# Stellaris® LM3S5632 Microcontroller

## DATA SHEET

DS-LM3S5632-15852.2743
SPMS087H

Copyright © 2007-2014
Texas Instruments Incorporated

# Copyright

Copyright © 2007-2014 Texas Instruments Incorporated All rights reserved. Stellaris and StellarisWare® are registered trademarks of Texas Instruments Incorporated. ARM and Thumb are registered trademarks and Cortex is a trademark of ARM Limited. Other names and brands may be claimed as the property of others.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

Texas Instruments Incorporated
108 Wild Basin, Suite 350
Austin, TX 78746
http://www.ti.com/stellaris
http://www-k.ext.ti.com/sc/technical-support/product-information-centers.htm

# Table of Contents

**Revision History** ..... **26**

**About This Document** ..... **33**

Audience ..... 33
About This Manual ..... 33
Related Documents ..... 33
Documentation Conventions ..... 34

**1 Architectural Overview** ..... **36**

1.1 Product Features ..... 36
1.2 Target Applications ..... 44
1.3 High-Level Block Diagram ..... 45
1.4 Functional Overview ..... 47
1.4.1 ARM Cortex™-M3 ..... 47
1.4.2 Motor Control Peripherals ..... 48
1.4.3 Analog Peripherals ..... 48
1.4.4 Serial Communications Peripherals ..... 49
1.4.5 System Peripherals ..... 50
1.4.6 Memory Peripherals ..... 51
1.4.7 Additional Features ..... 52
1.4.8 Hardware Details ..... 52

**2 The Cortex-M3 Processor** ..... **53**

2.1 Block Diagram ..... 54
2.2 Overview ..... 55
2.2.1 System-Level Interface ..... 55
2.2.2 Integrated Configurable Debug ..... 55
2.2.3 Trace Port Interface Unit (TPIU) ..... 56
2.2.4 Cortex-M3 System Component Details ..... 56
2.3 Programming Model ..... 57
2.3.1 Processor Mode and Privilege Levels for Software Execution ..... 57
2.3.2 Stacks ..... 57
2.3.3 Register Map ..... 58
2.3.4 Register Descriptions ..... 59
2.3.5 Exceptions and Interrupts ..... 72
2.3.6 Data Types ..... 72
2.4 Memory Model ..... 72
2.4.1 Memory Regions, Types and Attributes ..... 74
2.4.2 Memory System Ordering of Memory Accesses ..... 74
2.4.3 Behavior of Memory Accesses ..... 74
2.4.4 Software Ordering of Memory Accesses ..... 75
2.4.5 Bit-Banding ..... 76
2.4.6 Data Storage ..... 78
2.4.7 Synchronization Primitives ..... 79
2.5 Exception Model ..... 80
2.5.1 Exception States ..... 81
2.5.2 Exception Types ..... 81
2.5.3 Exception Handlers ..... 84

2.5.4 Vector Table ..... 84
2.5.5 Exception Priorities ..... 85
2.5.6 Interrupt Priority Grouping ..... 86
2.5.7 Exception Entry and Return ..... 86
2.6 Fault Handling ..... 88
2.6.1 Fault Types ..... 89
2.6.2 Fault Escalation and Hard Faults ..... 89
2.6.3 Fault Status Registers and Fault Address Registers ..... 90
2.6.4 Lockup ..... 90
2.7 Power Management ..... 90
2.7.1 Entering Sleep Modes ..... 91
2.7.2 Wake Up from Sleep Mode ..... 91
2.8 Instruction Set Summary ..... 92

**3 Cortex-M3 Peripherals ..... 95**
3.1 Functional Description ..... 95
3.1.1 System Timer (SysTick) ..... 95
3.1.2 Nested Vectored Interrupt Controller (NVIC) ..... 96
3.1.3 System Control Block (SCB) ..... 98
3.1.4 Memory Protection Unit (MPU) ..... 98
3.2 Register Map ..... 103
3.3 System Timer (SysTick) Register Descriptions ..... 105
3.4 NVIC Register Descriptions ..... 109
3.5 System Control Block (SCB) Register Descriptions ..... 122
3.6 Memory Protection Unit (MPU) Register Descriptions ..... 149

**4 JTAG Interface ..... 159**
4.1 Block Diagram ..... 160
4.2 Signal Description ..... 160
4.3 Functional Description ..... 161
4.3.1 JTAG Interface Pins ..... 161
4.3.2 JTAG TAP Controller ..... 162
4.3.3 Shift Registers ..... 163
4.3.4 Operational Considerations ..... 163
4.4 Initialization and Configuration ..... 166
4.5 Register Descriptions ..... 166
4.5.1 Instruction Register (IR) ..... 167
4.5.2 Data Registers ..... 169

**5 System Control ..... 171**
5.1 Signal Description ..... 171
5.2 Functional Description ..... 171
5.2.1 Device Identification ..... 171
5.2.2 Reset Control ..... 171
5.2.3 Non-Maskable Interrupt ..... 175
5.2.4 Power Control ..... 176
5.2.5 Clock Control ..... 176
5.2.6 System Control ..... 182
5.3 Initialization and Configuration ..... 183
5.4 Register Map ..... 183
5.5 Register Descriptions ..... 185

**6 Hibernation Module ..... 238**
6.1 Block Diagram ..... 239
6.2 Signal Description ..... 239
6.3 Functional Description ..... 240
6.3.1 Register Access Timing ..... 240
6.3.2 Clock Source ..... 240
6.3.3 Battery Management ..... 242
6.3.4 Real-Time Clock ..... 242
6.3.5 Battery-Backed Memory ..... 242
6.3.6 Power Control ..... 243
6.3.7 Initiating Hibernate ..... 243
6.3.8 Interrupts and Status ..... 243
6.4 Initialization and Configuration ..... 244
6.4.1 Initialization ..... 244
6.4.2 RTC Match Functionality (No Hibernation) ..... 244
6.4.3 RTC Match/Wake-Up from Hibernation ..... 244
6.4.4 External Wake-Up from Hibernation ..... 245
6.4.5 RTC/External Wake-Up from Hibernation ..... 245
6.5 Register Map ..... 245
6.6 Register Descriptions ..... 246
**7 Internal Memory ..... 260**
7.1 Block Diagram ..... 260
7.2 Functional Description ..... 260
7.2.1 SRAM Memory ..... 260
7.2.2 ROM Memory ..... 261
7.2.3 Flash Memory ..... 261
7.3 Flash Memory Initialization and Configuration ..... 264
7.3.1 Flash Programming ..... 264
7.3.2 Nonvolatile Register Programming ..... 264
7.4 Register Map ..... 265
7.5 ROM Register Descriptions (System Control Offset) ..... 266
7.6 Flash Register Descriptions (Flash Control Offset) ..... 267
7.7 Flash Register Descriptions (System Control Offset) ..... 275
**8 Micro Direct Memory Access (µDMA) ..... 290**
8.1 Block Diagram ..... 291
8.2 Functional Description ..... 291
8.2.1 Channel Assigments ..... 292
8.2.2 Priority ..... 292
8.2.3 Arbitration Size ..... 292
8.2.4 Request Types ..... 293
8.2.5 Channel Configuration ..... 293
8.2.6 Transfer Modes ..... 295
8.2.7 Transfer Size and Increment ..... 303
8.2.8 Peripheral Interface ..... 303
8.2.9 Software Request ..... 303
8.2.10 Interrupts and Errors ..... 304
8.3 Initialization and Configuration ..... 304
8.3.1 Module Initialization ..... 304

8.3.2 Configuring a Memory-to-Memory Transfer ..... 304
8.3.3 Configuring a Peripheral for Simple Transmit ..... 306
8.3.4 Configuring a Peripheral for Ping-Pong Receive ..... 307
8.4 Register Map ..... 310
8.5 µDMA Channel Control Structure ..... 311
8.6 µDMA Register Descriptions ..... 317
**9 General-Purpose Input/Outputs (GPIOs) ..... 351**
9.1 Signal Description ..... 351
9.2 Functional Description ..... 354
9.2.1 Data Control ..... 355
9.2.2 Interrupt Control ..... 356
9.2.3 Mode Control ..... 357
9.2.4 Commit Control ..... 357
9.2.5 Pad Control ..... 357
9.2.6 Identification ..... 358
9.3 Initialization and Configuration ..... 358
9.4 Register Map ..... 359
9.5 Register Descriptions ..... 361
**10 General-Purpose Timers ..... 399**
10.1 Block Diagram ..... 400
10.2 Signal Description ..... 400
10.3 Functional Description ..... 401
10.3.1 GPTM Reset Conditions ..... 401
10.3.2 32-Bit Timer Operating Modes ..... 401
10.3.3 16-Bit Timer Operating Modes ..... 403
10.4 Initialization and Configuration ..... 406
10.4.1 32-Bit One-Shot/Periodic Timer Mode ..... 406
10.4.2 32-Bit Real-Time Clock (RTC) Mode ..... 407
10.4.3 16-Bit One-Shot/Periodic Timer Mode ..... 407
10.4.4 16-Bit Input Edge Count Mode ..... 408
10.4.5 16-Bit Input Edge Timing Mode ..... 408
10.4.6 16-Bit PWM Mode ..... 409
10.5 Register Map ..... 409
10.6 Register Descriptions ..... 410
**11 Watchdog Timer ..... 433**
11.1 Block Diagram ..... 434
11.2 Functional Description ..... 434
11.3 Initialization and Configuration ..... 435
11.4 Register Map ..... 435
11.5 Register Descriptions ..... 436
**12 Analog-to-Digital Converter (ADC) ..... 457**
12.1 Block Diagram ..... 457
12.2 Signal Description ..... 458
12.3 Functional Description ..... 458
12.3.1 Sample Sequencers ..... 459
12.3.2 Module Control ..... 459
12.3.3 Hardware Sample Averaging Circuit ..... 460

12.3.4 Analog-to-Digital Converter ..... 460
12.3.5 Differential Sampling ..... 461
12.3.6 Internal Temperature Sensor ..... 463
12.4 Initialization and Configuration ..... 464
12.4.1 Module Initialization ..... 464
12.4.2 Sample Sequencer Configuration ..... 464
12.5 Register Map ..... 465
12.6 Register Descriptions ..... 466
**13 Universal Asynchronous Receivers/Transmitters (UARTs) ..... 492**
13.1 Block Diagram ..... 493
13.2 Signal Description ..... 493
13.3 Functional Description ..... 494
13.3.1 Transmit/Receive Logic ..... 494
13.3.2 Baud-Rate Generation ..... 494
13.3.3 Data Transmission ..... 495
13.3.4 Serial IR (SIR) ..... 495
13.3.5 FIFO Operation ..... 496
13.3.6 Interrupts ..... 497
13.3.7 Loopback Operation ..... 498
13.3.8 DMA Operation ..... 498
13.3.9 IrDA SIR block ..... 498
13.4 Initialization and Configuration ..... 499
13.5 Register Map ..... 499
13.6 Register Descriptions ..... 501
**14 Synchronous Serial Interface (SSI) ..... 536**
14.1 Block Diagram ..... 537
14.2 Signal Description ..... 537
14.3 Functional Description ..... 538
14.3.1 Bit Rate Generation ..... 538
14.3.2 FIFO Operation ..... 538
14.3.3 Interrupts ..... 539
14.3.4 Frame Formats ..... 539
14.3.5 DMA Operation ..... 547
14.4 Initialization and Configuration ..... 547
14.5 Register Map ..... 548
14.6 Register Descriptions ..... 549
**15 Inter-Integrated Circuit ($I^2C$) Interface ..... 576**
15.1 Block Diagram ..... 577
15.2 Signal Description ..... 577
15.3 Functional Description ..... 577
15.3.1 $I^2C$ Bus Functional Overview ..... 578
15.3.2 Available Speed Modes ..... 580
15.3.3 Interrupts ..... 581
15.3.4 Loopback Operation ..... 581
15.3.5 Command Sequence Flow Charts ..... 582
15.4 Initialization and Configuration ..... 589
15.5 Register Map ..... 590

15.6 Register Descriptions ($I^2C$ Master) ..... 591
15.7 Register Descriptions ($I^2C$ Slave) ..... 604

**16 Controller Area Network (CAN) Module ..... 613**
16.1 Block Diagram ..... 614
16.2 Signal Description ..... 614
16.3 Functional Description ..... 615
16.3.1 Initialization ..... 616
16.3.2 Operation ..... 616
16.3.3 Transmitting Message Objects ..... 617
16.3.4 Configuring a Transmit Message Object ..... 617
16.3.5 Updating a Transmit Message Object ..... 619
16.3.6 Accepting Received Message Objects ..... 619
16.3.7 Receiving a Data Frame ..... 619
16.3.8 Receiving a Remote Frame ..... 620
16.3.9 Receive/Transmit Priority ..... 620
16.3.10 Configuring a Receive Message Object ..... 620
16.3.11 Handling of Received Message Objects ..... 621
16.3.12 Handling of Interrupts ..... 624
16.3.13 Test Mode ..... 624
16.3.14 Bit Timing Configuration Error Considerations ..... 626
16.3.15 Bit Time and Bit Rate ..... 626
16.3.16 Calculating the Bit Timing Parameters ..... 628
16.4 Register Map ..... 631
16.5 CAN Register Descriptions ..... 633

**17 Universal Serial Bus (USB) Controller ..... 659**
17.1 Block Diagram ..... 660
17.2 Signal Description ..... 660
17.3 Functional Description ..... 660
17.3.1 Operation as a Device ..... 661
17.3.2 Operation as a Host ..... 666
17.3.3 DMA Operation ..... 670
17.4 Initialization and Configuration ..... 671
17.4.1 Pin Configuration ..... 671
17.4.2 Endpoint Configuration ..... 671
17.5 Register Map ..... 672
17.6 Register Descriptions ..... 675

**18 Pin Diagram ..... 753**

**19 Signal Tables ..... 754**
19.1 Signals by Pin Number ..... 754
19.2 Signals by Signal Name ..... 757
19.3 Signals by Function, Except for GPIO ..... 760
19.4 GPIO Pins and Alternate Functions ..... 762
19.5 Connections for Unused Signals ..... 763

**20 Operating Characteristics ..... 765**

**21 Electrical Characteristics ..... 766**
21.1 DC Characteristics ..... 766
21.1.1 Maximum Ratings ..... 766

21.1.2 Recommended DC Operating Conditions ..... 766
21.1.3 On-Chip Low Drop-Out (LDO) Regulator Characteristics ..... 767
21.1.4 GPIO Module Characteristics ..... 767
21.1.5 Power Specifications ..... 767
21.1.6 Flash Memory Characteristics ..... 769
21.1.7 Hibernation ..... 769
21.1.8 USB ..... 769
21.2 AC Characteristics ..... 770
21.2.1 Load Conditions ..... 770
21.2.2 Clocks ..... 770
21.2.3 JTAG and Boundary Scan ..... 772
21.2.4 Reset ..... 773
21.2.5 Sleep Modes ..... 775
21.2.6 Hibernation Module ..... 775
21.2.7 General-Purpose I/O (GPIO) ..... 775
21.2.8 Analog-to-Digital Converter ..... 776
21.2.9 Synchronous Serial Interface (SSI) ..... 777
21.2.10 Inter-Integrated Circuit ($I^2C$) Interface ..... 779
21.2.11 Universal Serial Bus (USB) Controller ..... 780

**A Boot Loader ..... 781**
A.1 Boot Loader ..... 781
A.2 Interfaces ..... 781
A.2.1 UART ..... 781
A.2.2 SSI ..... 782
A.2.3 $I^2C$ ..... 782
A.3 Packet Handling ..... 782
A.3.1 Packet Format ..... 782
A.3.2 Sending Packets ..... 782
A.3.3 Receiving Packets ..... 783
A.4 Commands ..... 783
A.4.1 COMMAND_PING (0X20) ..... 783
A.4.2 COMMAND_DOWNLOAD (0x21) ..... 783
A.4.3 COMMAND_RUN (0x22) ..... 784
A.4.4 COMMAND_GET_STATUS (0x23) ..... 784
A.4.5 COMMAND_SEND_DATA (0x24) ..... 784
A.4.6 COMMAND_RESET (0x25) ..... 785

**B ROM DriverLib Functions ..... 786**
B.1 DriverLib Functions Included in the Integrated ROM ..... 786

**C Register Quick Reference ..... 797**

**D Ordering and Contact Information ..... 826**
D.1 Ordering Information ..... 826
D.2 Part Markings ..... 826
D.3 Kits ..... 826
D.4 Support Information ..... 827

**E Package Information ..... 828**
E.1 64-Pin LQFP Package ..... 828
E.1.1 Package Dimensions ..... 828

E.1.2 Tray Dimensions ..... 830
E.1.3 Tape and Reel Dimensions ..... 831

# List of Figures


Figure 1-1. Stellaris LM3S5632 Microcontroller High-Level Block Diagram ..... 46
Figure 2-1. CPU Block Diagram ..... 55
Figure 2-2. TPIU Block Diagram ..... 56
Figure 2-3. Cortex-M3 Register Set ..... 58
Figure 2-4. Bit-Band Mapping ..... 78
Figure 2-5. Data Storage ..... 79
Figure 2-6. Vector Table ..... 85
Figure 2-7. Exception Stack Frame ..... 87
Figure 3-1. SRD Use Example ..... 101
Figure 4-1. JTAG Module Block Diagram ..... 160
Figure 4-2. Test Access Port State Machine ..... 163
Figure 4-3. IDCODE Register Format ..... 169
Figure 4-4. BYPASS Register Format ..... 169
Figure 4-5. Boundary Scan Register Format ..... 170
Figure 5-1. Basic $\overline{RST}$ Configuration ..... 173
Figure 5-2. External Circuitry to Extend Power-On Reset ..... 173
Figure 5-3. Reset Circuit Controlled by Switch ..... 174
Figure 5-4. Main Clock Tree ..... 178
Figure 6-1. Hibernation Module Block Diagram ..... 239
Figure 6-2. Clock Source Using Crystal ..... 241
Figure 6-3. Clock Source Using Dedicated Oscillator ..... 241
Figure 7-1. Flash Block Diagram ..... 260
Figure 8-1. µDMA Block Diagram ..... 291
Figure 8-2. Example of Ping-Pong DMA Transaction ..... 296
Figure 8-3. Memory Scatter-Gather, Setup and Configuration ..... 298
Figure 8-4. Memory Scatter-Gather, µDMA Copy Sequence ..... 299
Figure 8-5. Peripheral Scatter-Gather, Setup and Configuration ..... 301
Figure 8-6. Peripheral Scatter-Gather, µDMA Copy Sequence ..... 302
Figure 9-1. Digital I/O Pads ..... 354
Figure 9-2. Analog/Digital I/O Pads ..... 355
Figure 9-3. GPIODATA Write Example ..... 356
Figure 9-4. GPIODATA Read Example ..... 356
Figure 10-1. GPTM Module Block Diagram ..... 400
Figure 10-2. 16-Bit Input Edge Count Mode Example ..... 404
Figure 10-3. 16-Bit Input Edge Time Mode Example ..... 405
Figure 10-4. 16-Bit PWM Mode Example ..... 406
Figure 11-1. WDT Module Block Diagram ..... 434
Figure 12-1. ADC Module Block Diagram ..... 458
Figure 12-2. Differential Sampling Range, $V_{IN\_ODD}$ = 1.5 V ..... 462
Figure 12-3. Differential Sampling Range, $V_{IN\_ODD}$ = 0.75 V ..... 462
Figure 12-4. Differential Sampling Range, $V_{IN\_ODD}$ = 2.25 V ..... 463
Figure 12-5. Internal Temperature Sensor Characteristic ..... 464
Figure 13-1. UART Module Block Diagram ..... 493
Figure 13-2. UART Character Frame ..... 494
Figure 13-3. IrDA Data Modulation ..... 496
Figure 14-1. SSI Module Block Diagram ..... 537

Figure 14-2. TI Synchronous Serial Frame Format (Single Transfer) ..... 540
Figure 14-3. TI Synchronous Serial Frame Format (Continuous Transfer) ..... 541
Figure 14-4. Freescale SPI Format (Single Transfer) with SPO=0 and SPH=0 ..... 541
Figure 14-5. Freescale SPI Format (Continuous Transfer) with SPO=0 and SPH=0 ..... 542
Figure 14-6. Freescale SPI Frame Format with SPO=0 and SPH=1 ..... 543
Figure 14-7. Freescale SPI Frame Format (Single Transfer) with SPO=1 and SPH=0 ..... 543
Figure 14-8. Freescale SPI Frame Format (Continuous Transfer) with SPO=1 and SPH=0 ..... 544
Figure 14-9. Freescale SPI Frame Format with SPO=1 and SPH=1 ..... 545
Figure 14-10. MICROWIRE Frame Format (Single Frame) ..... 545
Figure 14-11. MICROWIRE Frame Format (Continuous Transfer) ..... 546
Figure 14-12. MICROWIRE Frame Format, SSIFss Input Setup and Hold Requirements ..... 547
Figure 15-1. $I^2C$ Block Diagram ..... 577
Figure 15-2. $I^2C$ Bus Configuration ..... 578
Figure 15-3. START and STOP Conditions ..... 578
Figure 15-4. Complete Data Transfer with a 7-Bit Address ..... 579
Figure 15-5. R/S Bit in First Byte ..... 579
Figure 15-6. Data Validity During Bit Transfer on the $I^2C$ Bus ..... 579
Figure 15-7. Master Single SEND ..... 583
Figure 15-8. Master Single RECEIVE ..... 584
Figure 15-9. Master Burst SEND ..... 585
Figure 15-10. Master Burst RECEIVE ..... 586
Figure 15-11. Master Burst RECEIVE after Burst SEND ..... 587
Figure 15-12. Master Burst SEND after Burst RECEIVE ..... 588
Figure 15-13. Slave Command Sequence ..... 589
Figure 16-1. CAN Controller Block Diagram ..... 614
Figure 16-2. CAN Data/Remote Frame ..... 615
Figure 16-3. Message Objects in a FIFO Buffer ..... 623
Figure 16-4. CAN Bit Time ..... 627
Figure 17-1. USB Module Block Diagram ..... 660
Figure 18-1. 64-Pin LQFP Package Pin Diagram ..... 753
Figure 21-1. Load Conditions ..... 770
Figure 21-2. JTAG Test Clock Input Timing ..... 772
Figure 21-3. JTAG Test Access Port (TAP) Timing ..... 773
Figure 21-4. External Reset Timing ($\overline{RST}$) ..... 773
Figure 21-5. Power-On Reset Timing ..... 774
Figure 21-6. Brown-Out Reset Timing ..... 774
Figure 21-7. Software Reset Timing ..... 774
Figure 21-8. Watchdog Reset Timing ..... 774
Figure 21-9. Hibernation Module Timing ..... 775
Figure 21-10. ADC Input Equivalency Diagram ..... 777
Figure 21-11. SSI Timing for TI Frame Format (FRF=01), Single Transfer Timing Measurement ..... 778
Figure 21-12. SSI Timing for MICROWIRE Frame Format (FRF=10), Single Transfer ..... 778
Figure 21-13. SSI Timing for SPI Frame Format (FRF=00), with SPH=1 ..... 779
Figure 21-14. $I^2C$ Timing ..... 780
Figure E-1. Stellaris LM3S5632 64-Pin LQFP Package ..... 828
Figure E-2. 64-Pin LQFP Tray Dimensions ..... 830
Figure E-3. 64-Pin LQFP Tape and Reel Dimensions ..... 831

# List of Tables


Table 1. Revision History ..... 26
Table 2. Documentation Conventions ..... 34
Table 2-1. Summary of Processor Mode, Privilege Level, and Stack Use ..... 58
Table 2-2. Processor Register Map ..... 59
Table 2-3. PSR Register Combinations ..... 64
Table 2-4. Memory Map ..... 72
Table 2-5. Memory Access Behavior ..... 74
Table 2-6. SRAM Memory Bit-Banding Regions ..... 77
Table 2-7. Peripheral Memory Bit-Banding Regions ..... 77
Table 2-8. Exception Types ..... 82
Table 2-9. Interrupts ..... 83
Table 2-10. Exception Return Behavior ..... 88
Table 2-11. Faults ..... 89
Table 2-12. Fault Status and Fault Address Registers ..... 90
Table 2-13. Cortex-M3 Instruction Summary ..... 92
Table 3-1. Core Peripheral Register Regions ..... 95
Table 3-2. Memory Attributes Summary ..... 98
Table 3-3. TEX, S, C, and B Bit Field Encoding ..... 101
Table 3-4. Cache Policy for Memory Attribute Encoding ..... 102
Table 3-5. AP Bit Field Encoding ..... 102
Table 3-6. Memory Region Attributes for Stellaris Microcontrollers ..... 102
Table 3-7. Peripherals Register Map ..... 103
Table 3-8. Interrupt Priority Levels ..... 128
Table 3-9. Example SIZE Field Values ..... 156
Table 4-1. JTAG_SWD_SWO Signals (64LQFP) ..... 160
Table 4-2. JTAG Port Pins Reset State ..... 161
Table 4-3. JTAG Instruction Register Commands ..... 167
Table 5-1. System Control & Clocks Signals (64LQFP) ..... 171
Table 5-2. Reset Sources ..... 172
Table 5-3. Clock Source Options ..... 177
Table 5-4. Possible System Clock Frequencies Using the SYSDIV Field ..... 179
Table 5-5. Examples of Possible System Clock Frequencies Using the SYSDIV2 Field ..... 179
Table 5-6. System Control Register Map ..... 183
Table 5-7. RCC2 Fields that Override RCC fields ..... 201
Table 6-1. Hibernate Signals (64LQFP) ..... 239
Table 6-2. Hibernation Module Register Map ..... 245
Table 7-1. Flash Protection Policy Combinations ..... 262
Table 7-2. User-Programmable Flash Memory Resident Registers ..... 265
Table 7-3. Flash Register Map ..... 265
Table 8-1. DMA Channel Assignments ..... 292
Table 8-2. Request Type Support ..... 293
Table 8-3. Control Structure Memory Map ..... 294
Table 8-4. Channel Control Structure ..... 294
Table 8-5. µDMA Read Example: 8-Bit Peripheral ..... 303
Table 8-6. µDMA Interrupt Assignments ..... 304
Table 8-7. Channel Control Structure Offsets for Channel 30 ..... 305

Table 8-8. Channel Control Word Configuration for Memory Transfer Example ..... 305
Table 8-9. Channel Control Structure Offsets for Channel 7 ..... 306
Table 8-10. Channel Control Word Configuration for Peripheral Transmit Example ..... 307
Table 8-11. Primary and Alternate Channel Control Structure Offsets for Channel 8 ..... 308
Table 8-12. Channel Control Word Configuration for Peripheral Ping-Pong Receive Example ..... 309
Table 8-13. µDMA Register Map ..... 310
Table 9-1. GPIO Pins With Non-Zero Reset Values ..... 352
Table 9-2. GPIO Pins and Alternate Functions (64LQFP) ..... 352
Table 9-3. GPIO Signals (64LQFP) ..... 353
Table 9-4. GPIO Pad Configuration Examples ..... 358
Table 9-5. GPIO Interrupt Configuration Example ..... 359
Table 9-6. GPIO Register Map ..... 360
Table 10-1. Available CCP Pins ..... 400
Table 10-2. General-Purpose Timers Signals (64LQFP) ..... 401
Table 10-3. 16-Bit Timer With Prescaler Configurations ..... 403
Table 10-4. Timers Register Map ..... 410
Table 11-1. Watchdog Timer Register Map ..... 435
Table 12-1. ADC Signals (64LQFP) ..... 458
Table 12-2. Samples and FIFO Depth of Sequencers ..... 459
Table 12-3. Differential Sampling Pairs ..... 461
Table 12-4. ADC Register Map ..... 465
Table 13-1. UART Signals (64LQFP) ..... 493
Table 13-2. UART Register Map ..... 500
Table 14-1. SSI Signals (64LQFP) ..... 538
Table 14-2. SSI Register Map ..... 549
Table 15-1. I2C Signals (64LQFP) ..... 577
Table 15-2. Examples of $I^2C$ Master Timer Period versus Speed Mode ..... 580
Table 15-3. Inter-Integrated Circuit ($I^2C$) Interface Register Map ..... 590
Table 15-4. Write Field Decoding for I2CMCS[3:0] Field (Sheet 1 of 3) ..... 595
Table 16-1. Controller Area Network Signals (64LQFP) ..... 614
Table 16-2. CAN Protocol Ranges ..... 627
Table 16-3. CANBIT Register Values ..... 627
Table 16-4. CAN Register Map ..... 631
Table 17-1. USB Signals (64LQFP) ..... 660
Table 17-2. Remainder (MAXLOAD/4) ..... 670
Table 17-3. Actual Bytes Read ..... 670
Table 17-4. Packet Sizes That Clear RXRDY ..... 670
Table 17-5. Universal Serial Bus (USB) Controller Register Map ..... 672
Table 19-1. Signals by Pin Number ..... 754
Table 19-2. Signals by Signal Name ..... 757
Table 19-3. Signals by Function, Except for GPIO ..... 760
Table 19-4. GPIO Pins and Alternate Functions ..... 762
Table 19-5. Connections for Unused Signals (64-pin LQFP) ..... 763
Table 20-1. Temperature Characteristics ..... 765
Table 20-2. Thermal Characteristics ..... 765
Table 20-3. ESD Absolute Maximum Ratings ..... 765
Table 21-1. Maximum Ratings ..... 766

Table 21-2. Recommended DC Operating Conditions ..... 766
Table 21-3. LDO Regulator Characteristics ..... 767
Table 21-4. GPIO Module DC Characteristics ..... 767
Table 21-5. Detailed Power Specifications ..... 768
Table 21-6. Flash Memory Characteristics ..... 769
Table 21-7. Hibernation Module DC Characteristics ..... 769
Table 21-8. USB Controller DC Characteristics ..... 769
Table 21-9. Phase Locked Loop (PLL) Characteristics ..... 770
Table 21-10. Actual PLL Frequency ..... 770
Table 21-11. Clock Characteristics ..... 771
Table 21-12. Crystal Characteristics ..... 771
Table 21-13. System Clock Characteristics with ADC Operation ..... 771
Table 21-14. System Clock Characteristics with USB Operation ..... 772
Table 21-15. JTAG Characteristics ..... 772
Table 21-16. Reset Characteristics ..... 773
Table 21-17. Sleep Modes AC Characteristics ..... 775
Table 21-18. Hibernation Module AC Characteristics ..... 775
Table 21-19. GPIO Characteristics ..... 776
Table 21-20. ADC Characteristics ..... 776
Table 21-21. ADC Module Internal Reference Characteristics ..... 777
Table 21-22. SSI Characteristics ..... 777
Table 21-23. $I^2C$ Characteristics ..... 779

# List of Registers

**The Cortex-M3 Processor ..... 53**
Register 1: Cortex General-Purpose Register 0 (R0) ..... 60
Register 2: Cortex General-Purpose Register 1 (R1) ..... 60
Register 3: Cortex General-Purpose Register 2 (R2) ..... 60
Register 4: Cortex General-Purpose Register 3 (R3) ..... 60
Register 5: Cortex General-Purpose Register 4 (R4) ..... 60
Register 6: Cortex General-Purpose Register 5 (R5) ..... 60
Register 7: Cortex General-Purpose Register 6 (R6) ..... 60
Register 8: Cortex General-Purpose Register 7 (R7) ..... 60
Register 9: Cortex General-Purpose Register 8 (R8) ..... 60
Register 10: Cortex General-Purpose Register 9 (R9) ..... 60
Register 11: Cortex General-Purpose Register 10 (R10) ..... 60
Register 12: Cortex General-Purpose Register 11 (R11) ..... 60
Register 13: Cortex General-Purpose Register 12 (R12) ..... 60
Register 14: Stack Pointer (SP) ..... 61
Register 15: Link Register (LR) ..... 62
Register 16: Program Counter (PC) ..... 63
Register 17: Program Status Register (PSR) ..... 64
Register 18: Priority Mask Register (PRIMASK) ..... 68
Register 19: Fault Mask Register (FAULTMASK) ..... 69
Register 20: Base Priority Mask Register (BASEPRI) ..... 70
Register 21: Control Register (CONTROL) ..... 71
**Cortex-M3 Peripherals ..... 95**
Register 1: SysTick Control and Status Register (STCTRL), offset 0x010 ..... 106
Register 2: SysTick Reload Value Register (STRELOAD), offset 0x014 ..... 108
Register 3: SysTick Current Value Register (STCURRENT), offset 0x018 ..... 109
Register 4: Interrupt 0-31 Set Enable (EN0), offset 0x100 ..... 110
Register 5: Interrupt 32-47 Set Enable (EN1), offset 0x104 ..... 111
Register 6: Interrupt 0-31 Clear Enable (DIS0), offset 0x180 ..... 112
Register 7: Interrupt 32-47 Clear Enable (DIS1), offset 0x184 ..... 113
Register 8: Interrupt 0-31 Set Pending (PEND0), offset 0x200 ..... 114
Register 9: Interrupt 32-47 Set Pending (PEND1), offset 0x204 ..... 115
Register 10: Interrupt 0-31 Clear Pending (UNPEND0), offset 0x280 ..... 116
Register 11: Interrupt 32-47 Clear Pending (UNPEND1), offset 0x284 ..... 117
Register 12: Interrupt 0-31 Active Bit (ACTIVE0), offset 0x300 ..... 118
Register 13: Interrupt 32-47 Active Bit (ACTIVE1), offset 0x304 ..... 119
Register 14: Interrupt 0-3 Priority (PRI0), offset 0x400 ..... 120
Register 15: Interrupt 4-7 Priority (PRI1), offset 0x404 ..... 120
Register 16: Interrupt 8-11 Priority (PRI2), offset 0x408 ..... 120
Register 17: Interrupt 12-15 Priority (PRI3), offset 0x40C ..... 120
Register 18: Interrupt 16-19 Priority (PRI4), offset 0x410 ..... 120
Register 19: Interrupt 20-23 Priority (PRI5), offset 0x414 ..... 120
Register 20: Interrupt 24-27 Priority (PRI6), offset 0x418 ..... 120
Register 21: Interrupt 28-31 Priority (PRI7), offset 0x41C ..... 120
Register 22: Interrupt 32-35 Priority (PRI8), offset 0x420 ..... 120

Register 23: Interrupt 36-39 Priority (PRI9), offset 0x424 ..... 120
Register 24: Interrupt 40-43 Priority (PRI10), offset 0x428 ..... 120
Register 25: Interrupt 44-47 Priority (PRI11), offset 0x42C ..... 120
Register 26: Software Trigger Interrupt (SWTRIG), offset 0xF00 ..... 122
Register 27: CPU ID Base (CPUID), offset 0xD00 ..... 123
Register 28: Interrupt Control and State (INTCTRL), offset 0xD04 ..... 124
Register 29: Vector Table Offset (VTABLE), offset 0xD08 ..... 127
Register 30: Application Interrupt and Reset Control (APINT), offset 0xD0C ..... 128
Register 31: System Control (SYSCTRL), offset 0xD10 ..... 130
Register 32: Configuration and Control (CFGCTRL), offset 0xD14 ..... 132
Register 33: System Handler Priority 1 (SYSPRI1), offset 0xD18 ..... 134
Register 34: System Handler Priority 2 (SYSPRI2), offset 0xD1C ..... 135
Register 35: System Handler Priority 3 (SYSPRI3), offset 0xD20 ..... 136
Register 36: System Handler Control and State (SYSHNDCTRL), offset 0xD24 ..... 137
Register 37: Configurable Fault Status (FAULTSTAT), offset 0xD28 ..... 141
Register 38: Hard Fault Status (HFAULTSTAT), offset 0xD2C ..... 147
Register 39: Memory Management Fault Address (MMADDR), offset 0xD34 ..... 148
Register 40: Bus Fault Address (FAULTADDR), offset 0xD38 ..... 149
Register 41: MPU Type (MPUTYPE), offset 0xD90 ..... 150
Register 42: MPU Control (MPUCTRL), offset 0xD94 ..... 151
Register 43: MPU Region Number (MPUNUMBER), offset 0xD98 ..... 153
Register 44: MPU Region Base Address (MPUBASE), offset 0xD9C ..... 154
Register 45: MPU Region Base Address Alias 1 (MPUBASE1), offset 0xDA4 ..... 154
Register 46: MPU Region Base Address Alias 2 (MPUBASE2), offset 0xDAC ..... 154
Register 47: MPU Region Base Address Alias 3 (MPUBASE3), offset 0xDB4 ..... 154
Register 48: MPU Region Attribute and Size (MPUATTR), offset 0xDA0 ..... 156
Register 49: MPU Region Attribute and Size Alias 1 (MPUATTR1), offset 0xDA8 ..... 156
Register 50: MPU Region Attribute and Size Alias 2 (MPUATTR2), offset 0xDB0 ..... 156
Register 51: MPU Region Attribute and Size Alias 3 (MPUATTR3), offset 0xDB8 ..... 156

**System Control ..... 171**

Register 1: Device Identification 0 (DID0), offset 0x000 ..... 186
Register 2: Brown-Out Reset Control (PBORCTL), offset 0x030 ..... 188
Register 3: LDO Power Control (LDOPCTL), offset 0x034 ..... 189
Register 4: Raw Interrupt Status (RIS), offset 0x050 ..... 190
Register 5: Interrupt Mask Control (IMC), offset 0x054 ..... 191
Register 6: Masked Interrupt Status and Clear (MISC), offset 0x058 ..... 192
Register 7: Reset Cause (RESC), offset 0x05C ..... 193
Register 8: Run-Mode Clock Configuration (RCC), offset 0x060 ..... 194
Register 9: XTAL to PLL Translation (PLLCFG), offset 0x064 ..... 198
Register 10: GPIO High-Performance Bus Control (GPIOHBCTL), offset 0x06C ..... 199
Register 11: Run-Mode Clock Configuration 2 (RCC2), offset 0x070 ..... 201
Register 12: Main Oscillator Control (MOSCCTL), offset 0x07C ..... 203
Register 13: Deep Sleep Clock Configuration (DSLPCLKCFG), offset 0x144 ..... 204
Register 14: Device Identification 1 (DID1), offset 0x004 ..... 205
Register 15: Device Capabilities 0 (DC0), offset 0x008 ..... 207
Register 16: Device Capabilities 1 (DC1), offset 0x010 ..... 208
Register 17: Device Capabilities 2 (DC2), offset 0x014 ..... 210
Register 18: Device Capabilities 3 (DC3), offset 0x018 ..... 211

Register 19: Device Capabilities 4 (DC4), offset 0x01C ..... 212
Register 20: Device Capabilities 5 (DC5), offset 0x020 ..... 213
Register 21: Device Capabilities 6 (DC6), offset 0x024 ..... 214
Register 22: Device Capabilities 7 (DC7), offset 0x028 ..... 215
Register 23: Run Mode Clock Gating Control Register 0 (RCGC0), offset 0x100 ..... 217
Register 24: Sleep Mode Clock Gating Control Register 0 (SCGC0), offset 0x110 ..... 219
Register 25: Deep Sleep Mode Clock Gating Control Register 0 (DCGC0), offset 0x120 ..... 221
Register 26: Run Mode Clock Gating Control Register 1 (RCGC1), offset 0x104 ..... 223
Register 27: Sleep Mode Clock Gating Control Register 1 (SCGC1), offset 0x114 ..... 225
Register 28: Deep Sleep Mode Clock Gating Control Register 1 (DCGC1), offset 0x124 ..... 227
Register 29: Run Mode Clock Gating Control Register 2 (RCGC2), offset 0x108 ..... 229
Register 30: Sleep Mode Clock Gating Control Register 2 (SCGC2), offset 0x118 ..... 231
Register 31: Deep Sleep Mode Clock Gating Control Register 2 (DCGC2), offset 0x128 ..... 233
Register 32: Software Reset Control 0 (SRCR0), offset 0x040 ..... 235
Register 33: Software Reset Control 1 (SRCR1), offset 0x044 ..... 236
Register 34: Software Reset Control 2 (SRCR2), offset 0x048 ..... 237

**Hibernation Module ..... 238**

Register 1: Hibernation RTC Counter (HIBRTCC), offset 0x000 ..... 247
Register 2: Hibernation RTC Match 0 (HIBRTCM0), offset 0x004 ..... 248
Register 3: Hibernation RTC Match 1 (HIBRTCM1), offset 0x008 ..... 249
Register 4: Hibernation RTC Load (HIBRTCLD), offset 0x00C ..... 250
Register 5: Hibernation Control (HIBCTL), offset 0x010 ..... 251
Register 6: Hibernation Interrupt Mask (HIBIM), offset 0x014 ..... 254
Register 7: Hibernation Raw Interrupt Status (HIBRIS), offset 0x018 ..... 255
Register 8: Hibernation Masked Interrupt Status (HIBMIS), offset 0x01C ..... 256
Register 9: Hibernation Interrupt Clear (HIBIC), offset 0x020 ..... 257
Register 10: Hibernation RTC Trim (HIBRTCT), offset 0x024 ..... 258
Register 11: Hibernation Data (HIBDATA), offset 0x030-0x12C ..... 259

**Internal Memory ..... 260**

Register 1: ROM Control (RMCTL), offset 0x0F0 ..... 267
Register 2: Flash Memory Address (FMA), offset 0x000 ..... 268
Register 3: Flash Memory Data (FMD), offset 0x004 ..... 269
Register 4: Flash Memory Control (FMC), offset 0x008 ..... 270
Register 5: Flash Controller Raw Interrupt Status (FCRIS), offset 0x00C ..... 272
Register 6: Flash Controller Interrupt Mask (FCIM), offset 0x010 ..... 273
Register 7: Flash Controller Masked Interrupt Status and Clear (FCMISC), offset 0x014 ..... 274
Register 8: USec Reload (USECRL), offset 0x140 ..... 276
Register 9: Flash Memory Protection Read Enable 0 (FMPRE0), offset 0x130 and 0x200 ..... 277
Register 10: Flash Memory Protection Program Enable 0 (FMPPE0), offset 0x134 and 0x400 ..... 278
Register 11: User Debug (USER_DBG), offset 0x1D0 ..... 279
Register 12: User Register 0 (USER_REG0), offset 0x1E0 ..... 280
Register 13: User Register 1 (USER_REG1), offset 0x1E4 ..... 281
Register 14: User Register 2 (USER_REG2), offset 0x1E8 ..... 282
Register 15: User Register 3 (USER_REG3), offset 0x1EC ..... 283
Register 16: Flash Memory Protection Read Enable 1 (FMPRE1), offset 0x204 ..... 284
Register 17: Flash Memory Protection Read Enable 2 (FMPRE2), offset 0x208 ..... 285
Register 18: Flash Memory Protection Read Enable 3 (FMPRE3), offset 0x20C ..... 286
Register 19: Flash Memory Protection Program Enable 1 (FMPPE1), offset 0x404 ..... 287

Register 20: Flash Memory Protection Program Enable 2 (FMPPE2), offset 0x408 ..... 288
Register 21: Flash Memory Protection Program Enable 3 (FMPPE3), offset 0x40C ..... 289

**Micro Direct Memory Access (µDMA) ..... 290**

Register 1: DMA Channel Source Address End Pointer (DMASRCENDP), offset 0x000 ..... 312
Register 2: DMA Channel Destination Address End Pointer (DMADSTENDP), offset 0x004 ..... 313
Register 3: DMA Channel Control Word (DMACHCTL), offset 0x008 ..... 314
Register 4: DMA Status (DMASTAT), offset 0x000 ..... 318
Register 5: DMA Configuration (DMACFG), offset 0x004 ..... 320
Register 6: DMA Channel Control Base Pointer (DMACTLBASE), offset 0x008 ..... 321
Register 7: DMA Alternate Channel Control Base Pointer (DMAALTBASE), offset 0x00C ..... 322
Register 8: DMA Channel Wait on Request Status (DMAWAITSTAT), offset 0x010 ..... 323
Register 9: DMA Channel Software Request (DMASWREQ), offset 0x014 ..... 324
Register 10: DMA Channel Useburst Set (DMAUSEBURSTSET), offset 0x018 ..... 325
Register 11: DMA Channel Useburst Clear (DMAUSEBURSTCLR), offset 0x01C ..... 327
Register 12: DMA Channel Request Mask Set (DMAREQMASKSET), offset 0x020 ..... 328
Register 13: DMA Channel Request Mask Clear (DMAREQMASKCLR), offset 0x024 ..... 330
Register 14: DMA Channel Enable Set (DMAENASET), offset 0x028 ..... 331
Register 15: DMA Channel Enable Clear (DMAENACLR), offset 0x02C ..... 333
Register 16: DMA Channel Primary Alternate Set (DMAALTSET), offset 0x030 ..... 334
Register 17: DMA Channel Primary Alternate Clear (DMAALTCLR), offset 0x034 ..... 336
Register 18: DMA Channel Priority Set (DMAPRIOSET), offset 0x038 ..... 337
Register 19: DMA Channel Priority Clear (DMAPRIOCLR), offset 0x03C ..... 339
Register 20: DMA Bus Error Clear (DMAERRCLR), offset 0x04C ..... 340
Register 21: DMA Peripheral Identification 0 (DMAPeriphID0), offset 0xFE0 ..... 342
Register 22: DMA Peripheral Identification 1 (DMAPeriphID1), offset 0xFE4 ..... 343
Register 23: DMA Peripheral Identification 2 (DMAPeriphID2), offset 0xFE8 ..... 344
Register 24: DMA Peripheral Identification 3 (DMAPeriphID3), offset 0xFEC ..... 345
Register 25: DMA Peripheral Identification 4 (DMAPeriphID4), offset 0xFD0 ..... 346
Register 26: DMA PrimeCell Identification 0 (DMAPCellID0), offset 0xFF0 ..... 347
Register 27: DMA PrimeCell Identification 1 (DMAPCellID1), offset 0xFF4 ..... 348
Register 28: DMA PrimeCell Identification 2 (DMAPCellID2), offset 0xFF8 ..... 349
Register 29: DMA PrimeCell Identification 3 (DMAPCellID3), offset 0xFFC ..... 350

**General-Purpose Input/Outputs (GPIOs) ..... 351**

Register 1: GPIO Data (GPIODATA), offset 0x000 ..... 362
Register 2: GPIO Direction (GPIODIR), offset 0x400 ..... 363
Register 3: GPIO Interrupt Sense (GPIOIS), offset 0x404 ..... 364
Register 4: GPIO Interrupt Both Edges (GPIOIBE), offset 0x408 ..... 365
Register 5: GPIO Interrupt Event (GPIOIEV), offset 0x40C ..... 366
Register 6: GPIO Interrupt Mask (GPIOIM), offset 0x410 ..... 367
Register 7: GPIO Raw Interrupt Status (GPIORIS), offset 0x414 ..... 368
Register 8: GPIO Masked Interrupt Status (GPIOMIS), offset 0x418 ..... 369
Register 9: GPIO Interrupt Clear (GPIOICR), offset 0x41C ..... 370
Register 10: GPIO Alternate Function Select (GPIOAFSEL), offset 0x420 ..... 371
Register 11: GPIO 2-mA Drive Select (GPIODR2R), offset 0x500 ..... 373
Register 12: GPIO 4-mA Drive Select (GPIODR4R), offset 0x504 ..... 374
Register 13: GPIO 8-mA Drive Select (GPIODR8R), offset 0x508 ..... 375
Register 14: GPIO Open Drain Select (GPIOODR), offset 0x50C ..... 376
Register 15: GPIO Pull-Up Select (GPIOPUR), offset 0x510 ..... 377

