruction is active. The default reset value of the CENSOR_CTRL register is 64'b0. The CENSOR_CTRL register transfers its value to a parallel hold register on the rising edge of TCK when the TAP controller state machine is in the Update-DR state. Once the ENABLE_CENSOR_CTRL instruction is executed, the register value will remain valid until a JTAG reset occurs.



1. The size of CENSOR_CTRL is 64 bits.
2. The reset value of CENSOR_CTRL is 64'b0.

Figure 627. CENSOR_CTRL Register

Table 439. CENSOR_CTRL field descriptions

Field	Description
CENSOR_CTRL	Censorship Control The CENSOR_CTRL bits are used to control chiptop censorship functions.

30.4.1.5 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. It is used to capture input pin data, force fixed values on output pins, and select a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in *Section 30.5.5: Boundary scan*. The size of the boundary scan register and bit ordering is device-dependent and can be found in the device BSDL file.

30.5 Functional description

30.5.1 JTAGC reset configuration

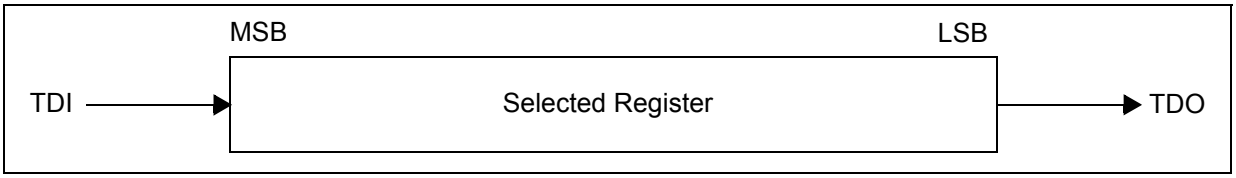
While in reset, the TAP controller is forced into the Test-Logic-Reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. In addition, the instruction register is loaded with the IDCODE instruction.

30.5.2 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 30.5.4.8: ACCESS_AUX_TAP_x instructions*.

Data is shifted between TDI and TDO through the selected register starting with the least significant bit, as illustrated in *Figure 628*. This applies for the instruction register, test data registers, and the bypass register.

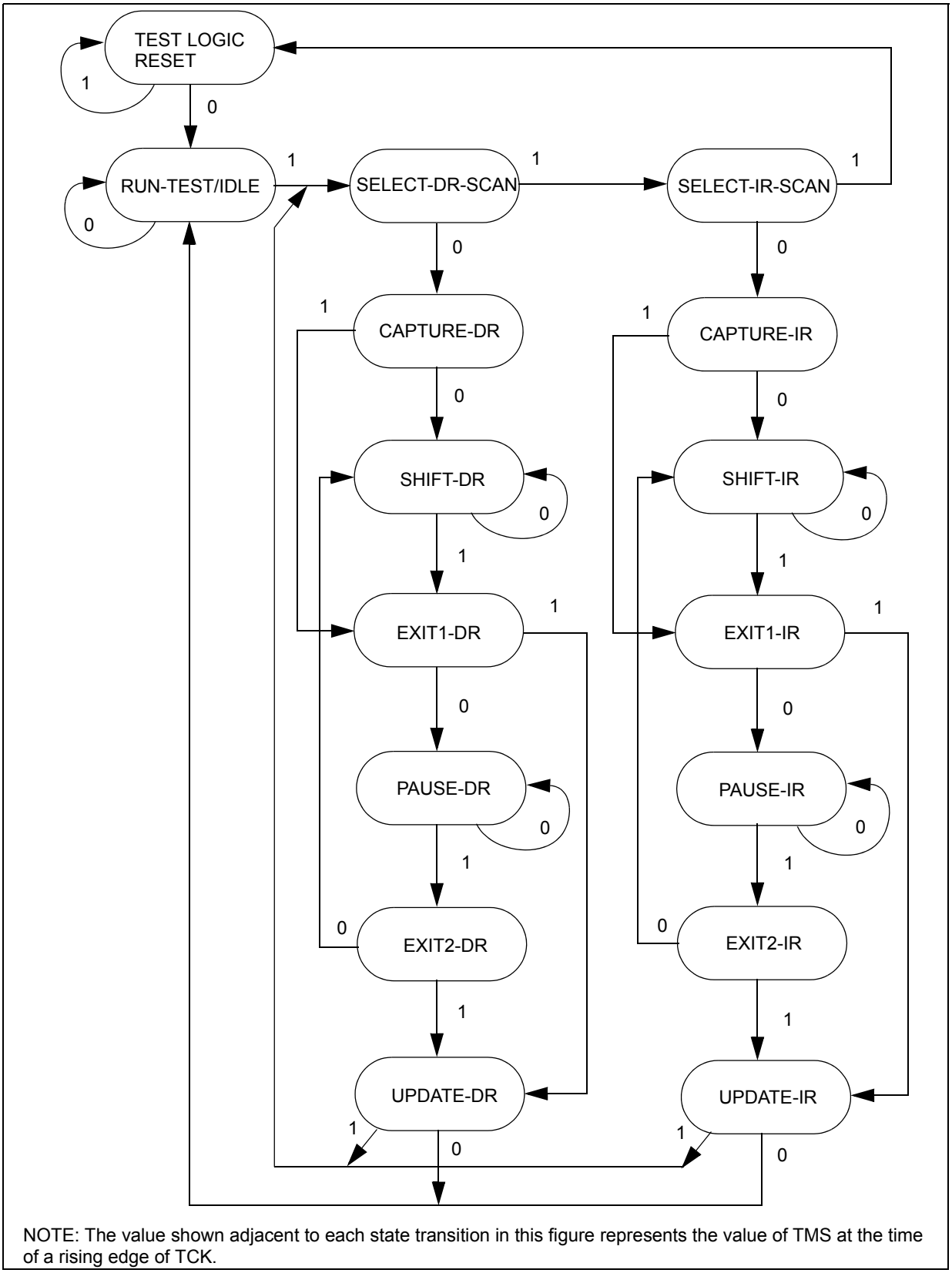
Figure 628. Shifting data through a register



30.5.3 TAP controller state machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. *Figure 629* 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 629* 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.

Figure 629. IEEE 1149.1-2001 TAP controller finite 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.

30.5.3.1 Enabling the TAP controller

The JTAGC TAP controller is enabled by setting JCOMP to a logic 1 value.

30.5.3.2 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.

30.5.4 JTAGC block instructions

The JTAGC block implements the IEEE 1149.1-2001 defined instructions listed in *Table 440*. 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 440. JTAG Instructions

Instruction	Code[4:0]	Instruction Summary
IDCODE	00001	Selects device identification register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
ENABLE_CENSOR_CTRL	00111	Selects CENSOR_CTRL register
HIGHZ	01001	Selects bypass register while three-stating all output pins and asserting functional reset
CLAMP	01100	Selects bypass register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_x	10000–11110	Grants one of the auxiliary TAP controllers ownership of the TAP as shown in the cells below. The number of auxiliary TAP controllers sharing the port is SHARE_CNT
BYPASS	11111	Selects bypass register for data operations
Factory debug reserved	00101, 00110, 01010, 00111	Intended for factory debug only
Reserved(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.

30.5.4.1 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.

30.5.4.2 SAMPLE/PRELOAD instruction

The SAMPLE/PRELOAD instruction has two functions:

- First, the SAMPLE portion of the instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the Capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. Both the data capture and the shift operation are transparent to system operation.
- Secondly, the PRELOAD portion of the instruction initializes the boundary scan register cells before selecting the EXTEST or CLAMP instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the Shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the Update-DR state. The data is applied to the external output pins by the EXTEST or CLAMP instruction. System operation is not affected.

30.5.4.3 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.

30.5.4.4 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.

30.5.4.5 ENABLE_CENSOR_CTRL instruction

The ENABLE_CENSOR_CTRL instruction selects the CENSOR_CTRL register for connection as the shift path between TDI and TDO.

30.5.4.6 HIGHZ instruction

HIGHZ selects the bypass register as the shift path between TDI and TDO. While HIGHZ is active all output drivers are placed in an inactive drive state (e.g., high impedance). HIGHZ also asserts the internal system reset for the MCU to force a predictable internal state.

30.5.4.7 CLAMP instruction

CLAMP allows the state of signals driven from MCU pins to be determined from the boundary scan register while the bypass register is selected as the serial path between TDI and TDO. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. CLAMP also asserts the internal system reset for the MCU to force a predictable internal state.

30.5.4.8 ACCESS_AUX_TAP_x instructions

The JTAGC is configurable to allow up to fifteen other TAP controllers on the device to share the port with it. This is done by providing ACCESS_AUX_TAP_x instructions for each of these TAP controllers. When this instruction is loaded, control of the JTAG pins are transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive. Instructions not used to access an auxiliary TAP controller on a device are treated like the BYPASS instruction.

30.5.4.9 BYPASS instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test functions. While the BYPASS instruction is active the system logic operates normally.

30.5.5 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.

30.6 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 Nexus Port Controller (NPC)

31.1 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.

31.1.1 Parameter values

Table 441 shows the parameter values for this device.

Table 441. Device parameter values

Parameter	Value
Number of Nexus trace clients on the device sharing the Nexus auxiliary port (not including the NPC)	1
SRC ID for e200z335 core	0b0000
Number of MDO pins available in reduced-port mode	4
Number of MDO pins available in full-port mode	12
Number of MSEO bits	2
Number of JCOMP bits	1
Number of breakpoint requests coming into the NPC from Nexus and non-Nexus sources on the device	1
Number of blocks that input an $\overline{\text{EVTO}}$	2
JCOMP value required to grant Nexus control of the JTAG port	0b1
Part identification number	0x200
Design code	0x2B

31.1.2 Unavailable features

The following features are not available or not supported on this device:

- Nexus double-data rate (DDR) mode
- The following bits in the Port Configuration Register:
  - DDR_EN
  - LP_DBG
  - LP2_SYN
  - LP1_SYN

31.1.3 Available features

In addition to the usual Nexus Class 2 features, this device also supports the following:

- Class 3 Nexus read/write access
- Class 4 Nexus watchpoint enabling of program trace

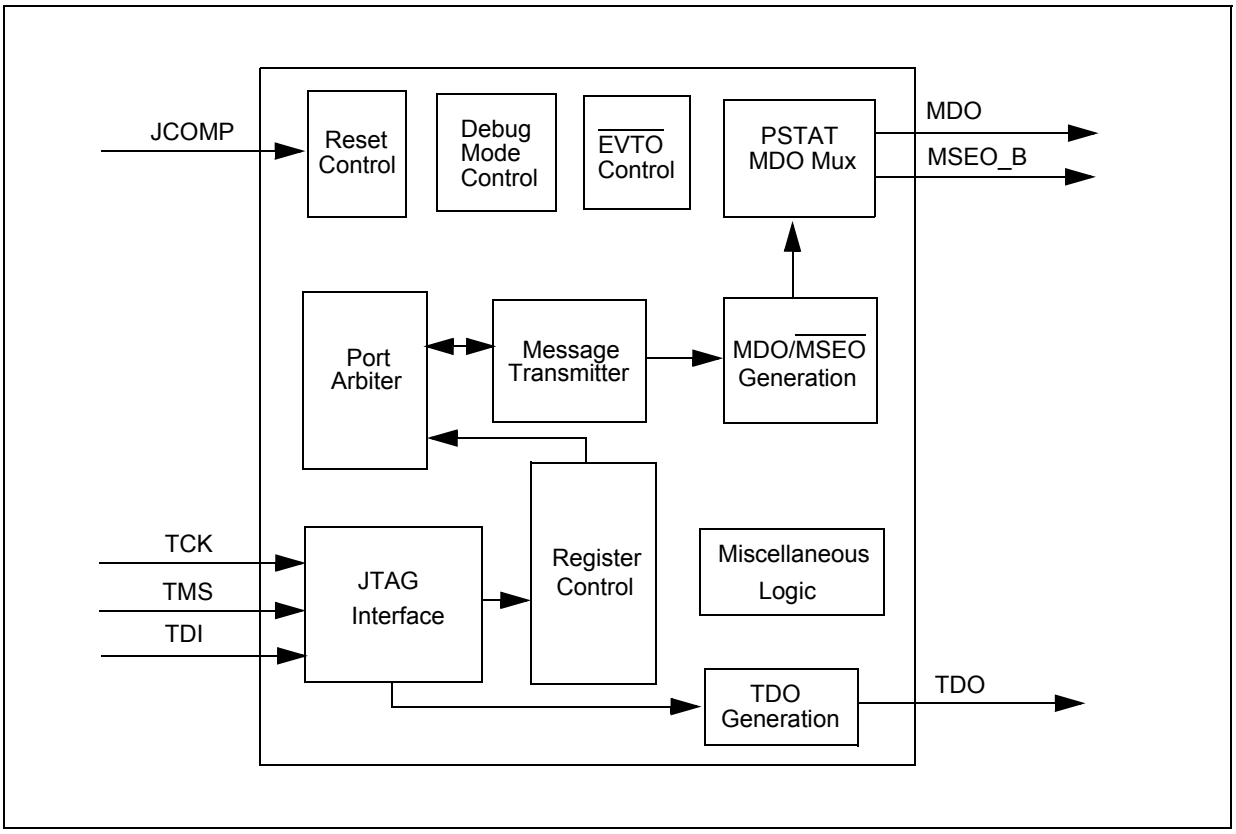
31.1.4 Nexus clients

This device has one Nexus client – the e200z335 Nexus Class 2 Interface (NZ3C2).