Register 16: GPIO Pull-Down Select (GPIOPDR), offset 0x514 ..... 379
Register 17: GPIO Slew Rate Control Select (GPIOSLR), offset 0x518 ..... 380
Register 18: GPIO Digital Enable (GPIODEN), offset 0x51C ..... 381
Register 19: GPIO Lock (GPIOLOCK), offset 0x520 ..... 383
Register 20: GPIO Commit (GPIOCR), offset 0x524 ..... 384
Register 21: GPIO Analog Mode Select (GPIOAMSEL), offset 0x528 ..... 386
Register 22: GPIO Peripheral Identification 4 (GPIOPeriphID4), offset 0xFD0 ..... 387
Register 23: GPIO Peripheral Identification 5 (GPIOPeriphID5), offset 0xFD4 ..... 388
Register 24: GPIO Peripheral Identification 6 (GPIOPeriphID6), offset 0xFD8 ..... 389
Register 25: GPIO Peripheral Identification 7 (GPIOPeriphID7), offset 0xFDC ..... 390
Register 26: GPIO Peripheral Identification 0 (GPIOPeriphID0), offset 0xFE0 ..... 391
Register 27: GPIO Peripheral Identification 1 (GPIOPeriphID1), offset 0xFE4 ..... 392
Register 28: GPIO Peripheral Identification 2 (GPIOPeriphID2), offset 0xFE8 ..... 393
Register 29: GPIO Peripheral Identification 3 (GPIOPeriphID3), offset 0xFEC ..... 394
Register 30: GPIO PrimeCell Identification 0 (GPIOPCellID0), offset 0xFF0 ..... 395
Register 31: GPIO PrimeCell Identification 1 (GPIOPCellID1), offset 0xFF4 ..... 396
Register 32: GPIO PrimeCell Identification 2 (GPIOPCellID2), offset 0xFF8 ..... 397
Register 33: GPIO PrimeCell Identification 3 (GPIOPCellID3), offset 0xFFC ..... 398

**General-Purpose Timers ..... 399**
Register 1: GPTM Configuration (GPTMCFG), offset 0x000 ..... 411
Register 2: GPTM TimerA Mode (GPTMTAMR), offset 0x004 ..... 412
Register 3: GPTM TimerB Mode (GPTMTBMR), offset 0x008 ..... 414
Register 4: GPTM Control (GPTMCTL), offset 0x00C ..... 416
Register 5: GPTM Interrupt Mask (GPTMIMR), offset 0x018 ..... 419
Register 6: GPTM Raw Interrupt Status (GPTMRIS), offset 0x01C ..... 421
Register 7: GPTM Masked Interrupt Status (GPTMMIS), offset 0x020 ..... 422
Register 8: GPTM Interrupt Clear (GPTMICR), offset 0x024 ..... 423
Register 9: GPTM TimerA Interval Load (GPTMTAILR), offset 0x028 ..... 425
Register 10: GPTM TimerB Interval Load (GPTMTBILR), offset 0x02C ..... 426
Register 11: GPTM TimerA Match (GPTMTAMATCHR), offset 0x030 ..... 427
Register 12: GPTM TimerB Match (GPTMTBMATCHR), offset 0x034 ..... 428
Register 13: GPTM TimerA Prescale (GPTMTAPR), offset 0x038 ..... 429
Register 14: GPTM TimerB Prescale (GPTMTBPR), offset 0x03C ..... 430
Register 15: GPTM TimerA (GPTMTAR), offset 0x048 ..... 431
Register 16: GPTM TimerB (GPTMTBR), offset 0x04C ..... 432

**Watchdog Timer ..... 433**
Register 1: Watchdog Load (WDTLOAD), offset 0x000 ..... 437
Register 2: Watchdog Value (WDTVALUE), offset 0x004 ..... 438
Register 3: Watchdog Control (WDTCTL), offset 0x008 ..... 439
Register 4: Watchdog Interrupt Clear (WDTICR), offset 0x00C ..... 440
Register 5: Watchdog Raw Interrupt Status (WDTRIS), offset 0x010 ..... 441
Register 6: Watchdog Masked Interrupt Status (WDTMIS), offset 0x014 ..... 442
Register 7: Watchdog Test (WDTTEST), offset 0x418 ..... 443
Register 8: Watchdog Lock (WDTLOCK), offset 0xC00 ..... 444
Register 9: Watchdog Peripheral Identification 4 (WDTPeriphID4), offset 0xFD0 ..... 445
Register 10: Watchdog Peripheral Identification 5 (WDTPeriphID5), offset 0xFD4 ..... 446
Register 11: Watchdog Peripheral Identification 6 (WDTPeriphID6), offset 0xFD8 ..... 447
Register 12: Watchdog Peripheral Identification 7 (WDTPeriphID7), offset 0xFDC ..... 448

Register 13: Watchdog Peripheral Identification 0 (WDTPeriphID0), offset 0xFE0 ..... 449
Register 14: Watchdog Peripheral Identification 1 (WDTPeriphID1), offset 0xFE4 ..... 450
Register 15: Watchdog Peripheral Identification 2 (WDTPeriphID2), offset 0xFE8 ..... 451
Register 16: Watchdog Peripheral Identification 3 (WDTPeriphID3), offset 0xFEC ..... 452
Register 17: Watchdog PrimeCell Identification 0 (WDTPCellID0), offset 0xFF0 ..... 453
Register 18: Watchdog PrimeCell Identification 1 (WDTPCellID1), offset 0xFF4 ..... 454
Register 19: Watchdog PrimeCell Identification 2 (WDTPCellID2), offset 0xFF8 ..... 455
Register 20: Watchdog PrimeCell Identification 3 (WDTPCellID3 ), offset 0xFFC ..... 456

**Analog-to-Digital Converter (ADC) ..... 457**

Register 1: ADC Active Sample Sequencer (ADCACTSS), offset 0x000 ..... 467
Register 2: ADC Raw Interrupt Status (ADCRIS), offset 0x004 ..... 468
Register 3: ADC Interrupt Mask (ADCIM), offset 0x008 ..... 469
Register 4: ADC Interrupt Status and Clear (ADCISC), offset 0x00C ..... 470
Register 5: ADC Overflow Status (ADCOSTAT), offset 0x010 ..... 471
Register 6: ADC Event Multiplexer Select (ADCEMUX), offset 0x014 ..... 472
Register 7: ADC Underflow Status (ADCUSTAT), offset 0x018 ..... 475
Register 8: ADC Sample Sequencer Priority (ADCSSPRI), offset 0x020 ..... 476
Register 9: ADC Processor Sample Sequence Initiate (ADCPSSI), offset 0x028 ..... 478
Register 10: ADC Sample Averaging Control (ADCSAC), offset 0x030 ..... 479
Register 11: ADC Sample Sequence Input Multiplexer Select 0 (ADCSSMUX0), offset 0x040 ..... 480
Register 12: ADC Sample Sequence Control 0 (ADCSSCTL0), offset 0x044 ..... 482
Register 13: ADC Sample Sequence Result FIFO 0 (ADCSSFIFO0), offset 0x048 ..... 485
Register 14: ADC Sample Sequence Result FIFO 1 (ADCSSFIFO1), offset 0x068 ..... 485
Register 15: ADC Sample Sequence Result FIFO 2 (ADCSSFIFO2), offset 0x088 ..... 485
Register 16: ADC Sample Sequence Result FIFO 3 (ADCSSFIFO3), offset 0x0A8 ..... 485
Register 17: ADC Sample Sequence FIFO 0 Status (ADCSSFSTAT0), offset 0x04C ..... 486
Register 18: ADC Sample Sequence FIFO 1 Status (ADCSSFSTAT1), offset 0x06C ..... 486
Register 19: ADC Sample Sequence FIFO 2 Status (ADCSSFSTAT2), offset 0x08C ..... 486
Register 20: ADC Sample Sequence FIFO 3 Status (ADCSSFSTAT3), offset 0x0AC ..... 486
Register 21: ADC Sample Sequence Input Multiplexer Select 1 (ADCSSMUX1), offset 0x060 ..... 487
Register 22: ADC Sample Sequence Input Multiplexer Select 2 (ADCSSMUX2), offset 0x080 ..... 487
Register 23: ADC Sample Sequence Control 1 (ADCSSCTL1), offset 0x064 ..... 488
Register 24: ADC Sample Sequence Control 2 (ADCSSCTL2), offset 0x084 ..... 488
Register 25: ADC Sample Sequence Input Multiplexer Select 3 (ADCSSMUX3), offset 0x0A0 ..... 490
Register 26: ADC Sample Sequence Control 3 (ADCSSCTL3), offset 0x0A4 ..... 491

**Universal Asynchronous Receivers/Transmitters (UARTs) ..... 492**

Register 1: UART Data (UARTDR), offset 0x000 ..... 502
Register 2: UART Receive Status/Error Clear (UARTRSR/UARTECR), offset 0x004 ..... 504
Register 3: UART Flag (UARTFR), offset 0x018 ..... 506
Register 4: UART IrDA Low-Power Register (UARTILPR), offset 0x020 ..... 508
Register 5: UART Integer Baud-Rate Divisor (UARTIBRD), offset 0x024 ..... 509
Register 6: UART Fractional Baud-Rate Divisor (UARTFBRD), offset 0x028 ..... 510
Register 7: UART Line Control (UARTLCRH), offset 0x02C ..... 511
Register 8: UART Control (UARTCTL), offset 0x030 ..... 513
Register 9: UART Interrupt FIFO Level Select (UARTIFLS), offset 0x034 ..... 515
Register 10: UART Interrupt Mask (UARTIM), offset 0x038 ..... 517
Register 11: UART Raw Interrupt Status (UARTRIS), offset 0x03C ..... 519
Register 12: UART Masked Interrupt Status (UARTMIS), offset 0x040 ..... 520

Register 13: UART Interrupt Clear (UARTICR), offset 0x044 ..... 521
Register 14: UART DMA Control (UARTDMACTL), offset 0x048 ..... 523
Register 15: UART Peripheral Identification 4 (UARTPeriphID4), offset 0xFD0 ..... 524
Register 16: UART Peripheral Identification 5 (UARTPeriphID5), offset 0xFD4 ..... 525
Register 17: UART Peripheral Identification 6 (UARTPeriphID6), offset 0xFD8 ..... 526
Register 18: UART Peripheral Identification 7 (UARTPeriphID7), offset 0xFDC ..... 527
Register 19: UART Peripheral Identification 0 (UARTPeriphID0), offset 0xFE0 ..... 528
Register 20: UART Peripheral Identification 1 (UARTPeriphID1), offset 0xFE4 ..... 529
Register 21: UART Peripheral Identification 2 (UARTPeriphID2), offset 0xFE8 ..... 530
Register 22: UART Peripheral Identification 3 (UARTPeriphID3), offset 0xFEC ..... 531
Register 23: UART PrimeCell Identification 0 (UARTPCellID0), offset 0xFF0 ..... 532
Register 24: UART PrimeCell Identification 1 (UARTPCellID1), offset 0xFF4 ..... 533
Register 25: UART PrimeCell Identification 2 (UARTPCellID2), offset 0xFF8 ..... 534
Register 26: UART PrimeCell Identification 3 (UARTPCellID3), offset 0xFFC ..... 535

**Synchronous Serial Interface (SSI) ..... 536**

Register 1: SSI Control 0 (SSICR0), offset 0x000 ..... 550
Register 2: SSI Control 1 (SSICR1), offset 0x004 ..... 552
Register 3: SSI Data (SSIDR), offset 0x008 ..... 554
Register 4: SSI Status (SSISR), offset 0x00C ..... 555
Register 5: SSI Clock Prescale (SSICPSR), offset 0x010 ..... 557
Register 6: SSI Interrupt Mask (SSIIM), offset 0x014 ..... 558
Register 7: SSI Raw Interrupt Status (SSIRIS), offset 0x018 ..... 560
Register 8: SSI Masked Interrupt Status (SSIMIS), offset 0x01C ..... 561
Register 9: SSI Interrupt Clear (SSIICR), offset 0x020 ..... 562
Register 10: SSI DMA Control (SSIDMACTL), offset 0x024 ..... 563
Register 11: SSI Peripheral Identification 4 (SSIPeriphID4), offset 0xFD0 ..... 564
Register 12: SSI Peripheral Identification 5 (SSIPeriphID5), offset 0xFD4 ..... 565
Register 13: SSI Peripheral Identification 6 (SSIPeriphID6), offset 0xFD8 ..... 566
Register 14: SSI Peripheral Identification 7 (SSIPeriphID7), offset 0xFDC ..... 567
Register 15: SSI Peripheral Identification 0 (SSIPeriphID0), offset 0xFE0 ..... 568
Register 16: SSI Peripheral Identification 1 (SSIPeriphID1), offset 0xFE4 ..... 569
Register 17: SSI Peripheral Identification 2 (SSIPeriphID2), offset 0xFE8 ..... 570
Register 18: SSI Peripheral Identification 3 (SSIPeriphID3), offset 0xFEC ..... 571
Register 19: SSI PrimeCell Identification 0 (SSIPCellID0), offset 0xFF0 ..... 572
Register 20: SSI PrimeCell Identification 1 (SSIPCellID1), offset 0xFF4 ..... 573
Register 21: SSI PrimeCell Identification 2 (SSIPCellID2), offset 0xFF8 ..... 574
Register 22: SSI PrimeCell Identification 3 (SSIPCellID3), offset 0xFFC ..... 575

**Inter-Integrated Circuit ($I^2C$) Interface ..... 576**

Register 1: $I^2C$ Master Slave Address (I2CMSA), offset 0x000 ..... 592
Register 2: $I^2C$ Master Control/Status (I2CMCS), offset 0x004 ..... 593
Register 3: $I^2C$ Master Data (I2CMDR), offset 0x008 ..... 597
Register 4: $I^2C$ Master Timer Period (I2CMTPR), offset 0x00C ..... 598
Register 5: $I^2C$ Master Interrupt Mask (I2CMIMR), offset 0x010 ..... 599
Register 6: $I^2C$ Master Raw Interrupt Status (I2CMRIS), offset 0x014 ..... 600
Register 7: $I^2C$ Master Masked Interrupt Status (I2CMMIS), offset 0x018 ..... 601
Register 8: $I^2C$ Master Interrupt Clear (I2CMICR), offset 0x01C ..... 602
Register 9: $I^2C$ Master Configuration (I2CMCR), offset 0x020 ..... 603

Register 10: I2C Slave Own Address (I2CSOAR), offset 0x800 ..... 605
Register 11: I2C Slave Control/Status (I2CSCSR), offset 0x804 ..... 606
Register 12: I2C Slave Data (I2CSDR), offset 0x808 ..... 608
Register 13: I2C Slave Interrupt Mask (I2CSIMR), offset 0x80C ..... 609
Register 14: I2C Slave Raw Interrupt Status (I2CSRIS), offset 0x810 ..... 610
Register 15: I2C Slave Masked Interrupt Status (I2CSMIS), offset 0x814 ..... 611
Register 16: I2C Slave Interrupt Clear (I2CSICR), offset 0x818 ..... 612

**Controller Area Network (CAN) Module ..... 613**
Register 1: CAN Control (CANCTL), offset 0x000 ..... 634
Register 2: CAN Status (CANSTS), offset 0x004 ..... 636
Register 3: CAN Error Counter (CANERR), offset 0x008 ..... 638
Register 4: CAN Bit Timing (CANBIT), offset 0x00C ..... 639
Register 5: CAN Interrupt (CANINT), offset 0x010 ..... 640
Register 6: CAN Test (CANTST), offset 0x014 ..... 641
Register 7: CAN Baud Rate Prescaler Extension (CANBRPE), offset 0x018 ..... 643
Register 8: CAN IF1 Command Request (CANIF1CRQ), offset 0x020 ..... 644
Register 9: CAN IF2 Command Request (CANIF2CRQ), offset 0x080 ..... 644
Register 10: CAN IF1 Command Mask (CANIF1CMSK), offset 0x024 ..... 645
Register 11: CAN IF2 Command Mask (CANIF2CMSK), offset 0x084 ..... 645
Register 12: CAN IF1 Mask 1 (CANIF1MSK1), offset 0x028 ..... 647
Register 13: CAN IF2 Mask 1 (CANIF2MSK1), offset 0x088 ..... 647
Register 14: CAN IF1 Mask 2 (CANIF1MSK2), offset 0x02C ..... 648
Register 15: CAN IF2 Mask 2 (CANIF2MSK2), offset 0x08C ..... 648
Register 16: CAN IF1 Arbitration 1 (CANIF1ARB1), offset 0x030 ..... 649
Register 17: CAN IF2 Arbitration 1 (CANIF2ARB1), offset 0x090 ..... 649
Register 18: CAN IF1 Arbitration 2 (CANIF1ARB2), offset 0x034 ..... 650
Register 19: CAN IF2 Arbitration 2 (CANIF2ARB2), offset 0x094 ..... 650
Register 20: CAN IF1 Message Control (CANIF1MCTL), offset 0x038 ..... 652
Register 21: CAN IF2 Message Control (CANIF2MCTL), offset 0x098 ..... 652
Register 22: CAN IF1 Data A1 (CANIF1DA1), offset 0x03C ..... 654
Register 23: CAN IF1 Data A2 (CANIF1DA2), offset 0x040 ..... 654
Register 24: CAN IF1 Data B1 (CANIF1DB1), offset 0x044 ..... 654
Register 25: CAN IF1 Data B2 (CANIF1DB2), offset 0x048 ..... 654
Register 26: CAN IF2 Data A1 (CANIF2DA1), offset 0x09C ..... 654
Register 27: CAN IF2 Data A2 (CANIF2DA2), offset 0x0A0 ..... 654
Register 28: CAN IF2 Data B1 (CANIF2DB1), offset 0x0A4 ..... 654
Register 29: CAN IF2 Data B2 (CANIF2DB2), offset 0x0A8 ..... 654
Register 30: CAN Transmission Request 1 (CANTXRQ1), offset 0x100 ..... 655
Register 31: CAN Transmission Request 2 (CANTXRQ2), offset 0x104 ..... 655
Register 32: CAN New Data 1 (CANNWDA1), offset 0x120 ..... 656
Register 33: CAN New Data 2 (CANNWDA2), offset 0x124 ..... 656
Register 34: CAN Message 1 Interrupt Pending (CANMSG1INT), offset 0x140 ..... 657
Register 35: CAN Message 2 Interrupt Pending (CANMSG2INT), offset 0x144 ..... 657
Register 36: CAN Message 1 Valid (CANMSG1VAL), offset 0x160 ..... 658
Register 37: CAN Message 2 Valid (CANMSG2VAL), offset 0x164 ..... 658

**Universal Serial Bus (USB) Controller ..... 659**
Register 1: USB Device Functional Address (USBFADDR), offset 0x000 ..... 676
Register 2: USB Power (USBPOWER), offset 0x001 ..... 677

Register 3: USB Transmit Interrupt Status (USBTXIS), offset 0x002 ..... 680
Register 4: USB Receive Interrupt Status (USBRXIS), offset 0x004 ..... 681
Register 5: USB Transmit Interrupt Enable (USBTXIE), offset 0x006 ..... 682
Register 6: USB Receive Interrupt Enable (USBRXIE), offset 0x008 ..... 683
Register 7: USB General Interrupt Status (USBIS), offset 0x00A ..... 684
Register 8: USB Interrupt Enable (USBIE), offset 0x00B ..... 687
Register 9: USB Frame Value (USBFRAME), offset 0x00C ..... 690
Register 10: USB Endpoint Index (USBEPIDX), offset 0x00E ..... 691
Register 11: USB Test Mode (USBTEST), offset 0x00F ..... 692
Register 12: USB FIFO Endpoint 0 (USBFIFO0), offset 0x020 ..... 694
Register 13: USB FIFO Endpoint 1 (USBFIFO1), offset 0x024 ..... 694
Register 14: USB FIFO Endpoint 2 (USBFIFO2), offset 0x028 ..... 694
Register 15: USB FIFO Endpoint 3 (USBFIFO3), offset 0x02C ..... 694
Register 16: USB Device Control (USBDEVCTL), offset 0x060 ..... 695
Register 17: USB Transmit Dynamic FIFO Sizing (USBTXFIFOSZ), offset 0x062 ..... 696
Register 18: USB Receive Dynamic FIFO Sizing (USBRXFIFOSZ), offset 0x063 ..... 696
Register 19: USB Transmit FIFO Start Address (USBTXFIFOADD), offset 0x064 ..... 697
Register 20: USB Receive FIFO Start Address (USBRXFIFOADD), offset 0x066 ..... 697
Register 21: USB Connect Timing (USBCONTIM), offset 0x07A ..... 698
Register 22: USB Full-Speed Last Transaction to End of Frame Timing (USBFSEOF), offset 0x07D ..... 699
Register 23: USB Low-Speed Last Transaction to End of Frame Timing (USBLSEOF), offset 0x07E ..... 700
Register 24: USB Transmit Functional Address Endpoint 0 (USBTXFUNCADDR0), offset 0x080 ..... 701
Register 25: USB Transmit Functional Address Endpoint 1 (USBTXFUNCADDR1), offset 0x088 ..... 701
Register 26: USB Transmit Functional Address Endpoint 2 (USBTXFUNCADDR2), offset 0x090 ..... 701
Register 27: USB Transmit Functional Address Endpoint 3 (USBTXFUNCADDR3), offset 0x098 ..... 701
Register 28: USB Transmit Hub Address Endpoint 0 (USBTXHUBADDR0), offset 0x082 ..... 702
Register 29: USB Transmit Hub Address Endpoint 1 (USBTXHUBADDR1), offset 0x08A ..... 702
Register 30: USB Transmit Hub Address Endpoint 2 (USBTXHUBADDR2), offset 0x092 ..... 702
Register 31: USB Transmit Hub Address Endpoint 3 (USBTXHUBADDR3), offset 0x09A ..... 702
Register 32: USB Transmit Hub Port Endpoint 0 (USBTXHUBPORT0), offset 0x083 ..... 703
Register 33: USB Transmit Hub Port Endpoint 1 (USBTXHUBPORT1), offset 0x08B ..... 703
Register 34: USB Transmit Hub Port Endpoint 2 (USBTXHUBPORT2), offset 0x093 ..... 703
Register 35: USB Transmit Hub Port Endpoint 3 (USBTXHUBPORT3), offset 0x09B ..... 703
Register 36: USB Receive Functional Address Endpoint 1 (USBRXFUNCADDR1), offset 0x08C ..... 704
Register 37: USB Receive Functional Address Endpoint 2 (USBRXFUNCADDR2), offset 0x094 ..... 704
Register 38: USB Receive Functional Address Endpoint 3 (USBRXFUNCADDR3), offset 0x09C ..... 704
Register 39: USB Receive Hub Address Endpoint 1 (USBRXHUBADDR1), offset 0x08E ..... 705
Register 40: USB Receive Hub Address Endpoint 2 (USBRXHUBADDR2), offset 0x096 ..... 705
Register 41: USB Receive Hub Address Endpoint 3 (USBRXHUBADDR3), offset 0x09E ..... 705
Register 42: USB Receive Hub Port Endpoint 1 (USBRXHUBPORT1), offset 0x08F ..... 706
Register 43: USB Receive Hub Port Endpoint 2 (USBRXHUBPORT2), offset 0x097 ..... 706
Register 44: USB Receive Hub Port Endpoint 3 (USBRXHUBPORT3), offset 0x09F ..... 706
Register 45: USB Maximum Transmit Data Endpoint 1 (USBTXMAXP1), offset 0x110 ..... 707
Register 46: USB Maximum Transmit Data Endpoint 2 (USBTXMAXP2), offset 0x120 ..... 707
Register 47: USB Maximum Transmit Data Endpoint 3 (USBTXMAXP3), offset 0x130 ..... 707
Register 48: USB Control and Status Endpoint 0 Low (USBCSRL0), offset 0x102 ..... 708
Register 49: USB Control and Status Endpoint 0 High (USBCSRH0), offset 0x103 ..... 712
Register 50: USB Receive Byte Count Endpoint 0 (USBCOUNT0), offset 0x108 ..... 714

Register 51: USB Type Endpoint 0 (USBTYPE0), offset 0x10A ..... 715
Register 52: USB NAK Limit (USBNAKLMT), offset 0x10B ..... 716
Register 53: USB Transmit Control and Status Endpoint 1 Low (USBTXCSRL1), offset 0x112 ..... 717
Register 54: USB Transmit Control and Status Endpoint 2 Low (USBTXCSRL2), offset 0x122 ..... 717
Register 55: USB Transmit Control and Status Endpoint 3 Low (USBTXCSRL3), offset 0x132 ..... 717
Register 56: USB Transmit Control and Status Endpoint 1 High (USBTXCSRH1), offset 0x113 ..... 721
Register 57: USB Transmit Control and Status Endpoint 2 High (USBTXCSRH2), offset 0x123 ..... 721
Register 58: USB Transmit Control and Status Endpoint 3 High (USBTXCSRH3), offset 0x133 ..... 721
Register 59: USB Maximum Receive Data Endpoint 1 (USBRXMAXP1), offset 0x114 ..... 725
Register 60: USB Maximum Receive Data Endpoint 2 (USBRXMAXP2), offset 0x124 ..... 725
Register 61: USB Maximum Receive Data Endpoint 3 (USBRXMAXP3), offset 0x134 ..... 725
Register 62: USB Receive Control and Status Endpoint 1 Low (USBRXCSRL1), offset 0x116 ..... 726
Register 63: USB Receive Control and Status Endpoint 2 Low (USBRXCSRL2), offset 0x126 ..... 726
Register 64: USB Receive Control and Status Endpoint 3 Low (USBRXCSRL3), offset 0x136 ..... 726
Register 65: USB Receive Control and Status Endpoint 1 High (USBRXCSRH1), offset 0x117 ..... 731
Register 66: USB Receive Control and Status Endpoint 2 High (USBRXCSRH2), offset 0x127 ..... 731
Register 67: USB Receive Control and Status Endpoint 3 High (USBRXCSRH3), offset 0x137 ..... 731
Register 68: USB Receive Byte Count Endpoint 1 (USBRXCOUNT1), offset 0x118 ..... 735
Register 69: USB Receive Byte Count Endpoint 2 (USBRXCOUNT2), offset 0x128 ..... 735
Register 70: USB Receive Byte Count Endpoint 3 (USBRXCOUNT3), offset 0x138 ..... 735
Register 71: USB Host Transmit Configure Type Endpoint 1 (USBTXTYPE1), offset 0x11A ..... 736
Register 72: USB Host Transmit Configure Type Endpoint 2 (USBTXTYPE2), offset 0x12A ..... 736
Register 73: USB Host Transmit Configure Type Endpoint 3 (USBTXTYPE3), offset 0x13A ..... 736
Register 74: USB Host Transmit Interval Endpoint 1 (USBTXINTERVAL1), offset 0x11B ..... 737
Register 75: USB Host Transmit Interval Endpoint 2 (USBTXINTERVAL2), offset 0x12B ..... 737
Register 76: USB Host Transmit Interval Endpoint 3 (USBTXINTERVAL3), offset 0x13B ..... 737
Register 77: USB Host Configure Receive Type Endpoint 1 (USBRXTYPE1), offset 0x11C ..... 738
Register 78: USB Host Configure Receive Type Endpoint 2 (USBRXTYPE2), offset 0x12C ..... 738
Register 79: USB Host Configure Receive Type Endpoint 3 (USBRXTYPE3), offset 0x13C ..... 738
Register 80: USB Host Receive Polling Interval Endpoint 1 (USBRXINTERVAL1), offset 0x11D ..... 739
Register 81: USB Host Receive Polling Interval Endpoint 2 (USBRXINTERVAL2), offset 0x12D ..... 739
Register 82: USB Host Receive Polling Interval Endpoint 3 (USBRXINTERVAL3), offset 0x13D ..... 739
Register 83: USB Request Packet Count in Block Transfer Endpoint 1 (USBRQPKTCOUNT1), offset 0x304 ..... 740
Register 84: USB Request Packet Count in Block Transfer Endpoint 2 (USBRQPKTCOUNT2), offset 0x308 ..... 740
Register 85: USB Request Packet Count in Block Transfer Endpoint 3 (USBRQPKTCOUNT3), offset 0x30C ..... 740
Register 86: USB Receive Double Packet Buffer Disable (USBRXDPKTBUFDIS), offset 0x340 ..... 741
Register 87: USB Transmit Double Packet Buffer Disable (USBTXDPKTBUFDIS), offset 0x342 ..... 742
Register 88: USB External Power Control (USBEPC), offset 0x400 ..... 743
Register 89: USB External Power Control Raw Interrupt Status (USBEPCRIS), offset 0x404 ..... 746
Register 90: USB External Power Control Interrupt Mask (USBEPCIM), offset 0x408 ..... 747
Register 91: USB External Power Control Interrupt Status and Clear (USBEPCISC), offset 0x40C ..... 748
Register 92: USB Device RESUME Raw Interrupt Status (USBDRRIS), offset 0x410 ..... 749
Register 93: USB Device RESUME Interrupt Mask (USBDRIM), offset 0x414 ..... 750
Register 94: USB Device RESUME Interrupt Status and Clear (USBDRISC), offset 0x418 ..... 751
Register 95: USB General-Purpose Control and Status (USBGPCS), offset 0x41C ..... 752

# Revision History

The revision history table notes changes made between the indicated revisions of the LM3S5632 data sheet.

**Table 1. Revision History**

| Date | Revision | Description |
|---|---|---|
| July 2014 | 15852.2743 | ■ In JTAG chapter, clarified procedure for recovering a locked device as well as sections on JTAG-to-SWD Switching and SWD-to-JTAG Switching.<br>■ In System Control chapter, clarified behavior of **Reset Cause (RESC)** register external reset bit.<br>■ In Internal Memory chapter:<br>– Added sections on Execute-Only Protection, Read-Only Protection, and Permanently Disabling Debug.<br>– Noted that the **Boot Configuration (BOOTCFG)** register requires a POR before committed changes to the Flash-resident registers take effect.<br>■ In UART chapter:<br>– Clarified that the transmit interrupt is based on a transition through level.<br>– Corrected reset for **UART Raw Interrupt Status (UARTRIS)** register.<br>■ In Electrical Characteristics chapter, updated Crystal Characteristics table.<br>■ In Ordering and Contact Information appendix, moved orderable part numbers table to addendum.<br>■ Additional minor data sheet clarifications and corrections. |

**Table 1. Revision History *(continued)***

| Date | Revision | Description |
|---|---|---|
| November 2011 | 11107 | ■ Clarified that when the USB module is in operation, MOSC must be provided with a clock source, and the system clock must be at least 30 MHz.<br>■ Added module-specific pin tables to each chapter in the new Signal Description sections.<br>■ In Hibernation chapter:<br>– Changed terminology from non-volatile memory to battery-backed memory.<br>– Clarified Hibernation module register reset conditions.<br>■ In Internal Memory chapter, corrected note in **USER_DBG** and **USER_REG0/1/2/3** registers, that once committed, the value of the register can never be restored to the factory default value.<br>■ In Timer chapter, clarified that in 16-Bit Input Edge Time Mode, the timer is capable of capturing three types of events: rising edge, falling edge, or both.<br>■ In UART chapter, clarified interrupt behavior.<br>■ In SSI chapter, corrected SSIClk in the figure "Synchronous Serial Frame Format (Single Transfer)".<br>■ In USB chapter:<br>– Removed `MULTTRAN` bit from **USB Transmit Hub Address Endpoint n (USBTXHUBADDRn)** and **USB Receive Hub Address Endpoint n (USBRXHUBADDRn)** registers.<br>– Removed `DISCON` bit from Device Mode table for **USB General Interrupt Status (USBIS)** register.<br>– Added `WTID` bit to **USB Connect Timing (USBCONTIM)** register.<br>■ In Signal Tables chapter:<br>– Corrected pin numbers in table "Connections for Unused Signals" (other pin tables were correct).<br>■ In Electrical Characteristics chapter:<br>– Corrected values in "Detailed Power Specifications" table.<br>– Added "System Clock Characteristics with USB Operation" table.<br>– Corrected Nom values for parameters "TCK clock Low time" and "TCK clock High time" in "JTAG Characteristics" table.<br>– Corrected missing values for "Conversion time" and "Conversion rate" parameters in "ADC Characteristics" table.<br>■ Additional minor data sheet clarifications and corrections. |

**Table 1. Revision History *(continued)***

| Date | Revision | Description |
|---|---|---|
| January 2011 | 9102 | ■ In **Application Interrupt and Reset Control (APINT)** register, changed bit name from SYSRESETREQ to SYSRESREQ.<br>■ Added DEBUG (Debug Priority) bit field to **System Handler Priority 3 (SYSPRI3)** register.<br>■ Added "Reset Sources" table to System Control chapter.<br>■ Removed mention of false-start bit detection in the UART chapter. This feature is not supported.<br>■ Added note that specific module clocks must be enabled before that module's registers can be programmed. There must be a delay of 3 system clocks after the module clock is enabled before any of that module's registers are accessed.<br>■ Changed $I^2C$ slave register base addresses and offsets to be relative to the $I^2C$ module base address of 0x4002.0000 and 0x4002.1000, so register bases and offsets were changed for all $I^2C$ slave registers. Note that the hw_i2c.h file in the StellarisWare® Driver Library uses a base address of 0x4002.0800 and 0x4002.1800 for the $I^2C$ slave registers. Be aware when using registers with offsets between 0x800 and 0x818 that StellarisWare uses the old slave base address for these offsets.<br>■ Corrected nonlinearity and offset error parameters ($E_L$, $E_D$ and $E_O$) in ADC Characteristics table.<br>■ Added specification for maximum input voltage on a non-power pin when the microcontroller is unpowered ($V_{NON}$ parameter in Maximum Ratings table).<br>■ Additional minor data sheet clarifications and corrections. |
| September 2010 | 7783 | ■ Reorganized ARM Cortex-M3 Processor Core, Memory Map and Interrupts chapters, creating two new chapters, The Cortex-M3 Processor and Cortex-M3 Peripherals. Much additional content was added, including all the Cortex-M3 registers.<br>■ Changed register names to be consistent with StellarisWare names: the Cortex-M3 **Interrupt Control and Status (ICSR)** register to the **Interrupt Control and State (INTCTRL)** register, and the Cortex-M3 **Interrupt Set Enable (SETNA)** register to the **Interrupt 0-31 Set Enable (EN0)** register.<br>■ In the Internal Memory chapter:<br>– Added clarification of instruction execution during Flash operations.<br>– Deleted **ROM Version (RMVER)** register as it is not used.<br>■ In the GPIO chapter:<br>– Renamed the **GPIO High-Speed Control (GPIOHSCTL)** register to the **GPIO High-Performance Bus Control (GPIOHBCTL)** register.<br>– Added clarification about the operation of the Advanced High-Performance Bus (AHB) and the legacy Advanced Peripheral Bus (APB).<br>– Modified Figure 9-1 on page 354 and Figure 9-2 on page 355 to clarify operation of the GPIO inputs when used as an alternate function.<br>■ In General-Purpose Timers chapter, clarified operation of the 32-bit RTC mode.<br>■ Numerous improvements and clarifications to the USB chapter. Also corrected definitions for bits 2 and 5 in the **USBIE** register.<br>■ In Electrical Characteristics chapter:<br>– Added "Input voltage for a GPIO configured as an analog input" value to Table 21-1 on page 766.<br>– Added $I_{LKG}$ parameter (GPIO input leakage current) to Table 21-4 on page 767.<br>– Corrected values for $t_{CLKRF}$ parameter (SSIClk rise/fall time) in Table 21-22 on page 777.<br>■ Added dimensions for Tray and Tape and Reel shipping mediums. |

**Table 1. Revision History *(continued)***

| Date | Revision | Description |
|---|---|---|
| June 2010 | 7403 | ■ Corrected base address for SRAM in architectural overview chapter.<br>■ Clarified system clock operation, adding content to “Clock Control” on page 176.<br>■ Clarified CAN bit timing examples.<br>■ In Signal Tables chapter, added table "Connections for Unused Signals."<br>■ In "Reset Characteristics" table, corrected value for supply voltage (VDD) rise time.<br>■ Additional minor data sheet clarifications and corrections. |
| April 2010 | 7021 | ■ Added caution note to the **I²C Master Timer Period (I2CMTPR)** register description and changed field width to 7 bits.<br>■ Added note about $\overline{\text{RST}}$ signal routing.<br>■ Clarified the function of the TnSTALL bit in the **GPTMCTL** register.<br>■ Additional minor data sheet clarifications and corrections. |
| January 2010 | 6707 | ■ In "System Control" section, clarified Debug Access Port operation after Sleep modes.<br>■ Clarified wording on Flash memory access errors.<br>■ Added section on Flash interrupts.<br>■ Changed the reset value of the **ADC Sample Sequence Result FIFO n (ADCSSFIFOn)** registers to be indeterminate.<br>■ Clarified operation of SSI transmit FIFO.<br>■ Made these changes to the Operating Characteristics chapter:<br>– Added storage temperature ratings to "Temperature Characteristics" table<br>– Added "ESD Absolute Maximum Ratings" table<br>■ Made these changes to the Electrical Characteristics chapter:<br>– In "Flash Memory Characteristics" table, corrected Mass erase time<br>– Added sleep and deep-sleep wake-up times ("Sleep Modes AC Characteristics" table)<br>– In "Reset Characteristics" table, corrected units for supply voltage (VDD) rise time |

**Table 1. Revision History *(continued)***

| Date | Revision | Description |
|---|---|---|
| October 2009 | 6449 | ■ Removed the MAXADCSPD bit field from the **DCGC0** register as it has no function in deep-sleep mode.<br>■ Deleted reset value for 16-bit mode from **GPTMTAILR**, **GPTMTAMATCHR**, and **GPTMTAR** registers because the module resets in 32-bit mode.<br>■ Clarified CAN bit timing and corrected examples.<br>■ Corrected description for ADDR bit field in **USBTXFIFOSZ** and **USBRXFIFOSZ** registers.<br>■ Clarified PWM source for ADC triggering<br>■ Made these changes to the Electrical Characteristics chapter:<br>– Removed $V_{SIH}$ and $V_{SIL}$ parameters from Operating Conditions table.<br>– Changed SSI set up and hold times to be expressed in system clocks, not ns.<br>– Revised ADC electrical specifications to clarify, including reorganizing and adding new data.<br>– Changed the name of the $t_{HIB\_REG\_WRITE}$ parameter to $t_{HIB\_REG\_ACCESS}$.<br>– Table added showing actual PLL frequency depending on input crystal.<br>■ Additional minor data sheet clarifications and corrections. |
| July 2009 | 5920 | ■ Clarified Power-on reset and $\overline{RST}$ pin operation; added new diagrams.<br>■ Corrected the reset value of the **Hibernation Data (HIBDATA)** and **Hibernation Control (HIBCTL)** registers.<br>■ Clarified explanation of nonvolatile register programming in Internal Memory chapter.<br>■ Added explanation of reset value to **FMPRE0/1/2/3**, **FMPPE0/1/2/3**, **USER_DBG**, and **USER_REG0/1** registers.<br>■ Special bulk handling and packet splitting has never been supported as the µDMA module can support the same function. As a result, all references to these topics has been removed. Bit 7 in the **USBTXCSRLn** register only functions as NAKTO in Host mode and is reserved in Device mode. In addition, bit 0 in the **USBRXCSRHn** register is reserved.<br>■ The DISCON and CONN bits in the **USBIS** and **USBIE** registers are not available in Device mode. When the USB controller is acting as a self-powered Device, a GPIO input or analog comparator input must be connected to VBUS and configured to generate an interrupt when the VBUS level drops. This interrupt is used to disable the pullup resistor on the USB0DP signal.<br>■ Changed buffer type for $\overline{WAKE}$ pin to TTL.<br>■ In ADC characteristics table, changed Max value for GAIN parameter from ±1 to ±3 and added $E_{IR}$(Internal voltage reference error) parameter.<br>■ Changed ordering numbers.<br>■ Additional minor data sheet clarifications and corrections. |

**Table 1. Revision History *(continued)***

| Date | Revision | Description |
|---|---|---|
| April 2009 | 5368 | ■ Added JTAG/SWD clarification (see “Communication with JTAG/SWD” on page 165).<br>■ Added clarification that the PLL operates at 400 MHz, but is divided by two prior to the application of the output divisor.<br>■ Corrected bits 2:1 in **I2CSIMR**, **I2CSRIS**, **I2CSMIS**, and **I2CSICR** registers to be reserved bits (cannot interrupt on start and stop conditions).<br>■ Corrected bits 15:11 in **USBTXMAXP0/1/2** and **USBRXMAXP0/1/2** registers to be reserved bits (cannot define multiplier).<br>■ Additional minor data sheet clarifications and corrections. |
| January 2009 | 4724 | ■ Corrected bit type for RELOAD bit field in SysTick Reload Value register; changed to R/W.<br>■ Added clarification as to what happens when the SSI in slave mode is required to transmit but there is no data in the TX FIFO.<br>■ Added section called "Setting the Device Address" for special considerations when writing the **USBFADDR** register.<br>■ Corrected **USBEPIDX** to be an 8-bit register.<br>■ Added comparator operating mode tables.<br>■ Corrected pin types of signals $\overline{\text{RST}}$ to "in" and USB0RBIAS to "out".<br>■ Additional minor data sheet clarifications and corrections. |
| November 2008 | 4283 | ■ Revised High-Level Block Diagram.<br>■ Additional minor data sheet clarifications and corrections were made. |

**Table 1. Revision History *(continued)***

| Date | Revision | Description |
|---|---|---|
| October 2008 | 4149 | ■ Added note on clearing interrupts to the Interrupts chapter:<br><br>**Note:** It may take several processor cycles after a write to clear an interrupt source in order for NVIC to see the interrupt source de-assert. This means if the interrupt clear is done as the last action in an interrupt handler, it is possible for the interrupt handler to complete while NVIC sees the interrupt as still asserted, causing the interrupt handler to be re-entered errantly. This can be avoided by either clearing the interrupt source at the beginning of the interrupt handler or by performing a read or write after the write to clear the interrupt source (and flush the write buffer)<br><br>■ Added clarification on JTAG reset to the JTAG chapter:<br><br>In order to reset the JTAG module after the device has been powered on, the TMS input must be held HIGH for five TCK clock cycles, resetting the TAP controller and all associated JTAG chains.<br><br>■ The binary value was incorrect in the JTAG 16-bit switch sequence in the JTAG-to-SWD Switching section in the JTAG chapter. Sentence changed to:<br><br>The 16-bit switch sequence for switching to JTAG mode is defined as b1110011100111100, transmitted LSB first.<br><br>■ The FMA value for the **FMPRE3** register was incorrect in the Flash Resident Registers table in the Internal Memory chapter. The correct value is 0x0000.0006.<br><br>■ Step 1 of the Initialization and Configuration procedure in the ADC chapter states the wrong register to use to enable the ADC clock. Sentence changed to:<br><br>1. Enable the ADC clock by writing a value of 0x0001.0000 to the **RCGC0** register.<br><br>■ In the CAN chapter, major improvements were made including a rewrite of the conceptual information and the addition of new figures to clarify how to use the Controller Area Network (CAN) module.<br><br>■ In the USB chapter, clarified endpoint terminology and added a new section on DMA Operation.<br><br>■ Additional minor data sheet clarifications and corrections were made. |
| June 2008 | 2972 | Started tracking revision history. |

# About This Document

This data sheet provides reference information for the LM3S5632 microcontroller, describing the functional blocks of the system-on-chip (SoC) device designed around the ARM® Cortex™-M3 core.

## Audience

This manual is intended for system software developers, hardware designers, and application developers.

## About This Manual

This document is organized into sections that correspond to each major feature.

## Related Documents

The following related documents are available on the Stellaris® web site at www.ti.com/stellaris:

- *Stellaris® Errata*
- *ARM® Cortex™-M3 Errata*
- *Cortex™-M3/M4 Instruction Set Technical User's Manual*
- *Stellaris® Boot Loader User's Guide*
- *Stellaris® Graphics Library User's Guide*
- *Stellaris® Peripheral Driver Library User's Guide*
- *Stellaris® ROM User's Guide*
- *Stellaris® USB Library User's Guide*

The following related documents are also referenced:

- *ARM® Debug Interface V5 Architecture Specification*
- *ARM® Embedded Trace Macrocell Architecture Specification*
- *IEEE Standard 1149.1-Test Access Port and Boundary-Scan Architecture*

This documentation list was current as of publication date. Please check the web site for additional documentation, including application notes and white papers.

# Documentation Conventions

This document uses the conventions shown in Table 2 on page 34.

**Table 2. Documentation Conventions**

| Notation | Meaning |
|---|---|
| **General Register Notation** | |
| **REGISTER** | APB registers are indicated in uppercase bold. For example, **PBORCTL** is the Power-On and Brown-Out Reset Control register. If a register name contains a lowercase n, it represents more than one register. For example, **SRCRn** represents any (or all) of the three Software Reset Control registers: **SRCR0, SRCR1** , and **SRCR2**. |
| bit | A single bit in a register. |
| bit field | Two or more consecutive and related bits. |
| offset 0x*nnn* | A hexadecimal increment to a register's address, relative to that module's base address as specified in Table 2-4 on page 72. |
| Register *N* | Registers are numbered consecutively throughout the document to aid in referencing them. The register number has no meaning to software. |
| reserved | Register bits marked *reserved* are reserved for future use. In most cases, reserved bits are set to 0; however, user software should not rely on the value of a reserved bit. To provide software compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| *yy:xx* | The range of register bits inclusive from xx to yy. For example, 31:15 means bits 15 through 31 in that register. |
| **Register Bit/Field Types** | This value in the register bit diagram indicates whether software running on the controller can change the value of the bit field. |
| RC | Software can read this field. The bit or field is cleared by hardware after reading the bit/field. |
| RO | Software can read this field. Always write the chip reset value. |
| R/W | Software can read or write this field. |
| R/WC | Software can read or write this field. Writing to it with any value clears the register. |
| R/W1C | Software can read or write this field. A write of a 0 to a W1C bit does not affect the bit value in the register. A write of a 1 clears the value of the bit in the register; the remaining bits remain unchanged.<br>This register type is primarily used for clearing interrupt status bits where the read operation provides the interrupt status and the write of the read value clears only the interrupts being reported at the time the register was read. |
| R/W1S | Software can read or write a 1 to this field. A write of a 0 to a R/W1S bit does not affect the bit value in the register. |
| W1C | Software can write this field. A write of a 0 to a W1C bit does not affect the bit value in the register. A write of a 1 clears the value of the bit in the register; the remaining bits remain unchanged. A read of the register returns no meaningful data.<br>This register is typically used to clear the corresponding bit in an interrupt register. |
| WO | Only a write by software is valid; a read of the register returns no meaningful data. |
| **Register Bit/Field Reset Value** | This value in the register bit diagram shows the bit/field value after any reset, unless noted. |
| 0 | Bit cleared to 0 on chip reset. |
| 1 | Bit set to 1 on chip reset. |
| - | Nondeterministic. |
| **Pin/Signal Notation** | |
| [ ] | Pin alternate function; a pin defaults to the signal without the brackets. |
| pin | Refers to the physical connection on the package. |
| signal | Refers to the electrical signal encoding of a pin. |

**Table 2. Documentation Conventions *(continued)***

| Notation | Meaning |
|---|---|
| assert a signal | Change the value of the signal from the logically False state to the logically True state. For active High signals, the asserted signal value is 1 (High); for active Low signals, the asserted signal value is 0 (Low). The active polarity (High or Low) is defined by the signal name (see SIGNAL and $\overline{\text{SIGNAL}}$ below). |
| deassert a signal | Change the value of the signal from the logically True state to the logically False state. |
| $\overline{\text{SIGNAL}}$ | Signal names are in uppercase and in the Courier font. An overbar on a signal name indicates that it is active Low. To assert $\overline{\text{SIGNAL}}$ is to drive it Low; to deassert $\overline{\text{SIGNAL}}$ is to drive it High. |
| SIGNAL | Signal names are in uppercase and in the Courier font. An active High signal has no overbar. To assert SIGNAL is to drive it High; to deassert SIGNAL is to drive it Low. |
| **Numbers** | |
| X | An uppercase X indicates any of several values is allowed, where X can be any legal pattern. For example, a binary value of 0X00 can be either 0100 or 0000, a hex value of 0xX is 0x0 or 0x1, and so on. |
| 0x | Hexadecimal numbers have a prefix of 0x. For example, 0x00FF is the hexadecimal number FF.<br>All other numbers within register tables are assumed to be binary. Within conceptual information, binary numbers are indicated with a b suffix, for example, 1011b, and decimal numbers are written without a prefix or suffix. |

# 1 Architectural Overview

The Stellaris® family of microcontrollers—the first ARM® Cortex™-M3 based controllers—brings high-performance 32-bit computing to cost-sensitive embedded microcontroller applications. These pioneering parts deliver customers 32-bit performance at a cost equivalent to legacy 8- and 16-bit devices, all in a package with a small footprint.

The Stellaris family offers efficient performance and extensive integration, favorably positioning the device into cost-conscious applications requiring significant control-processing and connectivity capabilities. The Stellaris LM3S3000 series provides the industry's first ARM® Cortex™-M3 microcontrollers with USB 2.0 Full-Speed On-The-Go/Host/Device combinations.

The LM3S5632 microcontroller is targeted for industrial applications, including remote monitoring, electronic point-of-sale machines, test and measurement equipment, network appliances and switches, factory automation, HVAC and building control, gaming equipment, motion control, medical instrumentation, and fire and security.

For applications requiring extreme conservation of power, the LM3S5632 microcontroller features a battery-backed Hibernation module to efficiently power down the LM3S5632 to a low-power state during extended periods of inactivity. With a power-up/power-down sequencer, a continuous time counter (RTC), a pair of match registers, an APB interface to the system bus, and dedicated non-volatile memory, the Hibernation module positions the LM3S5632 microcontroller perfectly for battery applications.

In addition, the LM3S5632 microcontroller offers the advantages of ARM's widely available development tools, System-on-Chip (SoC) infrastructure IP applications, and a large user community. Additionally, the microcontroller uses ARM's Thumb®-compatible Thumb-2 instruction set to reduce memory requirements and, thereby, cost. Finally, the LM3S5632 microcontroller is code-compatible to all members of the extensive Stellaris family; providing flexibility to fit our customers' precise needs.

Texas Instruments offers a complete solution to get to market quickly, with evaluation and development boards, white papers and application notes, an easy-to-use peripheral driver library, and a strong support, sales, and distributor network. See “Ordering and Contact Information” on page 826 for ordering information for Stellaris family devices.

## 1.1 Product Features

The LM3S5632 microcontroller includes the following product features:

- 32-Bit RISC Performance
  - 32-bit ARM® Cortex™-M3 v7M architecture optimized for small-footprint embedded applications
  - System timer (SysTick), providing a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism
  - Thumb®-compatible Thumb-2-only instruction set processor core for high code density
  - 50-MHz operation
  - Hardware-division and single-cycle-multiplication

- Integrated Nested Vectored Interrupt Controller (NVIC) providing deterministic interrupt handling
- 28 interrupts with eight priority levels
- Memory protection unit (MPU), providing a privileged mode for protected operating system functionality
- Unaligned data access, enabling data to be efficiently packed into memory
- Atomic bit manipulation (bit-banding), delivering maximum memory utilization and streamlined peripheral control

■ ARM® Cortex™-M3 Processor Core

- Compact core.
- Thumb-2 instruction set, delivering the high-performance expected of an ARM core in the memory size usually associated with 8- and 16-bit devices; typically in the range of a few kilobytes of memory for microcontroller class applications.
- Rapid application execution through Harvard architecture characterized by separate buses for instruction and data.
- Exceptional interrupt handling, by implementing the register manipulations required for handling an interrupt in hardware.
- Deterministic, fast interrupt processing: always 12 cycles, or just 6 cycles with tail-chaining
- External non-maskable interrupt signal (NMI) available for immediate execution of NMI handler for safety critical applications.
- Memory protection unit (MPU) to provide a privileged mode of operation for complex applications.
- Migration from the ARM7™ processor family for better performance and power efficiency.
- Full-featured debug solution
  - Serial Wire JTAG Debug Port (SWJ-DP)
  - Flash Patch and Breakpoint (FPB) unit for implementing breakpoints
  - Data Watchpoint and Trigger (DWT) unit for implementing watchpoints, trigger resources, and system profiling
  - Instrumentation Trace Macrocell (ITM) for support of printf style debugging
  - Trace Port Interface Unit (TPIU) for bridging to a Trace Port Analyzer
- Optimized for single-cycle flash usage
- Three sleep modes with clock gating for low power
- Single-cycle multiply instruction and hardware divide

  - Atomic operations
  - ARM Thumb2 mixed 16-/32-bit instruction set
  - 1.25 DMIPS/MHz
- JTAG
  - IEEE 1149.1-1990 compatible Test Access Port (TAP) controller
  - Four-bit Instruction Register (IR) chain for storing JTAG instructions
  - IEEE standard instructions: BYPASS, IDCODE, SAMPLE/PRELOAD, EXTEST and INTEST
  - ARM additional instructions: APACC, DPACC and ABORT
  - Integrated ARM Serial Wire Debug (SWD)
- Hibernation
  - System power control using discrete external regulator
  - Dedicated pin for waking from an external signal
  - Low-battery detection, signaling, and interrupt generation
  - 32-bit real-time clock (RTC)
  - Two 32-bit RTC match registers for timed wake-up and interrupt generation
  - Clock source from a 32.768-kHz external oscillator or a 4.194304-MHz crystal
  - RTC predivider trim for making fine adjustments to the clock rate
  - 64 32-bit words of non-volatile memory
  - Programmable interrupts for RTC match, external wake, and low battery events
- Internal Memory
  - 128 KB single-cycle flash
    - User-managed flash block protection on a 2-KB block basis
    - User-managed flash data programming
    - User-defined and managed flash-protection block
  - 32 KB single-cycle SRAM
  - Pre-programmed ROM
    - Stellaris family peripheral driver library (DriverLib)
    - Stellaris boot loader
- DMA Controller

- ARM PrimeCell® 32-channel configurable µDMA controller
- Support for multiple transfer modes
  - Basic, for simple transfer scenarios
  - Ping-pong, for continuous data flow to/from peripherals
  - Scatter-gather, from a programmable list of up to 256 arbitrary transfers initiated from a single request
- Dedicated channels for supported peripherals
- One channel each for receive and transmit path for bidirectional peripherals
- Dedicated channel for software-initiated transfers
- Independently configured and operated channels
- Per-channel configurable bus arbitration scheme
- Two levels of priority
- Design optimizations for improved bus access performance between µDMA controller and the processor core
  - µDMA controller access is subordinate to core access
  - RAM striping
  - Peripheral bus segmentation
- Data sizes of 8, 16, and 32 bits
- Source and destination address increment size of byte, half-word, word, or no increment
- Maskable device requests
- Optional software initiated requests for any channel
- Interrupt on transfer completion, with a separate interrupt per channel

■ GPIOs

- 1-33 GPIOs, depending on configuration
- 5-V-tolerant in input configuration
- Two means of port access: either Advanced High-Performance Bus (AHB) with better back-to-back access performance, or the legacy Advanced Peripheral Bus (APB) for backwards-compatibility with existing code
- Fast toggle capable of a change every clock cycle for ports on AHB, every two clock cycles for ports on APB
- Programmable control for GPIO interrupts

        - Interrupt generation masking
        - Edge-triggered on rising, falling, or both
        - Level-sensitive on High or Low values
    - Bit masking in both read and write operations through address lines
    - Can initiate an ADC sample sequence
    - Pins configured as digital inputs are Schmitt-triggered.
    - Programmable control for GPIO pad configuration
        - Weak pull-up or pull-down resistors
        - 2-mA, 4-mA, and 8-mA pad drive for digital communication; up to four pads can be configured with an 18-mA pad drive for high-current applications
        - Slew rate control for the 8-mA drive
        - Open drain enables
        - Digital input enables
- General-Purpose Timers
    - Three General-Purpose Timer Modules (GPTM), each of which provides two 16-bit timers/counters. Each GPTM can be configured to operate independently:
        - As a single 32-bit timer
        - As one 32-bit Real-Time Clock (RTC) to event capture
        - For Pulse Width Modulation (PWM)
        - To trigger analog-to-digital conversions
    - 32-bit Timer modes
        - Programmable one-shot timer
        - Programmable periodic timer
        - Real-Time Clock when using an external 32.768-KHz clock as the input
        - User-enabled stalling when the controller asserts CPU Halt flag during debug
        - ADC event trigger
    - 16-bit Timer modes
        - General-purpose timer function with an 8-bit prescaler (for one-shot and periodic modes only)
        - Programmable one-shot timer

        - Programmable periodic timer
        - User-enabled stalling when the controller asserts CPU Halt flag during debug
        - ADC event trigger
    - 16-bit Input Capture modes
        - Input edge count capture
        - Input edge time capture
    - 16-bit PWM mode
        - Simple PWM mode with software-programmable output inversion of the PWM signal
- ARM FiRM-compliant Watchdog Timer
    - 32-bit down counter with a programmable load register
    - Separate watchdog clock with an enable
    - Programmable interrupt generation logic with interrupt masking
    - Lock register protection from runaway software
    - Reset generation logic with an enable/disable
    - User-enabled stalling when the controller asserts the CPU Halt flag during debug
- ADC
    - Six analog input channels
    - Single-ended and differential-input configurations
    - On-chip internal temperature sensor
    - Sample rate of 500 thousand samples/second
    - Flexible, configurable analog-to-digital conversion
    - Four programmable sample conversion sequences from one to eight entries long, with corresponding conversion result FIFOs
    - Flexible trigger control
        - Controller (software)
        - Timers
        - GPIO
    - Hardware averaging of up to 64 samples for improved accuracy
    - Converter uses an internal 3-V reference

  - Power and ground for the analog circuitry is separate from the digital power and ground

- UART
  - Two fully programmable 16C550-type UARTs with IrDA support
  - Separate 16x8 transmit (TX) and receive (RX) FIFOs to reduce CPU interrupt service loading
  - Programmable baud-rate generator allowing speeds up to 3.125 Mbps
  - Programmable FIFO length, including 1-byte deep operation providing conventional double-buffered interface
  - FIFO trigger levels of 1/8, 1/4, 1/2, 3/4, and 7/8
  - Standard asynchronous communication bits for start, stop, and parity
  - Line-break generation and detection
  - Fully programmable serial interface characteristics
    - 5, 6, 7, or 8 data bits
    - Even, odd, stick, or no-parity bit generation/detection
    - 1 or 2 stop bit generation
  - IrDA serial-IR (SIR) encoder/decoder providing
    - Programmable use of IrDA Serial Infrared (SIR) or UART input/output
    - Support of IrDA SIR encoder/decoder functions for data rates up to 115.2 Kbps half-duplex
    - Support of normal 3/16 and low-power (1.41-2.23 µs) bit durations
    - Programmable internal clock generator enabling division of reference clock by 1 to 256 for low-power mode bit duration
  - Dedicated Direct Memory Access (DMA) transmit and receive channels
- Synchronous Serial Interface (SSI)
  - Master or slave operation
  - Support for Direct Memory Access (DMA)
  - Programmable clock bit rate and prescale
  - Separate transmit and receive FIFOs, 16 bits wide, 8 locations deep
  - Programmable interface operation for Freescale SPI, MICROWIRE, or Texas Instruments synchronous serial interfaces
  - Programmable data frame size from 4 to 16 bits
  - Internal loopback test mode for diagnostic/debug testing

- $I^2C$
  - Two $I^2C$ modules, each with the following features:
  - Devices on the $I^2C$ bus can be designated as either a master or a slave
    - Supports both sending and receiving data as either a master or a slave
    - Supports simultaneous master and slave operation
  - Four $I^2C$ modes
    - Master transmit
    - Master receive
    - Slave transmit
    - Slave receive
  - Two transmission speeds: Standard (100 Kbps) and Fast (400 Kbps)
  - Master and slave interrupt generation
    - Master generates interrupts when a transmit or receive operation completes (or aborts due to an error)
    - Slave generates interrupts when data has been sent or requested by a master
  - Master with arbitration and clock synchronization, multimaster support, and 7-bit addressing mode
- Controller Area Network (CAN)
  - CAN protocol version 2.0 part A/B
  - Bit rates up to 1 Mbps
  - 32 message objects with individual identifier masks
  - Maskable interrupt
  - Disable Automatic Retransmission mode for Time-Triggered CAN (TTCAN) applications
  - Programmable Loopback mode for self-test operation
  - Programmable FIFO mode enables storage of multiple message objects
  - Gluelessly attaches to an external CAN interface through the CANnTX and CANnRX signals
- USB
  - Standards-based
  - USB 2.0 full-speed (12 Mbps) and low-speed (1.5 Mbps) operation
  - USB Device or Host mode

  - Integrated PHY
  - 4 transfer types: Control, Interrupt, Bulk, and Isochronous
  - 8 endpoints
    - 1 dedicated control IN endpoint and 1 dedicated control OUT endpoint
    - 3 configurable IN endpoints and 3 configurable OUT endpoints
  - 2 KB dedicated endpoint memory
    - Direct memory access (DMA)
    - One endpoint may be defined for double-buffered 1023-byte isochronous packet size
- Power
  - On-chip Low Drop-Out (LDO) voltage regulator, with programmable output user-adjustable from 2.25 V to 2.75 V
  - Hibernation module handles the power-up/down 3.3 V sequencing and control for the core digital logic and analog circuits
  - Low-power options on controller: Sleep and Deep-sleep modes
  - Low-power options for peripherals: software controls shutdown of individual peripherals
  - 3.3-V supply brown-out detection and reporting via interrupt or reset
- Flexible Reset Sources
  - Power-on reset (POR)
  - Reset pin assertion
  - Brown-out (BOR) detector alerts to system power drops
  - Software reset
  - Watchdog timer reset
  - Internal low drop-out (LDO) regulator output goes unregulated
- Industrial-range 64-pin RoHS-compliant LQFP package

## 1.2 Target Applications

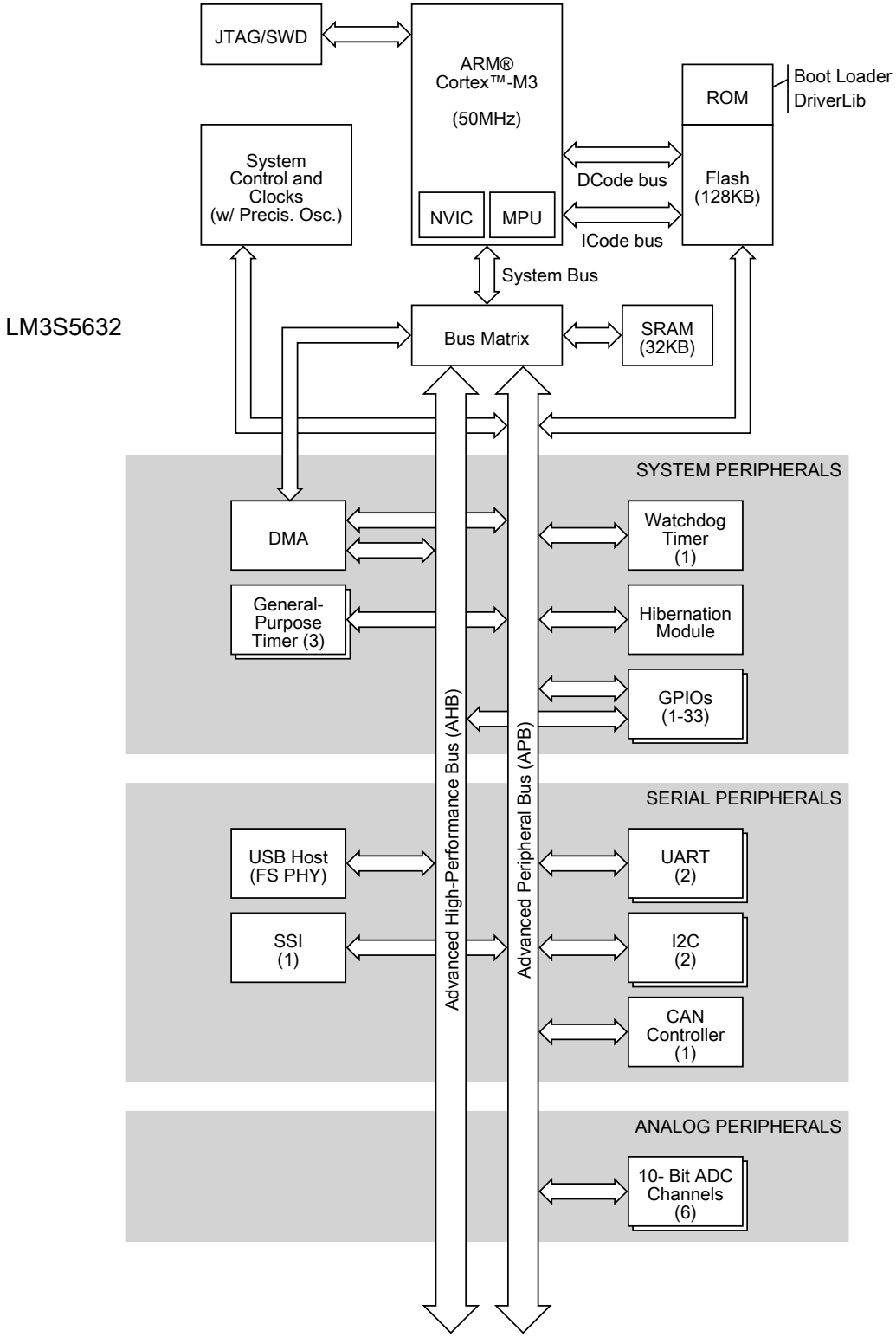
- Remote monitoring
- Electronic point-of-sale (POS) machines
- Test and measurement equipment
- Network appliances and switches

- Factory automation
- HVAC and building control
- Gaming equipment
- Motion control
- Medical instrumentation
- Fire and security
- Power and energy
- Transportation

## 1.3 High-Level Block Diagram

Figure 1-1 on page 46 depicts the features on the Stellaris LM3S5632 microcontroller.

**Figure 1-1. Stellaris LM3S5632 Microcontroller High-Level Block Diagram**



# 1.4 Functional Overview

The following sections provide an overview of the features of the LM3S5632 microcontroller. The page number in parenthesis indicates where that feature is discussed in detail. Ordering and support information can be found in “Ordering and Contact Information” on page 826.

## 1.4.1 ARM Cortex™-M3

### 1.4.1.1 Processor Core (see page 53)

All members of the Stellaris product family, including the LM3S5632 microcontroller, are designed around an ARM Cortex™-M3 processor core. The ARM Cortex-M3 processor provides the core for a high-performance, low-cost platform that meets the needs of minimal memory implementation, reduced pin count, and low-power consumption, while delivering outstanding computational performance and exceptional system response to interrupts.

### 1.4.1.2 Memory Map (see page 72)

A memory map lists the location of instructions and data in memory. The memory map for the LM3S5632 controller can be found in Table 2-4 on page 72. Register addresses are given as a hexadecimal increment, relative to the module's base address as shown in the memory map.

### 1.4.1.3 System Timer (SysTick) (see page 95)

Cortex-M3 includes an integrated system timer, SysTick. SysTick provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism. The counter can be used in several different ways, for example:

- An RTOS tick timer which fires at a programmable rate (for example, 100 Hz) and invokes a SysTick routine.
- A high-speed alarm timer using the system clock.
- A variable rate alarm or signal timer—the duration is range-dependent on the reference clock used and the dynamic range of the counter.
- A simple counter. Software can use this to measure time to completion and time used.
- An internal clock source control based on missing/meeting durations. The COUNTFLAG bit-field in the control and status register can be used to determine if an action completed within a set duration, as part of a dynamic clock management control loop.

### 1.4.1.4 Nested Vectored Interrupt Controller (NVIC) (see page 96)

The LM3S5632 controller includes the ARM Nested Vectored Interrupt Controller (NVIC) on the ARM® Cortex™-M3 core. The NVIC and Cortex-M3 prioritize and handle all exceptions. All exceptions are handled in Handler Mode. The processor state is automatically stored to the stack on an exception, and automatically restored from the stack at the end of the Interrupt Service Routine (ISR). The vector is fetched in parallel to the state saving, which enables efficient interrupt entry. The processor supports tail-chaining, which enables back-to-back interrupts to be performed without the overhead of state saving and restoration. Software can set eight priority levels on 7 exceptions (system handlers) and 28 interrupts.

#### 1.4.1.5 System Control Block (SCB) (see page 98)

The SCB provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

#### 1.4.1.6 Memory Protection Unit (MPU) (see page 98)

The MPU supports the standard ARMv7 Protected Memory System Architecture (PMSA) model. The MPU provides full support for protection regions, overlapping protection regions, access permissions, and exporting memory attributes to the system.

#### 1.4.1.7 Direct Memory Access (see page 290)

The LM3S5632 microcontroller includes a Direct Memory Access (DMA) controller, known as micro-DMA (µDMA). The µDMA controller provides a way to offload data transfer tasks from the Cortex-M3 processor, allowing for more efficient use of the processor and the expanded available bus bandwidth. The µDMA controller can perform transfers between memory and peripherals. It has dedicated channels for each supported peripheral and can be programmed to automatically perform transfers between peripherals and memory as the peripheral is ready to transfer more data. The µDMA controller also supports sophisticated transfer modes such as ping-pong and scatter-gather, which allows the processor to set up a list of transfer tasks for the controller.

### 1.4.2 Motor Control Peripherals

To enhance motor control, the LM3S5632 controller features Pulse Width Modulation (PWM) outputs.

#### 1.4.2.1 PWM

Pulse width modulation (PWM) is a powerful technique for digitally encoding analog signal levels. High-resolution counters are used to generate a square wave, and the duty cycle of the square wave is modulated to encode an analog signal. Typical applications include switching power supplies and motor control.

On the LM3S5632, PWM motion control functionality can be achieved through:

- The motion control features of the general-purpose timers using the CCP pins

***CCP Pins (see page 405)***

The General-Purpose Timer Module's CCP (Capture Compare PWM) pins are software programmable to support a simple PWM mode with a software-programmable output inversion of the PWM signal.

### 1.4.3 Analog Peripherals

To handle analog signals, the LM3S5632 microcontroller offers an Analog-to-Digital Converter (ADC).

#### 1.4.3.1 ADC (see page 457)

An analog-to-digital converter (ADC) is a peripheral that converts a continuous analog voltage to a discrete digital number.

The LM3S5632 ADC module features 10-bit conversion resolution and supports six input channels, plus an internal temperature sensor. Four buffered sample sequences allow rapid sampling of up to eight analog input sources without controller intervention. Each sample sequence provides flexible programming with fully configurable input source, trigger events, interrupt generation, and sequence priority.

### 1.4.4 Serial Communications Peripherals

The LM3S5632 controller supports both asynchronous and synchronous serial communications with:

- Two fully programmable 16C550-type UARTs
- One SSI module
- Two $I^2C$ modules
- One USB 2.0 full-speed controller
- One CAN unit

#### 1.4.4.1 UART (see page 492)

A Universal Asynchronous Receiver/Transmitter (UART) is an integrated circuit used for RS-232C serial communications, containing a transmitter (parallel-to-serial converter) and a receiver (serial-to-parallel converter), each clocked separately.

The LM3S5632 controller includes two fully programmable 16C550-type UARTs that support data transfer speeds up to 3.125 Mbps. (Although similar in functionality to a 16C550 UART, it is not register-compatible.) In addition, each UART is capable of supporting IrDA.

Separate 16x8 transmit (TX) and receive (RX) FIFOs reduce CPU interrupt service loading. The UART can generate individually masked interrupts from the RX, TX, modem status, and error conditions. The module provides a single combined interrupt when any of the interrupts are asserted and are unmasked.

#### 1.4.4.2 SSI (see page 536)

Synchronous Serial Interface (SSI) is a four-wire bi-directional full and low-speed communications interface.

The LM3S5632 controller includes one SSI module that provides the functionality for synchronous serial communications with peripheral devices, and can be configured to use the Freescale SPI, MICROWIRE, or TI synchronous serial interface frame formats. The size of the data frame is also configurable, and can be set between 4 and 16 bits, inclusive.

The SSI module performs serial-to-parallel conversion on data received from a peripheral device, and parallel-to-serial conversion on data transmitted to a peripheral device. The TX and RX paths are buffered with internal FIFOs, allowing up to eight 16-bit values to be stored independently.

The SSI module can be configured as either a master or slave device. As a slave device, the SSI module can also be configured to disable its output, which allows a master device to be coupled with multiple slave devices.

The SSI module also includes a programmable bit rate clock divider and prescaler to generate the output serial clock derived from the SSI module's input clock. Bit rates are generated based on the input clock and the maximum bit rate is determined by the connected peripheral.

#### 1.4.4.3 $I^2C$ (see page 576)

The Inter-Integrated Circuit ($I^2C$) bus provides bi-directional data transfer through a two-wire design (a serial data line SDA and a serial clock line SCL).

The I²C bus interfaces to external I²C devices such as serial memory (RAMs and ROMs), networking devices, LCDs, tone generators, and so on. The I²C bus may also be used for system testing and diagnostic purposes in product development and manufacture.

The LM3S5632 controller includes two I²C modules that provide the ability to communicate to other IC devices over an I²C bus. The I²C bus supports devices that can both transmit and receive (write and read) data.

Devices on the I²C bus can be designated as either a master or a slave. Each I²C module supports both sending and receiving data as either a master or a slave, and also supports the simultaneous operation as both a master and a slave. The four I²C modes are: Master Transmit, Master Receive, Slave Transmit, and Slave Receive.

A Stellaris I²C module can operate at two speeds: Standard (100 Kbps) and Fast (400 Kbps).

Both the I²C master and slave can generate interrupts. The I²C master generates interrupts when a transmit or receive operation completes (or aborts due to an error). The I²C slave generates interrupts when data has been sent or requested by a master.

#### 1.4.4.4 USB (see page 659)

Universal Serial Bus (USB) is a serial bus standard designed to allow peripherals to be connected and disconnected using a standardized interface without rebooting the system.

The LM3S5632 controller supports the USB 2.0 full-speed configuration with Device or USB Host mode. The specified throughput for a USB 2.0 full-speed controller is 12 Mbps.

#### 1.4.4.5 Controller Area Network (see page 613)

Controller Area Network (CAN) is a multicast shared serial-bus standard for connecting electronic control units (ECUs). CAN was specifically designed to be robust in electromagnetically noisy environments and can utilize a differential balanced line like RS-485 or a more robust twisted-pair wire. Originally created for automotive purposes, now it is used in many embedded control applications (for example, industrial or medical). Bit rates up to 1Mb/s are possible at network lengths below 40 meters. Decreased bit rates allow longer network distances (for example, 125 Kb/s at 500m).

A transmitter sends a message to all CAN nodes (broadcasting). Each node decides on the basis of the identifier received whether it should process the message. The identifier also determines the priority that the message enjoys in competition for bus access. Each CAN message can transmit from 0 to 8 bytes of user information. The LM3S5632 includes one CAN unit.

### 1.4.5 System Peripherals

#### 1.4.5.1 Programmable GPIOs (see page 351)

General-purpose input/output (GPIO) pins offer flexibility for a variety of connections.

The Stellaris GPIO module is comprised of five physical GPIO blocks, each corresponding to an individual GPIO port. The GPIO module is FiRM-compliant (compliant to the ARM Foundation IP for Real-Time Microcontrollers specification) and supports 1-33 programmable input/output pins. The number of GPIOs available depends on the peripherals being used (see “Signal Tables” on page 754 for the signals available to each GPIO pin).

The GPIO module features programmable interrupt generation as either edge-triggered or level-sensitive on all pins, programmable control for GPIO pad configuration, and bit masking in both read and write operations through address lines. Pins configured as digital inputs are Schmitt-triggered.

#### 1.4.5.2 Three Programmable Timers (see page 399)

Programmable timers can be used to count or time external events that drive the Timer input pins.

The Stellaris General-Purpose Timer Module (GPTM) contains three GPTM blocks. Each GPTM block provides two 16-bit timers/counters that can be configured to operate independently as timers or event counters, or configured to operate as one 32-bit timer or one 32-bit Real-Time Clock (RTC). Timers can also be used to trigger analog-to-digital (ADC) conversions.

When configured in 32-bit mode, a timer can run as a Real-Time Clock (RTC), one-shot timer or periodic timer. When in 16-bit mode, a timer can run as a one-shot timer or periodic timer, and can extend its precision by using an 8-bit prescaler. A 16-bit timer can also be configured for event capture or Pulse Width Modulation (PWM) generation.

#### 1.4.5.3 Watchdog Timer (see page 433)

A watchdog timer can generate an interrupt or a reset when a time-out value is reached. The watchdog timer is used to regain control when a system has failed due to a software error or to the failure of an external device to respond in the expected way.

The Stellaris Watchdog Timer module consists of a 32-bit down counter, a programmable load register, interrupt generation logic, and a locking register.

The Watchdog Timer can be configured to generate an interrupt to the controller on its first time-out, and to generate a reset signal on its second time-out. Once the Watchdog Timer has been configured, the lock register can be written to prevent the timer configuration from being inadvertently altered.

### 1.4.6 Memory Peripherals

The LM3S5632 controller offers both single-cycle SRAM and single-cycle Flash memory.

#### 1.4.6.1 SRAM (see page 260)

The LM3S5632 static random access memory (SRAM) controller supports 32 KB SRAM. The internal SRAM of the Stellaris devices starts at base address 0x2000.0000 of the device memory map. To reduce the number of time-consuming read-modify-write (RMW) operations, ARM has introduced *bit-banding* technology in the new Cortex-M3 processor. With a bit-band-enabled processor, certain regions in the memory map (SRAM and peripheral space) can use address aliases to access individual bits in a single, atomic operation.

#### 1.4.6.2 Flash (see page 261)

The LM3S5632 Flash controller supports 128 KB of flash memory. The flash is organized as a set of 1-KB blocks that can be individually erased. Erasing a block causes the entire contents of the block to be reset to all 1s. These blocks are paired into a set of 2-KB blocks that can be individually protected. The blocks can be marked as read-only or execute-only, providing different levels of code protection. Read-only blocks cannot be erased or programmed, protecting the contents of those blocks from being modified. Execute-only blocks cannot be erased or programmed, and can only be read by the controller instruction fetch mechanism, protecting the contents of those blocks from being read by either the controller or by a debugger.

#### 1.4.6.3 ROM (see page 786)

The LM3S5632 microcontroller ships with the Stellaris family Peripheral Driver Library conveniently preprogrammed in read-only memory (ROM). The Stellaris Peripheral Driver Library is a royalty-free software library for controlling on-chip peripherals, and includes a boot-loader capability. The library performs both peripheral initialization and peripheral control functions, with a choice of polled or interrupt-driven peripheral support, and takes full advantage of the stellar interrupt performance of

the ARM® Cortex™-M3 core. No special pragmas or custom assembly code prologue/epilogue functions are required. For applications that require in-field programmability, the royalty-free Stellaris boot loader included in the Stellaris Peripheral Driver Library can act as an application loader and support in-field firmware updates.

## 1.4.7 Additional Features

### 1.4.7.1 JTAG TAP Controller (see page 159)

The Joint Test Action Group (JTAG) port is an IEEE standard that defines a Test Access Port and Boundary Scan Architecture for digital integrated circuits and provides a standardized serial interface for controlling the associated test logic. The TAP, Instruction Register (IR), and Data Registers (DR) can be used to test the interconnections of assembled printed circuit boards and obtain manufacturing information on the components. The JTAG Port also provides a means of accessing and controlling design-for-test features such as I/O pin observation and control, scan testing, and debugging.

The JTAG port is composed of the standard four pins: TCK, TMS, TDI, and TDO. Data is transmitted serially into the controller on TDI and out of the controller on TDO. The interpretation of this data is dependent on the current state of the TAP controller. For detailed information on the operation of the JTAG port and TAP controller, please refer to the *IEEE Standard 1149.1-Test Access Port and Boundary-Scan Architecture*.

The Stellaris JTAG controller works with the ARM JTAG controller built into the Cortex-M3 core. This is implemented by multiplexing the TDO outputs from both JTAG controllers. ARM JTAG instructions select the ARM TDO output while Stellaris JTAG instructions select the Stellaris TDO outputs. The multiplexer is controlled by the Stellaris JTAG controller, which has comprehensive programming for the ARM, Stellaris, and unimplemented JTAG instructions.

### 1.4.7.2 System Control and Clocks (see page 171)

System control determines the overall operation of the device. It provides information about the device, controls the clocking of the device and individual peripherals, and handles reset detection and reporting.

### 1.4.7.3 Hibernation Module (see page 238)

The Hibernation module provides logic to switch power off to the main processor and peripherals, and to wake on external or time-based events. The Hibernation module includes power-sequencing logic, a real-time clock with a pair of match registers, low-battery detection circuitry, and interrupt signalling to the processor. It also includes 64 32-bit words of non-volatile memory that can be used for saving state during hibernation.

## 1.4.8 Hardware Details

Details on the pins and package can be found in the following sections:

- “Pin Diagram” on page 753
- “Signal Tables” on page 754
- “Operating Characteristics” on page 765
- “Electrical Characteristics” on page 766
- “Package Information” on page 828

# 2 The Cortex-M3 Processor

The ARM® Cortex™-M3 processor provides a high-performance, low-cost platform that meets the system requirements of minimal memory implementation, reduced pin count, and low power consumption, while delivering outstanding computational performance and exceptional system response to interrupts. Features include:

- Compact core.
- Thumb-2 instruction set, delivering the high-performance expected of an ARM core in the memory size usually associated with 8- and 16-bit devices; typically in the range of a few kilobytes of memory for microcontroller class applications.
- Rapid application execution through Harvard architecture characterized by separate buses for instruction and data.
- Exceptional interrupt handling, by implementing the register manipulations required for handling an interrupt in hardware.
- Deterministic, fast interrupt processing: always 12 cycles, or just 6 cycles with tail-chaining
- External non-maskable interrupt signal (NMI) available for immediate execution of NMI handler for safety critical applications.
- Memory protection unit (MPU) to provide a privileged mode of operation for complex applications.
- Migration from the ARM7™ processor family for better performance and power efficiency.
- Full-featured debug solution
  - Serial Wire JTAG Debug Port (SWJ-DP)
  - Flash Patch and Breakpoint (FPB) unit for implementing breakpoints
  - Data Watchpoint and Trigger (DWT) unit for implementing watchpoints, trigger resources, and system profiling
  - Instrumentation Trace Macrocell (ITM) for support of printf style debugging
  - Trace Port Interface Unit (TPIU) for bridging to a Trace Port Analyzer
- Optimized for single-cycle flash usage
- Three sleep modes with clock gating for low power
- Single-cycle multiply instruction and hardware divide
- Atomic operations
- ARM Thumb2 mixed 16-/32-bit instruction set
- 1.25 DMIPS/MHz

The Stellaris® family of microcontrollers builds on this core to bring high-performance 32-bit computing to cost-sensitive embedded microcontroller applications, such as factory automation and control, industrial control power devices, building and home automation, and stepper motor control.

This chapter provides information on the Stellaris implementation of the Cortex-M3 processor, including the programming model, the memory model, the exception model, fault handling, and power management.

For technical details on the instruction set, see the *Cortex™-M3/M4 Instruction Set Technical User's Manual*.

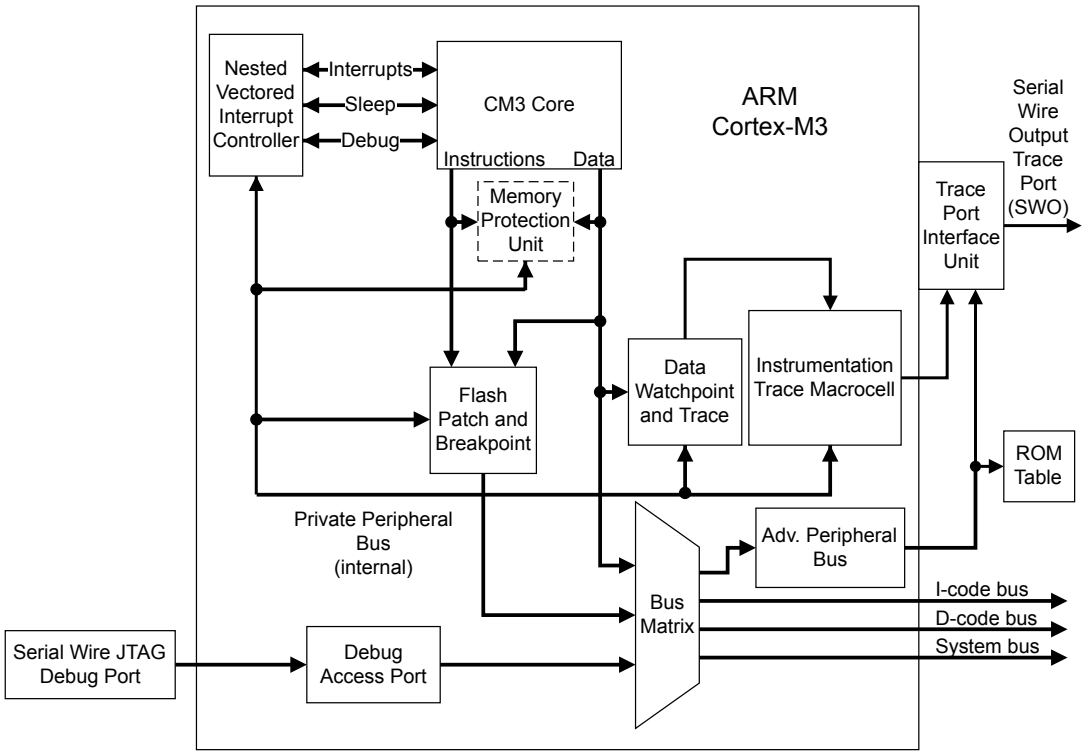
## 2.1 Block Diagram

The Cortex-M3 processor is built on a high-performance processor core, with a 3-stage pipeline Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design, providing high-end processing hardware including a range of single-cycle and SIMD multiplication and multiply-with-accumulate capabilities, saturating arithmetic and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M3 processor implements tightly coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M3 processor implements a version of the Thumb® instruction set based on Thumb-2 technology, ensuring high code density and reduced program memory requirements. The Cortex-M3 instruction set provides the exceptional performance expected of a modern 32-bit architecture, with the high code density of 8-bit and 16-bit microcontrollers.

The Cortex-M3 processor closely integrates a nested interrupt controller (NVIC), to deliver industry-leading interrupt performance. The Stellaris NVIC includes a non-maskable interrupt (NMI) and provides eight interrupt priority levels. The tight integration of the processor core and NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing interrupt latency. The hardware stacking of registers and the ability to suspend load-multiple and store-multiple operations further reduce interrupt latency. Interrupt handlers do not require any assembler stubs which removes code overhead from the ISRs. Tail-chaining optimization also significantly reduces the overhead when switching from one ISR to another. To optimize low-power designs, the NVIC integrates with the sleep modes, including Deep-sleep mode, which enables the entire device to be rapidly powered down.

**Figure 2-1. CPU Block Diagram**




## 2.2 Overview

### 2.2.1 System-Level Interface

The Cortex-M3 processor provides multiple interfaces using AMBA® technology to provide high-speed, low-latency memory accesses. The core supports unaligned data accesses and implements atomic bit manipulation that enables faster peripheral controls, system spinlocks, and thread-safe Boolean data handling.

The Cortex-M3 processor has a memory protection unit (MPU) that provides fine-grain memory control, enabling applications to implement security privilege levels and separate code, data and stack on a task-by-task basis.

### 2.2.2 Integrated Configurable Debug

The Cortex-M3 processor implements a complete hardware debug solution, providing high system visibility of the processor and memory through either a traditional JTAG port or a 2-pin Serial Wire Debug (SWD) port that is ideal for microcontrollers and other small package devices. The Stellaris implementation replaces the ARM SW-DP and JTAG-DP with the ARM CoreSight™-compliant Serial Wire JTAG Debug Port (SWJ-DP) interface. The SWJ-DP interface combines the SWD and JTAG debug ports into one module. See the *ARM® Debug Interface V5 Architecture Specification* for details on SWJ-DP.

For system trace, the processor integrates an Instrumentation Trace Macrocell (ITM) alongside data watchpoints and a profiling unit. To enable simple and cost-effective profiling of the system trace events, a Serial Wire Viewer (SWV) can export a stream of software-generated messages, data trace, and profiling information through a single pin.

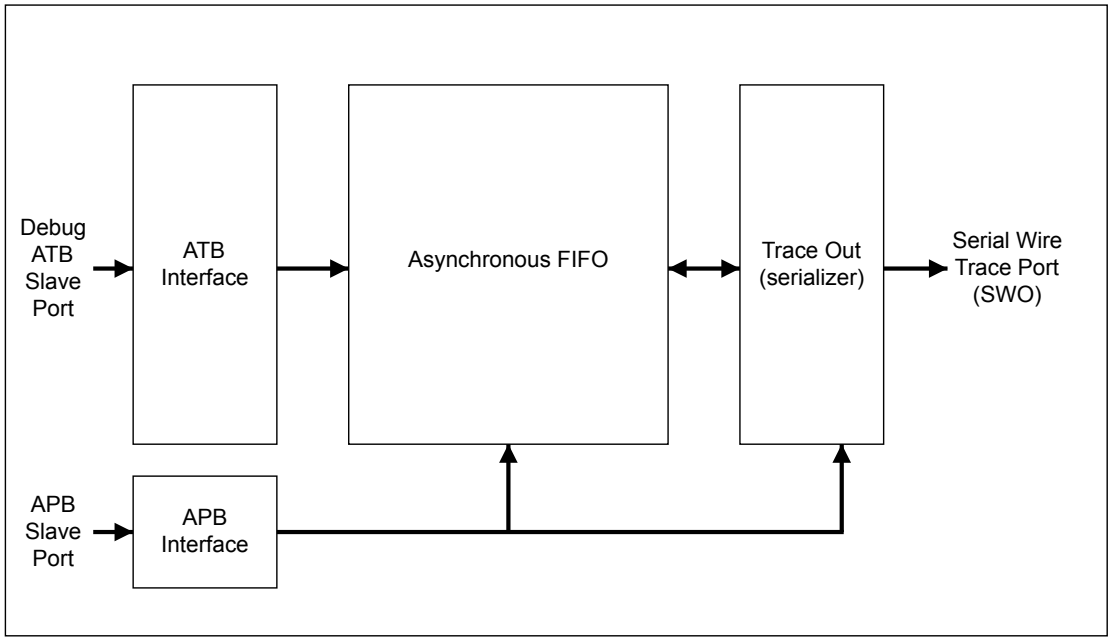
The Flash Patch and Breakpoint Unit (FPB) provides up to eight hardware breakpoint comparators that debuggers can use. The comparators in the FPB also provide remap functions of up to eight words in the program code in the CODE memory region. This enables applications stored in a read-only area of Flash memory to be patched in another area of on-chip SRAM or Flash memory. If a patch is required, the application programs the FPB to remap a number of addresses. When those addresses are accessed, the accesses are redirected to a remap table specified in the FPB configuration.

For more information on the Cortex-M3 debug capabilities, see the*ARM® Debug Interface V5 Architecture Specification*.

### 2.2.3 Trace Port Interface Unit (TPIU)

The TPIU acts as a bridge between the Cortex-M3 trace data from the ITM, and an off-chip Trace Port Analyzer, as shown in Figure 2-2 on page 56.

**Figure 2-2. TPIU Block Diagram**




### 2.2.4 Cortex-M3 System Component Details

The Cortex-M3 includes the following system components:

- SysTick

  A 24-bit count-down timer that can be used as a Real-Time Operating System (RTOS) tick timer or as a simple counter (see “System Timer (SysTick)” on page 95).

- Nested Vectored Interrupt Controller (NVIC)

  An embedded interrupt controller that supports low latency interrupt processing (see “Nested Vectored Interrupt Controller (NVIC)” on page 96).

- System Control Block (SCB)

The programming model interface to the processor. The SCB provides system implementation information and system control, including configuration, control, and reporting of system exceptions (see “System Control Block (SCB)” on page 98).

- Memory Protection Unit (MPU)

  Improves system reliability by defining the memory attributes for different memory regions. The MPU provides up to eight different regions and an optional predefined background region (see “Memory Protection Unit (MPU)” on page 98).

## 2.3 Programming Model

This section describes the Cortex-M3 programming model. In addition to the individual core register descriptions, information about the processor modes and privilege levels for software execution and stacks is included.

### 2.3.1 Processor Mode and Privilege Levels for Software Execution

The Cortex-M3 has two modes of operation:

- Thread mode

  Used to execute application software. The processor enters Thread mode when it comes out of reset.

- Handler mode

  Used to handle exceptions. When the processor has finished exception processing, it returns to Thread mode.

In addition, the Cortex-M3 has two privilege levels:

- Unprivileged

  In this mode, software has the following restrictions:

  – Limited access to the MSR and MRS instructions and no use of the CPS instruction

  – No access to the system timer, NVIC, or system control block

  – Possibly restricted access to memory or peripherals

- Privileged

  In this mode, software can use all the instructions and has access to all resources.

In Thread mode, the **CONTROL** register (see page 71) controls whether software execution is privileged or unprivileged. In Handler mode, software execution is always privileged.

Only privileged software can write to the **CONTROL** register to change the privilege level for software execution in Thread mode. Unprivileged software can use the SVC instruction to make a supervisor call to transfer control to privileged software.

### 2.3.2 Stacks

The processor uses a full descending stack, meaning that the stack pointer indicates the last stacked item on the memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location. The processor implements two stacks:

the main stack and the process stack, with a pointer for each held in independent registers (see the **SP** register on page 61).

In Thread mode, the **CONTROL** register (see page 71) controls whether the processor uses the main stack or the process stack. In Handler mode, the processor always uses the main stack. The options for processor operations are shown in Table 2-1 on page 58.

**Table 2-1. Summary of Processor Mode, Privilege Level, and Stack Use**

| Processor Mode | Use | Privilege Level | Stack Used |
|---|---|---|---|
| Thread | Applications | Privileged or unprivileged [a] | Main stack or process stack [a] |
| Handler | Exception handlers | Always privileged | Main stack |

a. See **CONTROL** (page 71).

## 2.3.3 Register Map

Figure 2-3 on page 58 shows the Cortex-M3 register set. Table 2-2 on page 59 lists the Core registers. The core registers are not memory mapped and are accessed by register name, so the base address is n/a (not applicable) and there is no offset.