31.2 Introduction

Figure 630 is a block diagram of the Nexus Port Controller (NPC) block.

Figure 630. Nexus Port Controller block diagram



31.2.1 Overview

On a system-on-chip device, there are often multiple blocks that require development support. Each of these blocks implements a development interface based on the IEEE-ISTO 5001-2001 standard. The blocks share input and output ports that interface with the development tool. The NPC controls the usage of the input and output port in a manner that allows all the individual development interface blocks to share the port, and appear to the development tool to be a single block.

31.2.2 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
- Controls 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

31.2.3 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.

31.2.3.1 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 (Test Access Port) controller state machine is in the Test-Logic-Reset state. Holding TMS high for five 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

31.2.3.2 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 $\overline{\text{EVTO}}$.

31.2.3.3 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.

31.2.3.4 Reduced-Port Mode

Reduced-port mode (RPM) is entered by asserting the MCKO_EN bit and negating 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.

31.2.3.5 Censored Mode

In censored mode, it should not be possible to read the contents of internal Flash externally. The NPC and all other Nexus blocks are held in reset when in censored mode, preventing Nexus read/write to memory mapped resources and the transmission of Nexus trace messages.

31.2.3.6 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 on each edge (both rising and falling) of MCKO, effectively doubling message throughput.

31.3 External signal description

31.3.1 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 442*.

Table 442. NPC signal properties

Name	Port	Function	Reset state	Pull(1)
EVTO_B	Auxiliary	Event Out pin	0b1	—
JCOMP	JTAG	JTAG Compliancy and TAP Sharing Control	—	Down
MDO	Auxiliary	Message Data Out pins	0(2)	—
MSEO	Auxiliary	Message Start/End Out pins	0b11	—
TCK	JTAG	Test Clock Input	—	Down
TDI	JTAG	Test Data Input	—	Up
TDO	JTAG	Test Data Output	High Z(3)	—
TMS	JTAG	Test Mode Select Input	—	Up

1. The pull is not implemented in this block. Pullup/pulldown devices are implemented in the pads.
2. Following a power-on reset, MDO[0] remains asserted until power-on reset is exited and the system clock achieves lock.
3. TDO output buffer enable is negated when the NPC is not in the Shift-IR or Shift-DR states. A weak pull may be implemented on TDO at the SoC level.

31.3.2 Detailed signal descriptions

This section describes each of the signals listed in *Table 442* in more detail. Note that 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.

31.3.2.1 EVTO_B – 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.

31.3.2.2 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.

31.3.2.3 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.

31.3.2.4 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.

31.3.2.5 TCK – Test Clock Input

The Test Clock Input (TCK) pin is used to synchronize the test logic and control register access through the JTAG port.

31.3.2.6 TDI – Test Data Input

The Test Data Input (TDI) pin receives serial test instruction and data. TDI is sampled on the rising edge of TCK.

31.3.2.7 TDO – Nexus Test Data Output

The 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.

31.3.2.8 TMS – Test Mode Select

The 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.

31.4 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 443 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 443. NPC registers

Index	Register
0	Device ID Register (DID)
127	Port Configuration Register (PCR)

31.4.1 Register descriptions

This section provides the NPC register descriptions.

31.4.1.1 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.

31.4.1.2 Instruction Register

The NPC block uses a 4-bit instruction register as shown in *Table 631*. 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.

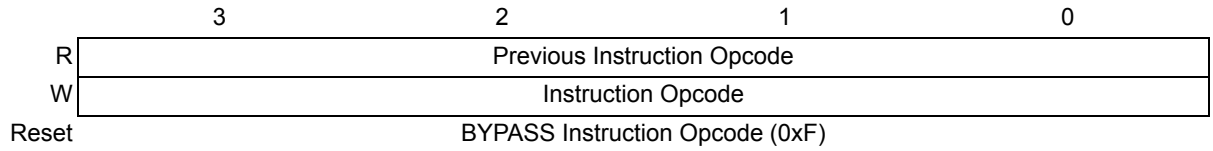


Figure 631. 4-bit Instruction Register

31.4.1.3 Nexus Device ID Register (DID)

The device identification register, shown in *Figure 632*, allows the part revision number, design center, part identification number, and manufacturer's identification code of the part to be determined through the auxiliary output port.

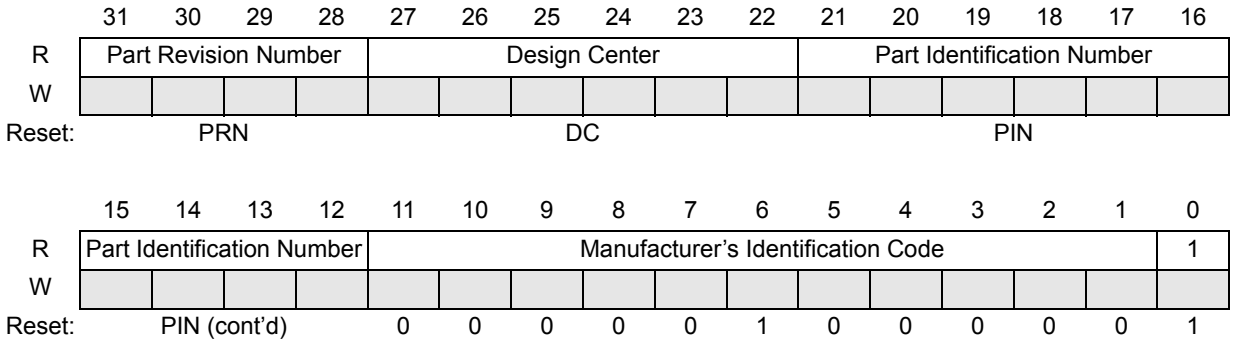


Figure 632. Nexus Device ID Register (DID)

Table 444. DID field descriptions

Field	Description
PRN	Part Revision Number Bits [31:28] contain the revision number of the part.
DC	Design Center Bits [27:22] indicate the design center.
PIN	Part Identification Number Bits [21:12] contain the part number of the device.
MIC	Manufacturer's Identification Code Bits [11:1] contain the JEDEC (Joint Electron Device Engineering Council) manufacturer's identification code. 0x020: STMicroelectronics
Bit [0]	IDCODE Register ID Bit [0] identifies this register as the device identification register and not the bypass register.

31.4.1.4 Port Configuration Register (PCR)

The PCR, shown in *Figure 633*, 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 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.

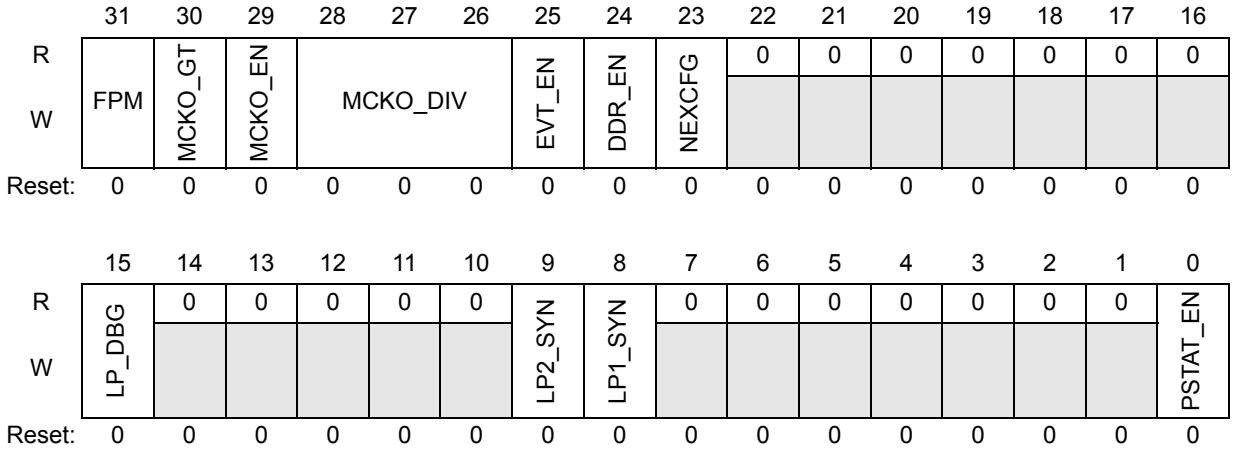


Figure 633. Port Configuration Register (PCR)

Table 445. PCR field descriptions

Bit	Name	Description
31	FPM	Full Port Mode The value of the FPM bit determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 1: All MDO pins are used to transmit messages 0: A subset of MDO pins are used to transmit messages
30	MCKO_GT	MCKO Clock Gating Control This bit is used to enable or disable MCKO clock gating. If clock gating is enabled, the MCKO clock is gated when the NPC is in enabled mode but not actively transmitting messages on the auxiliary output port. When clock gating is disabled, MCKO is allowed to run even if no auxiliary output port messages are being transmitted. 1: MCKO gating is enabled 0: MCKO gating is disabled
29	MCKO_EN	MCKO Enable This bit enables the MCKO clock to run. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. 1: MCKO clock is enabled 0: MCKO clock is driven to zero
28:26	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 446* shows the meaning of MCKO_DIV Values. In this table, SYS_CLK represents the system clock frequency.
25	EVT_EN	EVTO/EVTI Enable This bit enables the EVTO/EVTI port functions. 1: EVTO/EVTI port enabled 0: EVTO/EVTI port disabled

Table 445. PCR field descriptions (continued)

Bit	Name	Description
24	DDR_EN	Double Data Rate Mode Enable This bit enables Nexus double data rate (DDR) mode. In DDR mode, message data is updated on both rising and falling edges of MCKO, effectively doubling message throughput. 1: DDR mode enabled 0: DDR mode disabled
23	NEXCFG	Nexus Configuration Select Generic Nexus control bit. Function is SoC specific. 1: NEXCFG set 0: NEXCFG cleared
22:16	—	Reserved
15	LP_DBG_EN	Low Power Debug Enable This bit enables debug functionality on exit from low power modes on supported devices. 1: Low power debug enabled 0: Low power debug disabled
14:10	—	Reserved
9:8	LPn_SYN	Low Power Mode n Synchronization These bits are used to synchronize the entry into low power modes between the device and debug tool. Supported devices set these bits before a pending entry into low power mode. After reading the bit as set, the debug tool then clears the bit to acknowledge to the device that it may enter the low power mode. 1: Low power mode entry pending 0: Low power mode entry acknowledged
7:1	—	Reserved
0	PSTAT_EN	Processor Status Mode Enable(1) This bit enables processor status (PSTAT) mode. In PSTAT mode, all auxiliary output port MDO pins are used to transmit processor status information, and Nexus messaging is unavailable. 1: PSTAT mode enabled 0: PSTAT mode disabled

1. PSTAT Mode is intended for factory processor debug only. The PSTAT_EN bit should be written to disable PSTAT mode if Nexus messaging is desired. No Nexus messages are transmitted under any circumstances when PSTAT mode is enabled.