**Figure 2-3. Cortex-M3 Register Set**

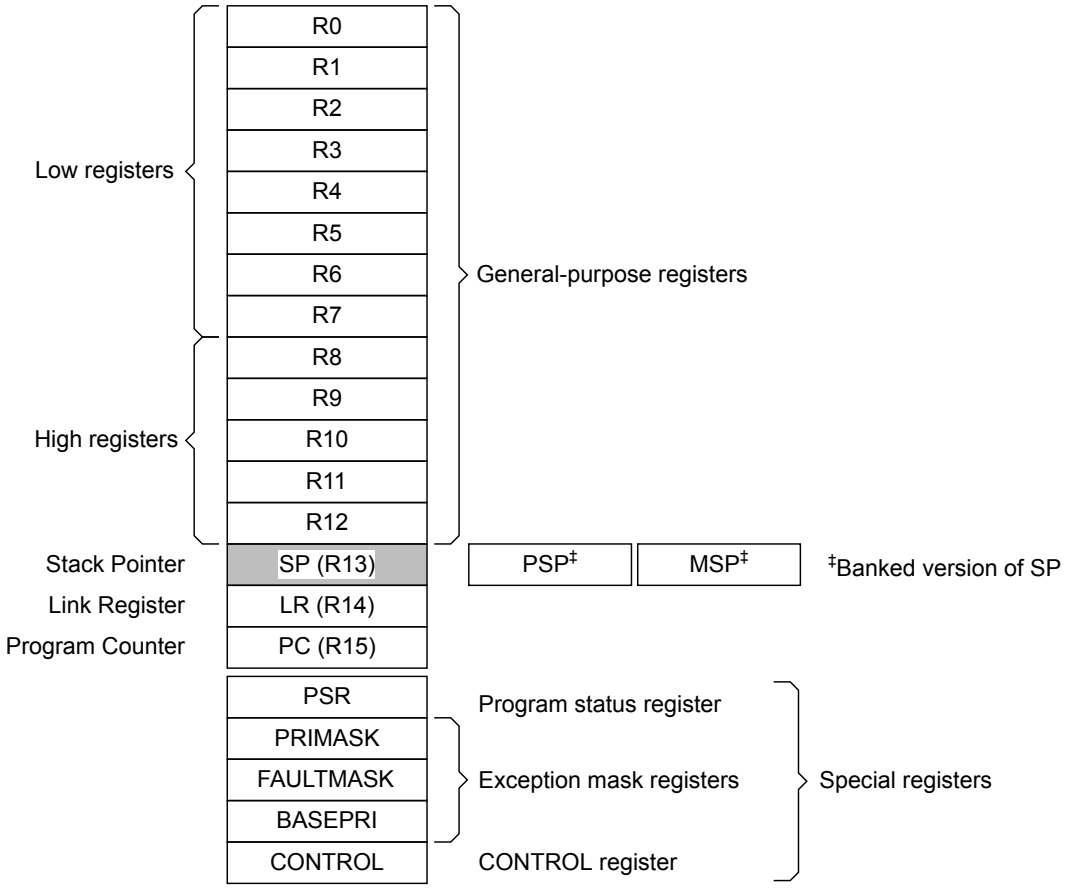


**Table 2-2. Processor Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| - | R0 | R/W | - | Cortex General-Purpose Register 0 | 60 |
| - | R1 | R/W | - | Cortex General-Purpose Register 1 | 60 |
| - | R2 | R/W | - | Cortex General-Purpose Register 2 | 60 |
| - | R3 | R/W | - | Cortex General-Purpose Register 3 | 60 |
| - | R4 | R/W | - | Cortex General-Purpose Register 4 | 60 |
| - | R5 | R/W | - | Cortex General-Purpose Register 5 | 60 |
| - | R6 | R/W | - | Cortex General-Purpose Register 6 | 60 |
| - | R7 | R/W | - | Cortex General-Purpose Register 7 | 60 |
| - | R8 | R/W | - | Cortex General-Purpose Register 8 | 60 |
| - | R9 | R/W | - | Cortex General-Purpose Register 9 | 60 |
| - | R10 | R/W | - | Cortex General-Purpose Register 10 | 60 |
| - | R11 | R/W | - | Cortex General-Purpose Register 11 | 60 |
| - | R12 | R/W | - | Cortex General-Purpose Register 12 | 60 |
| - | SP | R/W | - | Stack Pointer | 61 |
| - | LR | R/W | 0xFFFF.FFFF | Link Register | 62 |
| - | PC | R/W | - | Program Counter | 63 |
| - | PSR | R/W | 0x0100.0000 | Program Status Register | 64 |
| - | PRIMASK | R/W | 0x0000.0000 | Priority Mask Register | 68 |
| - | FAULTMASK | R/W | 0x0000.0000 | Fault Mask Register | 69 |
| - | BASEPRI | R/W | 0x0000.0000 | Base Priority Mask Register | 70 |
| - | CONTROL | R/W | 0x0000.0000 | Control Register | 71 |

### 2.3.4 Register Descriptions

This section lists and describes the Cortex-M3 registers, in the order shown in Figure 2-3 on page 58. The core registers are not memory mapped and are accessed by register name rather than offset.

**Note:** The register type shown in the register descriptions refers to type during program execution in Thread mode and Handler mode. Debug access can differ.

### Register 1: Cortex General-Purpose Register 0 (R0)

### Register 2: Cortex General-Purpose Register 1 (R1)

### Register 3: Cortex General-Purpose Register 2 (R2)

### Register 4: Cortex General-Purpose Register 3 (R3)

### Register 5: Cortex General-Purpose Register 4 (R4)

### Register 6: Cortex General-Purpose Register 5 (R5)

### Register 7: Cortex General-Purpose Register 6 (R6)

### Register 8: Cortex General-Purpose Register 7 (R7)

### Register 9: Cortex General-Purpose Register 8 (R8)

### Register 10: Cortex General-Purpose Register 9 (R9)

### Register 11: Cortex General-Purpose Register 10 (R10)

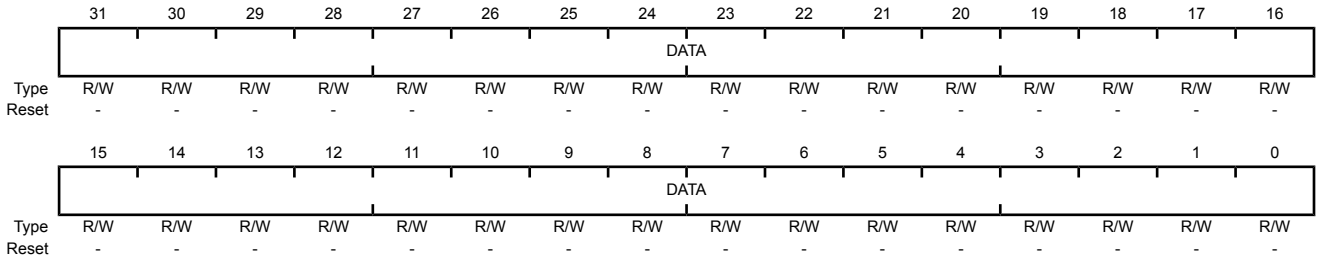
### Register 12: Cortex General-Purpose Register 11 (R11)

### Register 13: Cortex General-Purpose Register 12 (R12)

The **Rn** registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.

Cortex General-Purpose Register 0 (R0)

Type R/W, reset -



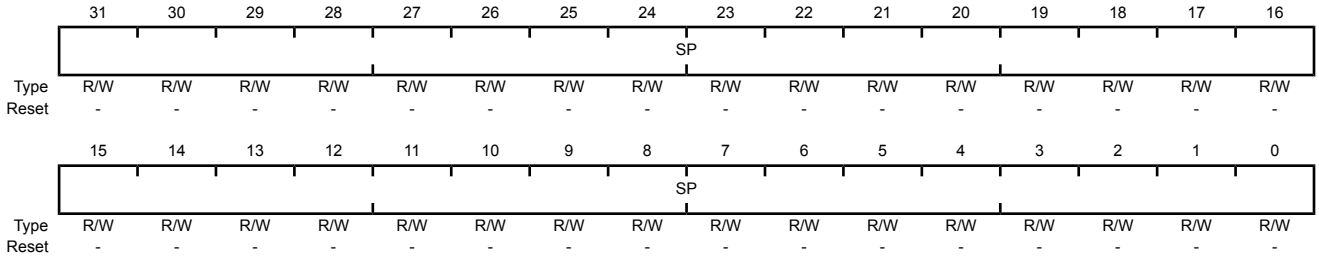

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R/W | - | Register data. |

### Register 14: Stack Pointer (SP)

The **Stack Pointer (SP)** is register R13. In Thread mode, the function of this register changes depending on the ASP bit in the **Control Register (CONTROL)** register. When the ASP bit is clear, this register is the **Main Stack Pointer (MSP)**. When the ASP bit is set, this register is the **Process Stack Pointer (PSP)**. On reset, the ASP bit is clear, and the processor loads the **MSP** with the value from address 0x0000.0000. The **MSP** can only be accessed in privileged mode; the **PSP** can be accessed in either privileged or unprivileged mode.

Stack Pointer (SP)

Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SP | R/W | - | This field is the address of the stack pointer. |

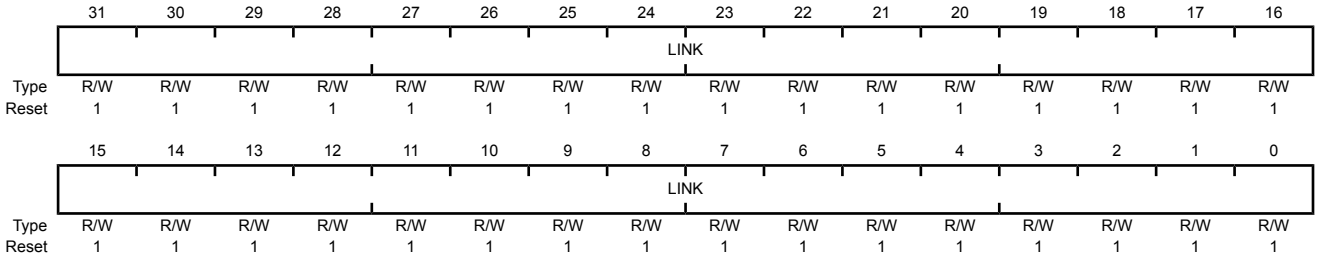
### Register 15: Link Register (LR)

The **Link Register (LR)** is register R14, and it stores the return information for subroutines, function calls, and exceptions. **LR** can be accessed from either privileged or unprivileged mode.

EXC_RETURN is loaded into **LR** on exception entry. See Table 2-10 on page 88 for the values and description.

Link Register (LR)

Type R/W, reset 0xFFFF.FFFF



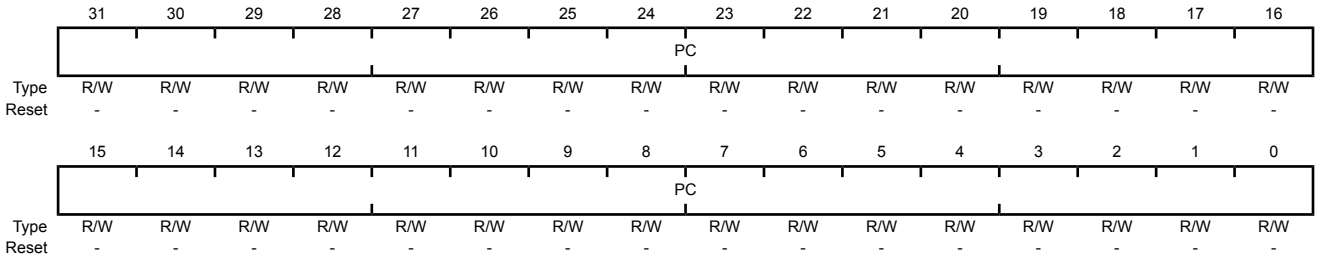

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | LINK | R/W | 0xFFFF.FFFF | This field is the return address. |

### Register 16: Program Counter (PC)

The **Program Counter (PC)** is register R15, and it contains the current program address. On reset, the processor loads the **PC** with the value of the reset vector, which is at address 0x0000.0004. Bit 0 of the reset vector is loaded into the THUMB bit of the **EPSR** at reset and must be 1. The **PC** register can be accessed in either privileged or unprivileged mode.

Program Counter (PC)

Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | PC | R/W | - | This field is the current program address. |

### Register 17: Program Status Register (PSR)

**Note:** This register is also referred to as **xPSR**.

The **Program Status Register (PSR)** has three functions, and the register bits are assigned to the different functions:

- **Application Program Status Register (APSR)**, bits 31:27,
- **Execution Program Status Register (EPSR)**, bits 26:24, 15:10
- **Interrupt Program Status Register (IPSR)**, bits 6:0

The **PSR**, **IPSR**, and **EPSR** registers can only be accessed in privileged mode; the **APSR** register can be accessed in either privileged or unprivileged mode.

**APSR** contains the current state of the condition flags from previous instruction executions.

**EPSR** contains the Thumb state bit and the execution state bits for the If-Then (`IT`) instruction or the Interruptible-Continuable Instruction (`ICI`) field for an interrupted load multiple or store multiple instruction. Attempts to read the **EPSR** directly through application software using the `MSR` instruction always return zero. Attempts to write the **EPSR** using the `MSR` instruction in application software are always ignored. Fault handlers can examine the **EPSR** value in the stacked **PSR** to determine the operation that faulted (see “Exception Entry and Return” on page 86).

**IPSR** contains the exception type number of the current Interrupt Service Routine (ISR).

These registers can be accessed individually or as a combination of any two or all three registers, using the register name as an argument to the `MSR` or `MRS` instructions. For example, all of the registers can be read using **PSR** with the `MRS` instruction, or **APSR** only can be written to using **APSR** with the `MSR` instruction. page 64 shows the possible register combinations for the **PSR**. See the `MRS` and `MSR` instruction descriptions in the *Cortex™-M3/M4 Instruction Set Technical User's Manual* for more information about how to access the program status registers.

**Table 2-3. PSR Register Combinations**

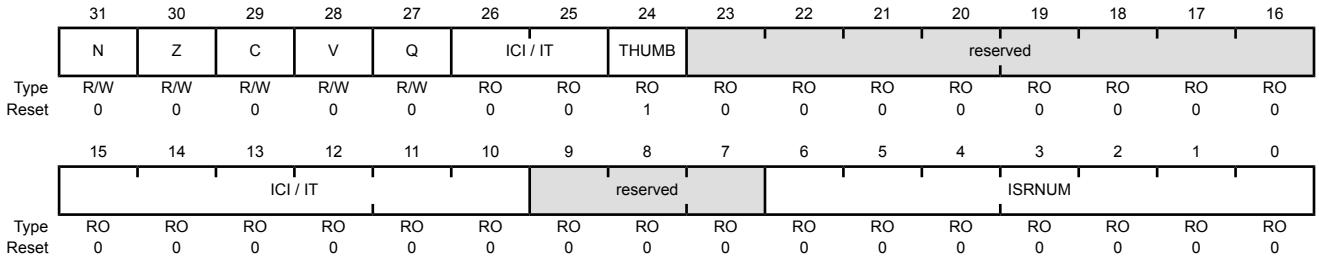
| Register | Type | Combination |
|---|---|---|
| **PSR** | R/W[a, b] | **APSR**, **EPSR**, and **IPSR** |
| **IEPSR** | RO | **EPSR** and **IPSR** |
| **IAPSR** | R/W[a] | **APSR** and **IPSR** |
| **EAPSR** | R/W[b] | **APSR** and **EPSR** |

a. The processor ignores writes to the **IPSR** bits.

b. Reads of the **EPSR** bits return zero, and the processor ignores writes to these bits.

Program Status Register (PSR)

Type R/W, reset 0x0100.0000



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | N | R/W | 0 | **APSR** Negative or Less Flag<br><br>Value Description<br>1 The previous operation result was negative or less than.<br>0 The previous operation result was positive, zero, greater than, or equal.<br><br>The value of this bit is only meaningful when accessing **PSR** or **APSR**. |
| 30 | Z | R/W | 0 | **APSR** Zero Flag<br><br>Value Description<br>1 The previous operation result was zero.<br>0 The previous operation result was non-zero.<br><br>The value of this bit is only meaningful when accessing **PSR** or **APSR**. |
| 29 | C | R/W | 0 | **APSR** Carry or Borrow Flag<br><br>Value Description<br>1 The previous add operation resulted in a carry bit or the previous subtract operation did not result in a borrow bit.<br>0 The previous add operation did not result in a carry bit or the previous subtract operation resulted in a borrow bit.<br><br>The value of this bit is only meaningful when accessing **PSR** or **APSR**. |
| 28 | V | R/W | 0 | **APSR** Overflow Flag<br><br>Value Description<br>1 The previous operation resulted in an overflow.<br>0 The previous operation did not result in an overflow.<br><br>The value of this bit is only meaningful when accessing **PSR** or **APSR**. |
| 27 | Q | R/W | 0 | **APSR** DSP Overflow and Saturation Flag<br><br>Value Description<br>1 DSP Overflow or saturation has occurred.<br>0 DSP overflow or saturation has not occurred since reset or since the bit was last cleared.<br><br>The value of this bit is only meaningful when accessing **PSR** or **APSR**.<br>This bit is cleared by software using an MRS instruction. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 26:25 | ICI / IT | RO | 0x0 | **EPSR** ICI / IT status<br>These bits, along with bits 15:10, contain the Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instruction or the execution state bits of the IT instruction.<br>When **EPSR** holds the ICI execution state, bits 26:25 are zero.<br>The If-Then block contains up to four instructions following an IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See the *Cortex™-M3/M4 Instruction Set Technical User's Manual* for more information.<br>The value of this field is only meaningful when accessing **PSR** or **EPSR**. |
| 24 | THUMB | RO | 1 | **EPSR** Thumb State<br>This bit indicates the Thumb state and should always be set.<br>The following can clear the THUMB bit:<br>■ The BLX, BX and POP{PC} instructions<br>■ Restoration from the stacked **xPSR** value on an exception return<br>■ Bit 0 of the vector value on an exception entry or reset<br>Attempting to execute instructions when this bit is clear results in a fault or lockup. See “Lockup” on page 90 for more information.<br>The value of this bit is only meaningful when accessing **PSR** or **EPSR**. |
| 23:16 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:10 | ICI / IT | RO | 0x0 | **EPSR** ICI / IT status<br>These bits, along with bits 26:25, contain the Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instruction or the execution state bits of the IT instruction.<br>When an interrupt occurs during the execution of an LDM, STM, PUSH or POP instruction, the processor stops the load multiple or store multiple instruction operation temporarily and stores the next register operand in the multiple operation to bits 15:12. After servicing the interrupt, the processor returns to the register pointed to by bits 15:12 and resumes execution of the multiple load or store instruction. When **EPSR** holds the ICI execution state, bits 11:10 are zero.<br>The If-Then block contains up to four instructions following a 16-bit IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See the *Cortex™-M3/M4 Instruction Set Technical User's Manual* for more information.<br>The value of this field is only meaningful when accessing **PSR** or **EPSR**. |
| 9:7 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

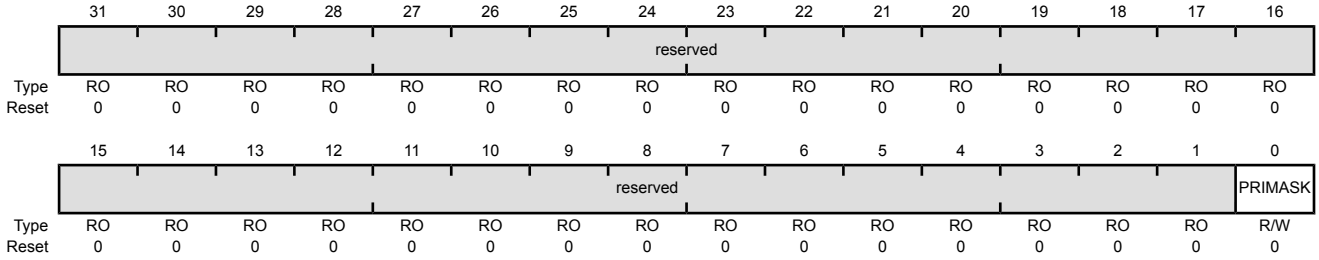
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6:0 | ISRNUM | RO | 0x00 | **IPSR** ISR Number<br>This field contains the exception type number of the current Interrupt Service Routine (ISR).<br><br>Value — Description<br>0x00 — Thread mode<br>0x01 — Reserved<br>0x02 — NMI<br>0x03 — Hard fault<br>0x04 — Memory management fault<br>0x05 — Bus fault<br>0x06 — Usage fault<br>0x07-0x0A — Reserved<br>0x0B — SVCall<br>0x0C — Reserved for Debug<br>0x0D — Reserved<br>0x0E — PendSV<br>0x0F — SysTick<br>0x10 — Interrupt Vector 0<br>0x11 — Interrupt Vector 1<br>... — ...<br>0x3F — Interrupt Vector 47<br>0x40-0x7F — Reserved<br><br>See “Exception Types” on page 81 for more information.<br>The value of this field is only meaningful when accessing **PSR** or **IPSR**. |

### Register 18: Priority Mask Register (PRIMASK)

The **PRIMASK** register prevents activation of all exceptions with programmable priority. Reset, non-maskable interrupt (NMI), and hard fault are the only exceptions with fixed priority. Exceptions should be disabled when they might impact the timing of critical tasks. This register is only accessible in privileged mode. The MSR and MRS instructions are used to access the **PRIMASK** register, and the CPS instruction may be used to change the value of the **PRIMASK** register. See the *Cortex™-M3/M4 Instruction Set Technical User's Manual* for more information on these instructions. For more information on exception priority levels, see “Exception Types” on page 81.

Priority Mask Register (PRIMASK)

Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | PRIMASK | R/W | 0 | Priority Mask |

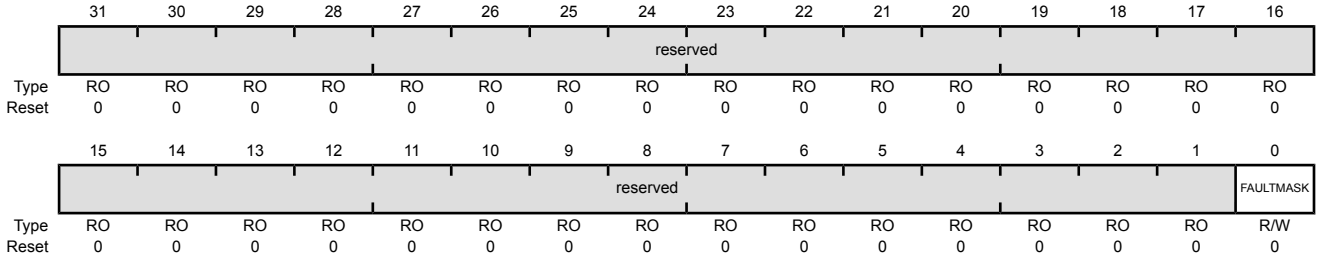
| Value | Description |
|---|---|
| 1 | Prevents the activation of all exceptions with configurable priority. |
| 0 | No effect. |

### Register 19: Fault Mask Register (FAULTMASK)

The **FAULTMASK** register prevents activation of all exceptions except for the Non-Maskable Interrupt (NMI). Exceptions should be disabled when they might impact the timing of critical tasks. This register is only accessible in privileged mode. The MSR and MRS instructions are used to access the **FAULTMASK** register, and the CPS instruction may be used to change the value of the **FAULTMASK** register. See the *Cortex™-M3/M4 Instruction Set Technical User's Manual* for more information on these instructions. For more information on exception priority levels, see “Exception Types” on page 81.

Fault Mask Register (FAULTMASK)

Type R/W, reset 0x0000.0000



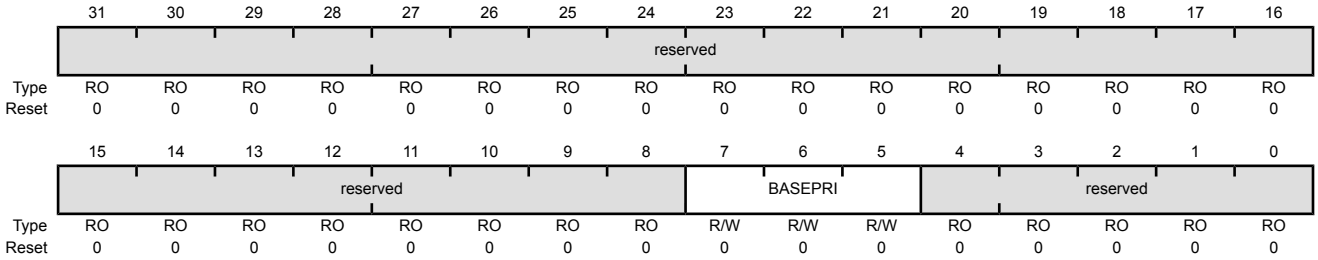

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | FAULTMASK | R/W | 0 | Fault Mask<br><br>Value Description<br>1 Prevents the activation of all exceptions except for NMI.<br>0 No effect.<br><br>The processor clears the FAULTMASK bit on exit from any exception handler except the NMI handler. |

### Register 20: Base Priority Mask Register (BASEPRI)

The **BASEPRI** register defines the minimum priority for exception processing. When **BASEPRI** is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the **BASEPRI** value. Exceptions should be disabled when they might impact the timing of critical tasks. This register is only accessible in privileged mode. For more information on exception priority levels, see “Exception Types” on page 81.

Base Priority Mask Register (BASEPRI)

Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:5 | BASEPRI | R/W | 0x0 | Base Priority<br>Any exception that has a programmable priority level with the same or lower priority as the value of this field is masked. The **PRIMASK** register can be used to mask all exceptions with programmable priority levels. Higher priority exceptions have lower priority levels.<br><br>Value Description<br>0x0 All exceptions are unmasked.<br>0x1 All exceptions with priority level 1-7 are masked.<br>0x2 All exceptions with priority level 2-7 are masked.<br>0x3 All exceptions with priority level 3-7 are masked.<br>0x4 All exceptions with priority level 4-7 are masked.<br>0x5 All exceptions with priority level 5-7 are masked.<br>0x6 All exceptions with priority level 6-7 are masked.<br>0x7 All exceptions with priority level 7 are masked. |
| 4:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

### Register 21: Control Register (CONTROL)

The **CONTROL** register controls the stack used and the privilege level for software execution when the processor is in Thread mode. This register is only accessible in privileged mode.

Handler mode always uses **MSP**, so the processor ignores explicit writes to the ASP bit of the **CONTROL** register when in Handler mode. The exception entry and return mechanisms automatically update the **CONTROL** register based on the EXC_RETURN value (see Table 2-10 on page 88). In an OS environment, threads running in Thread mode should use the process stack and the kernel and exception handlers should use the main stack. By default, Thread mode uses **MSP**. To switch the stack pointer used in Thread mode to **PSP**, either use the MSR instruction to set the ASP bit, as detailed in the *Cortex™-M3/M4 Instruction Set Technical User's Manual*, or perform an exception return to Thread mode with the appropriate EXC_RETURN value, as shown in Table 2-10 on page 88.

**Note:** When changing the stack pointer, software must use an ISB instruction immediately after the MSR instruction, ensuring that instructions after the ISB execute use the new stack pointer. See the *Cortex™-M3/M4 Instruction Set Technical User's Manual*.

Control Register (CONTROL)

Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | ASP | TMPL |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | ASP | R/W | 0 | Active Stack Pointer<br><br>Value Description<br>1 **PSP** is the current stack pointer.<br>0 **MSP** is the current stack pointer<br><br>In Handler mode, this bit reads as zero and ignores writes. The Cortex-M3 updates this bit automatically on exception return. |
| 0 | TMPL | R/W | 0 | Thread Mode Privilege Level<br><br>Value Description<br>1 Unprivileged software can be executed in Thread mode.<br>0 Only privileged software can be executed in Thread mode. |

### 2.3.5 Exceptions and Interrupts

The Cortex-M3 processor supports interrupts and system exceptions. The processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses Handler mode to handle all exceptions except for reset. See “Exception Entry and Return” on page 86 for more information.

The NVIC registers control interrupt handling. See “Nested Vectored Interrupt Controller (NVIC)” on page 96 for more information.

### 2.3.6 Data Types

The Cortex-M3 supports 32-bit words, 16-bit halfwords, and 8-bit bytes. The processor also supports 64-bit data transfer instructions. All instruction and data memory accesses are little endian. See “Memory Regions, Types and Attributes” on page 74 for more information.

## 2.4 Memory Model

This section describes the processor memory map, the behavior of memory accesses, and the bit-banding features. The processor has a fixed memory map that provides up to 4 GB of addressable memory.

The memory map for the LM3S5632 controller is provided in Table 2-4 on page 72. In this manual, register addresses are given as a hexadecimal increment, relative to the module’s base address as shown in the memory map.

The regions for SRAM and peripherals include bit-band regions. Bit-banding provides atomic operations to bit data (see “Bit-Banding” on page 76).

The processor reserves regions of the Private peripheral bus (PPB) address range for core peripheral registers (see “Cortex-M3 Peripherals” on page 95).

**Note:** Within the memory map, all reserved space returns a bus fault when read or written.

**Table 2-4. Memory Map**

| Start | End | Description | For details, see page ... |
|---|---|---|---|
| **Memory** | | | |
| 0x0000.0000 | 0x0001.FFFF | On-chip Flash | 267 |
| 0x0002.0000 | 0x00FF.FFFF | Reserved | - |
| 0x0100.0000 | 0x1FFF.FFFF | Reserved for ROM | 261 |
| 0x2000.0000 | 0x2000.7FFF | Bit-banded on-chip SRAM | 260 |
| 0x2000.8000 | 0x21FF.FFFF | Reserved | - |
| 0x2200.0000 | 0x220F.FFFF | Bit-band alias of bit-banded on-chip SRAM starting at 0x2000.0000 | 260 |
| 0x2210.0000 | 0x3FFF.FFFF | Reserved | - |
| **FiRM Peripherals** | | | |
| 0x4000.0000 | 0x4000.0FFF | Watchdog timer 0 | 436 |
| 0x4000.1000 | 0x4000.3FFF | Reserved | - |
| 0x4000.4000 | 0x4000.4FFF | GPIO Port A | 361 |
| 0x4000.5000 | 0x4000.5FFF | GPIO Port B | 361 |
| 0x4000.6000 | 0x4000.6FFF | GPIO Port C | 361 |
| 0x4000.7000 | 0x4000.7FFF | GPIO Port D | 361 |

**Table 2-4. Memory Map *(continued)***

| Start | End | Description | For details, see page ... |
|---|---|---|---|
| 0x4000.8000 | 0x4000.8FFF | SSI0 | 549 |
| 0x4000.9000 | 0x4000.BFFF | Reserved | - |
| 0x4000.C000 | 0x4000.CFFF | UART0 | 501 |
| 0x4000.D000 | 0x4000.DFFF | UART1 | 501 |
| 0x4000.E000 | 0x4001.FFFF | Reserved | - |
| **Peripherals** | | | |
| 0x4002.0000 | 0x4002.0FFF | $I^2C$ 0 | 591 |
| 0x4002.1000 | 0x4002.1FFF | $I^2C$ 1 | 591 |
| 0x4002.2000 | 0x4002.3FFF | Reserved | - |
| 0x4002.4000 | 0x4002.4FFF | GPIO Port E | 361 |
| 0x4002.5000 | 0x4002.FFFF | Reserved | - |
| 0x4003.0000 | 0x4003.0FFF | Timer 0 | 410 |
| 0x4003.1000 | 0x4003.1FFF | Timer 1 | 410 |
| 0x4003.2000 | 0x4003.2FFF | Timer 2 | 410 |
| 0x4003.3000 | 0x4003.7FFF | Reserved | - |
| 0x4003.8000 | 0x4003.8FFF | ADC0 | 466 |
| 0x4003.9000 | 0x4003.FFFF | Reserved | - |
| 0x4004.0000 | 0x4004.0FFF | CAN0 Controller | 633 |
| 0x4004.1000 | 0x4004.FFFF | Reserved | - |
| 0x4005.0000 | 0x4005.0FFF | USB | 675 |
| 0x4005.1000 | 0x4005.7FFF | Reserved | - |
| 0x4005.8000 | 0x4005.8FFF | GPIO Port A (AHB aperture) | 361 |
| 0x4005.9000 | 0x4005.9FFF | GPIO Port B (AHB aperture) | 361 |
| 0x4005.A000 | 0x4005.AFFF | GPIO Port C (AHB aperture) | 361 |
| 0x4005.B000 | 0x4005.BFFF | GPIO Port D (AHB aperture) | 361 |
| 0x4005.C000 | 0x4005.CFFF | GPIO Port E (AHB aperture) | 361 |
| 0x4005.D000 | 0x400F.BFFF | Reserved | - |
| 0x400F.C000 | 0x400F.CFFF | Hibernation Module | 246 |
| 0x400F.D000 | 0x400F.DFFF | Flash memory control | 267 |
| 0x400F.E000 | 0x400F.EFFF | System control | 185 |
| 0x400F.F000 | 0x400F.FFFF | µDMA | 310 |
| 0x4010.0000 | 0x41FF.FFFF | Reserved | - |
| 0x4200.0000 | 0x43FF.FFFF | Bit-banded alias of 0x4000.0000 through 0x400F.FFFF | - |
| 0x4400.0000 | 0xDFFF.FFFF | Reserved | - |
| **Private Peripheral Bus** | | | |
| 0xE000.0000 | 0xE000.0FFF | Instrumentation Trace Macrocell (ITM) | 55 |
| 0xE000.1000 | 0xE000.1FFF | Data Watchpoint and Trace (DWT) | 55 |
| 0xE000.2000 | 0xE000.2FFF | Flash Patch and Breakpoint (FPB) | 55 |
| 0xE000.3000 | 0xE000.DFFF | Reserved | - |
| 0xE000.E000 | 0xE000.EFFF | Cortex-M3 Peripherals (SysTick, NVIC, MPU and SCB) | 103 |
| 0xE000.F000 | 0xE003.FFFF | Reserved | - |

**Table 2-4. Memory Map *(continued)***

| Start | End | Description | For details, see page ... |
|---|---|---|---|
| 0xE004.0000 | 0xE004.0FFF | Trace Port Interface Unit (TPIU) | 56 |
| 0xE004.1000 | 0xFFFF.FFFF | Reserved | - |

### 2.4.1 Memory Regions, Types and Attributes

The memory map and the programming of the MPU split the memory map into regions. Each region has a defined memory type, and some regions have additional memory attributes. The memory type and attributes determine the behavior of accesses to the region.

The memory types are:

- Normal: The processor can re-order transactions for efficiency and perform speculative reads.
- Device: The processor preserves transaction order relative to other transactions to Device or Strongly Ordered memory.
- Strongly Ordered: The processor preserves transaction order relative to all other transactions.

The different ordering requirements for Device and Strongly Ordered memory mean that the memory system can buffer a write to Device memory but must not buffer a write to Strongly Ordered memory.

An additional memory attribute is Execute Never (XN), which means the processor prevents instruction accesses. A fault exception is generated only on execution of an instruction executed from an XN region.

### 2.4.2 Memory System Ordering of Memory Accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not guarantee that the order in which the accesses complete matches the program order of the instructions, providing the order does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, software must insert a memory barrier instruction between the memory access instructions (see “Software Ordering of Memory Accesses” on page 75).

However, the memory system does guarantee ordering of accesses to Device and Strongly Ordered memory. For two memory access instructions A1 and A2, if both A1 and A2 are accesses to either Device or Strongly Ordered memory, and if A1 occurs before A2 in program order, A1 is always observed before A2.

### 2.4.3 Behavior of Memory Accesses

Table 2-5 on page 74 shows the behavior of accesses to each region in the memory map. See “Memory Regions, Types and Attributes” on page 74 for more information on memory types and the XN attribute. Stellaris devices may have reserved memory areas within the address ranges shown below (refer to Table 2-4 on page 72 for more information).

**Table 2-5. Memory Access Behavior**

| Address Range | Memory Region | Memory Type | Execute Never (XN) | Description |
|---|---|---|---|---|
| 0x0000.0000 - 0x1FFF.FFFF | Code | Normal | - | This executable region is for program code. Data can also be stored here. |

**Table 2-5. Memory Access Behavior *(continued)***

| Address Range | Memory Region | Memory Type | Execute Never (XN) | Description |
|---|---|---|---|---|
| 0x2000.0000 - 0x3FFF.FFFF | SRAM | Normal | - | This executable region is for data. Code can also be stored here. This region includes bit band and bit band alias areas (see Table 2-6 on page 77). |
| 0x4000.0000 - 0x5FFF.FFFF | Peripheral | Device | XN | This region includes bit band and bit band alias areas (see Table 2-7 on page 77). |
| 0x6000.0000 - 0x9FFF.FFFF | External RAM | Normal | - | This executable region is for data. |
| 0xA000.0000 - 0xDFFF.FFFF | External device | Device | XN | This region is for external device memory. |
| 0xE000.0000- 0xE00F.FFFF | Private peripheral bus | Strongly Ordered | XN | This region includes the NVIC, system timer, and system control block. |
| 0xE010.0000- 0xFFFF.FFFF | Reserved | - | - | - |

The Code, SRAM, and external RAM regions can hold programs. However, it is recommended that programs always use the Code region because the Cortex-M3 has separate buses that can perform instruction fetches and data accesses simultaneously.

The MPU can override the default memory access behavior described in this section. For more information, see “Memory Protection Unit (MPU)” on page 98.

The Cortex-M3 prefetches instructions ahead of execution and speculatively prefetches from branch target addresses.

### 2.4.4 Software Ordering of Memory Accesses

The order of instructions in the program flow does not always guarantee the order of the corresponding memory transactions for the following reasons:

- The processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence.
- The processor has multiple bus interfaces.
- Memory or devices in the memory map have different wait states.
- Some memory accesses are buffered or speculative.

“Memory System Ordering of Memory Accesses” on page 74 describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The Cortex-M3 has the following memory barrier instructions:

- The Data Memory Barrier (`DMB`) instruction ensures that outstanding memory transactions complete before subsequent memory transactions.
- The Data Synchronization Barrier (`DSB`) instruction ensures that outstanding memory transactions complete before subsequent instructions execute.
- The Instruction Synchronization Barrier (`ISB`) instruction ensures that the effect of all completed memory transactions is recognizable by subsequent instructions.

Memory barrier instructions can be used in the following situations:

- MPU programming
  - If the MPU settings are changed and the change must be effective on the very next instruction, use a DSB instruction to ensure the effect of the MPU takes place immediately at the end of context switching.
  - Use an ISB instruction to ensure the new MPU setting takes effect immediately after programming the MPU region or regions, if the MPU configuration code was accessed using a branch or call. If the MPU configuration code is entered using exception mechanisms, then an ISB instruction is not required.
- Vector table

  If the program changes an entry in the vector table and then enables the corresponding exception, use a DMB instruction between the operations. The DMB instruction ensures that if the exception is taken immediately after being enabled, the processor uses the new exception vector.
- Self-modifying code

  If a program contains self-modifying code, use an ISB instruction immediately after the code modification in the program. The ISB instruction ensures subsequent instruction execution uses the updated program.
- Memory map switching

  If the system contains a memory map switching mechanism, use a DSB instruction after switching the memory map in the program. The DSB instruction ensures subsequent instruction execution uses the updated memory map.
- Dynamic exception priority change

  When an exception priority has to change when the exception is pending or active, use DSB instructions after the change. The change then takes effect on completion of the DSB instruction.

Memory accesses to Strongly Ordered memory, such as the System Control Block, do not require the use of DMB instructions.

For more information on the memory barrier instructions, see the *Cortex™-M3/M4 Instruction Set Technical User's Manual*.

### 2.4.5 Bit-Banding

A bit-band region maps each word in a bit-band alias region to a single bit in the bit-band region. The bit-band regions occupy the lowest 1 MB of the SRAM and peripheral memory regions. Accesses to the 32-MB SRAM alias region map to the 1-MB SRAM bit-band region, as shown in Table 2-6 on page 77. Accesses to the 32-MB peripheral alias region map to the 1-MB peripheral bit-band region, as shown in Table 2-7 on page 77. For the specific address range of the bit-band regions, see Table 2-4 on page 72.

**Note:** A word access to the SRAM or the peripheral bit-band alias region maps to a single bit in the SRAM or peripheral bit-band region.

A word access to a bit band address results in a word access to the underlying memory, and similarly for halfword and byte accesses. This allows bit band accesses to match the access requirements of the underlying peripheral.

**Table 2-6. SRAM Memory Bit-Banding Regions**

| Address Range | | Memory Region | Instruction and Data Accesses |
|---|---|---|---|
| Start | End | | |
| 0x2000.0000 | 0x2000.7FFF | SRAM bit-band region | Direct accesses to this memory range behave as SRAM memory accesses, but this region is also bit addressable through bit-band alias. |
| 0x2200.0000 | 0x220F.FFFF | SRAM bit-band alias | Data accesses to this region are remapped to bit band region. A write operation is performed as read-modify-write. Instruction accesses are not remapped. |

**Table 2-7. Peripheral Memory Bit-Banding Regions**

| Address Range | | Memory Region | Instruction and Data Accesses |
|---|---|---|---|
| Start | End | | |
| 0x4000.0000 | 0x400F.FFFF | Peripheral bit-band region | Direct accesses to this memory range behave as peripheral memory accesses, but this region is also bit addressable through bit-band alias. |
| 0x4200.0000 | 0x43FF.FFFF | Peripheral bit-band alias | Data accesses to this region are remapped to bit band region. A write operation is performed as read-modify-write. Instruction accesses are not permitted. |

The following formula shows how the alias region maps onto the bit-band region:

```
bit_word_offset = (byte_offset x 32) + (bit_number x 4)
```

```
bit_word_addr = bit_band_base + bit_word_offset
```

where:

bit_word_offset
The position of the target bit in the bit-band memory region.

bit_word_addr
The address of the word in the alias memory region that maps to the targeted bit.

bit_band_base
The starting address of the alias region.

byte_offset
The number of the byte in the bit-band region that contains the targeted bit.

bit_number
The bit position, 0-7, of the targeted bit.

Figure 2-4 on page 78 shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at 0x23FF.FFE0 maps to bit 0 of the bit-band byte at 0x200F.FFFF:

  ```
  0x23FF.FFE0 = 0x2200.0000 + (0x000F.FFFF*32) + (0*4)
  ```

- The alias word at 0x23FF.FFFC maps to bit 7 of the bit-band byte at 0x200F.FFFF:

  ```
  0x23FF.FFFC = 0x2200.0000 + (0x000F.FFFF*32) + (7*4)
  ```

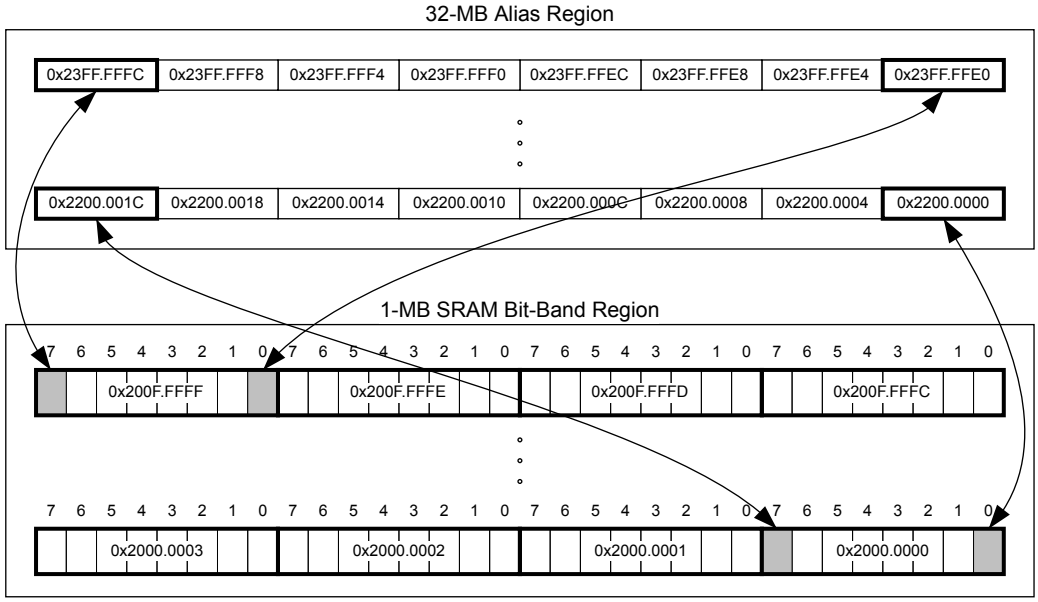
- The alias word at 0x2200.0000 maps to bit 0 of the bit-band byte at 0x2000.0000:

  `0x2200.0000 = 0x2200.0000 + (0*32) + (0*4)`

- The alias word at 0x2200.001C maps to bit 7 of the bit-band byte at 0x2000.0000:

  `0x2200.001C = 0x2200.0000+ (0*32) + (7*4)`

**Figure 2-4. Bit-Band Mapping**




### 2.4.5.1 Directly Accessing an Alias Region

Writing to a word in the alias region updates a single bit in the bit-band region.

Bit 0 of the value written to a word in the alias region determines the value written to the targeted bit in the bit-band region. Writing a value with bit 0 set writes a 1 to the bit-band bit, and writing a value with bit 0 clear writes a 0 to the bit-band bit.

Bits 31:1 of the alias word have no effect on the bit-band bit. Writing 0x01 has the same effect as writing 0xFF. Writing 0x00 has the same effect as writing 0x0E.

When reading a word in the alias region, 0x0000.0000 indicates that the targeted bit in the bit-band region is clear and 0x0000.0001 indicates that the targeted bit in the bit-band region is set.

### 2.4.5.2 Directly Accessing a Bit-Band Region

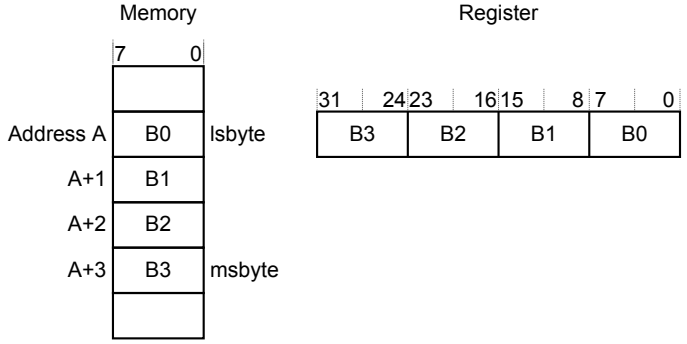
“Behavior of Memory Accesses” on page 74 describes the behavior of direct byte, halfword, or word accesses to the bit-band regions.

## 2.4.6 Data Storage

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word. Data is stored in little-endian format, with the least-significant byte (lsbyte) of a word stored at the

lowest-numbered byte, and the most-significant byte (msbyte) stored at the highest-numbered byte. Figure 2-5 on page 79 illustrates how data is stored.

**Figure 2-5. Data Storage**




## 2.4.7 Synchronization Primitives

The Cortex-M3 instruction set includes pairs of synchronization primitives which provide a non-blocking mechanism that a thread or process can use to obtain exclusive access to a memory location. Software can use these primitives to perform a guaranteed read-modify-write memory update sequence or for a semaphore mechanism.

A pair of synchronization primitives consists of:

- A Load-Exclusive instruction, which is used to read the value of a memory location and requests exclusive access to that location.
- A Store-Exclusive instruction, which is used to attempt to write to the same memory location and returns a status bit to a register. If this status bit is clear, it indicates that the thread or process gained exclusive access to the memory and the write succeeds; if this status bit is set, it indicates that the thread or process did not gain exclusive access to the memory and no write was performed.

The pairs of Load-Exclusive and Store-Exclusive instructions are:

- The word instructions LDREX and STREX
- The halfword instructions LDREXH and STREXH
- The byte instructions LDREXB and STREXB

Software must use a Load-Exclusive instruction with the corresponding Store-Exclusive instruction.

To perform an exclusive read-modify-write of a memory location, software must:

1. Use a Load-Exclusive instruction to read the value of the location.
2. Modify the value, as required.
3. Use a Store-Exclusive instruction to attempt to write the new value back to the memory location.
4. Test the returned status bit.

If the status bit is clear, the read-modify-write completed successfully. If the status bit is set, no write was performed, which indicates that the value returned at step 1 might be out of date. The software must retry the entire read-modify-write sequence.

Software can use the synchronization primitives to implement a semaphore as follows:

1. Use a Load-Exclusive instruction to read from the semaphore address to check whether the semaphore is free.
2. If the semaphore is free, use a Store-Exclusive to write the claim value to the semaphore address.
3. If the returned status bit from step 2 indicates that the Store-Exclusive succeeded, then the software has claimed the semaphore. However, if the Store-Exclusive failed, another process might have claimed the semaphore after the software performed step 1.

The Cortex-M3 includes an exclusive access monitor that tags the fact that the processor has executed a Load-Exclusive instruction. The processor removes its exclusive access tag if:

- It executes a CLREX instruction.
- It executes a Store-Exclusive instruction, regardless of whether the write succeeds.
- An exception occurs, which means the processor can resolve semaphore conflicts between different threads.

For more information about the synchronization primitive instructions, see the *Cortex™-M3/M4 Instruction Set Technical User's Manual*.

## 2.5 Exception Model

The ARM Cortex-M3 processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions in Handler Mode. The processor state is automatically stored to the stack on an exception and automatically restored from the stack at the end of the Interrupt Service Routine (ISR). The vector is fetched in parallel to the state saving, enabling efficient interrupt entry. The processor supports tail-chaining, which enables back-to-back interrupts to be performed without the overhead of state saving and restoration.

Table 2-8 on page 82 lists all exception types. Software can set eight priority levels on seven of these exceptions (system handlers) as well as on 28 interrupts (listed in Table 2-9 on page 83).

Priorities on the system handlers are set with the NVIC **System Handler Priority n (SYSPRIn)** registers. Interrupts are enabled through the NVIC **Interrupt Set Enable n (ENn)** register and prioritized with the NVIC **Interrupt Priority n (PRIn)** registers. Priorities can be grouped by splitting priority levels into preemption priorities and subpriorities. All the interrupt registers are described in “Nested Vectored Interrupt Controller (NVIC)” on page 96.

Internally, the highest user-programmable priority (0) is treated as fourth priority, after a Reset, Non-Maskable Interrupt (NMI), and a Hard Fault, in that order. Note that 0 is the default priority for all the programmable priorities.

**Important:** After a write to clear an interrupt source, it may take several processor cycles for the NVIC to see the interrupt source de-assert. Thus if the interrupt clear is done as the last action in an interrupt handler, it is possible for the interrupt handler to complete while the NVIC sees the interrupt as still asserted, causing the interrupt handler to be

re-entered errantly. This situation can be avoided by either clearing the interrupt source at the beginning of the interrupt handler or by performing a read or write after the write to clear the interrupt source (and flush the write buffer).

See “Nested Vectored Interrupt Controller (NVIC)” on page 96 for more information on exceptions and interrupts.

### 2.5.1 Exception States

Each exception is in one of the following states:

- **Inactive.** The exception is not active and not pending.
- **Pending.** The exception is waiting to be serviced by the processor. An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.
- **Active.** An exception that is being serviced by the processor but has not completed.

  **Note:** An exception handler can interrupt the execution of another exception handler. In this case, both exceptions are in the active state.

- **Active and Pending.** The exception is being serviced by the processor, and there is a pending exception from the same source.

### 2.5.2 Exception Types

The exception types are:

- **Reset.** Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in Thread mode.
- **NMI.** A non-maskable Interrupt (NMI) can be signaled using the NMI signal or triggered by software using the **Interrupt Control and State (INTCTRL)** register. This exception has the highest priority other than reset. NMI is permanently enabled and has a fixed priority of -2. NMIs cannot be masked or prevented from activation by any other exception or preempted by any exception other than reset.
- **Hard Fault.** A hard fault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism. Hard faults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.
- **Memory Management Fault.** A memory management fault is an exception that occurs because of a memory protection related fault, including access violation and no match. The MPU or the fixed memory protection constraints determine this fault, for both instruction and data memory transactions. This fault is used to abort instruction accesses to Execute Never (XN) memory regions, even if the MPU is disabled.
- **Bus Fault.** A bus fault is an exception that occurs because of a memory-related fault for an instruction or data memory transaction such as a prefetch fault or a memory access fault. This fault can be enabled or disabled.

- **Usage Fault.** A usage fault is an exception that occurs because of a fault related to instruction execution, such as:
  - An undefined instruction
  - An illegal unaligned access
  - Invalid state on instruction execution
  - An error on exception return

  An unaligned address on a word or halfword memory access or division by zero can cause a usage fault when the core is properly configured.
- **SVCall.** A supervisor call (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.
- **Debug Monitor.** This exception is caused by the debug monitor (when not halting). This exception is only active when enabled. This exception does not activate if it is a lower priority than the current activation.
- **PendSV.** PendSV is a pendable, interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active. PendSV is triggered using the **Interrupt Control and State (INTCTRL)** register.
- **SysTick.** A SysTick exception is an exception that the system timer generates when it reaches zero when it is enabled to generate an interrupt. Software can also generate a SysTick exception using the **Interrupt Control and State (INTCTRL)** register. In an OS environment, the processor can use this exception as system tick.
- **Interrupt (IRQ).** An interrupt, or IRQ, is an exception signaled by a peripheral or generated by a software request and fed through the NVIC (prioritized). All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor. Table 2-9 on page 83 lists the interrupts on the LM3S5632 controller.

For an asynchronous exception, other than reset, the processor can execute another instruction between when the exception is triggered and when the processor enters the exception handler.

Privileged software can disable the exceptions that Table 2-8 on page 82 shows as having configurable priority (see the **SYSHNDCTRL** register on page 137 and the **DIS0** register on page 112).

For more information about hard faults, memory management faults, bus faults, and usage faults, see “Fault Handling” on page 88.

**Table 2-8. Exception Types**

| Exception Type | Vector Number | Priority[a] | Vector Address or Offset[b] | Activation |
|---|---|---|---|---|
| - | 0 | - | 0x0000.0000 | Stack top is loaded from the first entry of the vector table on reset. |
| Reset | 1 | -3 (highest) | 0x0000.0004 | Asynchronous |
| Non-Maskable Interrupt (NMI) | 2 | -2 | 0x0000.0008 | Asynchronous |
| Hard Fault | 3 | -1 | 0x0000.000C | - |
| Memory Management | 4 | programmable[c] | 0x0000.0010 | Synchronous |

**Table 2-8. Exception Types *(continued)***

| Exception Type | Vector Number | Priority[a] | Vector Address or Offset[b] | Activation |
|---|---|---|---|---|
| Bus Fault | 5 | programmable[c] | 0x0000.0014 | Synchronous when precise and asynchronous when imprecise |
| Usage Fault | 6 | programmable[c] | 0x0000.0018 | Synchronous |
| - | 7-10 | - | - | Reserved |
| SVCall | 11 | programmable[c] | 0x0000.002C | Synchronous |
| Debug Monitor | 12 | programmable[c] | 0x0000.0030 | Synchronous |
| - | 13 | - | - | Reserved |
| PendSV | 14 | programmable[c] | 0x0000.0038 | Asynchronous |
| SysTick | 15 | programmable[c] | 0x0000.003C | Asynchronous |
| Interrupts | 16 and above | programmable[d] | 0x0000.0040 and above | Asynchronous |

a. 0 is the default priority for all the programmable priorities.

b. See “Vector Table” on page 84.

c. See **SYSPRI1** on page 134.

d. See **PRIn** registers on page 120.

**Table 2-9. Interrupts**

| Vector Number | Interrupt Number (Bit in Interrupt Registers) | Vector Address or Offset | Description |
|---|---|---|---|
| 0-15 | - | 0x0000.0000 - 0x0000.003C | Processor exceptions |
| 16 | 0 | 0x0000.0040 | GPIO Port A |
| 17 | 1 | 0x0000.0044 | GPIO Port B |
| 18 | 2 | 0x0000.0048 | GPIO Port C |
| 19 | 3 | 0x0000.004C | GPIO Port D |
| 20 | 4 | 0x0000.0050 | GPIO Port E |
| 21 | 5 | 0x0000.0054 | UART0 |
| 22 | 6 | 0x0000.0058 | UART1 |
| 23 | 7 | 0x0000.005C | SSI0 |
| 24 | 8 | 0x0000.0060 | $I^2C0$ |
| 25-29 | 9-13 | - | Reserved |
| 30 | 14 | 0x0000.0078 | ADC0 Sequence 0 |
| 31 | 15 | 0x0000.007C | ADC0 Sequence 1 |
| 32 | 16 | 0x0000.0080 | ADC0 Sequence 2 |
| 33 | 17 | 0x0000.0084 | ADC0 Sequence 3 |
| 34 | 18 | 0x0000.0088 | Watchdog Timer 0 |
| 35 | 19 | 0x0000.008C | Timer 0A |
| 36 | 20 | 0x0000.0090 | Timer 0B |
| 37 | 21 | 0x0000.0094 | Timer 1A |
| 38 | 22 | 0x0000.0098 | Timer 1B |
| 39 | 23 | 0x0000.009C | Timer 2A |
| 40 | 24 | 0x0000.00A0 | Timer 2B |
| 41-43 | 25-27 | - | Reserved |

**Table 2-9. Interrupts *(continued)***

| Vector Number | Interrupt Number (Bit in Interrupt Registers) | Vector Address or Offset | Description |
|---|---|---|---|
| 44 | 28 | 0x0000.00B0 | System Control |
| 45 | 29 | 0x0000.00B4 | Flash Memory Control |
| 46-52 | 30-36 | - | Reserved |
| 53 | 37 | 0x0000.00D4 | $I^2C1$ |
| 54 | 38 | - | Reserved |
| 55 | 39 | 0x0000.00DC | CAN0 |
| 56-58 | 40-42 | - | Reserved |
| 59 | 43 | 0x0000.00EC | Hibernation Module |
| 60 | 44 | 0x0000.00F0 | USB |
| 61 | 45 | - | Reserved |
| 62 | 46 | 0x0000.00F8 | µDMA Software |
| 63 | 47 | 0x0000.00FC | µDMA Error |

### 2.5.3 Exception Handlers

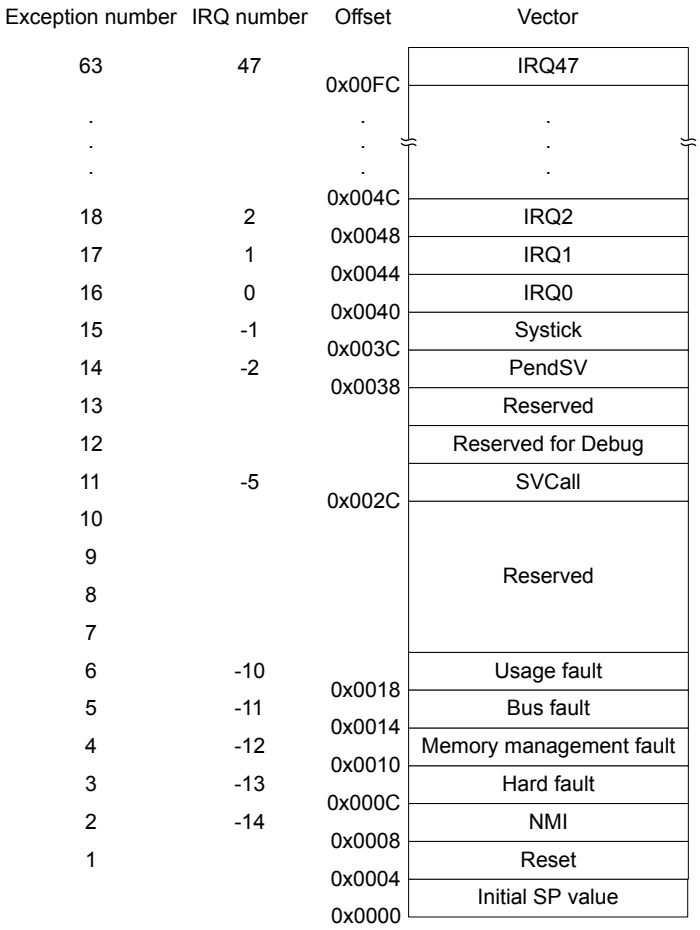
The processor handles exceptions using:

- **Interrupt Service Routines (ISRs).** Interrupts (IRQx) are the exceptions handled by ISRs.
- **Fault Handlers.** Hard fault, memory management fault, usage fault, and bus fault are fault exceptions handled by the fault handlers.
- **System Handlers.** NMI, PendSV, SVCall, SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.

### 2.5.4 Vector Table

The vector table contains the reset value of the stack pointer and the start addresses, also called exception vectors, for all exception handlers. The vector table is constructed using the vector address or offset shown in Table 2-8 on page 82. Figure 2-6 on page 85 shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is Thumb code

**Figure 2-6. Vector Table**




On system reset, the vector table is fixed at address 0x0000.0000. Privileged software can write to the **Vector Table Offset (VTABLE)** register to relocate the vector table start address to a different memory location, in the range 0x0000.0100 to 0x3FFF.FF00 (see "Vector Table" on page 84). Note that when configuring the **VTABLE** register, the offset must be aligned on a 256-byte boundary.

## 2.5.5 Exception Priorities

As Table 2-8 on page 82 shows, all exceptions have an associated priority, with a lower priority value indicating a higher priority and configurable priorities for all exceptions except Reset, Hard fault, and NMI. If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities, see page 134 and page 120.

**Note:** Configurable priority values for the Stellaris implementation are in the range 0-7. This means that the Reset, Hard fault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

For example, assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

### 2.5.6 Interrupt Priority Grouping

To increase priority control in systems with interrupts, the NVIC supports priority grouping. This grouping divides each interrupt priority register entry into two fields:

- An upper field that defines the group priority
- A lower field that defines a subpriority within the group

Only the group priority determines preemption of interrupt exceptions. When the processor is executing an interrupt exception handler, another interrupt with the same group priority as the interrupt being handled does not preempt the handler.

If multiple pending interrupts have the same group priority, the subpriority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and subpriority, the interrupt with the lowest IRQ number is processed first.

For information about splitting the interrupt priority fields into group priority and subpriority, see page 128.

### 2.5.7 Exception Entry and Return

Descriptions of exception handling use the following terms:

- **Preemption.** When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled. See “Interrupt Priority Grouping” on page 86 for more information about preemption by an interrupt. When one exception preempts another, the exceptions are called nested exceptions. See “Exception Entry” on page 87 more information.
- **Return.** Return occurs when the exception handler is completed, and there is no pending exception with sufficient priority to be serviced and the completed exception handler was not handling a late-arriving exception. The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See “Exception Return” on page 88 for more information.
- **Tail-Chaining.** This mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.
- **Late-Arriving.** This mechanism speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception. State saving is not affected by late arrival because the state saved is the same for both exceptions. Therefore, the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor. On

return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.

### 2.5.7.1 Exception Entry

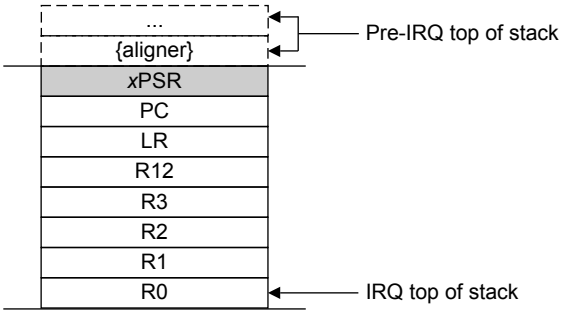
Exception entry occurs when there is a pending exception with sufficient priority and either the processor is in Thread mode or the new exception is of higher priority than the exception being handled, in which case the new exception preempts the original exception.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has more priority than any limits set by the mask registers (see **PRIMASK** on page 68, **FAULTMASK** on page 69, and **BASEPRI** on page 70). An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is referred to as *stacking* and the structure of eight data words is referred to as *stack frame*.

**Figure 2-7. Exception Stack Frame**




Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. Unless stack alignment is disabled, the stack frame is aligned to a double-word address. If the STKALIGN bit of the **Configuration Control (CCR)** register is set, stack align adjustment is performed during stacking.

The stack frame includes the return address, which is the address of the next instruction in the interrupted program. This value is restored to the **PC** at exception return so that the interrupted program resumes.

In parallel to the stacking operation, the processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC_RETURN value to the **LR**, indicating which stack pointer corresponds to the stack frame and what operation mode the processor was in before the entry occurred.

If no higher-priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher-priority exception occurs during exception entry, known as late arrival, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception.

### 2.5.7.2 Exception Return

Exception return occurs when the processor is in Handler mode and executes one of the following instructions to load the EXC_RETURN value into the **PC**:

- An LDM or POP instruction that loads the **PC**
- A BX instruction using any register
- An LDR instruction with the **PC** as the destination

EXC_RETURN is the value loaded into the **LR** on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. The lowest four bits of this value provide information on the return stack and processor mode. Table 2-10 on page 88 shows the EXC_RETURN values with a description of the exception return behavior.

EXC_RETURN bits 31:4 are all set. When this value is loaded into the **PC**, it indicates to the processor that the exception is complete, and the processor initiates the appropriate exception return sequence.

**Table 2-10. Exception Return Behavior**

| EXC_RETURN[31:0] | Description |
|---|---|
| 0xFFFF.FFF0 | Reserved |
| 0xFFFF.FFF1 | Return to Handler mode.<br>Exception return uses state from **MSP**.<br>Execution uses **MSP** after return. |
| 0xFFFF.FFF2 - 0xFFFF.FFF8 | Reserved |
| 0xFFFF.FFF9 | Return to Thread mode.<br>Exception return uses state from **MSP**.<br>Execution uses **MSP** after return. |
| 0xFFFF.FFFA - 0xFFFF.FFFC | Reserved |
| 0xFFFF.FFFD | Return to Thread mode.<br>Exception return uses state from **PSP**.<br>Execution uses **PSP** after return. |
| 0xFFFF.FFFE - 0xFFFF.FFFF | Reserved |

## 2.6 Fault Handling

Faults are a subset of the exceptions (see “Exception Model” on page 80). The following conditions generate a fault:

- A bus error on an instruction fetch or vector table load or a data access.
- An internally detected error such as an undefined instruction or an attempt to change state with a BX instruction.
- Attempting to execute an instruction from a memory region marked as Non-Executable (XN).
- An MPU fault because of a privilege violation or an attempt to access an unmanaged region.

### 2.6.1 Fault Types

Table 2-11 on page 89 shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates the fault has occurred. See page 141 for more information about the fault status registers.

**Table 2-11. Faults**

| Fault | Handler | Fault Status Register | Bit Name |
|---|---|---|---|
| Bus error on a vector read | Hard fault | **Hard Fault Status (HFAULTSTAT)** | VECT |
| Fault escalated to a hard fault | Hard fault | **Hard Fault Status (HFAULTSTAT)** | FORCED |
| MPU or default memory mismatch on instruction access | Memory management fault | **Memory Management Fault Status (MFAULTSTAT)** | IERR [a] |
| MPU or default memory mismatch on data access | Memory management fault | **Memory Management Fault Status (MFAULTSTAT)** | DERR |
| MPU or default memory mismatch on exception stacking | Memory management fault | **Memory Management Fault Status (MFAULTSTAT)** | MSTKE |
| MPU or default memory mismatch on exception unstacking | Memory management fault | **Memory Management Fault Status (MFAULTSTAT)** | MUSTKE |
| Bus error during exception stacking | Bus fault | **Bus Fault Status (BFAULTSTAT)** | BSTKE |
| Bus error during exception unstacking | Bus fault | **Bus Fault Status (BFAULTSTAT)** | BUSTKE |
| Bus error during instruction prefetch | Bus fault | **Bus Fault Status (BFAULTSTAT)** | IBUS |
| Precise data bus error | Bus fault | **Bus Fault Status (BFAULTSTAT)** | PRECISE |
| Imprecise data bus error | Bus fault | **Bus Fault Status (BFAULTSTAT)** | IMPRE |
| Attempt to access a coprocessor | Usage fault | **Usage Fault Status (UFAULTSTAT)** | NOCP |
| Undefined instruction | Usage fault | **Usage Fault Status (UFAULTSTAT)** | UNDEF |
| Attempt to enter an invalid instruction set state [b] | Usage fault | **Usage Fault Status (UFAULTSTAT)** | INVSTAT |
| Invalid EXC_RETURN value | Usage fault | **Usage Fault Status (UFAULTSTAT)** | INVPC |
| Illegal unaligned load or store | Usage fault | **Usage Fault Status (UFAULTSTAT)** | UNALIGN |
| Divide by 0 | Usage fault | **Usage Fault Status (UFAULTSTAT)** | DIV0 |

a. Occurs on an access to an XN region even if the MPU is disabled.

b. Attempting to use an instruction set other than the Thumb instruction set, or returning to a non load-store-multiple instruction with ICI continuation.

### 2.6.2 Fault Escalation and Hard Faults

All fault exceptions except for hard fault have configurable exception priority (see **SYSPRI1** on page 134). Software can disable execution of the handlers for these faults (see **SYSHNDCTRL** on page 137).

Usually, the exception priority, together with the values of the exception mask registers, determines whether the processor enters the fault handler, and whether a fault handler can preempt another fault handler as described in “Exception Model” on page 80.

In some situations, a fault with configurable priority is treated as a hard fault. This process is called priority escalation, and the fault is described as *escalated to hard fault*. Escalation to hard fault occurs when:

- A fault handler causes the same kind of fault as the one it is servicing. This escalation to hard fault occurs because a fault handler cannot preempt itself because it must have the same priority as the current priority level.

- A fault handler causes a fault with the same or lower priority as the fault it is servicing. This situation happens because the handler for the new fault cannot preempt the currently executing fault handler.
- An exception handler causes a fault for which the priority is the same as or lower than the currently executing exception.
- A fault occurs and the handler for that fault is not enabled.

If a bus fault occurs during a stack push when entering a bus fault handler, the bus fault does not escalate to a hard fault. Thus if a corrupted stack causes a fault, the fault handler executes even though the stack push for the handler failed. The fault handler operates but the stack contents are corrupted.

**Note:** Only Reset and NMI can preempt the fixed priority hard fault. A hard fault can preempt any exception other than Reset, NMI, or another hard fault.

### 2.6.3 Fault Status Registers and Fault Address Registers

The fault status registers indicate the cause of a fault. For bus faults and memory management faults, the fault address register indicates the address accessed by the operation that caused the fault, as shown in Table 2-12 on page 90.

**Table 2-12. Fault Status and Fault Address Registers**

| Handler | Status Register Name | Address Register Name | Register Description |
|---|---|---|---|
| Hard fault | **Hard Fault Status (HFAULTSTAT)** | - | page 147 |
| Memory management fault | **Memory Management Fault Status (MFAULTSTAT)** | **Memory Management Fault Address (MMADDR)** | page 141<br>page 148 |
| Bus fault | **Bus Fault Status (BFAULTSTAT)** | **Bus Fault Address (FAULTADDR)** | page 141<br>page 149 |
| Usage fault | **Usage Fault Status (UFAULTSTAT)** | - | page 141 |

### 2.6.4 Lockup

The processor enters a lockup state if a hard fault occurs when executing the NMI or hard fault handlers. When the processor is in the lockup state, it does not execute any instructions. The processor remains in lockup state until it is reset, an NMI occurs, or it is halted by a debugger.

**Note:** If the lockup state occurs from the NMI handler, a subsequent NMI does not cause the processor to leave the lockup state.

## 2.7 Power Management

The Cortex-M3 processor sleep modes reduce power consumption:

- Sleep mode stops the processor clock.
- Deep-sleep mode stops the system clock and switches off the PLL and Flash memory.

The SLEEPDEEP bit of the **System Control (SYSCTRL)** register selects which sleep mode is used (see page 130). For more information about the behavior of the sleep modes, see “System Control” on page 182.

This section describes the mechanisms for entering sleep mode and the conditions for waking up from sleep mode, both of which apply to Sleep mode and Deep-sleep mode.

## 2.7.1 Entering Sleep Modes

This section describes the mechanisms software can use to put the processor into one of the sleep modes.

The system can generate spurious wake-up events, for example a debug operation wakes up the processor. Therefore, software must be able to put the processor back into sleep mode after such an event. A program might have an idle loop to put the processor back to sleep mode.

### 2.7.1.1 Wait for Interrupt

The wait for interrupt instruction, WFI, causes immediate entry to sleep mode unless the wake-up condition is true (see “Wake Up from WFI or Sleep-on-Exit” on page 91). When the processor executes a WFI instruction, it stops executing instructions and enters sleep mode. See the *Cortex™-M3/M4 Instruction Set Technical User's Manual* for more information.

### 2.7.1.2 Wait for Event

The wait for event instruction, WFE, causes entry to sleep mode conditional on the value of a one-bit event register. When the processor executes a WFE instruction, it checks the event register. If the register is 0, the processor stops executing instructions and enters sleep mode. If the register is 1, the processor clears the register and continues executing instructions without entering sleep mode.

If the event register is 1, the processor must not enter sleep mode on execution of a WFE instruction. Typically, this situation occurs if an SEV instruction has been executed. Software cannot access this register directly.

See the *Cortex™-M3/M4 Instruction Set Technical User's Manual* for more information.

### 2.7.1.3 Sleep-on-Exit

If the SLEEPEXIT bit of the **SYSCTRL** register is set, when the processor completes the execution of all exception handlers, it returns to Thread mode and immediately enters sleep mode. This mechanism can be used in applications that only require the processor to run when an exception occurs.

## 2.7.2 Wake Up from Sleep Mode

The conditions for the processor to wake up depend on the mechanism that cause it to enter sleep mode.

### 2.7.2.1 Wake Up from WFI or Sleep-on-Exit

Normally, the processor wakes up only when the NVIC detects an exception with sufficient priority to cause exception entry. Some embedded systems might have to execute system restore tasks after the processor wakes up and before executing an interrupt handler. Entry to the interrupt handler can be delayed by setting the PRIMASK bit and clearing the FAULTMASK bit. If an interrupt arrives that is enabled and has a higher priority than current exception priority, the processor wakes up but does not execute the interrupt handler until the processor clears PRIMASK. For more information about **PRIMASK** and **FAULTMASK**, see page 68 and page 69.

### 2.7.2.2 Wake Up from WFE

The processor wakes up if it detects an exception with sufficient priority to cause exception entry.

In addition, if the SEVONPEND bit in the **SYSCTRL** register is set, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause exception entry. For more information about **SYSCTRL**, see page 130.

## 2.8 Instruction Set Summary

The processor implements a version of the Thumb instruction set. Table 2-13 on page 92 lists the supported instructions.

**Note:** In Table 2-13 on page 92:

- Angle brackets, <>, enclose alternative forms of the operand
- Braces, {}, enclose optional operands
- The Operands column is not exhaustive
- Op2 is a flexible second operand that can be either a register or a constant
- Most instructions can use an optional condition code suffix

For more information on the instructions and operands, see the instruction descriptions in the *Cortex™-M3/M4 Instruction Set Technical User's Manual*.

**Table 2-13. Cortex-M3 Instruction Summary**

| Mnemonic | Operands | Brief Description | Flags |
|---|---|---|---|
| ADC, ADCS | {Rd,} Rn, Op2 | Add with carry | N,Z,C,V |
| ADD, ADDS | {Rd,} Rn, Op2 | Add | N,Z,C,V |
| ADD, ADDW | {Rd,} Rn , #imm12 | Add | N,Z,C,V |
| ADR | Rd, label | Load PC-relative address | - |
| AND, ANDS | {Rd,} Rn, Op2 | Logical AND | N,Z,C |
| ASR, ASRS | Rd, Rm, <Rs\|#n> | Arithmetic shift right | N,Z,C |
| B | label | Branch | - |
| BFC | Rd, #lsb, #width | Bit field clear | - |
| BFI | Rd, Rn, #lsb, #width | Bit field insert | - |
| BIC, BICS | {Rd,} Rn, Op2 | Bit clear | N,Z,C |
| BKPT | #imm | Breakpoint | - |
| BL | label | Branch with link | - |
| BLX | Rm | Branch indirect with link | - |
| BX | Rm | Branch indirect | - |
| CBNZ | Rn, label | Compare and branch if non-zero | - |
| CBZ | Rn, label | Compare and branch if zero | - |
| CLREX | - | Clear exclusive | - |
| CLZ | Rd, Rm | Count leading zeros | - |
| CMN | Rn, Op2 | Compare negative | N,Z,C,V |
| CMP | Rn, Op2 | Compare | N,Z,C,V |
| CPSID | i | Change processor state, disable interrupts | - |
| CPSIE | i | Change processor state, enable interrupts | - |
| DMB | - | Data memory barrier | - |
| DSB | - | Data synchronization barrier | - |

**Table 2-13. Cortex-M3 Instruction Summary *(continued)***

| Mnemonic | Operands | Brief Description | Flags |
|---|---|---|---|
| EOR, EORS | {Rd,} Rn, Op2 | Exclusive OR | N,Z,C |
| ISB | - | Instruction synchronization barrier | - |
| IT | - | If-Then condition block | - |
| LDM | Rn{!}, reglist | Load multiple registers, increment after | - |
| LDMDB, LDMEA | Rn{!}, reglist | Load multiple registers, decrement before | - |
| LDMFD, LDMIA | Rn{!}, reglist | Load multiple registers, increment after | - |
| LDR | Rt, [Rn, #offset] | Load register with word | - |
| LDRB, LDRBT | Rt, [Rn, #offset] | Load register with byte | - |
| LDRD | Rt, Rt2, [Rn, #offset] | Load register with two bytes | - |
| LDREX | Rt, [Rn, #offset] | Load register exclusive | - |
| LDREXB | Rt, [Rn] | Load register exclusive with byte | - |
| LDREXH | Rt, [Rn] | Load register exclusive with halfword | - |
| LDRH, LDRHT | Rt, [Rn, #offset] | Load register with halfword | - |
| LDRSB, LDRSBT | Rt, [Rn, #offset] | Load register with signed byte | - |
| LDRSH, LDRSHT | Rt, [Rn, #offset] | Load register with signed halfword | - |
| LDRT | Rt, [Rn, #offset] | Load register with word | - |
| LSL, LSLS | Rd, Rm, <Rs\|#n> | Logical shift left | N,Z,C |
| LSR, LSRS | Rd, Rm, <Rs\|#n> | Logical shift right | N,Z,C |
| MLA | Rd, Rn, Rm, Ra | Multiply with accumulate, 32-bit result | - |
| MLS | Rd, Rn, Rm, Ra | Multiply and subtract, 32-bit result | - |
| MOV, MOVS | Rd, Op2 | Move | N,Z,C |
| MOV, MOVW | Rd, #imm16 | Move 16-bit constant | N,Z,C |
| MOVT | Rd, #imm16 | Move top | - |
| MRS | Rd, spec_reg | Move from special register to general register | - |
| MSR | spec_reg, Rm | Move from general register to special register | N,Z,C,V |
| MUL, MULS | {Rd,} Rn, Rm | Multiply, 32-bit result | N,Z |
| MVN, MVNS | Rd, Op2 | Move NOT | N,Z,C |
| NOP | - | No operation | - |
| ORN, ORNS | {Rd,} Rn, Op2 | Logical OR NOT | N,Z,C |
| ORR, ORRS | {Rd,} Rn, Op2 | Logical OR | N,Z,C |
| POP | reglist | Pop registers from stack | - |
| PUSH | reglist | Push registers onto stack | - |
| RBIT | Rd, Rn | Reverse bits | - |
| REV | Rd, Rn | Reverse byte order in a word | - |
| REV16 | Rd, Rn | Reverse byte order in each halfword | - |
| REVSH | Rd, Rn | Reverse byte order in bottom halfword and sign extend | - |
| ROR, RORS | Rd, Rm, <Rs\|#n> | Rotate right | N,Z,C |
| RRX, RRXS | Rd, Rm | Rotate right with extend | N,Z,C |

**Table 2-13. Cortex-M3 Instruction Summary *(continued)***

| Mnemonic | Operands | Brief Description | Flags |
|---|---|---|---|
| RSB, RSBS | {Rd,} Rn, Op2 | Reverse subtract | N,Z,C,V |
| SBC, SBCS | {Rd,} Rn, Op2 | Subtract with carry | N,Z,C,V |
| SBFX | Rd, Rn, #lsb, #width | Signed bit field extract | - |
| SDIV | {Rd,} Rn, Rm | Signed divide | - |
| SEV | - | Send event | - |
| SMLAL | RdLo, RdHi, Rn, Rm | Signed multiply with accumulate (32x32+64), 64-bit result | - |
| SMULL | RdLo, RdHi, Rn, Rm | Signed multiply (32x32), 64-bit result | - |
| SSAT | Rd, #n, Rm {,shift #s} | Signed saturate | Q |
| STM | Rn{!}, reglist | Store multiple registers, increment after | - |
| STMDB, STMEA | Rn{!}, reglist | Store multiple registers, decrement before | - |
| STMFD, STMIA | Rn{!}, reglist | Store multiple registers, increment after | - |
| STR | Rt, [Rn {, #offset}] | Store register word | - |
| STRB, STRBT | Rt, [Rn {, #offset}] | Store register byte | - |
| STRD | Rt, Rt2, [Rn {, #offset}] | Store register two words | - |
| STREX | Rt, Rt, [Rn {, #offset}] | Store register exclusive | - |
| STREXB | Rd, Rt, [Rn] | Store register exclusive byte | - |
| STREXH | Rd, Rt, [Rn] | Store register exclusive halfword | - |
| STRH, STRHT | Rt, [Rn {, #offset}] | Store register halfword | - |
| STRSB, STRSBT | Rt, [Rn {, #offset}] | Store register signed byte | - |
| STRSH, STRSHT | Rt, [Rn {, #offset}] | Store register signed halfword | - |
| STRT | Rt, [Rn {, #offset}] | Store register word | - |
| SUB, SUBS | {Rd,} Rn, Op2 | Subtract | N,Z,C,V |
| SUB, SUBW | {Rd,} Rn, #imm12 | Subtract 12-bit constant | N,Z,C,V |
| SVC | #imm | Supervisor call | - |
| SXTB | {Rd,} Rm {,ROR #n} | Sign extend a byte | - |
| SXTH | {Rd,} Rm {,ROR #n} | Sign extend a halfword | - |
| TBB | [Rn, Rm] | Table branch byte | - |
| TBH | [Rn, Rm, LSL #1] | Table branch halfword | - |
| TEQ | Rn, Op2 | Test equivalence | N,Z,C |
| TST | Rn, Op2 | Test | N,Z,C |
| UBFX | Rd, Rn, #lsb, #width | Unsigned bit field extract | - |
| UDIV | {Rd,} Rn, Rm | Unsigned divide | - |
| UMLAL | RdLo, RdHi, Rn, Rm | Unsigned multiply with accumulate (32x32+32+32), 64-bit result | - |
| UMULL | RdLo, RdHi, Rn, Rm | Unsigned multiply (32x 2), 64-bit result | - |
| USAT | Rd, #n, Rm {,shift #s} | Unsigned Saturate | Q |
| UXTB | {Rd,} Rm, {,ROR #n} | Zero extend a Byte | - |
| UXTH | {Rd,} Rm, {,ROR #n} | Zero extend a Halfword | - |
| WFE | - | Wait for event | - |
| WFI | - | Wait for interrupt | - |

# 3 Cortex-M3 Peripherals

This chapter provides information on the Stellaris® implementation of the Cortex-M3 processor peripherals, including:

- SysTick (see page 95)

  Provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism.

- Nested Vectored Interrupt Controller (NVIC) (see page 96)
  - Facilitates low-latency exception and interrupt handling
  - Controls power management
  - Implements system control registers

- System Control Block (SCB) (see page 98)

  Provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

- Memory Protection Unit (MPU) (see page 98)

  Supports the standard ARMv7 Protected Memory System Architecture (PMSA) model. The MPU provides full support for protection regions, overlapping protection regions, access permissions, and exporting memory attributes to the system.

Table 3-1 on page 95 shows the address map of the Private Peripheral Bus (PPB). Some peripheral register regions are split into two address regions, as indicated by two addresses listed.

**Table 3-1. Core Peripheral Register Regions**

| Address | Core Peripheral | Description (see page ...) |
|---|---|---|
| 0xE000.E010-0xE000.E01F | System Timer | 95 |
| 0xE000.E100-0xE000.E4EF<br>0xE000.EF00-0xE000.EF03 | Nested Vectored Interrupt Controller | 96 |
| 0xE000.ED00-0xE000.ED3F | System Control Block | 98 |
| 0xE000.ED90-0xE000.EDB8 | Memory Protection Unit | 98 |

## 3.1 Functional Description

This chapter provides information on the Stellaris implementation of the Cortex-M3 processor peripherals: SysTick, NVIC, SCB and MPU.

### 3.1.1 System Timer (SysTick)

Cortex-M3 includes an integrated system timer, SysTick, which provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism. The counter can be used in several different ways, for example as:

- An RTOS tick timer that fires at a programmable rate (for example, 100 Hz) and invokes a SysTick routine.

- A high-speed alarm timer using the system clock.

- A variable rate alarm or signal timer—the duration is range-dependent on the reference clock used and the dynamic range of the counter.
- A simple counter used to measure time to completion and time used.
- An internal clock source control based on missing/meeting durations. The COUNT bit in the **STCTRL** control and status register can be used to determine if an action completed within a set duration, as part of a dynamic clock management control loop.

The timer consists of three registers:

- **SysTick Control and Status (STCTRL)**: A control and status counter to configure its clock, enable the counter, enable the SysTick interrupt, and determine counter status.
- **SysTick Reload Value (STRELOAD)**: The reload value for the counter, used to provide the counter's wrap value.
- **SysTick Current Value (STCURRENT)**: The current value of the counter.

When enabled, the timer counts down on each clock from the reload value to zero, reloads (wraps) to the value in the **STRELOAD** register on the next clock edge, then decrements on subsequent clocks. Clearing the **STRELOAD** register disables the counter on the next wrap. When the counter reaches zero, the COUNT status bit is set. The COUNT bit clears on reads.

Writing to the **STCURRENT** register clears the register and the COUNT status bit. The write does not trigger the SysTick exception logic. On a read, the current value is the value of the register at the time the register is accessed.

The SysTick counter runs on the system clock. If this clock signal is stopped for low power mode, the SysTick counter stops. Ensure software uses aligned word accesses to access the SysTick registers.

**Note:** When the processor is halted for debugging, the counter does not decrement.

### 3.1.2 Nested Vectored Interrupt Controller (NVIC)

This section describes the Nested Vectored Interrupt Controller (NVIC) and the registers it uses. The NVIC supports:

- 28 interrupts.
- A programmable priority level of 0-7 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Low-latency exception and interrupt handling.
- Level and pulse detection of interrupt signals.
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- An external Non-maskable interrupt (NMI).

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead, providing low latency exception handling.

#### 3.1.2.1 Level-Sensitive and Pulse Interrupts

The processor supports both level-sensitive and pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request. A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock. To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC detects the pulse and latches the interrupt.

When the processor enters the ISR, it automatically removes the pending state from the interrupt (see “Hardware and Software Control of Interrupts” on page 97 for more information). For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. As a result, the peripheral can hold the interrupt signal asserted until it no longer needs servicing.

#### 3.1.2.2 Hardware and Software Control of Interrupts

The Cortex-M3 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- The NVIC detects that the interrupt signal is High and the interrupt is not active.
- The NVIC detects a rising edge on the interrupt signal.
- Software writes to the corresponding interrupt set-pending register bit, or to the **Software Trigger Interrupt (SWTRIG)** register to make a Software-Generated Interrupt pending. See the `INT` bit in the **PEND0** register on page 114 or **SWTRIG** on page 122.

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt, changing the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.
  - For a pulse interrupt, the NVIC continues to monitor the interrupt signal, and if this is pulsed the state of the interrupt changes to pending and active. In this case, when the processor returns from the ISR the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR.

    If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit
  - For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.

– For a pulse interrupt, the state of the interrupt changes to inactive, if the state was pending or to active, if the state was active and pending.

### 3.1.3 System Control Block (SCB)

The System Control Block (SCB) provides system implementation information and system control, including configuration, control, and reporting of the system exceptions.

### 3.1.4 Memory Protection Unit (MPU)

This section describes the Memory protection unit (MPU). The MPU divides the memory map into a number of regions and defines the location, size, access permissions, and memory attributes of each region. The MPU supports independent attribute settings for each region, overlapping regions, and export of memory attributes to the system.

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M3 MPU defines eight separate memory regions, 0-7, and a background region.

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M3 MPU memory map is unified, meaning that instruction accesses and data accesses have the same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a memory management fault, causing a fault exception and possibly causing termination of the process in an OS environment. In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

Configuration of MPU regions is based on memory types (see “Memory Regions, Types and Attributes” on page 74 for more information).

Table 3-2 on page 98 shows the possible MPU region attributes. See the section called “MPU Configuration for a Stellaris Microcontroller” on page 102 for guidelines for programming a microcontroller implementation.

**Table 3-2. Memory Attributes Summary**

| Memory Type | Description |
|---|---|
| Strongly Ordered | All accesses to Strongly Ordered memory occur in program order. |
| Device | Memory-mapped peripherals |
| Normal | Normal memory |

To avoid unexpected behavior, disable the interrupts before updating the attributes of a region that the interrupt handlers might access.

Ensure software uses aligned accesses of the correct size to access MPU registers:

- Except for the **MPU Region Attribute and Size (MPUATTR)** register, all MPU registers must be accessed with aligned word accesses.
- The **MPUATTR** register can be accessed with byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to MPU registers.

When setting up the MPU, and if the MPU has previously been programmed, disable unused regions to prevent any previous region settings from affecting the new MPU setup.

#### 3.1.4.1 Updating an MPU Region

To update the attributes for an MPU region, the **MPU Region Number (MPUNUMBER)**, **MPU Region Base Address (MPUBASE)** and **MPUATTR** registers must be updated. Each register can be programmed separately or with a multiple-word write to program all of these registers. You can use the **MPUBASEx** and **MPUATTRx** aliases to program up to four regions simultaneously using an STM instruction.

***Updating an MPU Region Using Separate Words***

This example simple code configures one region:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPUNUMBER        ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]       ; Region Number
STR R4, [R0, #0x4]       ; Region Base Address
STRH R2, [R0, #0x8]      ; Region Size and Enable
STRH R3, [R0, #0xA]      ; Region Attribute
```

Disable a region before writing new region settings to the MPU if you have previously enabled the region being changed. For example:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPUNUMBER         ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]        ; Region Number
BIC R2, R2, #1            ; Disable
STRH R2, [R0, #0x8]       ; Region Size and Enable
STR R4, [R0, #0x4]        ; Region Base Address
STRH R3, [R0, #0xA]       ; Region Attribute
ORR R2, #1                ; Enable
STRH R2, [R0, #0x8]       ; Region Size and Enable
```

Software must use memory barrier instructions:

- Before MPU setup, if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings.
- After MPU setup, if it includes memory transfers that must use the new MPU settings.

However, memory barrier instructions are not required if the MPU setup process starts by entering an exception handler, or is followed by an exception return, because the exception entry and exception return mechanism cause memory barrier behavior.

Software does not need any memory barrier instructions during MPU setup, because it accesses the MPU through the Private Peripheral Bus (PPB), which is a Strongly Ordered memory region.

For example, if all of the memory access behavior is intended to take effect immediately after the programming sequence, then a DSB instruction and an ISB instruction should be used. A DSB is required after changing MPU settings, such as at the end of context switch. An ISB is required if the code that programs the MPU region or regions is entered using a branch or call. If the programming sequence is entered using a return from exception, or by taking an exception, then an ISB is not required.