Table 446. 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 446. MCKO_DIV values (continued)

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.

31.5 Functional description

31.5.1 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 447 describes the NPC reset configuration options.

Table 447. NPC reset configuration options

JCOMP equal to npc_jcomp_plug?	Port Configuration Register bit		Configuration
	MCKO_EN	FPM	
No(1)	X	X	Reset
Yes	0	X	Disabled
Yes	1	1	Full-Port Mode
Yes	1	0	Reduced-Port Mode

1. Indicates that a block other than the Nexus block has control of the TAP and that all Nexus TAP controllers should be in reset.

31.5.2 Auxiliary output port

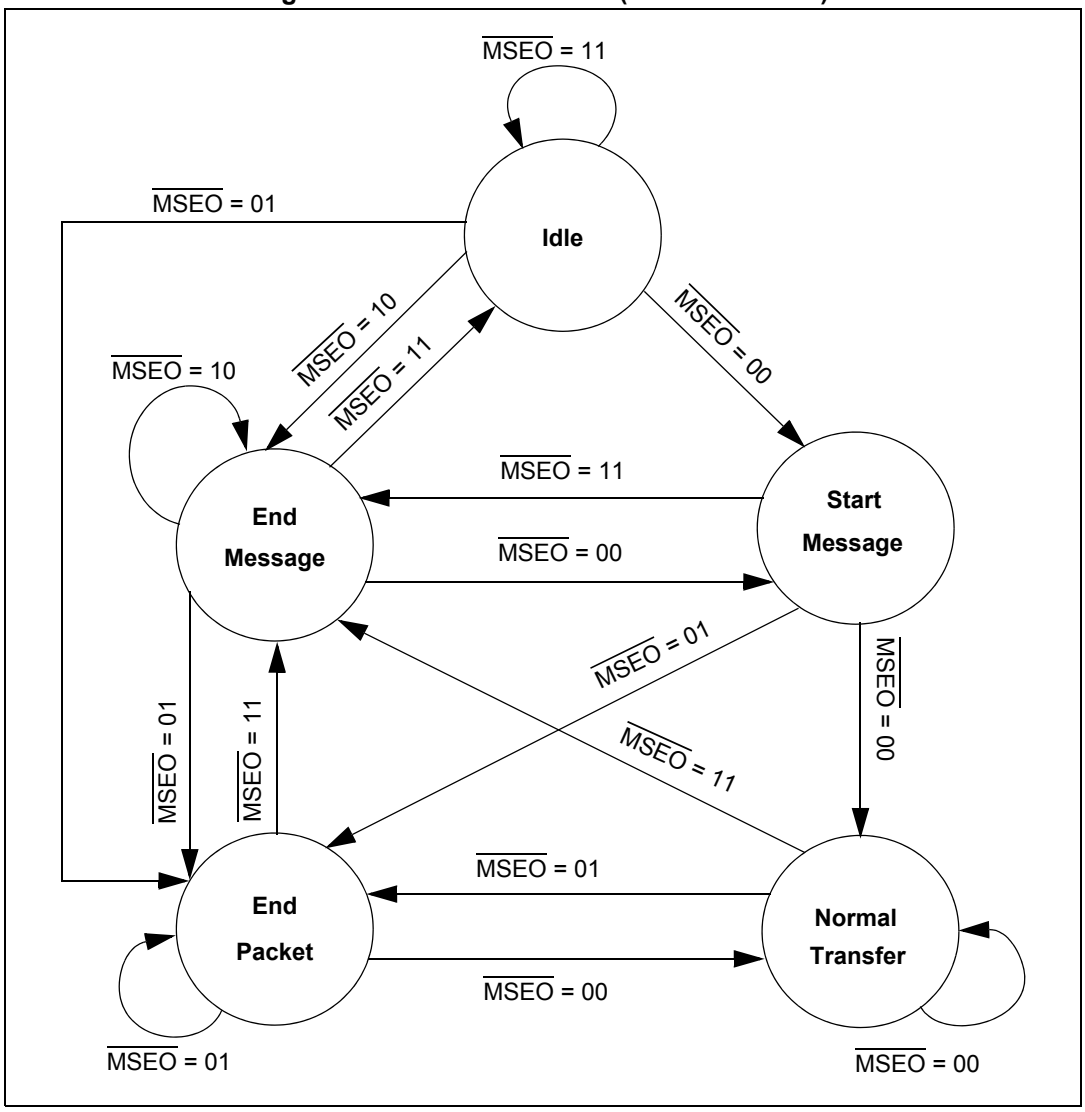
The auxiliary output port is shared by each of the Nexus modules on the device. The NPC communicates with each of the Nexus modules and arbitrates for access to the port.

31.5.2.1 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 634 illustrates the state diagram for MSEO transfers. All transitions not included in the figure are reserved, and must not be used.

Figure 634. $\overline{\text{MSEO}}$ Transfers (for 2-bit $\overline{\text{MSEO}}$)



31.5.2.2 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 448 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 448. 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 635 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).

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

Figure 635. Message field sizes

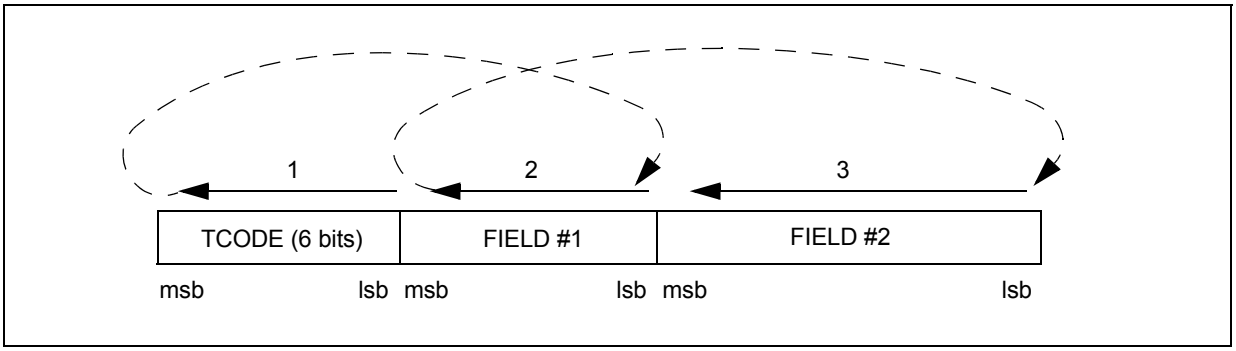
1. Minimum information size. The actual number of bits transmitted depends on the number of MDO pins.
2. Maximum information size. The actual number of bits transmitted depends on the number of MDO pins.

The double edges in *Figure 635* 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.

31.5.2.3 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 636* shows the transmission sequence of a message that is made up of a TCODE followed by two fields.

Figure 636. Transmission sequence of messages



31.5.3 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 638*. The Nexus controller state machine is defined by the IEEE-ISTO 5001-2001 standard. It is shown in *Figure 639*.

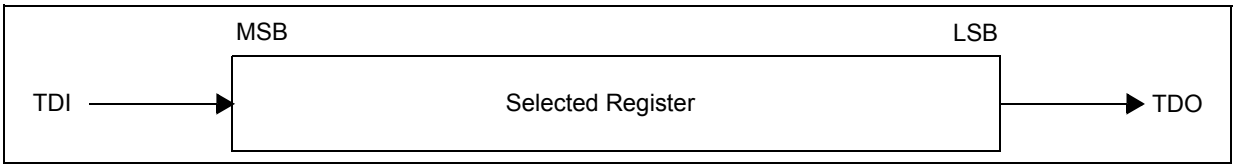
The instructions implemented by the NPC TAP controller are listed in *Table 449*. The value of the NEXUS-ENABLE instruction is 0b0000. Each unimplemented instruction acts like the BYPASS instruction. The size of the NPC instruction register is 4 bits.

Table 449. 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 637*. This applies for the instruction register and all Nexus tool-mapped registers.

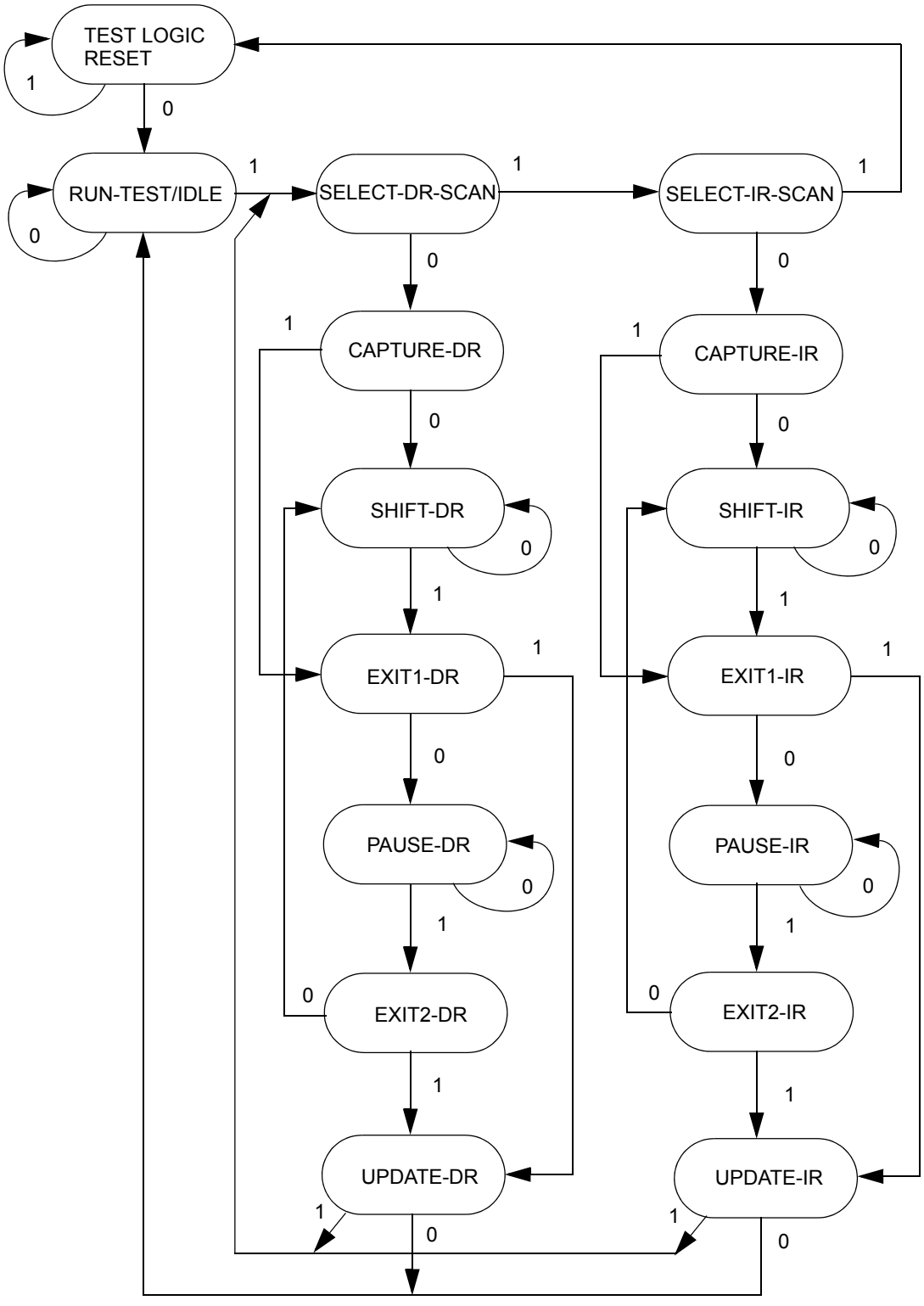
Figure 637. Shifting data into register



31.5.3.1 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 638. 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.

31.5.3.2 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 30.5.3.2: Selecting an IEEE 1149.1-2001 register*.

31.5.3.3 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 639*, transitions to the REG_SELECT state. The Nexus controller has three states: idle, register select, and data access. *Table 450* illustrates the IEEE 1149.1 sequence to load the NEXUS-ENABLE instruction.

Figure 639. NEXUS controller state machine

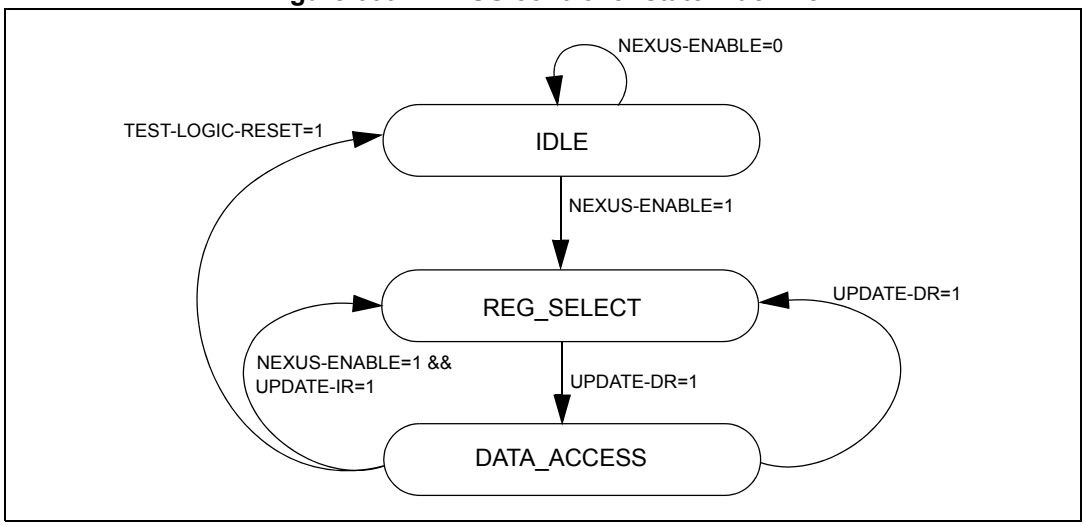


Table 450. Loading NEXUS-ENABLE instruction

Clock	TMS	IEEE 1149.1 state	Nexus state	Description
0	0	RUN-TEST/IDLE	IDLE	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	IDLE	Transitional state
2	1	SELECT-IR-SCAN	IDLE	Transitional state
3	0	CAPTURE-IR	IDLE	Internal shifter loaded with current instruction
4	0	SHIFT-IR	IDLE	TDO becomes active, and the IEEE 1149.1-2001 shifter is ready. Shift in all but the last bit of the NEXUS_ENABLE instruction.
3 TCKS				
12	1	EXIT1-IR	IDLE	Last bit of instruction shifted in
13	1	UPDATE-IR	IDLE	NEXUS-ENABLE loaded into instruction register
14	0	RUN-TEST/IDLE	REG_SELECT	Ready to be read/write Nexus registers

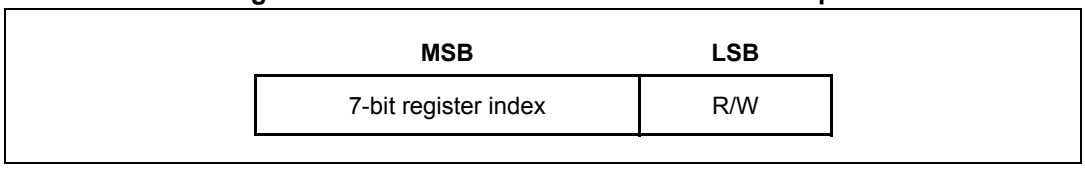
31.5.3.4 Selecting a Nexus client register

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 640*. The read/write control bit is set to 1 for writes and 0 for reads.

Figure 640. 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 451 illustrates a sequence which writes a 32-bit value to a register

Table 451. 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				
11	1	EXIT1-DR	REG_SELECT	Last bit of register index shifted into TDI
12	1	UPDATE-DR	REG_SELECT	Controller decodes and selects register
13	1	SELECT-DR-SCAN	DATA_ACCESS	Second pass through SELECT-DR-SCAN path
14	0	CAPTURE-DR	DATA_ACCESS	Internal shifter loaded with current value of register
15	0	SHIFT-DR	DATA_ACCESS	TDO becomes active, and outputs current value of register while new value is shifted in through TDI
31 TCKs				
47	1	EXIT1-DR	DATA_ACCESS	Last bit of current value shifted out TDO. Last bit of new value shifted in TDI.

Table 451. Write to a 32-bit Nexus client register (continued)

Clock	TMS	IEEE 1149.1 state	Nexus state	Description
48	1	UPDATE-DR	DATA_ACCESS	Value written to register
49	0	RUN-TEST/IDLE	REG_SELECT	Controller returned to idle state. It could also return to SELECT-DR-SCAN to write another register.

31.5.4 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.

31.5.5 MCKO and ipg_sync_mcko

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock and its frequency is determined by the value of the MCKO_DIV field in the PCR. Possible operating frequencies include system clock, one-half system clock, one-quarter system clock, and one-eighth system clock speed.

The NPC also generates an MCKO clock gating control output signal. This output can be used by the MCKO generation logic to gate the transmission of MCKO when the auxiliary port is enabled but not transmitting messages. The setting of the MCKO_GT bit inside the PCR determines whether or not MCKO gating control is active. The MCKO_GT bit resets to a logic 0. In this state gating of MCKO is disabled. To enable gating of MCKO, the MCKO_GT bit in the PCR is written to a logic 1.

31.5.6 $\overline{\text{EVTO}}$ 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.

31.5.7 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.

31.5.8 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.

31.6 Initialization/application information

31.6.1 Accessing NPC tool-mapped registers

To initialize the TAP for Nexus register accesses, the following sequence is required:

1. Enable the Nexus TAP controller
2. Load the TAP controller with the NEXUS-ENABLE instruction

To write control data to NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and set the write bit to select the register with a pass through the SELECT-DR-SCAN path in the TAP controller state machine.
2. Write the register value with a second pass through the SELECT-DR-SCAN path. Note that the prior value of this register is shifted out during the write.

To read status and control data from NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and clear the write bit to select register with a pass through SELECT-DR-SCAN path in the TAP controller state machine.
2. Read the register value with a second pass through the SELECT-DR-SCAN path. Data shifted in is ignored.

See the IEEE-ISTO 5001-2001 standard for more detail.

32 Temperature sensor

32.1 Overview

SPC563Mxx MCUs include an onboard temperature sensor that monitors device temperature and produces a voltage directly proportional to the internal junction temperature. Internal junction temperature must be calculated by software based on the sampled temperature sensor voltage, sampled bandgap voltage and calibration parameter values stored in internal flash memory.

32.2 Detailed description

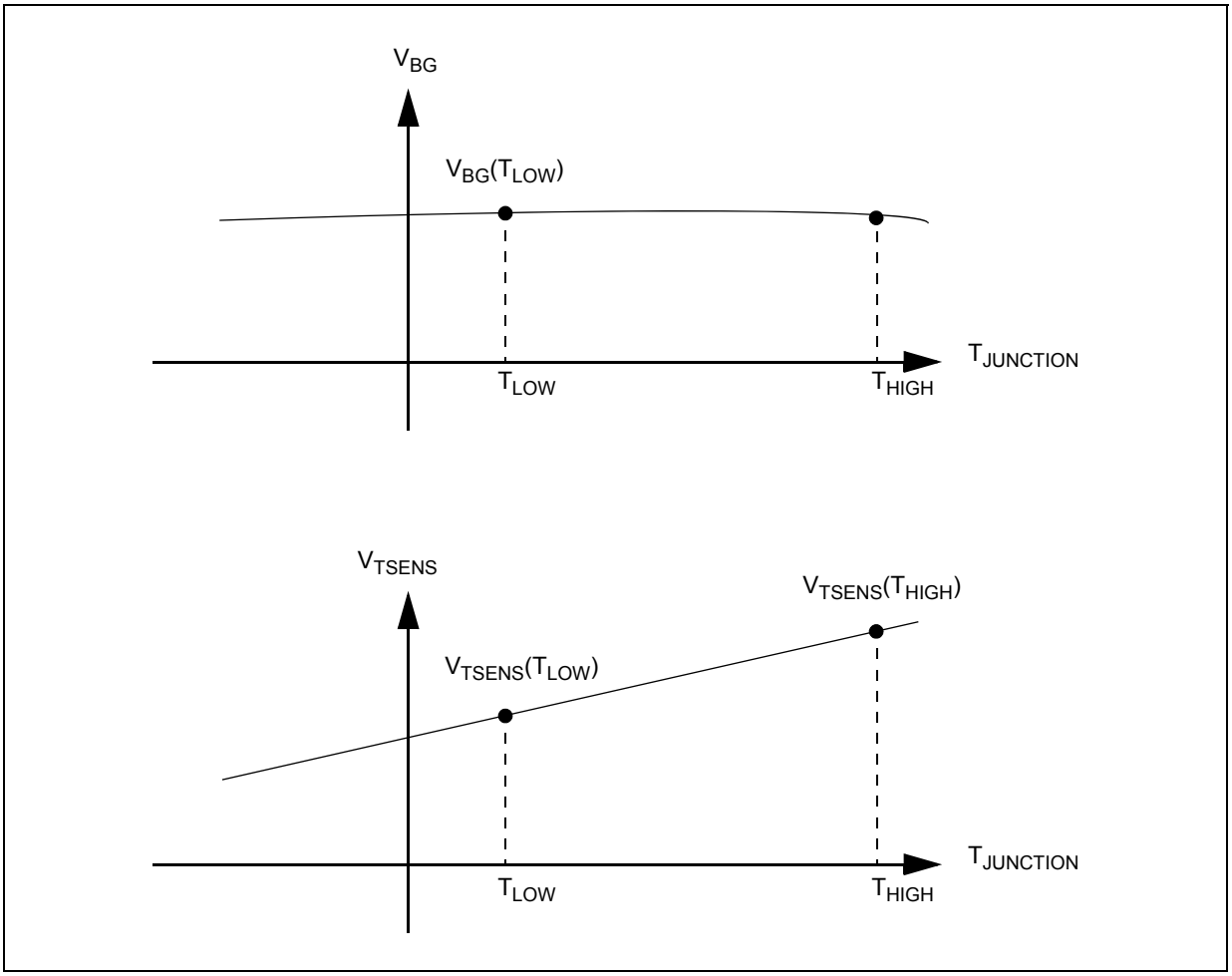
The temperature sensor generates a voltage that increases linearly with temperature. Since the voltage is an amplified version of a ΔV_{BE} (base-to-emitter voltage), it is proportional to absolute temperature. This voltage, $V_{TSENS}(T)$, is read by software using the onboard eQADC module and used with the bandgap voltage and constants stored in flash memory during factory test to calculate device junction temperature.

Five calibration parameters are stored in flash memory during factory test:

- T_{LOW} is the low temperature factory calibration temperature value.
- T_{HIGH} is the hot factory calibration temperature value.
- $V_{BG_CODE}(T_{LOW})$ is the bandgap voltage at low calibration temperature (T_{LOW}) sampled by the eQADC and converted to a 14-bit value.
- $T_{TSENS_CODE}(T_{LOW})$ is the temperature sensor voltage at low calibration temperature (T_{LOW}) sampled by the eQADC and converted to a 14-bit value.
- $T_{TSENS_CODE}(T_{HIGH})$ is the temperature sensor voltage at high calibration temperature (T_{HIGH}) sampled by the eQADC and converted to a 14-bit value.