### *Updating an MPU Region Using Multi-Word Writes*

The MPU can be programmed directly using multi-word writes, depending how the information is divided. Consider the following reprogramming:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPUNUMBER  ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]  ; Region Number
STR R2, [R0, #0x4]  ; Region Base Address
STR R3, [R0, #0x8]  ; Region Attribute, Size and Enable
```

An STM instruction can be used to optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPUNUMBER  ; 0xE000ED98, MPU region number register
STM R0, {R1-R3}     ; Region number, address, attribute, size and enable
```

This operation can be done in two words for pre-packed information, meaning that the **MPU Region Base Address (MPUBASE)** register (see page 154) contains the required region number and has the VALID bit set. This method can be used when the data is statically packed, for example in a boot loader:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0, =MPUBASE    ; 0xE000ED9C, MPU Region Base register
STR R1, [R0, #0x0]  ; Region base address and region number combined
                    ; with VALID (bit 4) set
STR R2, [R0, #0x4]  ; Region Attribute, Size and Enable
```

### *Subregions*

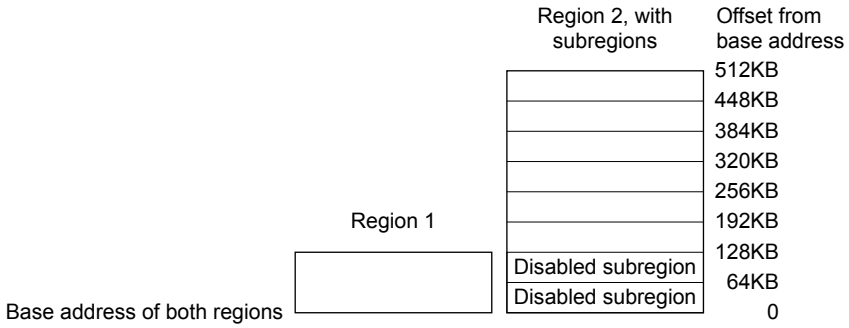
Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the **MPU Region Attribute and Size (MPUATTR)** register (see page 156) to disable a subregion. The least-significant bit of the SRD field controls the first subregion, and the most-significant bit controls the last subregion. Disabling a subregion means another region overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion, the MPU issues a fault.

Regions of 32, 64, and 128 bytes do not support subregions. With regions of these sizes, the SRD field must be configured to 0x00, otherwise the MPU behavior is unpredictable.

***Example of SRD Use***

Two regions with the same base address overlap. Region one is 128 KB, and region two is 512 KB. To ensure the attributes from region one apply to the first 128 KB region, configure the SRD field for region two to 0x03 to disable the first two subregions, as Figure 3-1 on page 101 shows.

**Figure 3-1. SRD Use Example**




### 3.1.4.2 MPU Access Permission Attributes

The access permission bits, TEX, S, C, B, AP, and XN of the **MPUATTR** register, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

Table 3-3 on page 101 shows the encodings for the TEX, C, B, and S access permission bits. All encodings are shown for completeness, however the current implementation of the Cortex-M3 does not support the concept of cacheability or shareability. Refer to the section called “MPU Configuration for a Stellaris Microcontroller” on page 102 for information on programming the MPU for Stellaris implementations.

**Table 3-3. TEX, S, C, and B Bit Field Encoding**

| TEX | S | C | B | Memory Type | Shareability | Other Attributes |
|---|---|---|---|---|---|---|
| 000b | x[a] | 0 | 0 | Strongly Ordered | Shareable | - |
| 000 | x[a] | 0 | 1 | Device | Shareable | - |
| 000 | 0 | 1 | 0 | Normal | Not shareable | Outer and inner write-through. No write allocate. |
| 000 | 1 | 1 | 0 | Normal | Shareable | |
| 000 | 0 | 1 | 1 | Normal | Not shareable | |
| 000 | 1 | 1 | 1 | Normal | Shareable | |
| 001 | 0 | 0 | 0 | Normal | Not shareable | Outer and inner noncacheable. |
| 001 | 1 | 0 | 0 | Normal | Shareable | |
| 001 | x[a] | 0 | 1 | Reserved encoding | - | - |
| 001 | x[a] | 1 | 0 | Reserved encoding | - | - |
| 001 | 0 | 1 | 1 | Normal | Not shareable | Outer and inner write-back. Write and read allocate. |
| 001 | 1 | 1 | 1 | Normal | Shareable | |
| 010 | x[a] | 0 | 0 | Device | Not shareable | Nonshared Device. |
| 010 | x[a] | 0 | 1 | Reserved encoding | - | - |
| 010 | x[a] | 1 | x[a] | Reserved encoding | - | - |

**Table 3-3. TEX, S, C, and B Bit Field Encoding *(continued)***

| TEX | S | C | B | Memory Type | Shareability | Other Attributes |
|---|---|---|---|---|---|---|
| 1BB | 0 | A | A | Normal | Not shareable | Cached memory (BB = outer policy, AA = inner policy).<br>See Table 3-4 for the encoding of the AA and BB bits. |
| 1BB | 1 | A | A | Normal | Shareable | |

a. The MPU ignores the value of this bit.

Table 3-4 on page 102 shows the cache policy for memory attribute encodings with a TEX value in the range of 0x4-0x7.

**Table 3-4. Cache Policy for Memory Attribute Encoding**

| Encoding, AA or BB | Corresponding Cache Policy |
|---|---|
| 00 | Non-cacheable |
| 01 | Write back, write and read allocate |
| 10 | Write through, no write allocate |
| 11 | Write back, no write allocate |

Table 3-5 on page 102 shows the AP encodings in the **MPUATTR** register that define the access permissions for privileged and unprivileged software.

**Table 3-5. AP Bit Field Encoding**

| AP Bit Field | Privileged Permissions | Unprivileged Permissions | Description |
|---|---|---|---|
| 000 | No access | No access | All accesses generate a permission fault. |
| 001 | R/W | No access | Access from privileged software only. |
| 010 | R/W | RO | Writes by unprivileged software generate a permission fault. |
| 011 | R/W | R/W | Full access. |
| 100 | Unpredictable | Unpredictable | Reserved. |
| 101 | RO | No access | Reads by privileged software only. |
| 110 | RO | RO | Read-only, by privileged or unprivileged software. |
| 111 | RO | RO | Read-only, by privileged or unprivileged software. |

### *MPU Configuration for a Stellaris Microcontroller*

Stellaris microcontrollers have only a single processor and no caches. As a result, the MPU should be programmed as shown in Table 3-6 on page 102.

**Table 3-6. Memory Region Attributes for Stellaris Microcontrollers**

| Memory Region | TEX | S | C | B | Memory Type and Attributes |
|---|---|---|---|---|---|
| Flash memory | 000b | 0 | 1 | 0 | Normal memory, non-shareable, write-through |
| Internal SRAM | 000b | 1 | 1 | 0 | Normal memory, shareable, write-through |
| External SRAM | 000b | 1 | 1 | 1 | Normal memory, shareable, write-back, write-allocate |
| Peripherals | 000b | 1 | 0 | 1 | Device memory, shareable |

In current Stellaris microcontroller implementations, the shareability and cache policy attributes do not affect the system behavior. However, using these settings for the MPU regions can make the application code more portable. The values given are for typical situations.

#### 3.1.4.3 MPU Mismatch

When an access violates the MPU permissions, the processor generates a memory management fault (see “Exceptions and Interrupts” on page 72 for more information). The **MFAULTSTAT** register indicates the cause of the fault. See page 141 for more information.

## 3.2 Register Map

Table 3-7 on page 103 lists the Cortex-M3 Peripheral SysTick, NVIC, MPU and SCB registers. The offset listed is a hexadecimal increment to the register's address, relative to the Core Peripherals base address of 0xE000.E000.

**Note:** Register spaces that are not used are reserved for future or internal use. Software should not modify any reserved memory address.

**Table 3-7. Peripherals Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| **System Timer (SysTick) Registers** | | | | | |
| 0x010 | STCTRL | R/W | 0x0000.0000 | SysTick Control and Status Register | 106 |
| 0x014 | STRELOAD | R/W | 0x0000.0000 | SysTick Reload Value Register | 108 |
| 0x018 | STCURRENT | R/WC | 0x0000.0000 | SysTick Current Value Register | 109 |
| **Nested Vectored Interrupt Controller (NVIC) Registers** | | | | | |
| 0x100 | EN0 | R/W | 0x0000.0000 | Interrupt 0-31 Set Enable | 110 |
| 0x104 | EN1 | R/W | 0x0000.0000 | Interrupt 32-47 Set Enable | 111 |
| 0x180 | DIS0 | R/W | 0x0000.0000 | Interrupt 0-31 Clear Enable | 112 |
| 0x184 | DIS1 | R/W | 0x0000.0000 | Interrupt 32-47 Clear Enable | 113 |
| 0x200 | PEND0 | R/W | 0x0000.0000 | Interrupt 0-31 Set Pending | 114 |
| 0x204 | PEND1 | R/W | 0x0000.0000 | Interrupt 32-47 Set Pending | 115 |
| 0x280 | UNPEND0 | R/W | 0x0000.0000 | Interrupt 0-31 Clear Pending | 116 |
| 0x284 | UNPEND1 | R/W | 0x0000.0000 | Interrupt 32-47 Clear Pending | 117 |
| 0x300 | ACTIVE0 | RO | 0x0000.0000 | Interrupt 0-31 Active Bit | 118 |
| 0x304 | ACTIVE1 | RO | 0x0000.0000 | Interrupt 32-47 Active Bit | 119 |
| 0x400 | PRI0 | R/W | 0x0000.0000 | Interrupt 0-3 Priority | 120 |
| 0x404 | PRI1 | R/W | 0x0000.0000 | Interrupt 4-7 Priority | 120 |
| 0x408 | PRI2 | R/W | 0x0000.0000 | Interrupt 8-11 Priority | 120 |
| 0x40C | PRI3 | R/W | 0x0000.0000 | Interrupt 12-15 Priority | 120 |
| 0x410 | PRI4 | R/W | 0x0000.0000 | Interrupt 16-19 Priority | 120 |

**Table 3-7. Peripherals Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x414 | PRI5 | R/W | 0x0000.0000 | Interrupt 20-23 Priority | 120 |
| 0x418 | PRI6 | R/W | 0x0000.0000 | Interrupt 24-27 Priority | 120 |
| 0x41C | PRI7 | R/W | 0x0000.0000 | Interrupt 28-31 Priority | 120 |
| 0x420 | PRI8 | R/W | 0x0000.0000 | Interrupt 32-35 Priority | 120 |
| 0x424 | PRI9 | R/W | 0x0000.0000 | Interrupt 36-39 Priority | 120 |
| 0x428 | PRI10 | R/W | 0x0000.0000 | Interrupt 40-43 Priority | 120 |
| 0x42C | PRI11 | R/W | 0x0000.0000 | Interrupt 44-47 Priority | 120 |
| 0xF00 | SWTRIG | WO | 0x0000.0000 | Software Trigger Interrupt | 122 |
| **System Control Block (SCB) Registers** | | | | | |
| 0xD00 | CPUID | RO | 0x411F.C231 | CPU ID Base | 123 |
| 0xD04 | INTCTRL | R/W | 0x0000.0000 | Interrupt Control and State | 124 |
| 0xD08 | VTABLE | R/W | 0x0000.0000 | Vector Table Offset | 127 |
| 0xD0C | APINT | R/W | 0xFA05.0000 | Application Interrupt and Reset Control | 128 |
| 0xD10 | SYSCTRL | R/W | 0x0000.0000 | System Control | 130 |
| 0xD14 | CFGCTRL | R/W | 0x0000.0000 | Configuration and Control | 132 |
| 0xD18 | SYSPRI1 | R/W | 0x0000.0000 | System Handler Priority 1 | 134 |
| 0xD1C | SYSPRI2 | R/W | 0x0000.0000 | System Handler Priority 2 | 135 |
| 0xD20 | SYSPRI3 | R/W | 0x0000.0000 | System Handler Priority 3 | 136 |
| 0xD24 | SYSHNDCTRL | R/W | 0x0000.0000 | System Handler Control and State | 137 |
| 0xD28 | FAULTSTAT | R/W1C | 0x0000.0000 | Configurable Fault Status | 141 |
| 0xD2C | HFAULTSTAT | R/W1C | 0x0000.0000 | Hard Fault Status | 147 |
| 0xD34 | MMADDR | R/W | - | Memory Management Fault Address | 148 |
| 0xD38 | FAULTADDR | R/W | - | Bus Fault Address | 149 |
| **Memory Protection Unit (MPU) Registers** | | | | | |
| 0xD90 | MPUTYPE | RO | 0x0000.0800 | MPU Type | 150 |
| 0xD94 | MPUCTRL | R/W | 0x0000.0000 | MPU Control | 151 |
| 0xD98 | MPUNUMBER | R/W | 0x0000.0000 | MPU Region Number | 153 |
| 0xD9C | MPUBASE | R/W | 0x0000.0000 | MPU Region Base Address | 154 |
| 0xDA0 | MPUATTR | R/W | 0x0000.0000 | MPU Region Attribute and Size | 156 |
| 0xDA4 | MPUBASE1 | R/W | 0x0000.0000 | MPU Region Base Address Alias 1 | 154 |
| 0xDA8 | MPUATTR1 | R/W | 0x0000.0000 | MPU Region Attribute and Size Alias 1 | 156 |
| 0xDAC | MPUBASE2 | R/W | 0x0000.0000 | MPU Region Base Address Alias 2 | 154 |

**Table 3-7. Peripherals Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0xDB0 | MPUATTR2 | R/W | 0x0000.0000 | MPU Region Attribute and Size Alias 2 | 156 |
| 0xDB4 | MPUBASE3 | R/W | 0x0000.0000 | MPU Region Base Address Alias 3 | 154 |
| 0xDB8 | MPUATTR3 | R/W | 0x0000.0000 | MPU Region Attribute and Size Alias 3 | 156 |

## 3.3 System Timer (SysTick) Register Descriptions

This section lists and describes the System Timer registers, in numerical order by address offset.

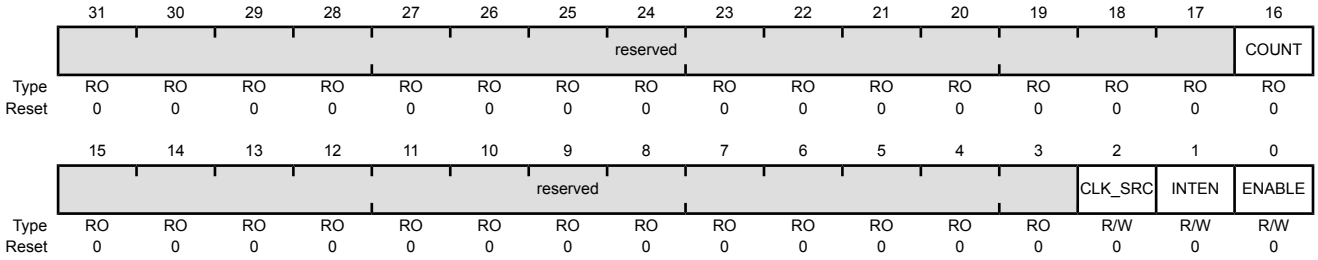
## Register 1: SysTick Control and Status Register (STCTRL), offset 0x010

**Note:** This register can only be accessed from privileged mode.

The SysTick **STCTRL** register enables the SysTick features.

SysTick Control and Status Register (STCTRL)

Base 0xE000.E000
Offset 0x010
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | COUNT | RO | 0 | Count Flag<br><br>Value Description<br>0 The SysTick timer has not counted to 0 since the last time this bit was read.<br>1 The SysTick timer has counted to 0 since the last time this bit was read.<br><br>This bit is cleared by a read of the register or if the **STCURRENT** register is written with any value.<br><br>If read by the debugger using the DAP, this bit is cleared only if the `MasterType` bit in the **AHB-AP Control Register** is clear. Otherwise, the `COUNT` bit is not changed by the debugger read. See the *ARM® Debug Interface V5 Architecture Specification* for more information on `MasterType`. |
| 15:3 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | CLK_SRC | R/W | 0 | Clock Source<br><br>Value Description<br>0 External reference clock. (Not implemented for most Stellaris microcontrollers.)<br>1 System clock<br><br>Because an external reference clock is not implemented, this bit must be set in order for SysTick to operate. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | INTEN | R/W | 0 | Interrupt Enable |

| Value | Description |
|---|---|
| 0 | Interrupt generation is disabled. Software can use the COUNT bit to determine if the counter has ever reached 0. |
| 1 | An interrupt is generated to the NVIC when SysTick counts to 0. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | ENABLE | R/W | 0 | Enable |

| Value | Description |
|---|---|
| 0 | The counter is disabled. |
| 1 | Enables SysTick to operate in a multi-shot way. That is, the counter loads the RELOAD value and begins counting down. On reaching 0, the COUNT bit is set and an interrupt is generated if enabled by INTEN. The counter then loads the RELOAD value again and begins counting. |

## Register 2: SysTick Reload Value Register (STRELOAD), offset 0x014

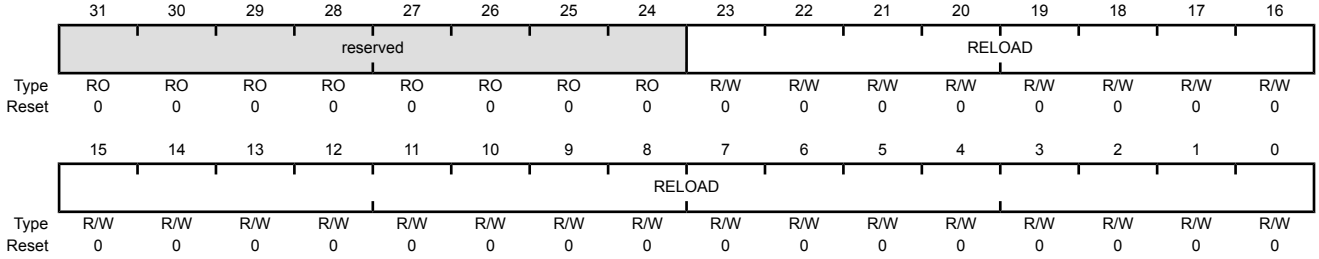
**Note:** This register can only be accessed from privileged mode.

The **STRELOAD** register specifies the start value to load into the **SysTick Current Value (STCURRENT)** register when the counter reaches 0. The start value can be between 0x1 and 0x00FF.FFFF. A start value of 0 is possible but has no effect because the SysTick interrupt and the COUNT bit are activated when counting from 1 to 0.

SysTick can be configured as a multi-shot timer, repeated over and over, firing every N+1 clock pulses, where N is any value from 1 to 0x00FF.FFFF. For example, if a tick interrupt is required every 100 clock pulses, 99 must be written into the RELOAD field.

SysTick Reload Value Register (STRELOAD)

Base 0xE000.E000
Offset 0x014
Type R/W, reset 0x0000.0000



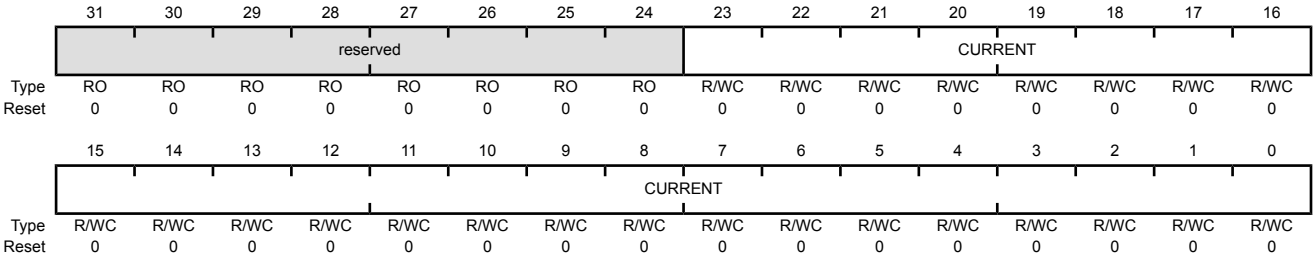

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:24 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:0 | RELOAD | R/W | 0x00.0000 | Reload Value<br>Value to load into the **SysTick Current Value (STCURRENT)** register when the counter reaches 0. |

### Register 3: SysTick Current Value Register (STCURRENT), offset 0x018

**Note:** This register can only be accessed from privileged mode.

The **STCURRENT** register contains the current value of the SysTick counter.

SysTick Current Value Register (STCURRENT)
Base 0xE000.E000
Offset 0x018
Type R/WC, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:24 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:0 | CURRENT | R/WC | 0x00.0000 | Current Value<br>This field contains the current value at the time the register is accessed. No read-modify-write protection is provided, so change with care.<br>This register is write-clear. Writing to it with any value clears the register. Clearing this register also clears the COUNT bit of the **STCTRL** register. |

## 3.4 NVIC Register Descriptions

This section lists and describes the NVIC registers, in numerical order by address offset.

The NVIC registers can only be fully accessed from privileged mode, but interrupts can be pended while in unprivileged mode by enabling the **Configuration and Control (CFGCTRL)** register. Any other unprivileged mode access causes a bus fault.

Ensure software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers.

An interrupt can enter the pending state even if it is disabled.

Before programming the **VTABLE** register to relocate the vector table, ensure the vector table entries of the new vector table are set up for fault handlers, NMI, and all enabled exceptions such as interrupts. For more information, see page 127.

## Register 4: Interrupt 0-31 Set Enable (EN0), offset 0x100

**Note:** This register can only be accessed from privileged mode.

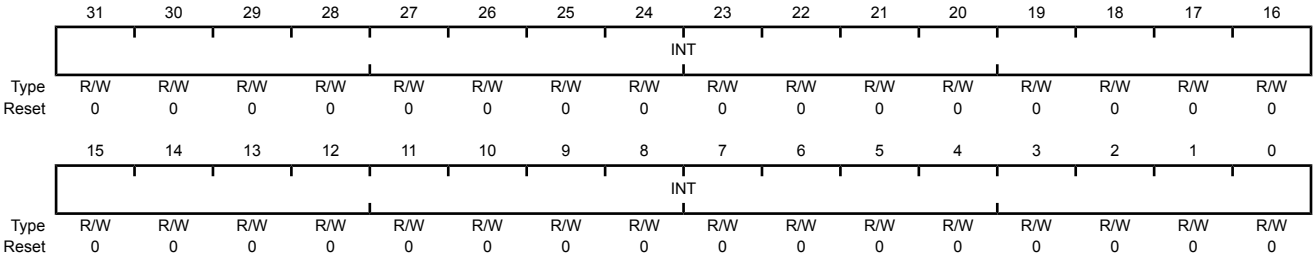
The **EN0** register enables interrupts and shows which interrupts are enabled. Bit 0 corresponds to Interrupt 0; bit 31 corresponds to Interrupt 31.

See Table 2-9 on page 83 for interrupt assignments.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

Interrupt 0-31 Set Enable (EN0)

Base 0xE000.E000
Offset 0x100
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | INT | R/W | 0x0000.0000 | Interrupt Enable |

| Value | Description |
|---|---|
| 0 | On a read, indicates the interrupt is disabled.<br>On a write, no effect. |
| 1 | On a read, indicates the interrupt is enabled.<br>On a write, enables the interrupt. |

A bit can only be cleared by setting the corresponding `INT[n]` bit in the **DISn** register.

## Register 5: Interrupt 32-47 Set Enable (EN1), offset 0x104

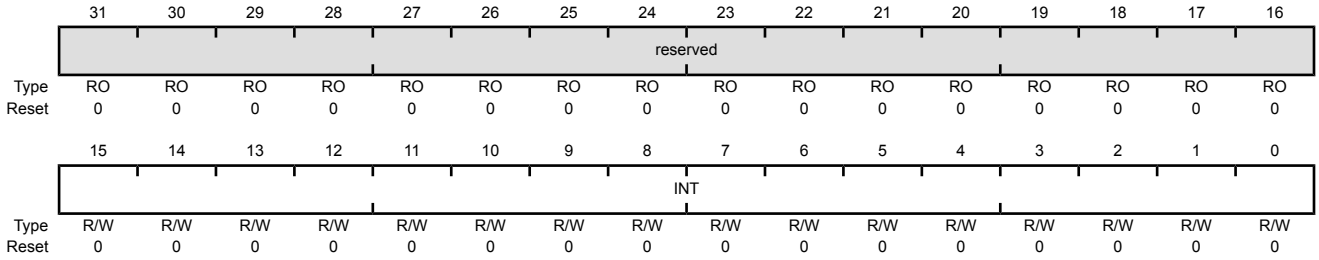
**Note:** This register can only be accessed from privileged mode.

The **EN1** register enables interrupts and shows which interrupts are enabled. Bit 0 corresponds to Interrupt 32; bit 15 corresponds to Interrupt 47. See Table 2-9 on page 83 for interrupt assignments.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

Interrupt 32-47 Set Enable (EN1)

Base 0xE000.E000
Offset 0x104
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | INT | R/W | 0x0.0000 | Interrupt Enable |

| Value | Description |
|---|---|
| 0 | On a read, indicates the interrupt is disabled. |
| | On a write, no effect. |
| 1 | On a read, indicates the interrupt is enabled. |
| | On a write, enables the interrupt. |

A bit can only be cleared by setting the corresponding `INT[n]` bit in the **DIS1** register.

## Register 6: Interrupt 0-31 Clear Enable (DIS0), offset 0x180

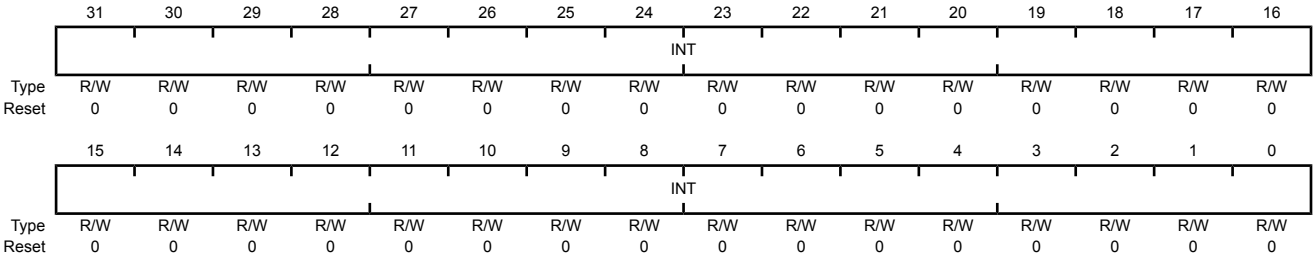
**Note:** This register can only be accessed from privileged mode.

The **DIS0** register disables interrupts. Bit 0 corresponds to Interrupt 0; bit 31 corresponds to Interrupt 31.

See Table 2-9 on page 83 for interrupt assignments.

Interrupt 0-31 Clear Enable (DIS0)

Base 0xE000.E000
Offset 0x180
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | INT | R/W | 0x0000.0000 | Interrupt Disable |

| Value | Description |
|---|---|
| 0 | On a read, indicates the interrupt is disabled.<br>On a write, no effect. |
| 1 | On a read, indicates the interrupt is enabled.<br>On a write, clears the corresponding `INT[n]` bit in the **EN0** register, disabling interrupt [n]. |

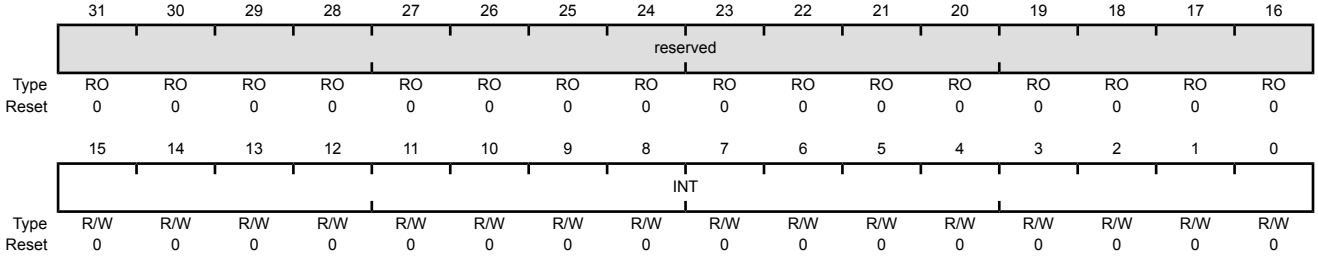
## Register 7: Interrupt 32-47 Clear Enable (DIS1), offset 0x184

**Note:** This register can only be accessed from privileged mode.

The **DIS1** register disables interrupts. Bit 0 corresponds to Interrupt 32; bit 15 corresponds to Interrupt 47. See Table 2-9 on page 83 for interrupt assignments.

Interrupt 32-47 Clear Enable (DIS1)

Base 0xE000.E000
Offset 0x184
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | INT | R/W | 0x0.0000 | Interrupt Disable |

| Value | Description |
|---|---|
| 0 | On a read, indicates the interrupt is disabled.<br>On a write, no effect. |
| 1 | On a read, indicates the interrupt is enabled.<br>On a write, clears the corresponding `INT[n]` bit in the **EN1** register, disabling interrupt [n]. |

## Register 8: Interrupt 0-31 Set Pending (PEND0), offset 0x200

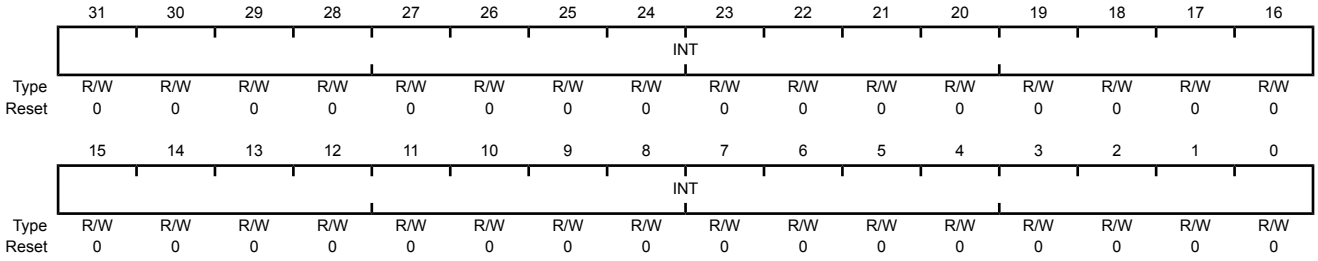
**Note:** This register can only be accessed from privileged mode.

The **PEND0** register forces interrupts into the pending state and shows which interrupts are pending. Bit 0 corresponds to Interrupt 0; bit 31 corresponds to Interrupt 31.

See Table 2-9 on page 83 for interrupt assignments.

Interrupt 0-31 Set Pending (PEND0)

Base 0xE000.E000
Offset 0x200
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | INT | R/W | 0x0000.0000 | Interrupt Set Pending |

| Value | Description |
|---|---|
| 0 | On a read, indicates that the interrupt is not pending.<br>On a write, no effect. |
| 1 | On a read, indicates that the interrupt is pending.<br>On a write, the corresponding interrupt is set to pending even if it is disabled. |

If the corresponding interrupt is already pending, setting a bit has no effect.

A bit can only be cleared by setting the corresponding `INT[n]` bit in the **UNPEND0** register.

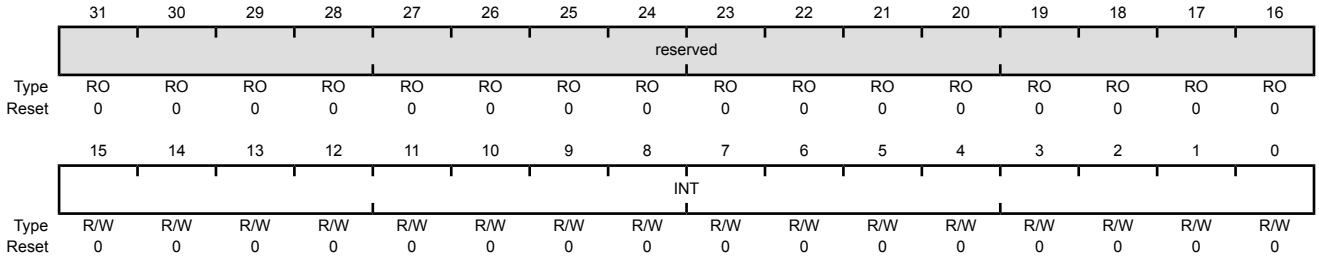
## Register 9: Interrupt 32-47 Set Pending (PEND1), offset 0x204

**Note:** This register can only be accessed from privileged mode.

The **PEND1** register forces interrupts into the pending state and shows which interrupts are pending. Bit 0 corresponds to Interrupt 32; bit 15 corresponds to Interrupt 47. See Table 2-9 on page 83 for interrupt assignments.

Interrupt 32-47 Set Pending (PEND1)

Base 0xE000.E000
Offset 0x204
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | INT | R/W | 0x0.0000 | Interrupt Set Pending |

| Value | Description |
|---|---|
| 0 | On a read, indicates that the interrupt is not pending.<br>On a write, no effect. |
| 1 | On a read, indicates that the interrupt is pending.<br>On a write, the corresponding interrupt is set to pending even if it is disabled. |

If the corresponding interrupt is already pending, setting a bit has no effect.

A bit can only be cleared by setting the corresponding `INT[n]` bit in the **UNPEND1** register.

## Register 10: Interrupt 0-31 Clear Pending (UNPEND0), offset 0x280

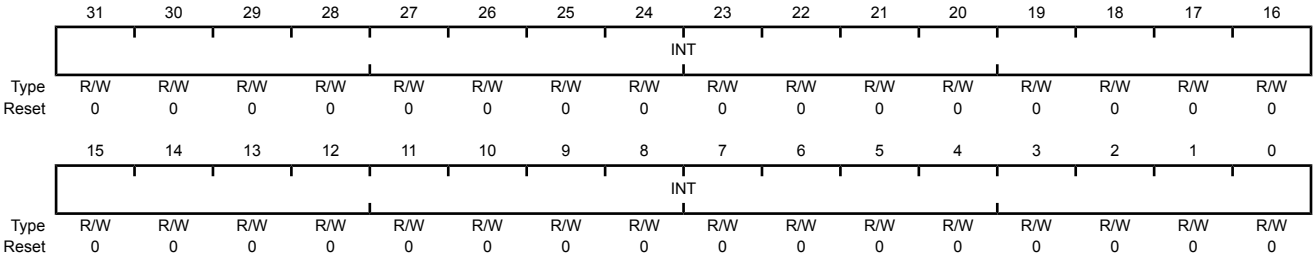
**Note:** This register can only be accessed from privileged mode.

The **UNPEND0** register shows which interrupts are pending and removes the pending state from interrupts. Bit 0 corresponds to Interrupt 0; bit 31 corresponds to Interrupt 31.

See Table 2-9 on page 83 for interrupt assignments.

Interrupt 0-31 Clear Pending (UNPEND0)

Base 0xE000.E000
Offset 0x280
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | INT | R/W | 0x0000.0000 | Interrupt Clear Pending |

| Value | Description |
|---|---|
| 0 | On a read, indicates that the interrupt is not pending.<br>On a write, no effect. |
| 1 | On a read, indicates that the interrupt is pending.<br>On a write, clears the corresponding `INT[n]` bit in the **PEND0** register, so that interrupt [n] is no longer pending.<br>Setting a bit does not affect the active state of the corresponding interrupt. |

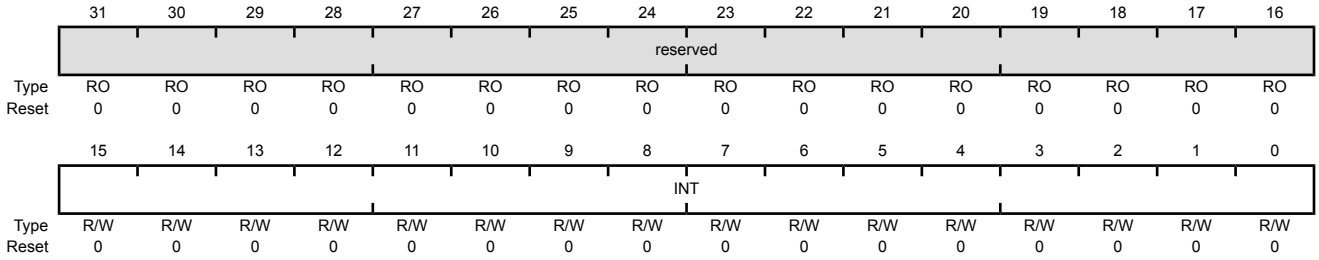
## Register 11: Interrupt 32-47 Clear Pending (UNPEND1), offset 0x284

**Note:** This register can only be accessed from privileged mode.

The **UNPEND1** register shows which interrupts are pending and removes the pending state from interrupts. Bit 0 corresponds to Interrupt 32; bit 15 corresponds to Interrupt 47. See Table 2-9 on page 83 for interrupt assignments.

Interrupt 32-47 Clear Pending (UNPEND1)

Base 0xE000.E000
Offset 0x284
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | INT | R/W | 0x0.0000 | Interrupt Clear Pending |

| Value | Description |
|---|---|
| 0 | On a read, indicates that the interrupt is not pending.<br>On a write, no effect. |
| 1 | On a read, indicates that the interrupt is pending.<br>On a write, clears the corresponding `INT[n]` bit in the **PEND1** register, so that interrupt [n] is no longer pending.<br>Setting a bit does not affect the active state of the corresponding interrupt. |

## Register 12: Interrupt 0-31 Active Bit (ACTIVE0), offset 0x300

**Note:** This register can only be accessed from privileged mode.

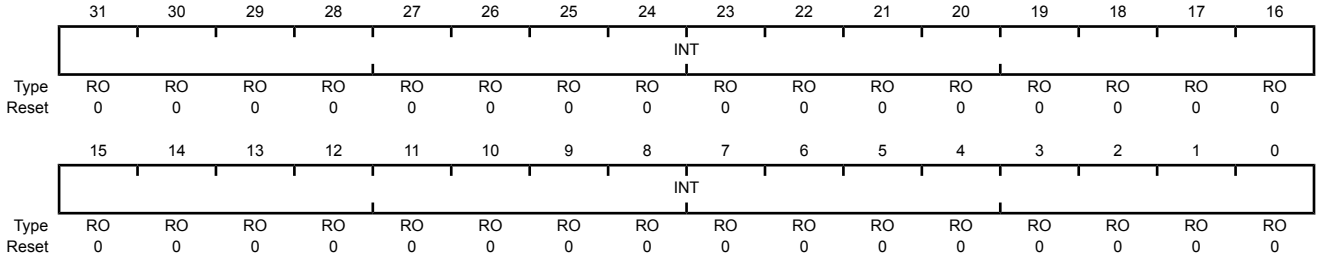
The **ACTIVE0** register indicates which interrupts are active. Bit 0 corresponds to Interrupt 0; bit 31 corresponds to Interrupt 31.

See Table 2-9 on page 83 for interrupt assignments.

**Caution – Do not manually set or clear the bits in this register.**

Interrupt 0-31 Active Bit (ACTIVE0)

Base 0xE000.E000
Offset 0x300
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | INT | RO | 0x0000.0000 | Interrupt Active |

| Value | Description |
|---|---|
| 0 | The corresponding interrupt is not active. |
| 1 | The corresponding interrupt is active, or active and pending. |

## Register 13: Interrupt 32-47 Active Bit (ACTIVE1), offset 0x304

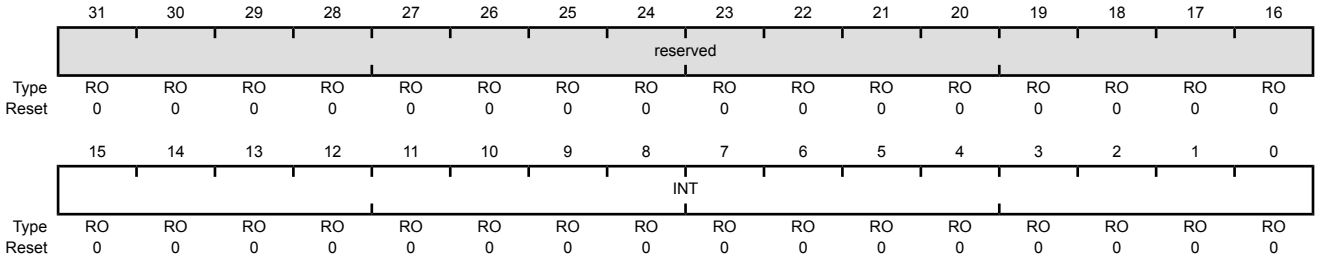
**Note:** This register can only be accessed from privileged mode.

The **ACTIVE1** register indicates which interrupts are active. Bit 0 corresponds to Interrupt 32; bit 15 corresponds to Interrupt 47. See Table 2-9 on page 83 for interrupt assignments.

**Caution – Do not manually set or clear the bits in this register.**

Interrupt 32-47 Active Bit (ACTIVE1)

Base 0xE000.E000
Offset 0x304
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | INT | RO | 0x0.0000 | Interrupt Active |

| Value | Description |
|---|---|
| 0 | The corresponding interrupt is not active. |
| 1 | The corresponding interrupt is active, or active and pending. |

### Register 14: Interrupt 0-3 Priority (PRI0), offset 0x400

### Register 15: Interrupt 4-7 Priority (PRI1), offset 0x404

### Register 16: Interrupt 8-11 Priority (PRI2), offset 0x408

### Register 17: Interrupt 12-15 Priority (PRI3), offset 0x40C

### Register 18: Interrupt 16-19 Priority (PRI4), offset 0x410

### Register 19: Interrupt 20-23 Priority (PRI5), offset 0x414

### Register 20: Interrupt 24-27 Priority (PRI6), offset 0x418

### Register 21: Interrupt 28-31 Priority (PRI7), offset 0x41C

### Register 22: Interrupt 32-35 Priority (PRI8), offset 0x420

### Register 23: Interrupt 36-39 Priority (PRI9), offset 0x424

### Register 24: Interrupt 40-43 Priority (PRI10), offset 0x428

### Register 25: Interrupt 44-47 Priority (PRI11), offset 0x42C

**Note:** This register can only be accessed from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

| PRIn Register Bit Field | Interrupt |
|---|---|
| Bits 31:29 | Interrupt [4n+3] |
| Bits 23:21 | Interrupt [4n+2] |
| Bits 15:13 | Interrupt [4n+1] |
| Bits 7:5 | Interrupt [4n] |

See Table 2-9 on page 83 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The PRIGROUP field in the **Application Interrupt and Reset Control (APINT)** register (see page 128) indicates the position of the binary point that splits the priority and subpriority fields.

These registers can only be accessed from privileged mode.

Interrupt 0-3 Priority (PRI0)

Base 0xE000.E000
Offset 0x400
Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | INTD | | | reserved | | | | | INTC | | | reserved | | | | |
| Type | R/W | R/W | R/W | RO | RO | RO | RO | RO | R/W | R/W | R/W | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | INTB | | | reserved | | | | | INTA | | | reserved | | | | |
| Type | R/W | R/W | R/W | RO | RO | RO | RO | RO | R/W | R/W | R/W | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:29 | INTD | R/W | 0x0 | Interrupt Priority for Interrupt [4n+3]<br>This field holds a priority value, 0-7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt. |
| 28:24 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:21 | INTC | R/W | 0x0 | Interrupt Priority for Interrupt [4n+2]<br>This field holds a priority value, 0-7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt. |
| 20:16 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:13 | INTB | R/W | 0x0 | Interrupt Priority for Interrupt [4n+1]<br>This field holds a priority value, 0-7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt. |
| 12:8 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:5 | INTA | R/W | 0x0 | Interrupt Priority for Interrupt [4n]<br>This field holds a priority value, 0-7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt. |
| 4:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

### Register 26: Software Trigger Interrupt (SWTRIG), offset 0xF00

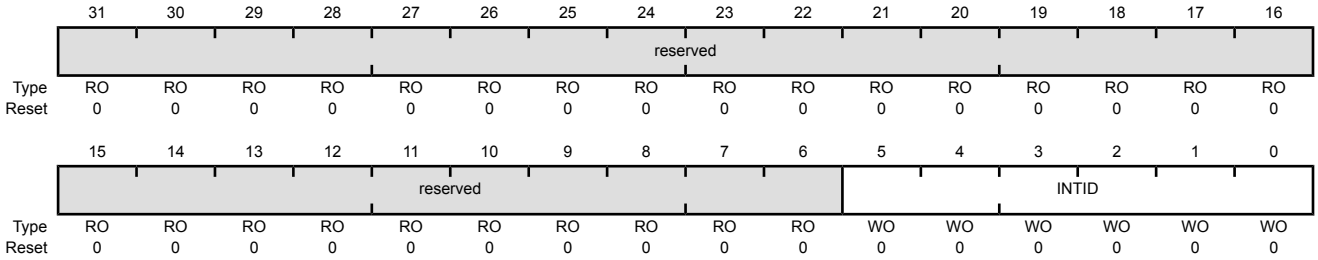
**Note:** Only privileged software can enable unprivileged access to the **SWTRIG** register.

Writing an interrupt number to the **SWTRIG** register generates a Software Generated Interrupt (SGI). See Table 2-9 on page 83 for interrupt assignments.

When the MAINPEND bit in the **Configuration and Control (CFGCTRL)** register (see page 132) is set, unprivileged software can access the **SWTRIG** register.

Software Trigger Interrupt (SWTRIG)

Base 0xE000.E000
Offset 0xF00
Type WO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:0 | INTID | WO | 0x00 | Interrupt ID<br>This field holds the interrupt ID of the required SGI. For example, a value of 0x3 generates an interrupt on IRQ3. |

## 3.5 System Control Block (SCB) Register Descriptions

This section lists and describes the System Control Block (SCB) registers, in numerical order by address offset. The SCB registers can only be accessed from privileged mode.

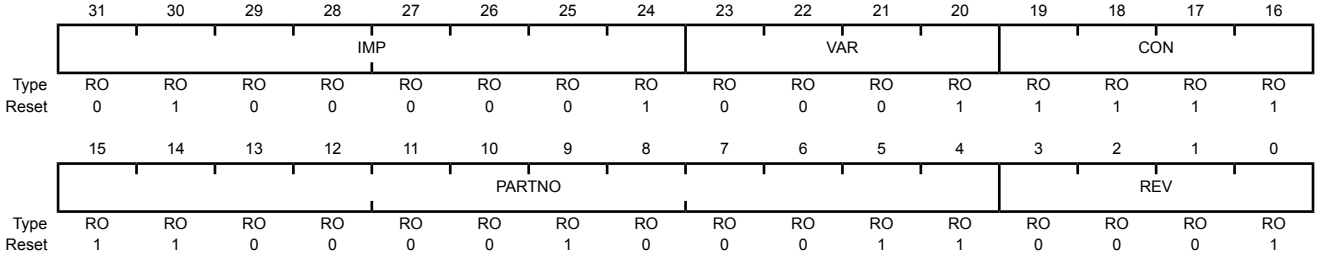
All registers must be accessed with aligned word accesses except for the **FAULTSTAT** and **SYSPRI1**-**SYSPRI3** registers, which can be accessed with byte or aligned halfword or word accesses. The processor does not support unaligned accesses to system control block registers.

## Register 27: CPU ID Base (CPUID), offset 0xD00

**Note:** This register can only be accessed from privileged mode.

The **CPUID** register contains the ARM® Cortex™-M3 processor part number, version, and implementation information.

CPU ID Base (CPUID)
Base 0xE000.E000
Offset 0xD00
Type RO, reset 0x411F.C231




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:24 | IMP | RO | 0x41 | Implementer Code<br><br>Value Description<br>0x41 ARM |
| 23:20 | VAR | RO | 0x1 | Variant Number<br><br>Value Description<br>0x1 The rn value in the rnpn product revision identifier, for example, the 1 in r1p1. |
| 19:16 | CON | RO | 0xF | Constant<br><br>Value Description<br>0xF Always reads as 0xF. |
| 15:4 | PARTNO | RO | 0xC23 | Part Number<br><br>Value Description<br>0xC23 Cortex-M3 processor. |
| 3:0 | REV | RO | 0x1 | Revision Number<br><br>Value Description<br>0x1 The pn value in the rnpn product revision identifier, for example, the 1 in r1p1. |

## Register 28: Interrupt Control and State (INTCTRL), offset 0xD04

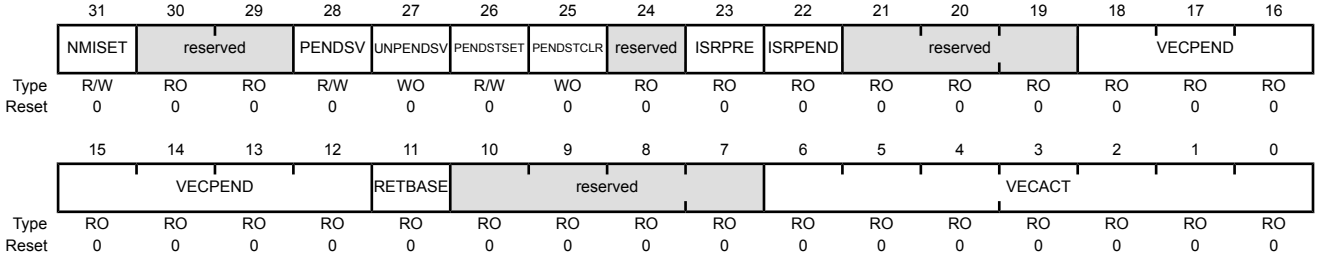
**Note:** This register can only be accessed from privileged mode.

The **INCTRL** register provides a set-pending bit for the NMI exception, and set-pending and clear-pending bits for the PendSV and SysTick exceptions. In addition, bits in this register indicate the exception number of the exception being processed, whether there are preempted active exceptions, the exception number of the highest priority pending exception, and whether any interrupts are pending.

When writing to **INCTRL**, the effect is unpredictable when writing a 1 to both the PENDSV and UNPENDSV bits, or writing a 1 to both the PENDSTSET and PENDSTCLR bits.

Interrupt Control and State (INTCTRL)

Base 0xE000.E000
Offset 0xD04
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | NMISET | R/W | 0 | NMI Set Pending<br><br>Value Description<br>0 On a read, indicates an NMI exception is not pending.<br>On a write, no effect.<br>1 On a read, indicates an NMI exception is pending.<br>On a write, changes the NMI exception state to pending.<br><br>Because NMI is the highest-priority exception, normally the processor enters the NMI exception handler as soon as it registers the setting of this bit, and clears this bit on entering the interrupt handler. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler. |
| 30:29 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 28 | PENDSV | R/W | 0 | PendSV Set Pending<br><br>Value Description<br>0 On a read, indicates a PendSV exception is not pending.<br>On a write, no effect.<br>1 On a read, indicates a PendSV exception is pending.<br>On a write, changes the PendSV exception state to pending.<br><br>Setting this bit is the only way to set the PendSV exception state to pending. This bit is cleared by writing a 1 to the UNPENDSV bit. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 27 | UNPENDSV | WO | 0 | PendSV Clear Pending<br><br>Value Description<br>0 On a write, no effect.<br>1 On a write, removes the pending state from the PendSV exception.<br><br>This bit is write only; on a register read, its value is unknown. |
| 26 | PENDSTSET | R/W | 0 | SysTick Set Pending<br><br>Value Description<br>0 On a read, indicates a SysTick exception is not pending.<br>On a write, no effect.<br>1 On a read, indicates a SysTick exception is pending.<br>On a write, changes the SysTick exception state to pending.<br><br>This bit is cleared by writing a 1 to the PENDSTCLR bit. |
| 25 | PENDSTCLR | WO | 0 | SysTick Clear Pending<br><br>Value Description<br>0 On a write, no effect.<br>1 On a write, removes the pending state from the SysTick exception.<br><br>This bit is write only; on a register read, its value is unknown. |
| 24 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23 | ISRPRE | RO | 0 | Debug Interrupt Handling<br><br>Value Description<br>0 The release from halt does not take an interrupt.<br>1 The release from halt takes an interrupt.<br><br>This bit is only meaningful in Debug mode and reads as zero when the processor is not in Debug mode. |
| 22 | ISRPEND | RO | 0 | Interrupt Pending<br><br>Value Description<br>0 No interrupt is pending.<br>1 An interrupt is pending.<br><br>This bit provides status for all interrupts excluding NMI and Faults. |
| 21:19 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 18:12 | VECPEND | RO | 0x00 | Interrupt Pending Vector Number<br>This field contains the exception number of the highest priority pending enabled exception. The value indicated by this field includes the effect of the **BASEPRI** and **FAULTMASK** registers, but not any effect of the **PRIMASK** register.<br>Value Description<br>0x00 No exceptions are pending<br>0x01 Reserved<br>0x02 NMI<br>0x03 Hard fault<br>0x04 Memory management fault<br>0x05 Bus fault<br>0x06 Usage fault<br>0x07-0x0A Reserved<br>0x0B SVCall<br>0x0C Reserved for Debug<br>0x0D Reserved<br>0x0E PendSV<br>0x0F SysTick<br>0x10 Interrupt Vector 0<br>0x11 Interrupt Vector 1<br>... ...<br>0x3F Interrupt Vector 47<br>0x40-0x7F Reserved |
| 11 | RETBASE | RO | 0 | Return to Base<br>Value Description<br>0 There are preempted active exceptions to execute.<br>1 There are no active exceptions, or the currently executing exception is the only active exception.<br>This bit provides status for all interrupts excluding NMI and Faults. This bit only has meaning if the processor is currently executing an ISR (the **Interrupt Program Status (IPSR)** register is non-zero). |
| 10:7 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | VECACT | RO | 0x00 | Interrupt Pending Vector Number<br>This field contains the active exception number. The exception numbers can be found in the description for the VECPEND field. If this field is clear, the processor is in Thread mode. This field contains the same value as the ISRNUM field in the **IPSR** register.<br>Subtract 16 from this value to obtain the IRQ number required to index into the **Interrupt Set Enable (ENn)**, **Interrupt Clear Enable (DISn**), **Interrupt Set Pending (PENDn)**, **Interrupt Clear Pending (UNPENDn)**, and **Interrupt Priority (PRIn)** registers (see page 64). |

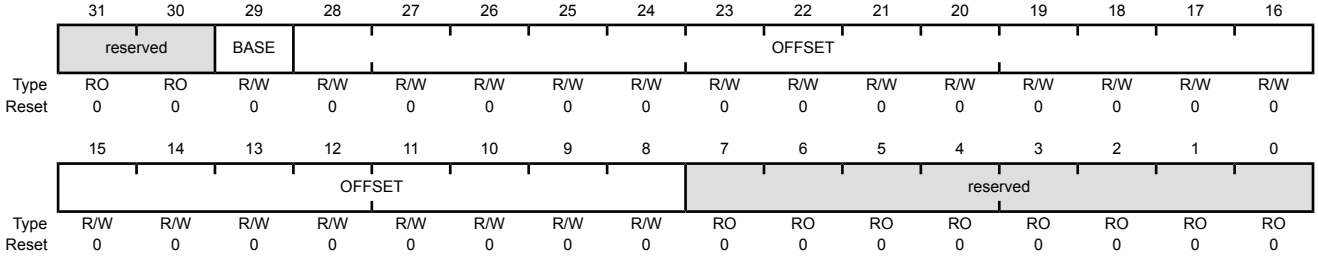
## Register 29: Vector Table Offset (VTABLE), offset 0xD08

**Note:** This register can only be accessed from privileged mode.

The **VTABLE** register indicates the offset of the vector table base address from memory address 0x0000.0000.

Vector Table Offset (VTABLE)

Base 0xE000.E000
Offset 0xD08
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 29 | BASE | R/W | 0 | Vector Table Base<br>Value Description<br>0 The vector table is in the code memory region.<br>1 The vector table is in the SRAM memory region. |
| 28:8 | OFFSET | R/W | 0x000.00 | Vector Table Offset<br>When configuring the OFFSET field, the offset must be aligned to the number of exception entries in the vector table. Because there are 47 interrupts, the offset must be aligned on a 256-byte boundary. |
| 7:0 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 30: Application Interrupt and Reset Control (APINT), offset 0xD0C

**Note:** This register can only be accessed from privileged mode.

The **APINT** register provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system. To write to this register, 0x05FA must be written to the VECTKEY field, otherwise the write is ignored.

The PRIGROUP field indicates the position of the binary point that splits the INTx fields in the **Interrupt Priority (PRIx)** registers into separate group priority and subpriority fields. Table 3-8 on page 128 shows how the PRIGROUP value controls this split. The bit numbers in the Group Priority Field and Subpriority Field columns in the table refer to the bits in the INTA field. For the INTB field, the corresponding bits are 15:13; for INTC, 23:21; and for INTD, 31:29.

**Note:** Determining preemption of an exception uses only the group priority field.

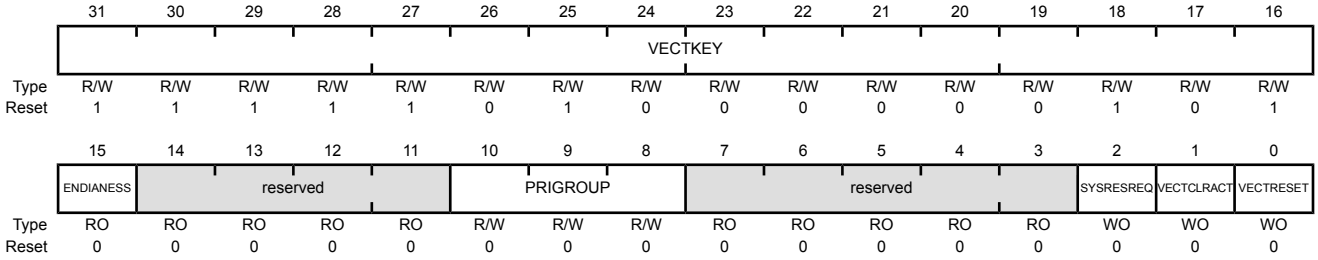
**Table 3-8. Interrupt Priority Levels**

| PRIGROUP Bit Field | Binary Point[a] | Group Priority Field | Subpriority Field | Group Priorities | Subpriorities |
|---|---|---|---|---|---|
| 0x0 - 0x4 | bxxx. | [7:5] | None | 8 | 1 |
| 0x5 | bxx.y | [7:6] | [5] | 4 | 2 |
| 0x6 | bx.yy | [7] | [6:5] | 2 | 4 |
| 0x7 | b.yyy | None | [7:5] | 1 | 8 |

a. INTx field showing the binary point. An x denotes a group priority field bit, and a y denotes a subpriority field bit.

Application Interrupt and Reset Control (APINT)

Base 0xE000.E000
Offset 0xD0C
Type R/W, reset 0xFA05.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | VECTKEY | R/W | 0xFA05 | Register Key<br>This field is used to guard against accidental writes to this register. 0x05FA must be written to this field in order to change the bits in this register. On a read, 0xFA05 is returned. |
| 15 | ENDIANESS | RO | 0 | Data Endianess<br>The Stellaris implementation uses only little-endian mode so this is cleared to 0. |
| 14:11 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 10:8 | PRIGROUP | R/W | 0x0 | Interrupt Priority Grouping<br>This field determines the split of group priority from subpriority (see Table 3-8 on page 128 for more information). |
| 7:3 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | SYSRESREQ | WO | 0 | System Reset Request<br>Value Description<br>0 No effect.<br>1 Resets the core and all on-chip peripherals except the Debug interface.<br>This bit is automatically cleared during the reset of the core and reads as 0. |
| 1 | VECTCLRACT | WO | 0 | Clear Active NMI / Fault<br>This bit is reserved for Debug use and reads as 0. This bit must be written as a 0, otherwise behavior is unpredictable. |