The calibration points are illustrated in *Figure 641*.

Figure 641. Calibration points



32.3 Temperature formula

The temperature formula is shown in *Figure 642*.

Figure 642. Temperature formula

$$T = T_{LOW} + \frac{T_{TSENS_CODE}(T) \times \beta - T_{TSENS_CODE}(T_{LOW})}{T_{TSENS_CODE}(T_{HIGH}) - T_{TSENS_CODE}(T_{LOW})} \times (T_{HIGH} - T_{LOW})$$

where:

$$T_{TSENS_CODE}(T_{LOW}) = \frac{V_{TSENS}(T_{LOW})}{V_{ref0}} \times 2^{14}$$ (Stored in device flash during factory calibration)

$$T_{TSENS_CODE}(T_{HIGH}) = \frac{V_{TSENS}(T_{HIGH})}{V_{ref0}} \times 2^{14}$$ (Stored in device flash during factory calibration)

$$V_{BG_CODE}(T_{LOW}) = \frac{V_{BG}(T_{LOW})}{V_{ref0}} \times 2^{14}$$ (Stored in device flash during factory calibration)

$$V_{BG_CODE}(T) = \frac{V_{BG}(T)}{V_{ref}} \times 2^{14}$$

$$T_{TSENS_CODE}(T) = \frac{V_{TSENS}(T)}{V_{ref}} \times 2^{14}$$

$$\beta = \frac{V_{BG_CODE}(T_{LOW})}{V_{BG_CODE}(T)}$$

Notes:

1. $V_{TSENS}(T)$ is the temperature sensor output sampled by the ADC
2. $V_{BG}(T)$ is the bandgap voltage sampled by the ADC
3. V_{ref} is the ADC reference voltage
4. V_{ref0} is the ADC reference voltage during factory calibration
5. T_{LOW} is the low temperature factory calibration temperature (stored in device flash)
6. T_{HIGH} is the hot factory calibration temperature (stored in device flash)

The following sections detail the values required and where to get them.

32.3.1 T_{LOW} and T_{HIGH}

T_{LOW} is the factory low calibration temperature; T_{HIGH} is the hot factory calibration temperature. These values are stored in shadow flash memory during factory calibration. See *Section 32.3.6.1: Temperature Calculation Constants Register 0* for details.

32.3.2 $T_{TSENS_CODE}(T_{LOW})$ and $T_{TSENS_CODE}(T_{HIGH})$

$T_{TSENS_CODE}(T_{LOW})$ is the sampled output voltage of the temperature sensor during low temperature factory calibration. $T_{TSENS_CODE}(T_{HIGH})$ is the sampled output voltage of the temperature sensor during hot temperature factory calibration. These values are stored in shadow flash memory during factory calibration. See *Section 32.3.6.1: Temperature Calculation Constants Register 0* for details.

32.3.3 $V_{BG_CODE}(T_{LOW})$

$V_{BG_CODE}(T_{LOW})$ is the value of the bandgap voltage sampled during low temperature factory calibration. This value is stored in shadow flash memory during factory calibration. See *Section 32.3.6.2: Temperature Calculation Constants Register 1* for details.

32.3.4 Temperature sensor voltage ($V_{TENS}(T)$)

$V_{TENS}(T)$ is the output voltage of the device temperature sensor. Software must sample the voltage from eQADC_A channel 128 (ADC0 and ADC1).

32.3.5 Bandgap reference voltage ($V_{BG_CODE}(T)$)

V_{BG} is the bandgap reference voltage. Software must sample the voltage from eQADC_A channel 144 (ADC0).

32.3.6 Registers

The calibration constants described previously, that is , T_{LOW}, T_{HIGH}, $T_{SENS_CODE}(T_{LOW})$, $T_{SENS_CODE}(T_{HIGH})$ and $V_{BG_CODE}(T_{LOW})$, are stored in device shadow flash memory during factory test. This section details the registers where the values reside.

32.3.6.1 Temperature Calculation Constants Register 0

This register contains the calibration temperatures and temperature sensor outputs measured during factory calibration:

- T_{HIGH}
- $T_{SENS_CODE}(T_{HIGH})$
- T_{LOW}
- $T_{SENS_CODE}(T_{LOW})$

Address: 0xFFFE_C000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	THIGH		TSCV2													
W																
RESET:	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TLOW		TSCV1													
W																
RESET:	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

= Unimplemented or Reserved

Figure 643. Temperature Calculation Constants Register 0

Table 452. Temperature Calculation Constants Register 0 field descriptions

Field	Bits	Description
THIGH	0–1	The THIGH field contains a value indicating the hot factory calibration temperature (T_{HIGH}). The values are as follows: 00: T_{HIGH} = Reserved 01: T_{HIGH} = Reserved 10: T_{HIGH} = 145 °C 11: T_{HIGH} = 150 °C
TSCV2	2–15	Temperature sensor output at hot factory calibration temperature ($T_{SENS_CODE}(T_{HIGH})$). TSCV2 is the temperature sensor voltage sampled and converted by the eQADC during factory test with device at hot temperature (T_{HIGH}). This is the $T_{SENS_CODE}(T_{HIGH})$ parameter value referenced in the temperature calculation formula (see *Figure 642*).
TLOW	16–17	The TLOW field contains a code indicating the low factory calibration temperature (T_{LOW}). The values are as follows: 00: T_{LOW} = 25 °C 01: T_{LOW} = 40 °C 10: T_{LOW} = Reserved 11: T_{LOW} = Reserved
TSCV1	18–31	Temperature sensor output at the low factory calibration temperature ($T_{SENS_CODE}(T_{LOW})$). TSCV1 is the temperature sensor voltage sampled and converted by the eQADC during factory test with device at the low calibration temperature. This is the $T_{SENS_CODE}(T_{LOW})$ parameter value referenced in the temperature calculation formula (see *Figure 642*).

32.3.6.2 Temperature Calculation Constants Register 1

This register contains the $V_{BG_CODE}(T_{LOW})$ parameter value used in the temperature calculation.

Address: 0xFFFE_C004

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	Reserved		TSCV3													
W																
RESET:	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x

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

= Unimplemented or Reserved

Figure 644. Temperature Calculation Constants Register 1

Table 453. Temperature Calculation Constants Register 1 field descriptions

Field	Bits	Description
Reserved	0–1	Reserved

Table 453. Temperature Calculation Constants Register 1 field descriptions (continued)

Field	Bits	Description
TSCV3	2–15	Bandgap voltage sampled and converted by ADC during factory test. This is the $V_{BG_CODE}(T_{LOW})$ parameter value referenced in the temperature calculation formula (see *Figure 642*).
Reserved	16–31	Reserved

33 Revision history

Table 454. Document revision history

Date	Revision	Changes
20-Mar-2008	1	Initial release.
04-Jun-2009	2	Formatting, style, and layout improvements in all the chapters. Chapter 2: Memory Map Several address ranges have changed, mostly in “Reserved” areas but also Memory Map: – Reserved memory range changed Detailed Memory Map: – Emulation Remapping of Flash Memory Array used size is now 1.5 MB – Bridge Peripherals (Mirror A)/Platform Flash Configuration used size is now 40 bytes – Used size on several Reserved areas changed Chapter 3: Signal Descriptions – All pinouts changed. Many of the signal names have changed in name only. See SIU changes for details. – I/O Power/Ground Segmentation section added – eTPU Pin Connections and Serialization section added – eMIOS Pin COnnections section added Chapter 11: General-Purpose Static RAM (SRAM) – Feature added: Memory repair done through redundancy provided (2 spare rows) Chapter 10: Flash Memory Functional description information added for memory and Flash controller. Memory organization has changed. Memory map and register definitions updated. – Previously, this chapter detailed C90FL Flash, which was used in rev. 1 devices. Later revisions use LC Flash. FL Flash specifics removed; Flash controller functional details updated—see Functional Description section. – Memory map and register definitions updated. – Flash memory block section replaced with updated info on LC Flash.

Table 454. Document revision history (continued)

Date	Revision	Changes
04-Jun-2009	2 (cont.)	Chapter 15: System Integration Unit (SIU) – BOOTCFG1_IRQ[3]_ETRIG[1]_GPIO[212] is now BOOTCFG1_IRQ[3]_ETRIG[3]_GPIO[212] – The SIU_PCR206 - SIU_PCR207 registers' WPE and WPS fields are now set to 0. – SIU_PCR206 - SIU_PCR207 registers WPE and WPS fields are now set to 0. – MCKO_CLKOUT_OSCCLK_GPIO[219] is now MCKO_GPIO[219] – DSPI, CAN and SCI signal naming conventions changed to be more clear and to match data sheet. Examples: PCS_B ==> DSPI_B_PCS SOUTB ==> DSPI_B_SOUT SINB ==> DSPI_B_SIN SCKB ==> DSPI_B_SCK CNTXB ==> CAN_B_TX CNRXB ==> CAN_B_RX TXDB ==> SCI_B_TX RXDB ==> SCI_B_RX – MUX Select Register 3 (SIU_ISEL3) eTSELx options expanded significantly. – MUX Select Register 8(SIU_ISEL8) section added. – MUX Select Register 8(SIU_ISEL9) section added. Appendix A: Flash Memory (C90FL) This type of Flash memory is not used on Revision 2 devices and later. The chapter is now an appendix.

Table 454. Document revision history (continued)

Date	Revision	Changes
19-Jan-2010	3	Chapter 2: Memory Map Added tables showing detailed Flash and SRAM mapping Chapter 3: Signal Descriptions – GPIO[139] and GPIO[87] pins changed to Medium pads – Some signal names have changed on 176-pin QFP package pinout: “CAL_x” signals renamed to “ALT_x”. – Changes to Signal Properties table: – VDDE7 removed as voltage segment from Calibration bus pins. Calibration bus pins are powered by VDDE12 only. – DSPI connectivity section updated. Diagrams simplified and information organized by DSPI instead of eTPU and eMIOS. Chapter 7: Direct Memory Access (DMA) The SIU outputs to the eDMA are not connected. If the SIU_DIRSR register selects the eDMA output, the effect will be to disable the interrupts. The following registers are not implemented: – DMA_ERQH – DMA_EEIH – DMA_INTH – DMA_ERRH – DCHPRI32-DCHPRI63 – TCD32 – TCD63 Registers/fields renamed to match header file: – DMACR register renamed to DMA_CR register. – All references to DMAERQH register deleted. It is not implemented in this device. – DMAERQL register renamed to DMA_ERQL register. – All references to DMAEEIH register deleted. It is not implemented in this device. – DMAEEIRL register renamed to DMA_EEIRL register. – DMASERQ register renamed DMA_SERQR register. – DMACERQ register renamed to DMA_CERQR register. – DMASEEI register renamed to DMA_SEEIR register. – DMACEEI register renamed to DMA_CEEIR register. – DMACINT register renamed to DMA_CIRQR register. – DMACERR register renamed to DMA_CER register. – DMASSRT register renamed to DMA_SSBR register. – DMACDNE register renamed to DMA_CDSBR register. – All references to DMAINTH register deleted. It is not implemented in this device. – DMAINTL register renamed to DMA_IRQRL register. – All references to DMAERRH register deleted. It is not implemented in this device. – DMAERRL register renamed to DMA_ERL register. – All references to DMAHRSH deleted. It is not implemented in this device. – DMAHRSL register renamed to DMA_HRSL register. – DMAGPOR register renamed to DMA_GPOR register. – All references to DCHPRI32-63 registers deleted. They are not implemented in this device. – DCHPRI0-31 registers renamed to DMA_CPR0-31. DMAES register renamed to DMA_ESR.