| 0 | VECTRESET | WO | 0 | System Reset<br>This bit is reserved for Debug use and reads as 0. This bit must be written as a 0, otherwise behavior is unpredictable. |

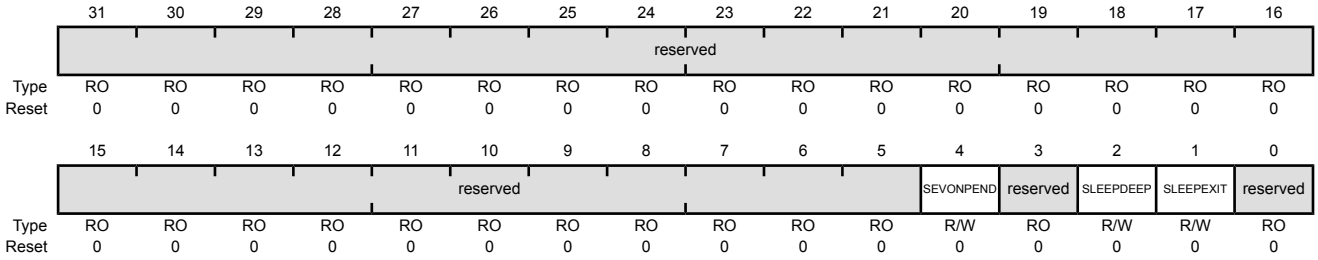
## Register 31: System Control (SYSCTRL), offset 0xD10

**Note:** This register can only be accessed from privileged mode.

The **SYSCTRL** register controls features of entry to and exit from low-power state.

System Control (SYSCTRL)

Base 0xE000.E000
Offset 0xD10
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SEVONPEND | R/W | 0 | Wake Up on Pending<br><br>Value Description<br>0 Only enabled interrupts or events can wake up the processor; disabled interrupts are excluded.<br>1 Enabled events and all interrupts, including disabled interrupts, can wake up the processor.<br><br>When an event or interrupt enters the pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE.<br>The processor also wakes up on execution of a SEV instruction or an external event. |
| 3 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | SLEEPDEEP | R/W | 0 | Deep Sleep Enable<br><br>Value Description<br>0 Use Sleep mode as the low power mode.<br>1 Use Deep-sleep mode as the low power mode. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | SLEEPEXIT | R/W | 0 | Sleep on ISR Exit<br><br>Value Description<br>0 When returning from Handler mode to Thread mode, do not sleep when returning to Thread mode.<br>1 When returning from Handler mode to Thread mode, enter sleep or deep sleep on return from an ISR.<br><br>Setting this bit enables an interrupt-driven application to avoid returning to an empty main application. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

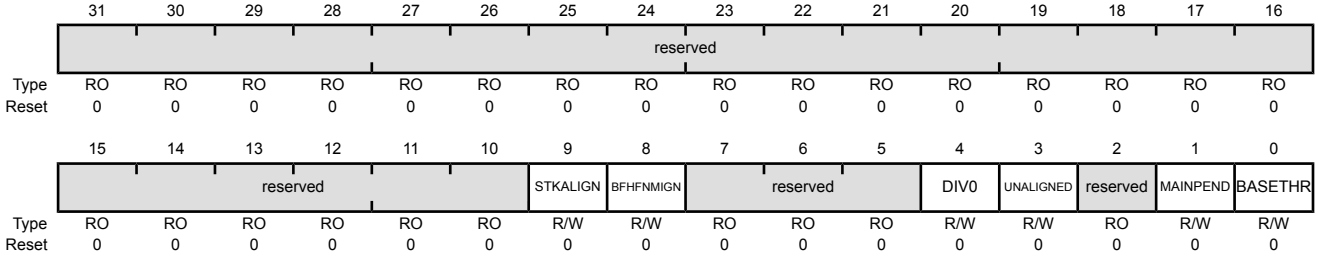
## Register 32: Configuration and Control (CFGCTRL), offset 0xD14

**Note:** This register can only be accessed from privileged mode.

The **CFGCTRL** register controls entry to Thread mode and enables: the handlers for NMI, hard fault and faults escalated by the **FAULTMASK** register to ignore bus faults; trapping of divide by zero and unaligned accesses; and access to the **SWTRIG** register by unprivileged software (see page 122).

Configuration and Control (CFGCTRL)

Base 0xE000.E000
Offset 0xD14
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9 | STKALIGN | R/W | 0 | Stack Alignment on Exception Entry<br><br>Value Description<br>0 The stack is 4-byte aligned.<br>1 The stack is 8-byte aligned.<br><br>On exception entry, the processor uses bit 9 of the stacked **PSR** to indicate the stack alignment. On return from the exception, it uses this stacked bit to restore the correct stack alignment. |
| 8 | BFHFNMIGN | R/W | 0 | Ignore Bus Fault in NMI and Fault<br><br>This bit enables handlers with priority -1 or -2 to ignore data bus faults caused by load and store instructions. The setting of this bit applies to the hard fault, NMI, and **FAULTMASK** escalated handlers.<br><br>Value Description<br>0 Data bus faults caused by load and store instructions cause a lock-up.<br>1 Handlers running at priority -1 and -2 ignore data bus faults caused by load and store instructions.<br><br>Set this bit only when the handler and its data are in absolutely safe memory. The normal use of this bit is to probe system devices and bridges to detect control path problems and fix them. |
| 7:5 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | DIV0 | R/W | 0 | Trap on Divide by 0<br>This bit enables faulting or halting when the processor executes an SDIV or UDIV instruction with a divisor of 0.<br><br>Value Description<br>0 Do not trap on divide by 0. A divide by zero returns a quotient of 0.<br>1 Trap on divide by 0. |
| 3 | UNALIGNED | R/W | 0 | Trap on Unaligned Access<br><br>Value Description<br>0 Do not trap on unaligned halfword and word accesses.<br>1 Trap on unaligned halfword and word accesses. An unaligned access generates a usage fault.<br><br>Unaligned LDM, STM, LDRD, and STRD instructions always fault regardless of whether UNALIGNED is set. |
| 2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | MAINPEND | R/W | 0 | Allow Main Interrupt Trigger<br><br>Value Description<br>0 Disables unprivileged software access to the **SWTRIG** register.<br>1 Enables unprivileged software access to the **SWTRIG** register (see page 122). |
| 0 | BASETHR | R/W | 0 | Thread State Control<br><br>Value Description<br>0 The processor can enter Thread mode only when no exception is active.<br>1 The processor can enter Thread mode from any level under the control of an EXC_RETURN value (see “Exception Return” on page 88 for more information). |

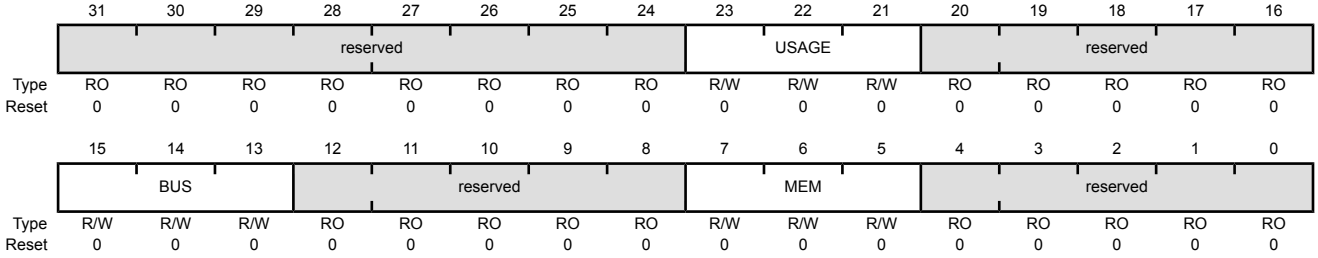
## Register 33: System Handler Priority 1 (SYSPRI1), offset 0xD18

**Note:** This register can only be accessed from privileged mode.

The **SYSPRI1** register configures the priority level, 0 to 7 of the usage fault, bus fault, and memory management fault exception handlers. This register is byte-accessible.

System Handler Priority 1 (SYSPRI1)

Base 0xE000.E000
Offset 0xD18
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:24 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:21 | USAGE | R/W | 0x0 | Usage Fault Priority<br>This field configures the priority level of the usage fault. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 20:16 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:13 | BUS | R/W | 0x0 | Bus Fault Priority<br>This field configures the priority level of the bus fault. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 12:8 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:5 | MEM | R/W | 0x0 | Memory Management Fault Priority<br>This field configures the priority level of the memory management fault. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 4:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

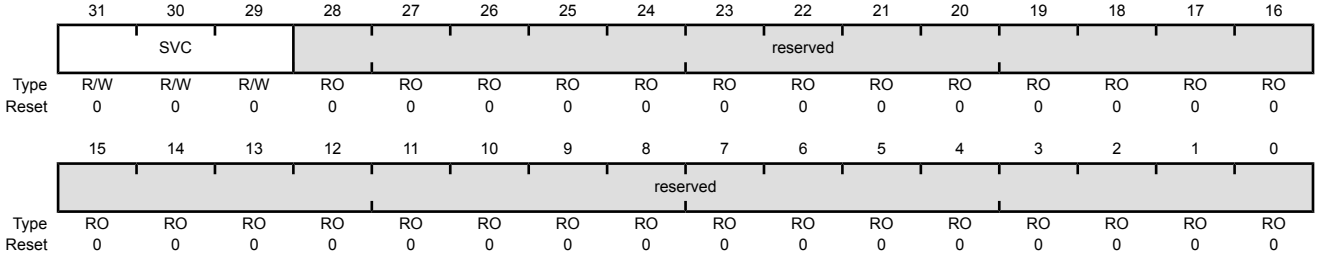
## Register 34: System Handler Priority 2 (SYSPRI2), offset 0xD1C

**Note:** This register can only be accessed from privileged mode.

The **SYSPRI2** register configures the priority level, 0 to 7 of the SVCall handler. This register is byte-accessible.

System Handler Priority 2 (SYSPRI2)

Base 0xE000.E000
Offset 0xD1C
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:29 | SVC | R/W | 0x0 | SVCall Priority<br>This field configures the priority level of SVCall. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 28:0 | reserved | RO | 0x000.0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

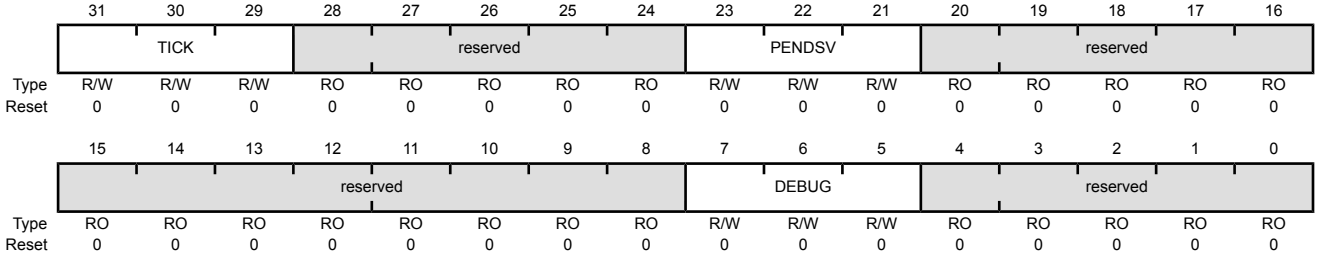
## Register 35: System Handler Priority 3 (SYSPRI3), offset 0xD20

**Note:** This register can only be accessed from privileged mode.

The **SYSPRI3** register configures the priority level, 0 to 7 of the SysTick exception and PendSV handlers. This register is byte-accessible.

System Handler Priority 3 (SYSPRI3)

Base 0xE000.E000
Offset 0xD20
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:29 | TICK | R/W | 0x0 | SysTick Exception Priority<br>This field configures the priority level of the SysTick exception. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 28:24 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:21 | PENDSV | R/W | 0x0 | PendSV Priority<br>This field configures the priority level of PendSV. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 20:8 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:5 | DEBUG | R/W | 0x0 | Debug Priority<br>This field configures the priority level of Debug. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 4:0 | reserved | RO | 0x0.0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 36: System Handler Control and State (SYSHNDCTRL), offset 0xD24

**Note:** This register can only be accessed from privileged mode.

The **SYSHNDCTRL** register enables the system handlers, and indicates the pending status of the usage fault, bus fault, memory management fault, and SVC exceptions as well as the active status of the system handlers.

If a system handler is disabled and the corresponding fault occurs, the processor treats the fault as a hard fault.

This register can be modified to change the pending or active status of system exceptions. An OS kernel can write to the active bits to perform a context switch that changes the current exception type.

**Caution – Software that changes the value of an active bit in this register without correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure software that writes to this register retains and subsequently restores the current active status.**

**If the value of a bit in this register must be modified after enabling the system handlers, a read-modify-write procedure must be used to ensure that only the required bit is modified.**

System Handler Control and State (SYSHNDCTRL)

Base 0xE000.E000
Offset 0xD24
Type R/W, reset 0x0000.0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | | | | | | | | | USAGE | BUS | MEM |
| Type RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SVC | BUSP | MEMP | USAGEP | TICK | PNDSV | reserved | MON | SVCA | reserved | | | USGA | reserved | BUSA | MEMA |
| Type R/W | R/W | R/W | R/W | R/W | R/W | RO | R/W | R/W | RO | RO | RO | R/W | RO | R/W | R/W |
| Reset 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | USAGE | R/W | 0 | Usage Fault Enable<br>Value Description<br>0 Disables the usage fault exception.<br>1 Enables the usage fault exception. |
| 17 | BUS | R/W | 0 | Bus Fault Enable<br>Value Description<br>0 Disables the bus fault exception.<br>1 Enables the bus fault exception. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 16 | MEM | R/W | 0 | Memory Management Fault Enable<br><br>Value Description<br>0 Disables the memory management fault exception.<br>1 Enables the memory management fault exception. |
| 15 | SVC | R/W | 0 | SVC Call Pending<br><br>Value Description<br>0 An SVC call exception is not pending.<br>1 An SVC call exception is pending.<br><br>This bit can be modified to change the pending status of the SVC call exception. |
| 14 | BUSP | R/W | 0 | Bus Fault Pending<br><br>Value Description<br>0 A bus fault exception is not pending.<br>1 A bus fault exception is pending.<br><br>This bit can be modified to change the pending status of the bus fault exception. |
| 13 | MEMP | R/W | 0 | Memory Management Fault Pending<br><br>Value Description<br>0 A memory management fault exception is not pending.<br>1 A memory management fault exception is pending.<br><br>This bit can be modified to change the pending status of the memory management fault exception. |
| 12 | USAGEP | R/W | 0 | Usage Fault Pending<br><br>Value Description<br>0 A usage fault exception is not pending.<br>1 A usage fault exception is pending.<br><br>This bit can be modified to change the pending status of the usage fault exception. |
| 11 | TICK | R/W | 0 | SysTick Exception Active<br><br>Value Description<br>0 A SysTick exception is not active.<br>1 A SysTick exception is active.<br><br>This bit can be modified to change the active status of the SysTick exception, however, see the Caution above before setting this bit. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 10 | PNDSV | R/W | 0 | PendSV Exception Active<br><br>Value Description<br>0 A PendSV exception is not active.<br>1 A PendSV exception is active.<br><br>This bit can be modified to change the active status of the PendSV exception, however, see the Caution above before setting this bit. |
| 9 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 8 | MON | R/W | 0 | Debug Monitor Active<br><br>Value Description<br>0 The Debug monitor is not active.<br>1 The Debug monitor is active. |
| 7 | SVCA | R/W | 0 | SVC Call Active<br><br>Value Description<br>0 SVC call is not active.<br>1 SVC call is active.<br><br>This bit can be modified to change the active status of the SVC call exception, however, see the Caution above before setting this bit. |
| 6:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | USGA | R/W | 0 | Usage Fault Active<br><br>Value Description<br>0 Usage fault is not active.<br>1 Usage fault is active.<br><br>This bit can be modified to change the active status of the usage fault exception, however, see the Caution above before setting this bit. |
| 2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | BUSA | R/W | 0 | Bus Fault Active<br><br>Value Description<br>0 Bus fault is not active.<br>1 Bus fault is active.<br><br>This bit can be modified to change the active status of the bus fault exception, however, see the Caution above before setting this bit. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | MEMA | R/W | 0 | Memory Management Fault Active |

| Value | Description |
|---|---|
| 0 | Memory management fault is not active. |
| 1 | Memory management fault is active. |

This bit can be modified to change the active status of the memory management fault exception, however, see the Caution above before setting this bit.

## Register 37: Configurable Fault Status (FAULTSTAT), offset 0xD28

**Note:** This register can only be accessed from privileged mode.

The **FAULTSTAT** register indicates the cause of a memory management fault, bus fault, or usage fault. Each of these functions is assigned to a subregister as follows:

- **Usage Fault Status (UFAULTSTAT)**, bits 31:16
- **Bus Fault Status (BFAULTSTAT)**, bits 15:8
- **Memory Management Fault Status (MFAULTSTAT)**, bits 7:0

**FAULTSTAT** is byte accessible. **FAULTSTAT** or its subregisters can be accessed as follows:

- The complete **FAULTSTAT** register, with a word access to offset 0xD28
- The **MFAULTSTAT**, with a byte access to offset 0xD28
- The **MFAULTSTAT** and **BFAULTSTAT**, with a halfword access to offset 0xD28
- The **BFAULTSTAT**, with a byte access to offset 0xD29
- The **UFAULTSTAT**, with a halfword access to offset 0xD2A

Bits are cleared by writing a 1 to them.

In a fault handler, the true faulting address can be determined by:

1. Read and save the **Memory Management Fault Address (MMADDR)** or **Bus Fault Address (FAULTADDR)** value.
2. Read the MMARV bit in **MFAULTSTAT**, or the BFARV bit in **BFAULTSTAT** to determine if the **MMADDR** or **FAULTADDR** contents are valid.

Software must follow this sequence because another higher priority exception might change the **MMADDR** or **FAULTADDR** value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the **MMADDR** or **FAULTADDR** value.

Configurable Fault Status (FAULTSTAT)

Base 0xE000.E000
Offset 0xD28
Type R/W1C, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | DIV0 | UNALIGN | reserved | | | | NOCP | INVPC | INVSTAT | UNDEF |
| Type | RO | RO | RO | RO | RO | RO | R/W1C | R/W1C | RO | RO | RO | RO | R/W1C | R/W1C | R/W1C | R/W1C |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BFARV | reserved | | BSTKE | BUSTKE | IMPRE | PRECISE | IBUS | MMARV | reserved | | MSTKE | MUSTKE | reserved | DERR | IERR |
| Type | R/W1C | RO | RO | R/W1C | R/W1C | R/W1C | R/W1C | R/W1C | R/W1C | RO | RO | R/W1C | R/W1C | RO | R/W1C | R/W1C |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:26 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 25 | DIV0 | R/W1C | 0 | Divide-by-Zero Usage Fault<br><br>Value Description<br>0 No divide-by-zero fault has occurred, or divide-by-zero trapping is not enabled.<br>1 The processor has executed an SDIV or UDIV instruction with a divisor of 0.<br><br>When this bit is set, the **PC** value stacked for the exception return points to the instruction that performed the divide by zero.<br>Trapping on divide-by-zero is enabled by setting the DIV0 bit in the **Configuration and Control (CFGCTRL)** register (see page 132).<br>This bit is cleared by writing a 1 to it. |
| 24 | UNALIGN | R/W1C | 0 | Unaligned Access Usage Fault<br><br>Value Description<br>0 No unaligned access fault has occurred, or unaligned access trapping is not enabled.<br>1 The processor has made an unaligned memory access.<br><br>Unaligned LDM, STM, LDRD, and STRD instructions always fault regardless of the configuration of this bit.<br>Trapping on unaligned access is enabled by setting the UNALIGNED bit in the **CFGCTRL** register (see page 132).<br>This bit is cleared by writing a 1 to it. |
| 23:20 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 19 | NOCP | R/W1C | 0 | No Coprocessor Usage Fault<br><br>Value Description<br>0 A usage fault has not been caused by attempting to access a coprocessor.<br>1 The processor has attempted to access a coprocessor.<br><br>This bit is cleared by writing a 1 to it. |
| 18 | INVPC | R/W1C | 0 | Invalid PC Load Usage Fault<br><br>Value Description<br>0 A usage fault has not been caused by attempting to load an invalid **PC** value.<br>1 The processor has attempted an illegal load of EXC_RETURN to the **PC** as a result of an invalid context or an invalid EXC_RETURN value.<br><br>When this bit is set, the **PC** value stacked for the exception return points to the instruction that tried to perform the illegal load of the **PC**.<br>This bit is cleared by writing a 1 to it. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 17 | INVSTAT | R/W1C | 0 | Invalid State Usage Fault<br><br>Value Description<br>0 A usage fault has not been caused by an invalid state.<br>1 The processor has attempted to execute an instruction that makes illegal use of the **EPSR** register.<br><br>When this bit is set, the **PC** value stacked for the exception return points to the instruction that attempted the illegal use of the **Execution Program Status Register (EPSR)** register.<br>This bit is not set if an undefined instruction uses the **EPSR** register.<br>This bit is cleared by writing a 1 to it. |
| 16 | UNDEF | R/W1C | 0 | Undefined Instruction Usage Fault<br><br>Value Description<br>0 A usage fault has not been caused by an undefined instruction.<br>1 The processor has attempted to execute an undefined instruction.<br><br>When this bit is set, the **PC** value stacked for the exception return points to the undefined instruction.<br>An undefined instruction is an instruction that the processor cannot decode.<br>This bit is cleared by writing a 1 to it. |
| 15 | BFARV | R/W1C | 0 | Bus Fault Address Register Valid<br><br>Value Description<br>0 The value in the **Bus Fault Address (FAULTADDR)** register is not a valid fault address.<br>1 The **FAULTADDR** register is holding a valid fault address.<br><br>This bit is set after a bus fault, where the address is known. Other faults can clear this bit, such as a memory management fault occurring later.<br>If a bus fault occurs and is escalated to a hard fault because of priority, the hard fault handler must clear this bit. This action prevents problems if returning to a stacked active bus fault handler whose **FAULTADDR** register value has been overwritten.<br>This bit is cleared by writing a 1 to it. |
| 14:13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 12 | BSTKE | R/W1C | 0 | Stack Bus Fault<br><br>Value Description<br>0 No bus fault has occurred on stacking for exception entry.<br>1 Stacking for an exception entry has caused one or more bus faults.<br><br>When this bit is set, the **SP** is still adjusted but the values in the context area on the stack might be incorrect. A fault address is not written to the **FAULTADDR** register.<br>This bit is cleared by writing a 1 to it. |
| 11 | BUSTKE | R/W1C | 0 | Unstack Bus Fault<br><br>Value Description<br>0 No bus fault has occurred on unstacking for a return from exception.<br>1 Unstacking for a return from exception has caused one or more bus faults.<br><br>This fault is chained to the handler. Thus, when this bit is set, the original return stack is still present. The **SP** is not adjusted from the failing return, a new save is not performed, and a fault address is not written to the **FAULTADDR** register.<br>This bit is cleared by writing a 1 to it. |
| 10 | IMPRE | R/W1C | 0 | Imprecise Data Bus Error<br><br>Value Description<br>0 An imprecise data bus error has not occurred.<br>1 A data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error.<br><br>When this bit is set, a fault address is not written to the **FAULTADDR** register.<br>This fault is asynchronous. Therefore, if the fault is detected when the priority of the current process is higher than the bus fault priority, the bus fault becomes pending and becomes active only when the processor returns from all higher-priority processes. If a precise fault occurs before the processor enters the handler for the imprecise bus fault, the handler detects that both the `IMPRE` bit is set and one of the precise fault status bits is set.<br>This bit is cleared by writing a 1 to it. |
| 9 | PRECISE | R/W1C | 0 | Precise Data Bus Error<br><br>Value Description<br>0 A precise data bus error has not occurred.<br>1 A data bus error has occurred, and the **PC** value stacked for the exception return points to the instruction that caused the fault.<br><br>When this bit is set, the fault address is written to the **FAULTADDR** register.<br>This bit is cleared by writing a 1 to it. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 8 | IBUS | R/W1C | 0 | Instruction Bus Error<br><br>Value Description<br>0 An instruction bus error has not occurred.<br>1 An instruction bus error has occurred.<br><br>The processor detects the instruction bus error on prefetching an instruction, but sets this bit only if it attempts to issue the faulting instruction.<br>When this bit is set, a fault address is not written to the **FAULTADDR** register.<br>This bit is cleared by writing a 1 to it. |
| 7 | MMARV | R/W1C | 0 | Memory Management Fault Address Register Valid<br><br>Value Description<br>0 The value in the **Memory Management Fault Address (MMADDR)** register is not a valid fault address.<br>1 The **MMADDR** register is holding a valid fault address.<br><br>If a memory management fault occurs and is escalated to a hard fault because of priority, the hard fault handler must clear this bit. This action prevents problems if returning to a stacked active memory management fault handler whose **MMADDR** register value has been overwritten.<br>This bit is cleared by writing a 1 to it. |
| 6:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | MSTKE | R/W1C | 0 | Stack Access Violation<br><br>Value Description<br>0 No memory management fault has occurred on stacking for exception entry.<br>1 Stacking for an exception entry has caused one or more access violations.<br><br>When this bit is set, the **SP** is still adjusted but the values in the context area on the stack might be incorrect. A fault address is not written to the **MMADDR** register.<br>This bit is cleared by writing a 1 to it. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | MUSTKE | R/W1C | 0 | Unstack Access Violation<br><br>Value Description<br>0 No memory management fault has occurred on unstacking for a return from exception.<br>1 Unstacking for a return from exception has caused one or more access violations.<br><br>This fault is chained to the handler. Thus, when this bit is set, the original return stack is still present. The **SP** is not adjusted from the failing return, a new save is not performed, and a fault address is not written to the **MMADDR** register.<br>This bit is cleared by writing a 1 to it. |
| 2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | DERR | R/W1C | 0 | Data Access Violation<br><br>Value Description<br>0 A data access violation has not occurred.<br>1 The processor attempted a load or store at a location that does not permit the operation.<br><br>When this bit is set, the **PC** value stacked for the exception return points to the faulting instruction and the address of the attempted access is written to the **MMADDR** register.<br>This bit is cleared by writing a 1 to it. |
| 0 | IERR | R/W1C | 0 | Instruction Access Violation<br><br>Value Description<br>0 An instruction access violation has not occurred.<br>1 The processor attempted an instruction fetch from a location that does not permit execution.<br><br>This fault occurs on any access to an XN region, even when the MPU is disabled or not present.<br>When this bit is set, the **PC** value stacked for the exception return points to the faulting instruction and the address of the attempted access is not written to the **MMADDR** register.<br>This bit is cleared by writing a 1 to it. |

## Register 38: Hard Fault Status (HFAULTSTAT), offset 0xD2C

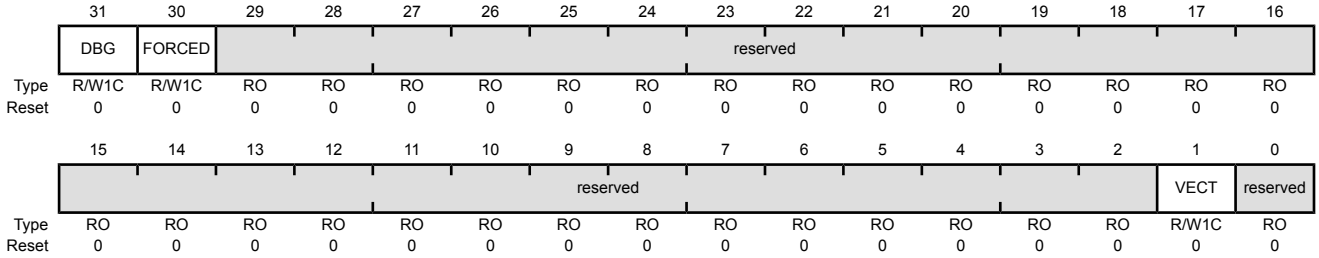
**Note:** This register can only be accessed from privileged mode.

The **HFAULTSTAT** register gives information about events that activate the hard fault handler.

Bits are cleared by writing a 1 to them.

Hard Fault Status (HFAULTSTAT)

Base 0xE000.E000
Offset 0xD2C
Type R/W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | DBG | R/W1C | 0 | Debug Event<br>This bit is reserved for Debug use. This bit must be written as a 0, otherwise behavior is unpredictable. |
| 30 | FORCED | R/W1C | 0 | Forced Hard Fault<br>Value Description<br>0 No forced hard fault has occurred.<br>1 A forced hard fault has been generated by escalation of a fault with configurable priority that cannot be handled, either because of priority or because it is disabled.<br>When this bit is set, the hard fault handler must read the other fault status registers to find the cause of the fault.<br>This bit is cleared by writing a 1 to it. |
| 29:2 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | VECT | R/W1C | 0 | Vector Table Read Fault<br>Value Description<br>0 No bus fault has occurred on a vector table read.<br>1 A bus fault occurred on a vector table read.<br>This error is always handled by the hard fault handler.<br>When this bit is set, the **PC** value stacked for the exception return points to the instruction that was preempted by the exception.<br>This bit is cleared by writing a 1 to it. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

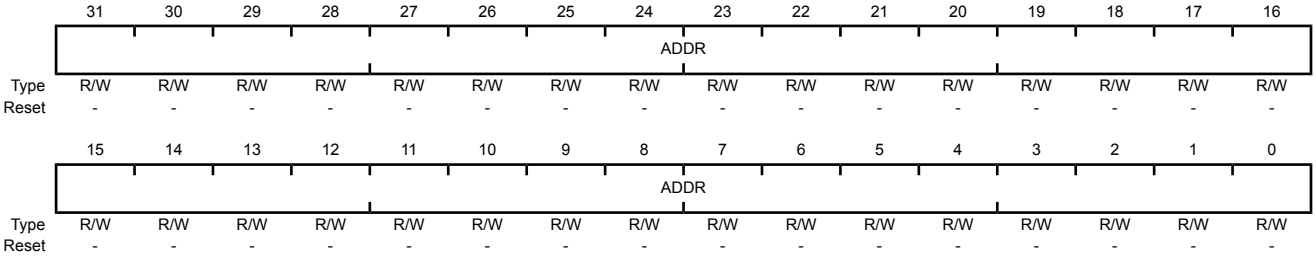
## Register 39: Memory Management Fault Address (MMADDR), offset 0xD34

**Note:** This register can only be accessed from privileged mode.

The **MMADDR** register contains the address of the location that generated a memory management fault. When an unaligned access faults, the address in the **MMADDR** register is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size. Bits in the **Memory Management Fault Status (MFAULTSTAT)** register indicate the cause of the fault and whether the value in the **MMADDR** register is valid (see page 141).

Memory Management Fault Address (MMADDR)

Base 0xE000.E000
Offset 0xD34
Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | ADDR | R/W | - | Fault Address<br>When the MMARV bit of **MFAULTSTAT** is set, this field holds the address of the location that generated the memory management fault. |

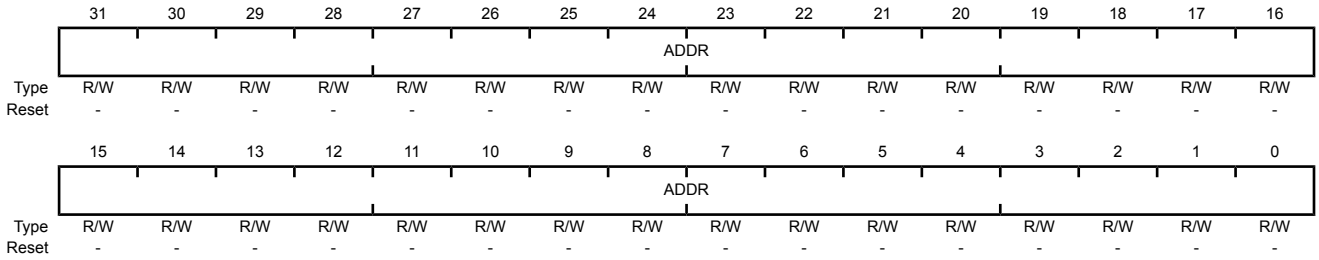
### Register 40: Bus Fault Address (FAULTADDR), offset 0xD38

**Note:** This register can only be accessed from privileged mode.

The **FAULTADDR** register contains the address of the location that generated a bus fault. When an unaligned access faults, the address in the **FAULTADDR** register is the one requested by the instruction, even if it is not the address of the fault. Bits in the **Bus Fault Status (BFAULTSTAT)** register indicate the cause of the fault and whether the value in the **FAULTADDR** register is valid (see page 141).

Bus Fault Address (FAULTADDR)

Base 0xE000.E000
Offset 0xD38
Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | ADDR | R/W | - | Fault Address<br>When the FAULTADDRV bit of **BFAULTSTAT** is set, this field holds the address of the location that generated the bus fault. |

## 3.6 Memory Protection Unit (MPU) Register Descriptions

This section lists and describes the Memory Protection Unit (MPU) registers, in numerical order by address offset.

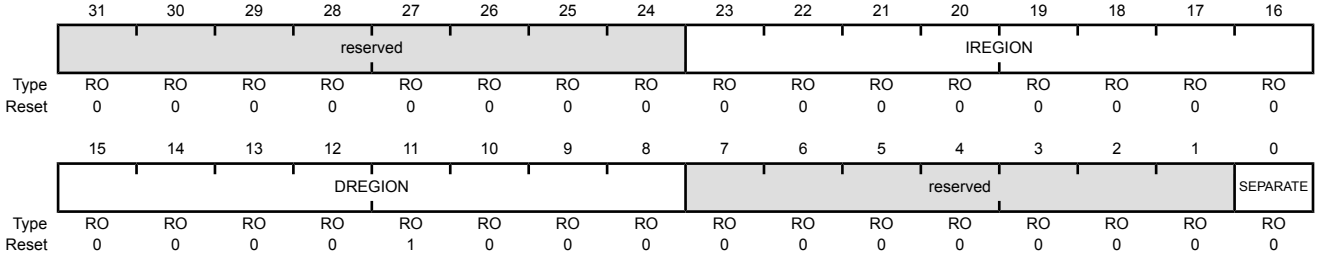
The MPU registers can only be accessed from privileged mode.

## Register 41: MPU Type (MPUTYPE), offset 0xD90

**Note:** This register can only be accessed from privileged mode.

The **MPUTYPE** register indicates whether the MPU is present, and if so, how many regions it supports.

MPU Type (MPUTYPE)
Base 0xE000.E000
Offset 0xD90
Type RO, reset 0x0000.0800




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:24 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:16 | IREGION | RO | 0x00 | Number of I Regions<br>This field indicates the number of supported MPU instruction regions. This field always contains 0x00. The MPU memory map is unified and is described by the `DREGION` field. |
| 15:8 | DREGION | RO | 0x08 | Number of D Regions<br>Value Description<br>0x08 Indicates there are eight supported MPU data regions. |
| 7:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | SEPARATE | RO | 0 | Separate or Unified MPU<br>Value Description<br>0 Indicates the MPU is unified. |

## Register 42: MPU Control (MPUCTRL), offset 0xD94

**Note:** This register can only be accessed from privileged mode.

The **MPUCTRL** register enables the MPU, enables the default memory map background region, and enables use of the MPU when in the hard fault, Non-maskable Interrupt (NMI), and **Fault Mask Register (FAULTMASK)** escalated handlers.

When the ENABLE and PRIVDEFEN bits are both set:

- For privileged accesses, the default memory map is as described in "Memory Model" on page 72. Any access by privileged software that does not address an enabled memory region behaves as defined by the default memory map.
- Any access by unprivileged software that does not address an enabled memory region causes a memory management fault.

Execute Never (XN) and Strongly Ordered rules always apply to the System Control Space regardless of the value of the ENABLE bit.

When the ENABLE bit is set, at least one region of the memory map must be enabled for the system to function unless the PRIVDEFEN bit is set. If the PRIVDEFEN bit is set and no regions are enabled, then only privileged software can operate.

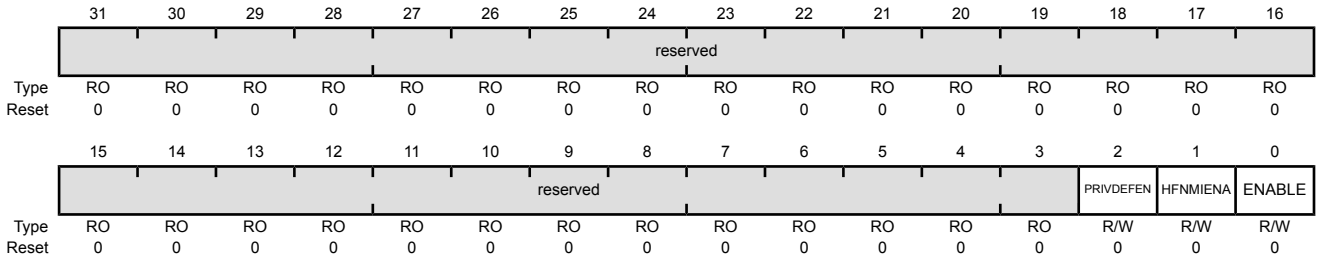
When the ENABLE bit is clear, the system uses the default memory map, which has the same memory attributes as if the MPU is not implemented (see Table 2-5 on page 74 for more information). The default memory map applies to accesses from both privileged and unprivileged software.

When the MPU is enabled, accesses to the System Control Space and vector table are always permitted. Other areas are accessible based on regions and whether PRIVDEFEN is set.

Unless HFNMIENA is set, the MPU is not enabled when the processor is executing the handler for an exception with priority –1 or –2. These priorities are only possible when handling a hard fault or NMI exception or when **FAULTMASK** is enabled. Setting the HFNMIENA bit enables the MPU when operating with these two priorities.

MPU Control (MPUCTRL)

Base 0xE000.E000
Offset 0xD94
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | PRIVDEFEN | R/W | 0 | MPU Default Region<br>This bit enables privileged software access to the default memory map.<br><br>Value Description<br>0 If the MPU is enabled, this bit disables use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault.<br>1 If the MPU is enabled, this bit enables use of the default memory map as a background region for privileged software accesses.<br><br>When this bit is set, the background region acts as if it is region number -1. Any region that is defined and enabled has priority over this default map.<br>If the MPU is disabled, the processor ignores this bit. |
| 1 | HFNMIENA | R/W | 0 | MPU Enabled During Faults<br>This bit controls the operation of the MPU during hard fault, NMI, and **FAULTMASK** handlers.<br><br>Value Description<br>0 The MPU is disabled during hard fault, NMI, and **FAULTMASK** handlers, regardless of the value of the ENABLE bit.<br>1 The MPU is enabled during hard fault, NMI, and **FAULTMASK** handlers.<br><br>When the MPU is disabled and this bit is set, the resulting behavior is unpredictable. |
| 0 | ENABLE | R/W | 0 | MPU Enable<br><br>Value Description<br>0 The MPU is disabled.<br>1 The MPU is enabled.<br><br>When the MPU is disabled and the HFNMIENA bit is set, the resulting behavior is unpredictable. |

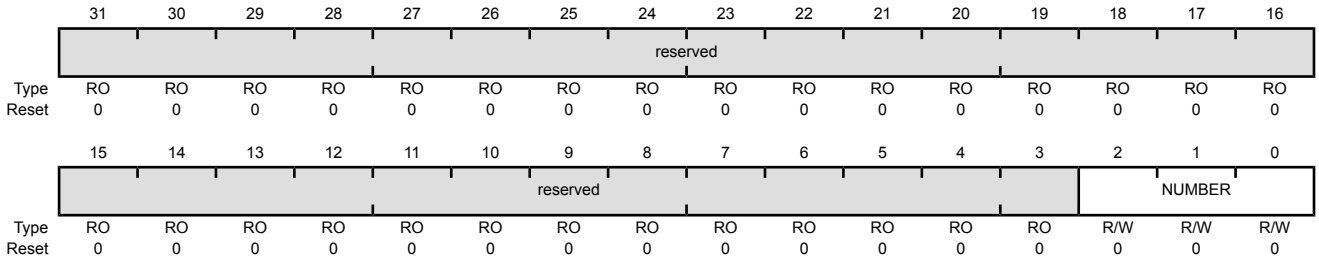
## Register 43: MPU Region Number (MPUNUMBER), offset 0xD98

**Note:** This register can only be accessed from privileged mode.

The **MPUNUMBER** register selects which memory region is referenced by the **MPU Region Base Address (MPUBASE)** and **MPU Region Attribute and Size (MPUATTR)** registers. Normally, the required region number should be written to this register before accessing the **MPUBASE** or the **MPUATTR** register. However, the region number can be changed by writing to the **MPUBASE** register with the VALID bit set (see page 154). This write updates the value of the REGION field.

MPU Region Number (MPUNUMBER)

Base 0xE000.E000
Offset 0xD98
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | NUMBER | R/W | 0x0 | MPU Region to Access<br>This field indicates the MPU region referenced by the **MPUBASE** and **MPUATTR** registers. The MPU supports eight memory regions. |

## Register 44: MPU Region Base Address (MPUBASE), offset 0xD9C

## Register 45: MPU Region Base Address Alias 1 (MPUBASE1), offset 0xDA4

## Register 46: MPU Region Base Address Alias 2 (MPUBASE2), offset 0xDAC

## Register 47: MPU Region Base Address Alias 3 (MPUBASE3), offset 0xDB4

**Note:** This register can only be accessed from privileged mode.

The **MPUBASE** register defines the base address of the MPU region selected by the **MPU Region Number (MPUNUMBER)** register and can update the value of the **MPUNUMBER** register. To change the current region number and update the **MPUNUMBER** register, write the **MPUBASE** register with the VALID bit set.

The ADDR field is bits 31:*N* of the **MPUBASE** register. Bits (*N*-1):5 are reserved. The region size, as specified by the SIZE field in the **MPU Region Attribute and Size (MPUATTR)** register, defines the value of *N* where:

$N = Log_2(\text{Region size in bytes})$

If the region size is configured to 4 GB in the **MPUATTR** register, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x0000.0000.

The base address is aligned to the size of the region. For example, a 64-KB region must be aligned on a multiple of 64 KB, for example, at 0x0001.0000 or 0x0002.0000.

MPU Region Base Address (MPUBASE)

Base 0xE000.E000
Offset 0xD9C
Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ADDR | | | | | | | | | | | | | | | |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ADDR | | | | | | | | | | | VALID | reserved | REGION | | |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | WO | RO | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | ADDR | R/W | 0x0000.000 | Base Address Mask<br>Bits 31:*N* in this field contain the region base address. The value of *N* depends on the region size, as shown above. The remaining bits (*N*-1):5 are reserved.<br>Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | VALID | WO | 0 | Region Number Valid<br><br>Value Description<br>0 The **MPUNUMBER** register is not changed and the processor updates the base address for the region specified in the **MPUNUMBER** register and ignores the value of the REGION field.<br>1 The **MPUNUMBER** register is updated with the value of the REGION field and the base address is updated for the region specified in the REGION field.<br><br>This bit is always read as 0. |
| 3 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | REGION | R/W | 0x0 | Region Number<br><br>On a write, contains the value to be written to the **MPUNUMBER** register. On a read, returns the current region number in the **MPUNUMBER** register. |

## Register 48: MPU Region Attribute and Size (MPUATTR), offset 0xDA0

## Register 49: MPU Region Attribute and Size Alias 1 (MPUATTR1), offset 0xDA8

## Register 50: MPU Region Attribute and Size Alias 2 (MPUATTR2), offset 0xDB0

## Register 51: MPU Region Attribute and Size Alias 3 (MPUATTR3), offset 0xDB8

**Note:** This register can only be accessed from privileged mode.

The **MPUATTR** register defines the region size and memory attributes of the MPU region specified by the **MPU Region Number (MPUNUMBER)** register and enables that region and any subregions.

The **MPUATTR** register is accessible using word or halfword accesses with the most-significant halfword holding the region attributes and the least-significant halfword holds the region size and the region and subregion enable bits.

The MPU access permission attribute bits, XN, AP, TEX, S, C, and B, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

The SIZE field defines the size of the MPU memory region specified by the **MPUNUMBER** register as follows:

(Region size in bytes) = $2^{(SIZE+1)}$

The smallest permitted region size is 32 bytes, corresponding to a SIZE value of 4. Table 3-9 on page 156 gives example SIZE values with the corresponding region size and value of N in the **MPU Region Base Address (MPUBASE)** register.

**Table 3-9. Example SIZE Field Values**

| SIZE Encoding | Region Size | Value of N[a] | Note |
|---|---|---|---|
| 00100b (0x4) | 32 B | 5 | Minimum permitted size |
| 01001b (0x9) | 1 KB | 10 | - |
| 10011b (0x13) | 1 MB | 20 | - |
| 11101b (0x1D) | 1 GB | 30 | - |
| 11111b (0x1F) | 4 GB | No valid ADDR field in **MPUBASE**; the region occupies the complete memory map. | Maximum possible size |

a. Refers to the N parameter in the **MPUBASE** register (see page 154).

MPU Region Attribute and Size (MPUATTR)

Base 0xE000.E000
Offset 0xDA0
Type R/W, reset 0x0000.0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | XN | reserved | AP | | | reserved | | TEX | | | S | C | B |
| Type | RO | RO | RO | R/W | RO | R/W | R/W | R/W | RO | RO | R/W | R/W | R/W | R/W | R/W | R/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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SRD | | | | | | | | reserved | | SIZE | | | | | ENABLE |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:29 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 28 | XN | R/W | 0 | Instruction Access Disable<br>Value Description<br>0 Instruction fetches are enabled.<br>1 Instruction fetches are disabled. |
| 27 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 26:24 | AP | R/W | 0 | Access Privilege<br>For information on using this bit field, see Table 3-5 on page 102. |
| 23:22 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 21:19 | TEX | R/W | 0x0 | Type Extension Mask<br>For information on using this bit field, see Table 3-3 on page 101. |
| 18 | S | R/W | 0 | Shareable<br>For information on using this bit, see Table 3-3 on page 101. |
| 17 | C | R/W | 0 | Cacheable<br>For information on using this bit, see Table 3-3 on page 101. |