Table 454. Document revision history (continued)

Date	Revision	Changes
19-Jan-2010	3 (cont.)	Chapter 8: Multi-Layer AHB Crossbar Switch (XBAR) – There are 4 master ports on this device, including one (Master 2) that is implemented but not used – Crossbar register addresses table corrected. Addresses for unused port registers deleted – Master x general purpose control registers address table added – Note added: High Priority Enable (HPEx) bits are only implemented for masters that are implemented. Chapter 10: Flash Memory Registers/fields renamed to match header file: – PFCR1 register name changed to BIUCR – BIUCR[M3PFE] changed to R/W instead of read-only – BIUCR[BFE] name changed to BIUCR[BFEN] – BIUCR[IPFE] name changed to BIUCR[IPFEN] – BIUCR[DPFE] name changed to BIUCR[DPFEN] – PFAPR register name changed to BIUAPR – PFCR2 register name changed to BIUCR2 – LML register name changed to LMLR – LMLR[MLK] changed to LMLR[MLOCK] – LMLR[SLK] changed to LMLR[SLOCK] – LMLR[LLK] changed to LMLR[LLOCK] – HBL register name changed to HLR – HBL[HLK] changed to HBL[HBLOCK] – SLL register name changed to SLMLR – SLMLR[SMK] changed to SLMLR[SMLOCK] – SLMLR[SSLK] changed to SLMLR[SSLOCK] – SLMLR[SLE] deleted – SLMLR[SLK] changed to SLMLR[SLLOCK] – LMS register name changed to LMSR – LMSR[MSL] changed to LMSR[MSEL] – LMSR[LSL] changed to LMSR[LSEL] – HBS register name changed to HSR – HSR[HSL] changed to HSR[HBSEL] – ADR register name changed to AR – AR[AD] changed to AR[ADDR] Read Reset operation is no longer supported Censorship control word offset from shadow Flash base changed to 0x3DE0 The LMLR register default reset value comes from the Test Flash - Address is 0xBF_FDE8. The HLR register default reset value comes from the Test Flash - Address is 0xBF_FDF0. The SLMLR register default reset value comes from the Test Flash - Address is 0xBF_FDF8. Added test block section including unique serial number info

Table 454. Document revision history (continued)

Date	Revision	Changes
19-Jan-2010	3 (cont.)	Chapter 13: Interrupt Controller (INTC) Multiprocessor features deleted from block diagram Registers/fields renamed to match header file: – Register INTC_BCR renamed to INTC_MCR – Register field INTC_MCR[VTES_PRC0] renamed to INTC_MCR[VTES] – Register field INTC_MCR[HVEN_PRC0] renamed to INTC_MCR[HVEN_PRC0] – Register field INTC_MCR[SIMPBC] deleted – Register INTC_CPR_PRC0 renamed to INTC_CPR – Register INTC_CPR_PRC1 deleted – Register INTC_IACKR_PRC0 renamed to INTC_IACKR – Register INTC_IACKR_PRC1 deleted – Register INTC_EOIR_PRC0 renamed to EOIR – Register INTC_EOIR_PRC1 deleted Chapter 15: System Integration Unit (SIU) Registers/fields renamed to match header file: – The ISEL0 register is now SIU_ETISR – The ISEL1 register is now SIU_EIISR – The ISEL2 register is now SIU_DISR – SIU_CMPAH register renamed to SIU_CARH – SIU_CMPAL register renamed to SIU_CARL – SIU_CMPBH register renamed to SIU_CBRH – SIU_CMPBL register renamed to SIU_CBRL Changes to SIU_MIDR2 register: – SIU_MIDR2[S/F] field is now SIU_MIDR2[S_F] – SIU_MIDR2[Flash Size 1] is now SIU_MIDR2[FLASH_SIZE_1] – SIU_MIDR2[Flash Size 2] is now SIU_MIDR2[FLASH_SIZE_2] – SIU_MIDR2[Temp Range] is now SIU_MIDR2[TEMP_RANGE] – SIU_MIDR2[Max Freq] is now SIU_MIDR2[MAX_FREQ] – SIU_MIDR2[Supply] is now SIU_MIDR2[SUPPLY] – SIU_MIDR2[Part Number] is now SIU_MIDR2[PART_NUMBER] Offset for SIU_SRCR register is 0x10 (was 0xE) Info added to SIU_ETISR section describing interaction with SIU_ISEL3 register. Register fields added: – SWT_SEL field added to SIU_DIRER register – SWTRE field added to SIU_IREER register – SWTFE added to SIU_IFEER register

Table 454. Document revision history (continued)

Date	Revision	Changes
19-Jan-2010	3 (cont.)	Chapter 16: Frequency-Modulated Phase Locked Loop (FMPLL) Descriptions added to clarify relationship between f_{SYS} and f_{VCO} in legacy mode Updated SYNCR[MFD] description Updated SYNCR[RFD] description Updated Clock Configuration section with formulas and text that describe the relationship in both legacy and enhanced modes Chapter 17: Error Correction Status Module (ECSM) Added EEGR register per direction from Randy Dees. This register, though marked as non_cust in some documents is used by customers to test their code. Registers/fields renamed to match header file: – Changed ECR register field EPRNCR to ERNCR – Changed ECR register field EPFNCR to EFNCR – Changed ESR register field PRNCE to RNCE – Changed ESR register field PFNCE to FNCE – Changed PFEAR register name to FEAR – Changed PFEMR register name to FEMR – Changed PFEAT register name to FEAT – Changed PFEDRH register name to FEDRH – Changed PFEDRL register name to FEDRL – Changed PREAR register name to REAR – Changed PREMR register name to REMR – Changed PREAT register name to REAT – Changed PREDRH register name to REDRH – Changed PREDRL register name to REDRL New fields added to the ECSM_ECR register: – ER1BR enables/disables single-bit error reporting for SRAM – EF1BR enables/disables single-bit error reporting for Flash New fields added to the ECSM_ESR register: – R1BC reports single-bit error correction for SRAM – F1BC reports single-bit error correction for Flash Chapter 20: Boot Assist Module (BAM Boot from the calibration bus is not supported Address for serial boot password corrected Section detailing how to enable debug in a censored device added

Table 454. Document revision history (continued)

Date	Revision	Changes
19-Jan-2010	3 (cont.)	Registers/fields renamed to match header file: – EMIOSS[n] register is now named EMIOS_CSR[n] – EMIOSGFLAG register is now named EMIOS_GFR – EMIOSMCR register is now named EMIOS_MCR – EMIOSOUDIS register is now named EMIOS_OUDR – EMIOSUCDIS register is now named EMIOS_UCDIS – EMIOSA register is now named EMIOS_CADR – EMIOSB register is now named EMIOS_CBDR – EMIOSCNT register is now named EMIOS_CCNTR – EMIOSC is now named EMIOS_CCR – EMIOSALTA is now named EMIOS_ALTA Chapter 22: Enhanced Time Processing Unit (eTPU2) Changed register field name: MISCCMPR[EMISCCMP] is now MISCCMPR[ETPUMISCCMP] Chapter 23: Enhanced Queued Analog-to_Digital Converter (eQADC) – REDLCCR register is not implemented Bias generator section removed. This device does not have a bias generator. Chapter 26: Enhanced Serial Communication Interface (eSCI) Device specific information section added. Registers converted to 16- and 32-bit format to match header file: – Fields formerly found in registers SCIBDH, SCIBDL, SCICR1 and SCICR2 are now contained in SCI_CR1 – Fields formerly found in registers SCICR3 and SCICR4 are now contained in SCI_CR2 – Registers SCIDRH and SCIDRL have been combined into a single register named SCI_DR – Fields formerly found in registers SCISR1, SCIRSR2, LINSTAT1 and LINSTAT2 are now contained in SCI_SR – Fields formerly found in registers LINCTRL1, LINCTRL2 and LINCTRL3 are now contained in SCI_LCR – The LINTX register is now SCI_LTR – The LINRX register is now SCI LRR – Fields formerly found in registers LINCRCP1, LINCRCP2 and SCICR5 are now contained in SCI_LPR. Field name changes: – SCICR3[BERIE] is now SCI_CR2[IEBERR] – SCICR3[BRCL] is now SCI_CR2[BRK13] – SCICR4[BESM] is now SCI_CR2[BESM13] – SCICR4[BESTP] is now SCI_CR2[SBSTP] – SCIDRH[RN] is now SCI_DR[R8] – SCIDRH[TN] is now SCI_DR[T8] – SCISR2[RACT] is now SCI_SR[RAF]

Table 454. Document revision history (continued)

Date	Revision	Changes
19-Jan-2010	3 (cont.)	Chapter 27: FlexCAN Module MCR register is now named CAN_MCR Changes to CAN_MCR register field names: – NOT_RDY is now NOTRDY – SOFT_RST is now SOFTRST – FRZ_ACK is now FRZACK – WRN_EN is now WRNEN – LPM_ACK is now MDISACK – SRX_DIS is now SRXDIS – BCC is now MBFEN CTRL register is now named CAN_CR Changes to CAN_CR register field names: – BOFF_MSK is now BOFFMSK – ERR_MSK is now ERRMSK – CLK_SRC is now CLKSRC – TWRN_MSK is now TWRNMSK – RWRN_MSK is now RWRNMSK – BOFF_REC is now BOFFREC ESR register is now named CAN_ESR Changes to CAN_ESR register field names: – TWRN_INT is now TWRNINT – RWRN_INT is now RWRNINT – BIT1_ERR is now BIT1ERR: – BIT0_ERR is now BIT0ERR – ACK_ERR is now ACKERR – CRC_ERR is now CRCERR – FRM_ERR is now FRMERR – STF_ERR is now STFERR – TX_WRN is now TXWRN – RX_WRN is now RXWRN:1 – FLT_CONF is now FLTCONF – BOFF_INT is now BOFFINT – ERR_INT is now ERRINT IMASK2 register is now named CAN_IMRH IMASK1 register is now named CAN_IMRL IFLAG2 register is now named CAN_IFRH IFLAG1 register is now named CAN_IFRL TIMER register is now named CAN_TIMER RXGMASK register is now named CAN_RXGMASK RX14MASK register is now named CAN_RX14MASK RX15MASK register is now named CAN_RX15MASK ECR register is now named CAN_ECR RXIMRn registers renamed to CAN_RXIMRn

Table 454. Document revision history (continued)

Date	Revision	Changes
19-Jan-2010	3 (cont.)	Note added to Global Receive Mask section: “If the FIFO is enabled and individual masks is not enabled (MBFEN=0) it is impossible to use the Global masks for FIFO and MB at same time. If FIFO is enabled, no MBs can be configured as RX. Note added to Rx14 Mask section: “It is impossible to use the Rx 14 mask for FIFO and MB14 at same time. If Note added to Rx15 Mask section: “It is impossible to use the Rx 15 mask for FIFO and MB15 at same time. If FIFO is enabled and MBFEN = 0, MB15 cannot be configured as RX. Chapter 29: Power Management Controller (PMC) – “Low power RAM test” added to list of operating modes. – In the Functional Description section, detail added regarding disabling the voltage regulators. – PMC Signals table updated: * VDDREG is 4.5-5.5V (was 4.0- 5.5V) * VDD3p3 is 3.3-3.6V (was 3.0-3.6V) * VDD1p1 is 1.2-1.32V (was 1.08-1.32V) – VDDREG requires decoupling capacitor on the order of 4.7 µF - 20 µF (was 1.0 µF - 20 µF) – Vdd1p2: bypass capacitor ESR max is 50mΩ (was 10mΩ). Ceramic capacitor value is 100nF (was 200nF). Deleted sentence stating: “ When switching current load is lower, it is possible to reduce the requirements of the bypass capacitor to 1 µF - 5 µF and 100 mΩ ESR.“ – CFGR register is now named PMC_MCR – PMC_MCR register reset value is 0x98000000 (was 0x00000000) – NVUSRO register info added--contains bit used to shutdown 3.3V regulator. NVUSRO[V33EN]: A logic 1 in this field enables the 3.3 V regulator and a 0 disables it. – Changes to PMC_MCR register: * LVRE5 field is now LVRE50 * LVIE5 field is now LVIE50 – TRIMR register is now named PMC_TRIMR – Changes to PMC_TRIMR register: * LVI50TRIM is now LVDREGTRIM * V33TRIM is now VDD33TRIM * LVI33TRIM is now LVD33TRIM * V12TRIM is now VDDCTRIM * LVI12TRIM field is now LVDCTRIM * Updated VDD33TRIM values * Updated VDDCTRIM values * Updated LVDCTRIM values – SR register is now named PMC_SR – PMC_SR register reset value is “0x03000000 or 0x06000000 (was 0x02000000 or 0x06000000)

Table 454. Document revision history (continued)

Date	Revision	Changes
19-Jan-2010	3 (cont.)	Changes to PMC_SR register field names: – BRW field is now LVFVSTBY – LVI5C is now LVFC50 – LVI3C is now LVFC33 – LVI1C is now LVFCC – BGS1 is now BGRDY – BGS2 is now BGTS – V33S is now V33DIS – LVIRC is now LVFCR – LVIHC is now LVFCH – LVI5 is now LVFC50 – LVI3 is now LVFC33 – LVI1 is now LVFCC – LVIRF is now LVFR – LVIHF is now LVFH – LVI5F is now LVF50 – LVI3F is now LVF33 – LVI1F is now LVFC – Updated PMC_SR VDD33TRIM values – In functional description bandgap reference of 2.7V deleted. – In bandgap section, description of 1.219V reference voltage “...that varies by ±4% before calibration and ±1% after calibration over temperature and lifetime” deleted. – In 5V LVI section, following statement deleted: “ Maximum hysteresis value between rising and falling trip points is 90 mV” – In 5V LVI section, “In case the monitored voltage falls below the *nominal* trip point, the LVI output goes to logical 0” changed to, “In case the monitored voltage falls below the *falling* trip point, the LVI output goes to logical 0“. – In LVI section, description of resistor chain deleted. – Nominal 4.29V trip value noted as being typical and subject to variance. – In 3.3V internal voltage regulator section, following text deleted: “Tolerance of the 3.3 V supply is -5% / +10% including line and load variation. Detail on disabling voltage regulator added. – In 3.3V LVI section, noted that there are 2 LV monitors. Also deleted hysteresis spec. – In 3.3V LVI section noted 3.09V (was 3.00V) trip value as being typical and subject to variance. – In 1.2V regulator section, “tolerance of ±10%” on 1.2V supply current deleted. – In 1.2V regulator section, default voltage is 1.28 V (was 1.270V). – 1.2V LVI section: hysteresis spec deleted. Rising trip point is 1.16 (was 1.08). – Power On Reset section: typical values table deleted. – Sections added describing modules affected by PMC. – ADC test mux section added – Electrical characteristics removed. See data sheet. – Replaced PMC block diagram – Replaced bandgap reference block diagram

Table 454. Document revision history (continued)

Date	Revision	Changes
19-Jan-2010	3 (cont.)	– Replaced Vreg 3.3 V power connection diagram – PMC ADC monitor channels table added to device specific information section. – Note added to External Signals section: The VDD33 supply on this device is routed to the VRC33 pin. – PMC ADC monitor channels table expanded Chapter 30: JTAG Controller (JTAGC) – Added JTAG ID table and CENSOR_CTRL code. – Added info ENABLE_CENSOR_CTRL instruction. Chapter 31: Nexus Port Controller (NPC) Censorship information added Chapter 32: Temperature Sensor New Revision History Updated revision history for changes between Rev 2 and Rev 3.
02-Jul-2010	4	Chapter Introduction Correction to features list: – eQADC supports four external 8-to-1 muxes which can expand the input channel number from 34 to 59 (was 31 to 59). Update to feature details: – Programming feature: eTPU2 channel flags can be tested – Amount of embedded memory in FlexCAN module A changed to 1056 bytes (was 1088 bytes) Chapter Signal Descriptions Pinout/ballmap changes: 100 pin LQFP package: – Pin 31 is now VDDEH1B (was VDDEH4A) – Pin 43 is now VDDEH6A (was VDDEH4B) 144 pin LQFP package: – Pin 46 is now VDDEH1B (was VDDEH4A) – Pin 61 is now VDDEH6A (was VDDEH4B) 176 pin LQFP package (1.5M devices) – Pin 55 is now VDDEH1B (was VDDEH4A) – Pin 74 is now VDDEH6A (was VDDEH4B) 176 pin LQFP package (1.5M devices) – Pin 55 is now VDDEH1B (was VDDEH4A) – Pin 74 is now VDDEH6A (was VDDEH4B) 208 ball BGA package (all devices) Ball N9 changed to VDDEH1/6 (was VDDEH6). In a future revision of the device this may be changed to NC (no connect).

Table 454. Document revision history (continued)

Date	Revision	Changes
02-Jul-2010	4 (cont.)	Chapter Signal Descriptions (continued) Changes to calibration ball names on devices with 1 MB Flash memory: – CAL_MDO0 changed to ALT_MDO0 – CAL_MDO1 changed to ALT_MDO1 – CAL_MDO2 changed to ALT_MDO2 – CAL_MDO3 changed to ALT_MDO3 – CAL_MSEO0 changed to ALT_$\overline{\text{MSEO0}}$ – CAL_MSEO1 changed to ALT_$\overline{\text{MSEO1}}$ – CAL_$\overline{\text{EVTI}}$ changed to ALT_$\overline{\text{EVTI}}$ – CAL_$\overline{\text{EVTO}}$ changed to ALT_$\overline{\text{EVTO}}$ – CAL_MCKO changed to ALT_MCKO Clarification: following pins are available only on the 496 CSP/MAPBGA calibration/development package: – CAL_ADDR[12:15] – NEXUSCFG – CAL_$\overline{\text{CS}}$[0] – CAL_$\overline{\text{CS}}$[2] / CAL_ADDR[10] – CAL_$\overline{\text{CS}}$[3] / CAL_ADDR[11] – CAL_DATA[0:9] – CAL_DATA[10:15] – CAL_$\overline{\text{OE}}$ – CAL_RD_$\overline{\text{WR}}$ – CAL_$\overline{\text{TS}}$_ALE – CAL_$\overline{\text{WE}}$_BE[0:1] Corrections to PCR[PA] values: – PCR[215] has a 3-bit PA field. 0b100 selects eTPU function – PCR[216] has a 3-bit PA field. 0b100 selects eTPU function Correction to pad type: – eTPU_A[6] / eTPU_A[18] / DSPI_B_SCK_LVDS+ / GPIO[120] pin has a Medium pad (was Slow). Correction to pad voltage: – CLKOUT pin is in VDDE5 power segment for all packages (was VDDEH7 for 208-ball package). Footnote deleted from signal properties table: “VDDE12/VSSE12 pins are to be used for decoupling capacitors only.“ (VDDE12 is a power supply input) In signal properties table VDDE12 renamed to VDDE7. They are the same but the signal is named VDDE12 in the calibration package and VDDE7 in all other packages.

Table 454. Document revision history (continued)

Date	Revision	Changes
02-Jul-2010	4 (cont.)	Chapter Signal Descriptions (continued) Power/ground segment changes: – Power segments VDDEH4, VDDEH4A and VDDEH4B have been removed. – CLKOUT power segment is VDDE5 (was VDDE12) – Note added: VDDEH1A, VDDEH1B, and VDDEH1AB are shorted together in all production packages. The separation of the signal names is present to support legacy naming, however they should be considered as the same signal in this document. – Note added: VDDEH6A, VDDEH6B are shorted together in all production packages. The separation of the signal names is present to support legacy naming, however they should be considered as the same signal in this document. – Correction: VDDE5 (208-ball package only) range is 1.8V-3.3V (was 3.3V - 5.0V) Updates to detailed signal descriptions: – VSTBY has two ranges – VDDE5 added Chapter Direct Memory Access (DMA) Channel groups 2 and 3 are not implemented. Chapter Flash Memory Overview memory map updated: – Bank 1, array 2 Test Block is now 992 bytes (was 768 bytes) and begins at base + 0x00B7_FB20. – Bank 1, array 1 Test Block is now 992 bytes (was 768 bytes) and begins at base + 0x00BB_FB20. – Bank 0, array 0Test Block is now 992 bytes (was 768 bytes) and begins at base + 0x00BF_FB20. Warning note added: The Analog power supply VDD (pins 82, 120, and 149 for 100-, 144-, and 176-pin packages) are shared by Array1 and Array2. Consequently, a read performed while reading or writing the other will generate a wrong value. Added OTP test blocks to Flash Memory Storage Address Map. Added LMLR register addresses for Bank1 Array1 and Bank1Array2. Also added note on SLOCK field description: field is also used to unlock the Flash test sector. Added note to SLMLR[SSLOCK] field description that field description: field is also used to unlock the Flash test block.

Table 454. Document revision history (continued)

Date	Revision	Changes
02-Jul-2010	4 (cont.)	Chapter External Bus Interface (EBI) Following pins are not supported: CAL_CS1, TA and TSIZ[0:1] In the EBI Base Registers (EBI_BR0-EBI_BR3, EBI_CAL_BR0-3): – The BI (Burst Inhibit) field is writable but is not supported since burst is not supported. It must always be set to 1. – The LWRN (Late RD_$\overline{WR}$ Negation) field is not supported. – The EOE (Early $\overline{OE}$) field is not supported. – The SBL (Short Burst Length) field is not supported. – The TBDIP (Toggle Burst Data in Progress) field is not supported. – The GCSN (Guarantee $\overline{CS}$ Negation) field is not supported. – The SETA (Select External Transfer Acknowledge) field is writable but is not supported since burst is not supported. EBI_MCR reset value changed. (Reset values for D16_31 anad DBM fields were x.) Chapter System Integration Unit (SIU) – PCR general description updated and sample PCR register map added. – PA value table added to SIU_PCR218 section SIU_ISEL9[eTSEL0A] values list updated: – 0x00000 is now reserved (was GPIO206 (eTRIG0)) – 0x00111 is now reserved (was BOOTCFG1 (eTRIG3) SIU_ECCR register reset value changed to 0x0000_1001 (was 0x0000_0001) Chapter Frequency-Modulated Phase Locked Loop (FMPLL) Changes to Clock Quality Monitor diagram: – RC OSC block is 16 MHz (was 4 MHz) – Divide-by-4 block added between RC Osc and Counter 1. Loss-of-Clock Detection section updated to state that a 16 MHz RC oscilator and clock divider are used to generate the reference time base. Chapter Software Watchdog Timer (SWT) Correction: the default configuration is for SWT enabled (WEN=1) out of reset (was WEN=0). Chapter Boot Assist Module (BAM) Correction to BAM program flow chart: “MIDR[15]=0?” decision block changed to MIDR[16]. SIU_MIDR[16]=1 indicates calibration package.

Table 454. Document revision history (continued)

Date	Revision	Changes
02-Jul-2010	4 (cont.)	Chapter Configurable Enhanced Modular I/O Subsystem (eMIOS200) Reset value of EMIOS_UCDIS register is 0x0000_0000 Chapter Enhanced Queued Analog-to-Digital Converter (EQADC) Added details about delay of eQADC abort feature. Streaming Mode section removed from “Modes of Operation” section. Streaming mode is not a module function—it is only related to CFIFO0. Clarification added to EQADC_CFCRx[AMODE]: It is necessary to set to "Disabled" before changing to different a different trigger type. EQADC_CFCRx[AMODE] value 0b0001 is reserved (was Software Trigger, Single Scan). Correction: EQADC_CFCRx[AMODE] is not related to single scan mode. Correction: ADC0/1_AORx register reset value is 0x0000_0000 (was 0x001F_0000). ADC_REDTBRx registers deleted. Changes to Non-multiplexed Channel Assignments Table – Channel 194/ADC1 is reserved (was VDD12) – Channel 195/ADC1 is reserved (was LVI12) – Channel 197/ADC1 is VDD33 (was LVI33) – Channel 198/ADC1 is VDD12 (was LVI50 50%) Changes to Multiplexed Channel Assignments Table – Channel 194/ADC1 is reserved (was VDD12) – Channel 195/ADC1 is reserved (was LVI12) – Channel 197/ADC1 is VDD33 (was LVI33) – Channel 198/ADC1 is VDD12 (was LVI50 50%) Chapter Enhanced Serial Communication Interface (eSCI) Corrections to LIN Control Register (SCI_LCR) field descriptions: – WUD0 and WUD1 field descriptions combined into WUD[0:1] – LDBG description corrected. – DSF description corrected. – PRTY description corrected. – LIN description corrected. Chapter JTAG Controller (JTAGC) – Part identification number (PIN) changed to 0x201 (was 0x200) – Boundary scan length changed to 264 (was 248)

Table 454. Document revision history (continued)

Date	Revision	Changes
02-Jul-2010	4 (cont.)	Chapter Power Management Controller (PMC) "Core Voltage Regulator Controller External Components Preferred Configuration" circuit diagram updated. Chapter Temperature Sensor Changes to Temperature Calculation Constants Register 0: – Split TSCV2 and TSCV1 into two fields each—a calibration temperature field and a temperature sensor calibration value. – Reset value changed. Change to Temperature Calculation Constants Register 0: – Reset value changed.
10-Nov-2011	5	Chapter Introduction Replaced all instances of "SPC563M54" and "SPC563M60" with "SPC563M54P" and "SPC563M60P", respectively. Changed the "Standby SRAM size" of the SPC563M60P device from "24" to "32". Replaced all instances of "Microsecond Bus" with "Microsecond Channel". Changed the "Processor core" information of the SPC563M54P, SPC563M60P, and SPC563M64 device to: "32-bit e200z335 with SPE and FPU support". Replaced all instances of "Microsecond Bus" with "Microsecond Channel". Replaced all instances of "downlink" with "downstream". Replaced all instances of "uplink" with "upstream". Made the following changes in the "Flash" section: – Changed "Program page size of 128 bits (four words) to accelerate programming" to "Program page size of 64 bits (two words)". – Changed "Erase suspend, program suspend and erase-suspended program" to "Erase suspend". Added the the following to the "Available only in the calibration package" sentence in the Features section ((496 CSP package). Chapter Memory Map Replaced all instances of "SPC563M60" with "SPC563M60P" and "SPC563M54" with "SPC563M54P", respectively. Chapter Operating modes Updated the "System clock divider" description. Chapter e200z335 Core Added the following to the "Features" section: "Memory Management Unit (MMU) with 8-entry fully-associative translation look-aside buffer".

Table 454. Document revision history (continued)

Date	Revision	Changes
10-Nov-2011	5 (cont.)	Chapter Flash memory – Updated the “APC, RWSC, WWSC Settings vs. Frequency of Operation” table. – Deleted the following sentence from the “Modify Operations” section: “If a modify operation is on-going in an array then it is forbidden to start any other modify operation in the other arrays on the device.” – Updated the description of the “SLE” bit of the “SLMLR” register. Chapter External Bus Interface for Calibration Bus Replaced “eSys” with “MPC55xx”. Chapter System Integration Unit (SIU) – Changed the reset value of the ‘BYPASS’ bit of the ‘SIU_SYSDIV’ register from ‘0’ to ‘1’. – Changed “*CAL_MDO*” to “*ALT_MDO*”. Chapter Configurable Enhanced Modular IO Subsystem (eMIOS200) Added a footnote for channel numbers 1, 13, and 15 in the “eMIOS Channel Connections” table. Chapter Enhanced Time Processing Unit (eTPU2) Made the following changes in the features section: – Added information about “eTPU2 enhancements over eTPU”. – Updated “Test and development support features”. Chapter Enhanced Queued Analog-to-Digital Converter (eQADC) – Added the following note to the “ADC pre-gain feature” section: “This feature is valid only for differential channels.” – Updated the “Multiplexed Channel Assignments” table. Chapter Deserial Serial Peripheral Interface (DSPI) Added the following to the “Device-Specific Features” section: “Parity is not implemented in this device.” Removed information about the following registers: – DSPI DSI Serialization Source Select Register (DSPI_SSR) – DSPI DSI Parallel Input Select Register 0 (DPSI_PISR0)4 – DSPI DSI Parallel Input Select Register 1 (DPSI_PISR1)4 – DSPI DSI Parallel Input Select Register 2 (DPSI_PISR2)4 – DSPI DSI Parallel Input Select Register 3 (DPSI_PISR3)4 – DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR) – DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR) – DSPI Hardware Configuration Register (DSPI_HCR) Removed a paragraph related to PISR from the “DSI Serialization” section.

Table 454. Document revision history (continued)

Date	Revision	Changes
11-Nov-2011	6	Chapter Signal Descriptions Made minor editorial changes in the “100 LQFP100 Pinout” section. Added PA settings for TS versus ALE for the “Calibration Transfer Start Address Latch Enable” function in the “MSPC563Mx signal properties” table. Added footnotes (26) and a sentence to footnote (11) of the “MSPC563Mx signal properties” table. Previous errata err002279 integrated into the reference manual: Added footnote (25) to the “MSPC563Mx signal properties” table. Chapter Peripheral Bridge Updated the note in the “Introduction” section. Chapter C90LC Flash Memory Changed the chapter title from “Flash Memory” to “C90LC Flash Memory”. Updated the warning in “Programming considerations” section. Removed the “APC, RWSC, WWSC settings vs. frequency of operation” table. Chapter System Integration Unit (SIU) Updated information about Pad Configuration Register 350 – 413. Removed the “MAX_FREQ” and “TBD” bit fields from the “MCU ID Register 2”. Updated the “SUPPLY” and “FR” descriptions in the “MCU ID Register 2 field descriptions” table. Chapter Boot Assist Module (BAM Previous errata err002279 integrated into the reference manual: Added a footnote to the “CAN/eSCI pin configuration for CAN/eSCI fixed baud rate boot modes” table. Previous errata err001722 integrated into the reference manual: Added a paragraph to the “Download protocol execution” section. Chapter Enhanced Time Processing Unit (eTPU2) Updated the description of the “Angle mode selection” bit field in the “ETPU_TBCR field descriptions” table. Chapter Enhanced Queued Analog-to-Digital Converter (eQADC) Previous errata err002449 integrated into reference manual: Added a note to the “ADC0/1_EMUX” description of the “EQADC_CFxRw Field Descriptions” table. Previous errata err0000652 integrated into reference manual: Updated a footnote of the “Non-multiplexed channel assignments” and “Multiplexed channel assignments” tables. Previous errata err001741 integrated into reference manual: Added a note to the “ADC result calibration” section and added a footnote to the “Calibration example” table. Chapter Deserial Serial Peripheral Interface (DSPI) Previous errata err001147 integrated into the reference manual: Added a note to the “Parallel Chaining” section. Previous errata err000575 integrated into the reference manual: Added a warning to the “Continuous Selection Format” section. Previous errata err003105 integrated into the reference manual: Added a note to the “Continuous Selection Format” section. Previous errata err003230 integrated into the reference manual: Updated the bullets in the “Continuous serial communications clock” section.

Table 454. Document revision history (continued)

Date	Revision	Changes
11-Nov-2011	6 (cont.)	Chapter FlexCAN Module Previous errata err002685 integrated into the reference manual: Made editorial changes in the “Module Disable Mode” descriptions (“Modes of Operation” and “Modes of Operation Details” sections). Previous errata err002656 integrated into the reference manual: Made the following changes: – Editorial changes in the “Transmit Process” section. – Added a note to the “Transmission Abort Mechanism” section. Previous errata err002360 integrated into the reference manual: Added “Precautions when using Global Mask and Individual Mask registers” section. Chapter Power Management Controller (PMC) Removed the “Application information” section. Chapter Flash Memory (C90FL) Deleted the chapter.
03-Oct-2012	7	Updated title of the document.
17-Sep-2013	8	Updated Disclaimer.
21-Apr-2016	9	Formatting, style, and layout improvements in all the chapters. Replaced all instances of “SPC563M64” with “SPC563Mxx” and “SPC563M60P” with “SPC563M60”, respectively. Removed all occurrences of “SPC563M54P”, “LQFP100”, “LBGA208”, “100 QFP”, and “208 BGA” *Chapter 1: Introduction* In *Table 1: SPC563Mxx family device summary*: – Removed “SPC563M54P” column – Removed “LQFP100” from packages row in “SPC563M60” column – Updated note from “The 176-pin and 208-pin packages...” to “The 176-pin package...” In *Section 1.4.1: Feature List*: – Changed bullet “32 channels (each channel has dedicated I/O pin in all packages except LQFP100)” in “”32 channels (each channel has dedicated I/O pin in all packages)” – Removed the occurrences of “208-pin packages”, “208-ball packages”, and “100 pin package” *Chapter 2: Memory Map* In *Table 7: SPC563Mxx family device SRAM memory map*, updated “End address” and “Description” in row with “start address” equal to 0x4000_0000 Removed Section “LQFP100 Pinout (all 100-pin devices)”

Table 454. Document revision history (continued)

Date	Revision	Changes
21-Apr-2016	9 (cont.)	*Chapter 3: Signal Descriptions* Renamed Section “LQFP176 Pinout (SPC563M64)” in *Section 3.1.2: LQFP176 Pinout (all 176–pin devices)* Removed Section “LQFP176 Pinout (SPC563M60)” Removed Section “LBGA208 Ballmap (SPC563M64)” Removed Section “LBGA208 Ballmap (SPC563M60 only)” In *Table 8: SPC563Mx signal properties*: – Removed “VDDE7” from “Voltage/Pad Type” column – Removed “VDDE5” row – Removed note “Not available on 1 MB version of 176-pin package” In *Section 3.3.13: Power / Ground*, removed “VDDE5 I/O Supply Input” and “VDDE7 I/O Supply Input” In *Table 10: SPC563Mx Power/Ground Segmentation*, removed row related to “VDDE5” power segment *Section 3.3.4: Nexus Port Controller (NPC)*, *Section 3.3.10: eTPU*, and *Section 3.3.11: eMIOS*: in selected subsection titles, changed “eTPU_A Channel” in “eTPU_A Channel (Output Only)” *Chapter 4: Resets* In *Table 15: Reset Configuration Half Word (RCHW) field descriptions*, changed “Classic PowePC BookE” with “Power Architecture technology” *Chapter 6: e200z335 Core* In *Section 6.2: Features*, updated bullet “Memory Management Unit (MMU) with 8-entry...” with “Memory Management Unit (MMU) with 16-entry...” *Chapter 10: C90LC Flash Memory* In *Section 10.3.3.1.2: Block Erase*, changed bullet list “Write to any address in flash. This is referred to as an erase interlock write.” in “Write to any address in the respective flash array. This is referred to as an erase interlock write” *Chapter 14: Interrupts* In *Table 115: Interrupt summary*, replaced “MCM_MSWTIR_SWTIC” with “SWT_IR_TIF” and “MCM.MSWTIR[SWTIC]” with “SWT.SWT_IR[TIF]” *Chapter 15: System Integration Unit (SIU)* In *Table 117: SIU address map*, set “SIU_TST” register as reserved. Updated *Figure 250: SIU DMA/Interrupt request diagram* In *Section 15.9.3: External interrupt* added Note “If either the SWT or the NMI...” *Chapter 24: Decimation Filter* In *Section 24.1.1: Device-specific features*, added bullet “Cascaded mode (only one instance of decimation filter is supported)” *Chapter 25: Deserial Serial Peripheral Interface (DSPI)* In *Section 25.3.2.1: DSPI Module Configuration Register (DSPI_MCR)*, set bits 29 and 30 to Reserved (were fields “FCPCS” and “PES”, respectively) *Chapter 26: Enhanced Serial Communication Interface (FMPLL)* In *Figure 551: SCI control register 1 (SCI_CR1)*, changed SCI_CR1[16;18:31] from read only to read/write

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2016 STMicroelectronics – All rights reserved