| 16 | B | R/W | 0 | Bufferable<br>For information on using this bit, see Table 3-3 on page 101. |
| 15:8 | SRD | R/W | 0x00 | Subregion Disable Bits<br>Value Description<br>0 The corresponding subregion is enabled.<br>1 The corresponding subregion is disabled.<br>Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, configure the SRD field as 0x00. See the section called “Subregions” on page 100 for more information. |
| 7:6 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:1 | SIZE | R/W | 0x0 | Region Size Mask<br>The SIZE field defines the size of the MPU memory region specified by the **MPUNUMBER** register. Refer to Table 3-9 on page 156 for more information. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | ENABLE | R/W | 0 | Region Enable |

| Value | Description |
|---|---|
| 0 | The region is disabled. |
| 1 | The region is enabled. |

# 4 JTAG Interface

The Joint Test Action Group (JTAG) port is an IEEE standard that defines a Test Access Port and Boundary Scan Architecture for digital integrated circuits and provides a standardized serial interface for controlling the associated test logic. The TAP, Instruction Register (IR), and Data Registers (DR) can be used to test the interconnections of assembled printed circuit boards and obtain manufacturing information on the components. The JTAG Port also provides a means of accessing and controlling design-for-test features such as I/O pin observation and control, scan testing, and debugging.

The JTAG port is comprised of four pins: TCK, TMS, TDI, and TDO. Data is transmitted serially into the controller on TDI and out of the controller on TDO. The interpretation of this data is dependent on the current state of the TAP controller. For detailed information on the operation of the JTAG port and TAP controller, please refer to the *IEEE Standard 1149.1-Test Access Port and Boundary-Scan Architecture*.

The Stellaris® JTAG controller works with the ARM JTAG controller built into the Cortex-M3 core. This is implemented by multiplexing the TDO outputs from both JTAG controllers. ARM JTAG instructions select the ARM TDO output while Stellaris JTAG instructions select the Stellaris TDO outputs. The multiplexer is controlled by the Stellaris JTAG controller, which has comprehensive programming for the ARM, Stellaris, and unimplemented JTAG instructions.

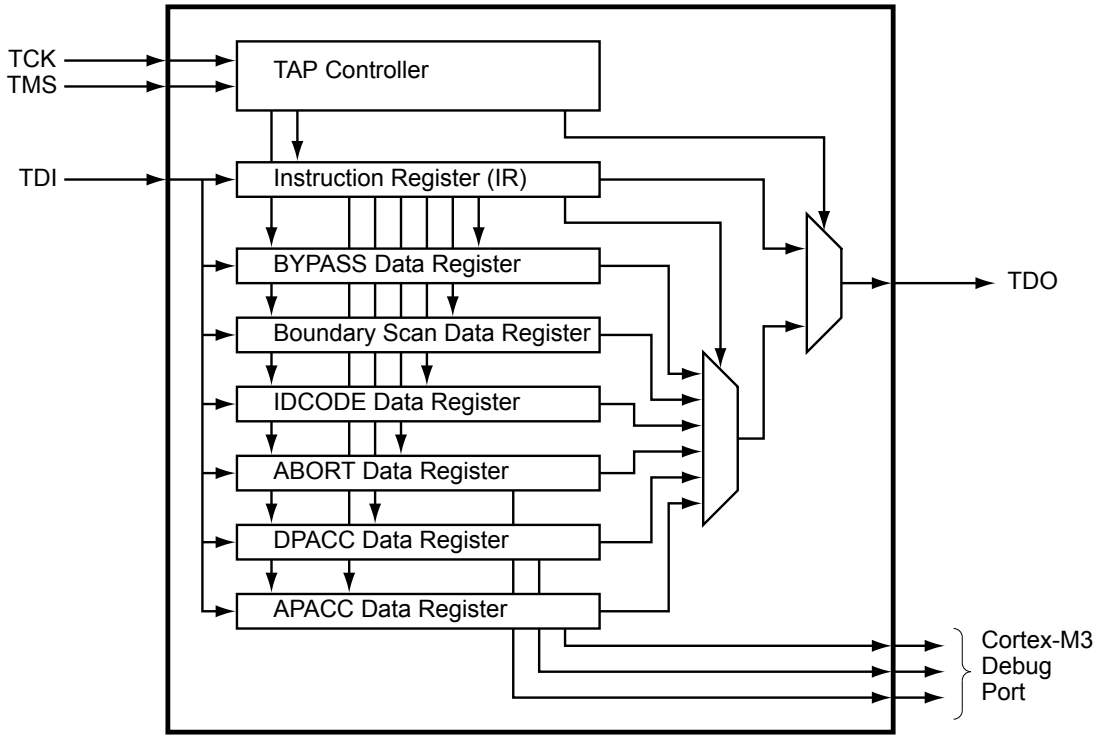
The Stellaris JTAG module has the following features:

- IEEE 1149.1-1990 compatible Test Access Port (TAP) controller
- Four-bit Instruction Register (IR) chain for storing JTAG instructions
- IEEE standard instructions: BYPASS, IDCODE, SAMPLE/PRELOAD, EXTEST and INTEST
- ARM additional instructions: APACC, DPACC and ABORT
- Integrated ARM Serial Wire Debug (SWD)

See the *ARM® Debug Interface V5 Architecture Specification* for more information on the ARM JTAG controller.

## 4.1 Block Diagram

**Figure 4-1. JTAG Module Block Diagram**




## 4.2 Signal Description

Table 4-1 on page 160 lists the external signals of the JTAG/SWD controller and describes the function of each. The JTAG/SWD controller signals are alternate functions for some GPIO signals, however note that the reset state of the pins is for the JTAG/SWD function. The JTAG/SWD controller signals are under commit protection and require a special process to be configured as GPIOs, see “Commit Control” on page 357. The column in the table below titled "Pin Assignment" lists the GPIO pin placement for the JTAG/SWD controller signals. The AFSEL bit in the **GPIO Alternate Function Select (GPIOAFSEL)** register (page 371) is set to choose the JTAG/SWD function. For more information on configuring GPIOs, see “General-Purpose Input/Outputs (GPIOs)” on page 351.

**Table 4-1. JTAG_SWD_SWO Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| SWCLK | 52 | I | TTL | JTAG/SWD CLK. |
| SWDIO | 51 | I/O | TTL | JTAG TMS and SWDIO. |
| SWO | 49 | O | TTL | JTAG TDO and SWO. |
| TCK | 52 | I | TTL | JTAG/SWD CLK. |
| TDI | 50 | I | TTL | JTAG TDI. |
| TDO | 49 | O | TTL | JTAG TDO and SWO. |
| TMS | 51 | I/O | TTL | JTAG TMS and SWDIO. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 4.3 Functional Description

A high-level conceptual drawing of the JTAG module is shown in Figure 4-1 on page 160. The JTAG module is composed of the Test Access Port (TAP) controller and serial shift chains with parallel update registers. The TAP controller is a simple state machine controlled by the TCK and TMS inputs. The current state of the TAP controller depends on the sequence of values captured on TMS at the rising edge of TCK. The TAP controller determines when the serial shift chains capture new data, shift data from TDI towards TDO, and update the parallel load registers. The current state of the TAP controller also determines whether the Instruction Register (IR) chain or one of the Data Register (DR) chains is being accessed.

The serial shift chains with parallel load registers are comprised of a single Instruction Register (IR) chain and multiple Data Register (DR) chains. The current instruction loaded in the parallel load register determines which DR chain is captured, shifted, or updated during the sequencing of the TAP controller.

Some instructions, like EXTEST and INTEST, operate on data currently in a DR chain and do not capture, shift, or update any of the chains. Instructions that are not implemented decode to the BYPASS instruction to ensure that the serial path between TDI and TDO is always connected (see Table 4-3 on page 167 for a list of implemented instructions).

See “JTAG and Boundary Scan” on page 772 for JTAG timing diagrams.

### 4.3.1 JTAG Interface Pins

The JTAG interface consists of four standard pins: TCK, TMS, TDI, and TDO. These pins and their associated reset state are given in Table 4-2 on page 161. Detailed information on each pin follows.

**Table 4-2. JTAG Port Pins Reset State**

| Pin Name | Data Direction | Internal Pull-Up | Internal Pull-Down | Drive Strength | Drive Value |
|---|---|---|---|---|---|
| TCK | Input | Enabled | Disabled | N/A | N/A |
| TMS | Input | Enabled | Disabled | N/A | N/A |
| TDI | Input | Enabled | Disabled | N/A | N/A |
| TDO | Output | Enabled | Disabled | 2-mA driver | High-Z |

#### 4.3.1.1 Test Clock Input (TCK)

The TCK pin is the clock for the JTAG module. This clock is provided so the test logic can operate independently of any other system clocks. In addition, it ensures that multiple JTAG TAP controllers that are daisy-chained together can synchronously communicate serial test data between components. During normal operation, TCK is driven by a free-running clock with a nominal 50% duty cycle. When necessary, TCK can be stopped at 0 or 1 for extended periods of time. While TCK is stopped at 0 or 1, the state of the TAP controller does not change and data in the JTAG Instruction and Data Registers is not lost.

By default, the internal pull-up resistor on the TCK pin is enabled after reset. This assures that no clocking occurs if the pin is not driven from an external source. The internal pull-up and pull-down resistors can be turned off to save internal power as long as the TCK pin is constantly being driven by an external source.

#### 4.3.1.2 Test Mode Select (TMS)

The TMS pin selects the next state of the JTAG TAP controller. TMS is sampled on the rising edge of TCK. Depending on the current TAP state and the sampled value of TMS, the next state is entered.

Because the TMS pin is sampled on the rising edge of TCK, the *IEEE Standard 1149.1* expects the value on TMS to change on the falling edge of TCK.

Holding TMS high for five consecutive TCK cycles drives the TAP controller state machine to the Test-Logic-Reset state. When the TAP controller enters the Test-Logic-Reset state, the JTAG module and associated registers are reset to their default values. This procedure should be performed to initialize the JTAG controller. The JTAG Test Access Port state machine can be seen in its entirety in Figure 4-2 on page 163.

By default, the internal pull-up resistor on the TMS pin is enabled after reset. Changes to the pull-up resistor settings on GPIO Port C should ensure that the internal pull-up resistor remains enabled on PC1/TMS; otherwise JTAG communication could be lost.

#### 4.3.1.3 Test Data Input (TDI)

The TDI pin provides a stream of serial information to the IR chain and the DR chains. TDI is sampled on the rising edge of TCK and, depending on the current TAP state and the current instruction, presents this data to the proper shift register chain. Because the TDI pin is sampled on the rising edge of TCK, the *IEEE Standard 1149.1* expects the value on TDI to change on the falling edge of TCK.

By default, the internal pull-up resistor on the TDI pin is enabled after reset. Changes to the pull-up resistor settings on GPIO Port C should ensure that the internal pull-up resistor remains enabled on PC2/TDI; otherwise JTAG communication could be lost.

#### 4.3.1.4 Test Data Output (TDO)

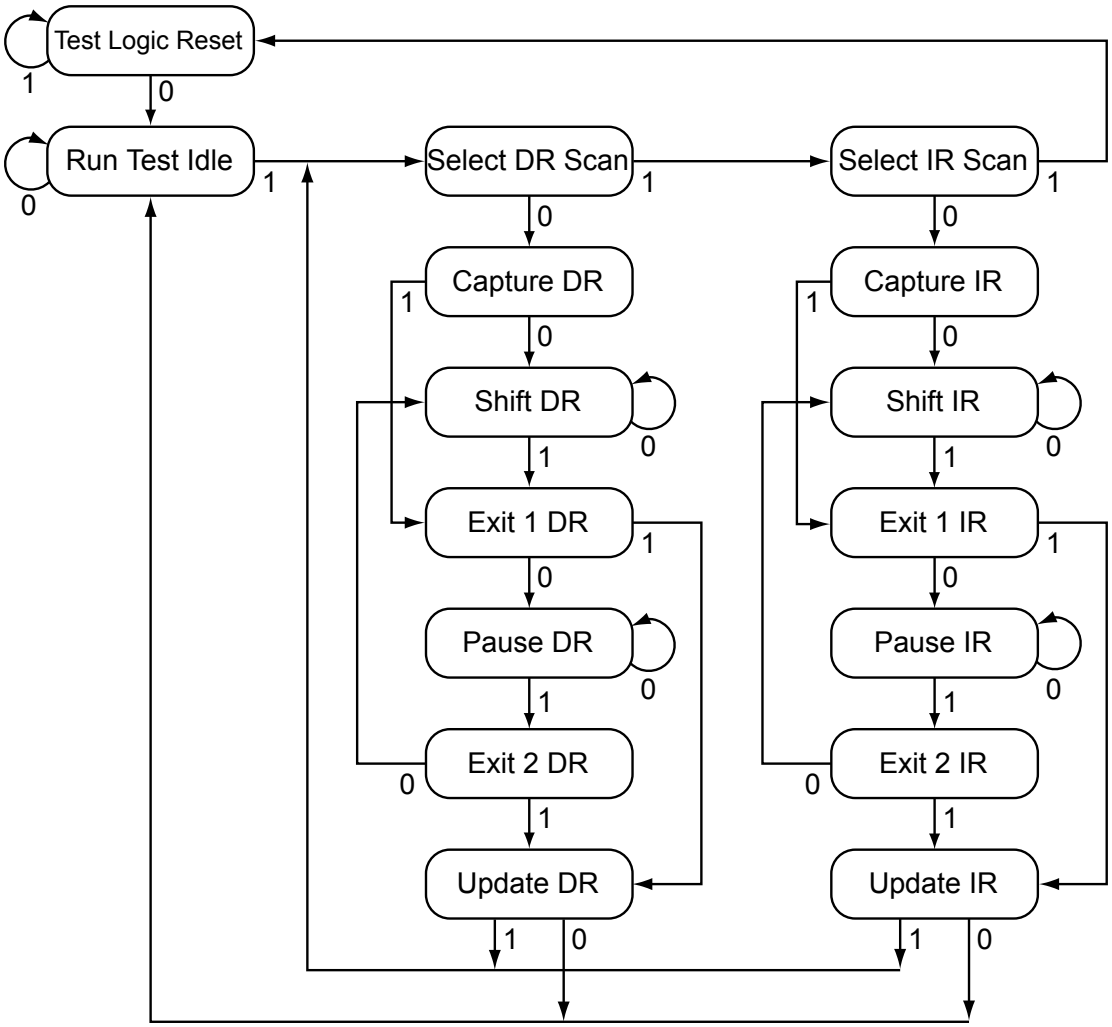
The TDO pin provides an output stream of serial information from the IR chain or the DR chains. The value of TDO depends on the current TAP state, the current instruction, and the data in the chain being accessed. In order to save power when the JTAG port is not being used, the TDO pin is placed in an inactive drive state when not actively shifting out data. Because TDO can be connected to the TDI of another controller in a daisy-chain configuration, the *IEEE Standard 1149.1* expects the value on TDO to change on the falling edge of TCK.

By default, the internal pull-up resistor on the TDO pin is enabled after reset. This assures that the pin remains at a constant logic level when the JTAG port is not being used. The internal pull-up and pull-down resistors can be turned off to save internal power if a High-Z output value is acceptable during certain TAP controller states.

### 4.3.2 JTAG TAP Controller

The JTAG TAP controller state machine is shown in Figure 4-2 on page 163. The TAP controller state machine is reset to the Test-Logic-Reset state on the assertion of a Power-On-Reset (POR). In order to reset the JTAG module after the device has been powered on, the TMS input must be held HIGH for five TCK clock cycles, resetting the TAP controller and all associated JTAG chains. Asserting the correct sequence on the TMS pin allows the JTAG module to shift in new instructions, shift in data, or idle during extended testing sequences. For detailed information on the function of the TAP controller and the operations that occur in each state, please refer to *IEEE Standard 1149.1*.

**Figure 4-2. Test Access Port State Machine**




### 4.3.3 Shift Registers

The Shift Registers consist of a serial shift register chain and a parallel load register. The serial shift register chain samples specific information during the TAP controller's CAPTURE states and allows this information to be shifted out of TDO during the TAP controller's SHIFT states. While the sampled data is being shifted out of the chain on TDO, new data is being shifted into the serial shift register on TDI. This new data is stored in the parallel load register during the TAP controller's UPDATE states. Each of the shift registers is discussed in detail in "Register Descriptions" on page 166.

### 4.3.4 Operational Considerations

There are certain operational considerations when using the JTAG module. Because the JTAG pins can be programmed to be GPIOs, board configuration and reset conditions on these pins must be considered. In addition, because the JTAG module has integrated ARM Serial Wire Debug, the method for switching between these two operational modes is described below.

#### 4.3.4.1 GPIO Functionality

When the controller is reset with either a POR or RST, the JTAG/SWD port pins default to their JTAG/SWD configurations. The default configuration includes enabling digital functionality (setting **GPIODEN** to 1), enabling the pull-up resistors (setting **GPIOPUR** to 1), and enabling the alternate hardware function (setting **GPIOAFSEL** to 1) for the PC[3:0] JTAG/SWD pins.

It is possible for software to configure these pins as GPIOs after reset by writing 0s to PC[3:0] in the **GPIOAFSEL** register. If the user does not require the JTAG/SWD port for debugging or board-level testing, this provides four more GPIOs for use in the design.

---

**Caution – It is possible to create a software sequence that prevents the debugger from connecting to the Stellaris microcontroller. If the program code loaded into flash immediately changes the JTAG pins to their GPIO functionality, the debugger may not have enough time to connect and halt the controller before the JTAG pin functionality switches. This may lock the debugger out of the part. This can be avoided with a software routine that restores JTAG functionality based on an external or software trigger.**

---

The GPIO commit control registers provide a layer of protection against accidental programming of critical hardware peripherals. Protection is currently provided for the NMI pin (PB7) and the four JTAG/SWD pins (PC[3:0]). Writes to protected bits of the **GPIO Alternate Function Select (GPIOAFSEL)** register (see page 371), **GPIO Pull-Up Select (GPIOPUR)** register (see page 377), and **GPIO Digital Enable (GPIODEN)** register (see page 381) are not committed to storage unless the **GPIO Lock (GPIOLOCK)** register (see page 383) has been unlocked and the appropriate bits of the **GPIO Commit (GPIOCR)** register (see page 384) have been set to 1.

***Recovering a "Locked" Device***

**Note:** Performing the sequence below causes the nonvolatile registers discussed in “Nonvolatile Register Programming” on page 264 to be restored to their factory default values. The mass erase of the flash memory caused by the below sequence occurs prior to the nonvolatile registers being restored.

If software configures any of the JTAG/SWD pins as GPIO and loses the ability to communicate with the debugger, there is a debug sequence that can be used to recover the device. Performing a total of ten JTAG-to-SWD and SWD-to-JTAG switch sequences while holding the device in reset mass erases the flash memory. The sequence to recover the device is:

1. Assert and hold the RST signal.
2. Apply power to the device.
3. Perform the JTAG-to-SWD switch sequence.
4. Perform the SWD-to-JTAG switch sequence.
5. Perform the JTAG-to-SWD switch sequence.
6. Perform the SWD-to-JTAG switch sequence.
7. Perform the JTAG-to-SWD switch sequence.
8. Perform the SWD-to-JTAG switch sequence.
9. Perform the JTAG-to-SWD switch sequence.

10. Perform the SWD-to-JTAG switch sequence.

11. Perform the JTAG-to-SWD switch sequence.

12. Perform the SWD-to-JTAG switch sequence.

13. Release the $\overline{\text{RST}}$ signal.

14. Wait 400 ms.

15. Power-cycle the device.

The JTAG-to-SWD and SWD-to-JTAG switch sequences are described in "ARM Serial Wire Debug (SWD)" on page 165. When performing switch sequences for the purpose of recovering the debug capabilities of the device, only steps 1 and 2 of the switch sequence in the section called "JTAG-to-SWD Switching" on page 165 must be performed.

#### 4.3.4.2 Communication with JTAG/SWD

Because the debug clock and the system clock can be running at different frequencies, care must be taken to maintain reliable communication with the JTAG/SWD interface. In the Capture-DR state, the result of the previous transaction, if any, is returned, together with a 3-bit ACK response. Software should check the ACK response to see if the previous operation has completed before initiating a new transaction. Alternatively, if the system clock is at least 8 times faster than the debug clock (TCK or SWCLK), the previous operation has enough time to complete and the ACK bits do not have to be checked.

#### 4.3.4.3 ARM Serial Wire Debug (SWD)

In order to seamlessly integrate the ARM Serial Wire Debug (SWD) functionality, a serial-wire debugger must be able to connect to the Cortex-M3 core without having to perform, or have any knowledge of, JTAG cycles. This is accomplished with a SWD preamble that is issued before the SWD session begins.

The switching preamble used to enable the SWD interface of the SWJ-DP module starts with the TAP controller in the Test-Logic-Reset state. From here, the preamble sequences the TAP controller through the following states: Run Test Idle, Select DR, Select IR, Test Logic Reset, Test Logic Reset, Run Test Idle, Run Test Idle, Select DR, Select IR, Test Logic Reset, Test Logic Reset, Run Test Idle, Run Test Idle, Select DR, Select IR, and Test Logic Reset states.

Stepping through this sequences of the TAP state machine enables the SWD interface and disables the JTAG interface. For more information on this operation and the SWD interface, see the *ARM® Debug Interface V5 Architecture Specification*.

Because this sequence is a valid series of JTAG operations that could be issued, the ARM JTAG TAP controller is not fully compliant to the *IEEE Standard 1149.1*. This is the only instance where the ARM JTAG TAP controller does not meet full compliance with the specification. Due to the low probability of this sequence occurring during normal operation of the TAP controller, it should not affect normal performance of the JTAG interface.

##### *JTAG-to-SWD Switching*

To switch the operating mode of the Debug Access Port (DAP) from JTAG to SWD mode, the external debug hardware must send the switching preamble to the microcontroller. The 16-bit TMS/SWDIO command for switching to SWD mode is defined as b1110.0111.1001.1110, transmitted LSB first. This command can also be represented as 0xE79E when transmitted LSB first. The

complete switch sequence should consist of the following transactions on the TCK/SWCLK and TMS/SWDIO signals:

1. Send at least 50 TCK/SWCLK cycles with TMS/SWDIO High to ensure that both JTAG and SWD are in their reset states.
2. Send the 16-bit JTAG-to-SWD switch command, 0xE79E, on TMS/SWDIO.
3. Send at least 50 TCK/SWCLK cycles with TMS/SWDIO High to ensure that if SWJ-DP was already in SWD mode before sending the switch sequence, the SWD goes into the line reset state.

To verify that the Debug Access Port (DAP) has switched to the Serial Wire Debug (SWD) operating mode, perform a SWD READID operation. The ID value can be compared against the device's known ID to verify the switch.

#### *SWD-to-JTAG Switching*

To switch the operating mode of the Debug Access Port (DAP) from SWD to JTAG mode, the external debug hardware must send a switch command to the microcontroller. The 16-bit TMS/SWDIO command for switching to JTAG mode is defined as b1110.0111.0011.1100, transmitted LSB first. This command can also be represented as 0xE73C when transmitted LSB first. The complete switch sequence should consist of the following transactions on the TCK/SWCLK and TMS/SWDIO signals:

1. Send at least 50 TCK/SWCLK cycles with TMS/SWDIO High to ensure that both JTAG and SWD are in their reset states.
2. Send the 16-bit SWD-to-JTAG switch command, 0xE73C, on TMS/SWDIO.
3. Send at least 50 TCK/SWCLK cycles with TMS/SWDIO High to ensure that if SWJ-DP was already in JTAG mode before sending the switch sequence, the JTAG goes into the Test Logic Reset state.

To verify that the Debug Access Port (DAP) has switched to the JTAG operating mode, set the JTAG Instruction Register (IR) to the IDCODE instruction and shift out the Data Register (DR). The DR value can be compared against the device's known IDCODE to verify the switch.

## 4.4 Initialization and Configuration

After a Power-On-Reset or an external reset ($\overline{\text{RST}}$), the JTAG pins are automatically configured for JTAG communication. No user-defined initialization or configuration is needed. However, if the user application changes these pins to their GPIO function, they must be configured back to their JTAG functionality before JTAG communication can be restored. This is done by enabling the four JTAG pins (PC[3:0]) for their alternate function using the **GPIOAFSEL** register. In addition to enabling the alternate functions, any other changes to the GPIO pad configurations on the four JTAG pins (PC[3:0]) should be reverted to their default settings.

## 4.5 Register Descriptions

There are no APB-accessible registers in the JTAG TAP Controller or Shift Register chains. The registers within the JTAG controller are all accessed serially through the TAP Controller. The registers can be broken down into two main categories: Instruction Registers and Data Registers.

### 4.5.1 Instruction Register (IR)

The JTAG TAP Instruction Register (IR) is a four-bit serial scan chain connected between the JTAG TDI and TDO pins with a parallel load register. When the TAP Controller is placed in the correct states, bits can be shifted into the Instruction Register. Once these bits have been shifted into the chain and updated, they are interpreted as the current instruction. The decode of the Instruction Register bits is shown in Table 4-3 on page 167. A detailed explanation of each instruction, along with its associated Data Register, follows.

**Table 4-3. JTAG Instruction Register Commands**

| IR[3:0] | Instruction | Description |
|---|---|---|
| 0000 | EXTEST | Drives the values preloaded into the Boundary Scan Chain by the SAMPLE/PRELOAD instruction onto the pads. |
| 0001 | INTEST | Drives the values preloaded into the Boundary Scan Chain by the SAMPLE/PRELOAD instruction into the controller. |
| 0010 | SAMPLE / PRELOAD | Captures the current I/O values and shifts the sampled values out of the Boundary Scan Chain while new preload data is shifted in. |
| 1000 | ABORT | Shifts data into the ARM Debug Port Abort Register. |
| 1010 | DPACC | Shifts data into and out of the ARM DP Access Register. |
| 1011 | APACC | Shifts data into and out of the ARM AC Access Register. |
| 1110 | IDCODE | Loads manufacturing information defined by the *IEEE Standard 1149.1* into the IDCODE chain and shifts it out. |
| 1111 | BYPASS | Connects TDI to TDO through a single Shift Register chain. |
| All Others | Reserved | Defaults to the BYPASS instruction to ensure that TDI is always connected to TDO. |

#### 4.5.1.1 EXTEST Instruction

The EXTEST instruction is not associated with its own Data Register chain. The EXTEST instruction uses the data that has been preloaded into the Boundary Scan Data Register using the SAMPLE/PRELOAD instruction. When the EXTEST instruction is present in the Instruction Register, the preloaded data in the Boundary Scan Data Register associated with the outputs and output enables are used to drive the GPIO pads rather than the signals coming from the core. This allows tests to be developed that drive known values out of the controller, which can be used to verify connectivity. While the EXTEST instruction is present in the Instruction Register, the Boundary Scan Data Register can be accessed to sample and shift out the current data and load new data into the Boundary Scan Data Register.

#### 4.5.1.2 INTEST Instruction

The INTEST instruction is not associated with its own Data Register chain. The INTEST instruction uses the data that has been preloaded into the Boundary Scan Data Register using the SAMPLE/PRELOAD instruction. When the INTEST instruction is present in the Instruction Register, the preloaded data in the Boundary Scan Data Register associated with the inputs are used to drive the signals going into the core rather than the signals coming from the GPIO pads. This allows tests to be developed that drive known values into the controller, which can be used for testing.While the INTEXT instruction is present in the Instruction Register, the Boundary Scan Data Register can be accessed to sample and shift out the current data and load new data into the Boundary Scan Data Register.

#### 4.5.1.3 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction connects the Boundary Scan Data Register chain between TDI and TDO. This instruction samples the current state of the pad pins for observation and preloads new test data. Each GPIO pad has an associated input, output, and output enable signal. When the TAP controller enters the Capture DR state during this instruction, the input, output, and output-enable signals to each of the GPIO pads are captured. These samples are serially shifted out of TDO while the TAP controller is in the Shift DR state and can be used for observation or comparison in various tests.

While these samples of the inputs, outputs, and output enables are being shifted out of the Boundary Scan Data Register, new data is being shifted into the Boundary Scan Data Register from TDI. Once the new data has been shifted into the Boundary Scan Data Register, the data is saved in the parallel load registers when the TAP controller enters the Update DR state. This update of the parallel load register preloads data into the Boundary Scan Data Register that is associated with each input, output, and output enable. This preloaded data can be used with the EXTEST and INTEST instructions to drive data into or out of the controller. Please see “Boundary Scan Data Register” on page 169 for more information.

#### 4.5.1.4 ABORT Instruction

The ABORT instruction connects the associated ABORT Data Register chain between TDI and TDO. This instruction provides read and write access to the ABORT Register of the ARM Debug Access Port (DAP). Shifting the proper data into this Data Register clears various error bits or initiates a DAP abort of a previous request. Please see the “ABORT Data Register” on page 170 for more information.

#### 4.5.1.5 DPACC Instruction

The DPACC instruction connects the associated DPACC Data Register chain between TDI and TDO. This instruction provides read and write access to the DPACC Register of the ARM Debug Access Port (DAP). Shifting the proper data into this register and reading the data output from this register allows read and write access to the ARM debug and status registers. Please see “DPACC Data Register” on page 170 for more information.

#### 4.5.1.6 APACC Instruction

The APACC instruction connects the associated APACC Data Register chain between TDI and TDO. This instruction provides read and write access to the APACC Register of the ARM Debug Access Port (DAP). Shifting the proper data into this register and reading the data output from this register allows read and write access to internal components and buses through the Debug Port. Please see “APACC Data Register” on page 170 for more information.

#### 4.5.1.7 IDCODE Instruction

The IDCODE instruction connects the associated IDCODE Data Register chain between TDI and TDO. This instruction provides information on the manufacturer, part number, and version of the ARM core. This information can be used by testing equipment and debuggers to automatically configure their input and output data streams. IDCODE is the default instruction that is loaded into the JTAG Instruction Register when a Power-On-Reset (POR) is asserted, or the Test-Logic-Reset state is entered. Please see “IDCODE Data Register” on page 169 for more information.

#### 4.5.1.8 BYPASS Instruction

The BYPASS instruction connects the associated BYPASS Data Register chain between TDI and TDO. This instruction is used to create a minimum length serial path between the TDI and TDO ports.

The BYPASS Data Register is a single-bit shift register. This instruction improves test efficiency by allowing components that are not needed for a specific test to be bypassed in the JTAG scan chain by loading them with the BYPASS instruction. Please see “BYPASS Data Register” on page 169 for more information.

## 4.5.2 Data Registers

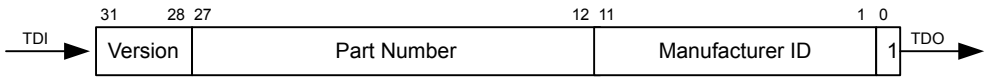
The JTAG module contains six Data Registers. These include: IDCODE, BYPASS, Boundary Scan, APACC, DPACC, and ABORT serial Data Register chains. Each of these Data Registers is discussed in the following sections.

### 4.5.2.1 IDCODE Data Register

The format for the 32-bit IDCODE Data Register defined by the *IEEE Standard 1149.1* is shown in Figure 4-3 on page 169. The standard requires that every JTAG-compliant device implement either the IDCODE instruction or the BYPASS instruction as the default instruction. The LSB of the IDCODE Data Register is defined to be a 1 to distinguish it from the BYPASS instruction, which has an LSB of 0. This allows auto configuration test tools to determine which instruction is the default instruction.

The major uses of the JTAG port are for manufacturer testing of component assembly, and program development and debug. To facilitate the use of auto-configuration debug tools, the IDCODE instruction outputs a value of 0x3BA0.0477. This allows the debuggers to automatically configure themselves to work correctly with the Cortex-M3 during debug.

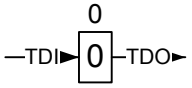
**Figure 4-3. IDCODE Register Format**




### 4.5.2.2 BYPASS Data Register

The format for the 1-bit BYPASS Data Register defined by the *IEEE Standard 1149.1* is shown in Figure 4-4 on page 169. The standard requires that every JTAG-compliant device implement either the BYPASS instruction or the IDCODE instruction as the default instruction. The LSB of the BYPASS Data Register is defined to be a 0 to distinguish it from the IDCODE instruction, which has an LSB of 1. This allows auto configuration test tools to determine which instruction is the default instruction.

**Figure 4-4. BYPASS Register Format**

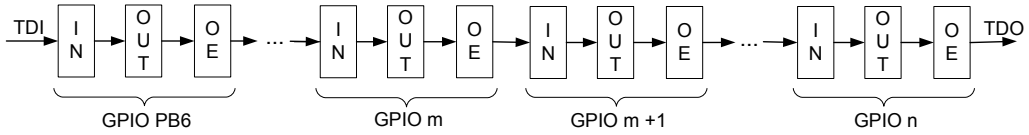



### 4.5.2.3 Boundary Scan Data Register

The format of the Boundary Scan Data Register is shown in Figure 4-5 on page 170. Each GPIO pin, starting with a GPIO pin next to the JTAG port pins, is included in the Boundary Scan Data Register. Each GPIO pin has three associated digital signals that are included in the chain. These signals are input, output, and output enable, and are arranged in that order as can be seen in the figure.

When the Boundary Scan Data Register is accessed with the SAMPLE/PRELOAD instruction, the input, output, and output enable from each digital pad are sampled and then shifted out of the chain to be verified. The sampling of these values occurs on the rising edge of TCK in the Capture DR state of the TAP controller. While the sampled data is being shifted out of the Boundary Scan chain in the Shift DR state of the TAP controller, new data can be preloaded into the chain for use with the EXTEST and INTEST instructions. These instructions either force data out of the controller, with the EXTEST instruction, or into the controller, with the INTEST instruction.

**Figure 4-5. Boundary Scan Register Format**




#### 4.5.2.4 APACC Data Register

The format for the 35-bit APACC Data Register defined by ARM is described in the *ARM® Debug Interface V5 Architecture Specification*.

#### 4.5.2.5 DPACC Data Register

The format for the 35-bit DPACC Data Register defined by ARM is described in the *ARM® Debug Interface V5 Architecture Specification*.

#### 4.5.2.6 ABORT Data Register

The format for the 35-bit ABORT Data Register defined by ARM is described in the *ARM® Debug Interface V5 Architecture Specification*.

# 5 System Control

System control determines the overall operation of the device. It provides information about the device, controls the clocking to the core and individual peripherals, and handles reset detection and reporting.

## 5.1 Signal Description

Table 5-1 on page 171 lists the external signals of the System Control module and describes the function of each. The NMI signal is the alternate function for and functions as a GPIO after reset. under commit protection and require a special process to be configured as any alternate function or to subsequently return to the GPIO function, see “Commit Control” on page 357. The column in the table below titled "Pin Assignment" lists the GPIO pin placement for the NMI signal. The AFSEL bit in the **GPIO Alternate Function Select (GPIOAFSEL)** register (page 371) should be set to choose the NMI function. For more information on configuring GPIOs, see “General-Purpose Input/Outputs (GPIOs)” on page 351. The remaining signals (with the word "fixed" in the Pin Assignment column) have a fixed pin assignment and function.

**Table 5-1. System Control & Clocks Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| NMI | 55 | I | TTL | Non-maskable interrupt. |
| OSC0 | 30 | I | Analog | Main oscillator crystal input or an external clock reference input. |
| OSC1 | 31 | O | Analog | Main oscillator crystal output. Leave unconnected when using a single-ended clock source. |
| RST | 40 | I | TTL | System reset input. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 5.2 Functional Description

The System Control module provides the following capabilities:

- Device identification (see “Device Identification” on page 171)
- Local control, such as reset (see “Reset Control” on page 171), power (see “Power Control” on page 176) and clock control (see “Clock Control” on page 176)
- System control (Run, Sleep, and Deep-Sleep modes); see “System Control” on page 182

### 5.2.1 Device Identification

Several read-only registers provide software with information on the microcontroller, such as version, part number, SRAM size, flash size, and other features. See the **DID0**, **DID1**, and **DC0**-**DC7** registers.

### 5.2.2 Reset Control

This section discusses aspects of hardware functions during reset as well as system software requirements following the reset sequence.

#### 5.2.2.1 Reset Sources

The controller has six sources of reset:

1. External reset input pin ($\overline{\text{RST}}$) assertion; see "External $\overline{\text{RST}}$ Pin" on page 173.
2. Power-on reset (POR); see "Power-On Reset (POR)" on page 172.
3. Internal brown-out (BOR) detector; see "Brown-Out Reset (BOR)" on page 174.
4. Software-initiated reset (with the software reset registers); see "Software Reset" on page 174.
5. A watchdog timer reset condition violation; see "Watchdog Timer Reset" on page 175.
6. MOSC failure; see "Main Oscillator Verification Failure" on page 176.

Table 5-2 provides a summary of results of the various reset operations.

**Table 5-2. Reset Sources**

| Reset Source | Core Reset? | JTAG Reset? | On-Chip Peripherals Reset? |
|---|---|---|---|
| Power-On Reset | Yes | Yes | Yes |
| $\overline{\text{RST}}$ | Yes | Pin Config Only | Yes |
| Brown-Out Reset | Yes | No | Yes |
| Software System Request Reset[a] | Yes | No | Yes |
| Software Peripheral Reset | No | No | Yes[b] |
| Watchdog Reset | Yes | No | Yes |
| MOSC Failure Reset | Yes | No | Yes |

a. By using the SYSRESREQ bit in the ARM Cortex-M3 **Application Interrupt and Reset Control (APINT)** register

b. Programmable on a module-by-module basis using the Software Reset Control Registers.

After a reset, the **Reset Cause (RESC)** register is set with the reset cause. The bits in this register are sticky and maintain their state across multiple reset sequences, except when an internal POR or an external reset is the cause, and then all the other bits in the **RESC** register are cleared except for the POR or EXT indicator.

### 5.2.2.2 Power-On Reset (POR)

**Note:** The power-on reset also resets the JTAG controller. An external reset does not.

The internal Power-On Reset (POR) circuit monitors the power supply voltage ($V_{DD}$) and generates a reset signal to all of the internal logic including JTAG when the power supply ramp reaches a threshold value ($V_{TH}$). The microcontroller must be operating within the specified operating parameters when the on-chip power-on reset pulse is complete. The 3.3-V power supply to the microcontroller must reach 3.0 V within 10 msec of $V_{DD}$ crossing 2.0 V to guarantee proper operation. For applications that require the use of an external reset signal to hold the microcontroller in reset longer than the internal POR, the $\overline{\text{RST}}$ input may be used as discussed in "External $\overline{\text{RST}}$ Pin" on page 173.

The Power-On Reset sequence is as follows:

1. The microcontroller waits for internal POR to go inactive.
2. The internal reset is released and the core loads from memory the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution.

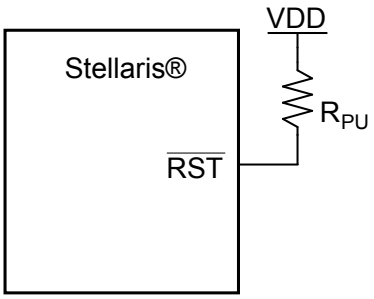
The internal POR is only active on the initial power-up of the microcontroller. The Power-On Reset timing is shown in Figure 21-5 on page 774.

### 5.2.2.3 External $\overline{\text{RST}}$ Pin

**Note:** It is recommended that the trace for the $\overline{\text{RST}}$ signal must be kept as short as possible. Be sure to place any components connected to the $\overline{\text{RST}}$ signal as close to the microcontroller as possible.

If the application only uses the internal POR circuit, the $\overline{\text{RST}}$ input must be connected to the power supply ($V_{DD}$) through an optional pull-up resistor (0 to 100K Ω) as shown in Figure 5-1 on page 173.

**Figure 5-1. Basic $\overline{\text{RST}}$ Configuration**



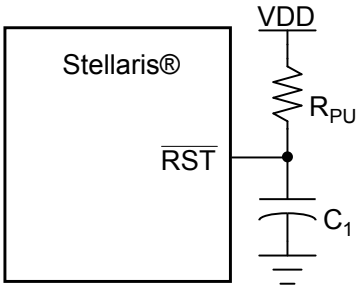

$R_{PU}$ = 0 to 100 kΩ

The external reset pin ($\overline{\text{RST}}$) resets the microcontroller including the core and all the on-chip peripherals except the JTAG TAP controller (see “JTAG Interface” on page 159). The external reset sequence is as follows:

1. The external reset pin ($\overline{\text{RST}}$) is asserted for the duration specified by $T_{MIN}$ and then de-asserted (see “Reset” on page 773).
2. The internal reset is released and the core loads from memory the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution.

To improve noise immunity and/or to delay reset at power up, the $\overline{\text{RST}}$ input may be connected to an RC network as shown in Figure 5-2 on page 173.

**Figure 5-2. External Circuitry to Extend Power-On Reset**

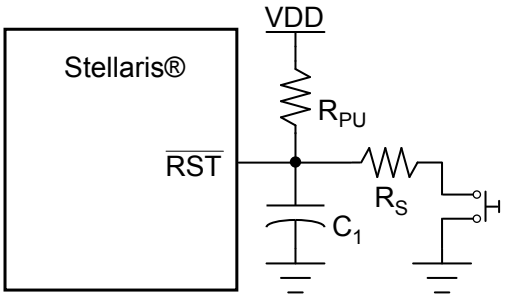



$R_{PU}$ = 1 kΩ to 100 kΩ

$C_1$ = 1 nF to 10 µF

If the application requires the use of an external reset switch, Figure 5-3 on page 174 shows the proper circuitry to use.

**Figure 5-3. Reset Circuit Controlled by Switch**




Typical $R_{PU}$ = 10 kΩ

Typical $R_S$ = 470 Ω

$C_1$ = 10 nF

The $R_{PU}$ and $C_1$ components define the power-on delay.

The external reset timing is shown in Figure 21-4 on page 773.

#### 5.2.2.4 Brown-Out Reset (BOR)

A drop in the input voltage resulting in the assertion of the internal brown-out detector can be used to reset the controller. This is initially disabled and may be enabled by software.

The system provides a brown-out detection circuit that triggers if the power supply ($V_{DD}$) drops below a brown-out threshold voltage ($V_{BTH}$). If a brown-out condition is detected, the system may generate a controller interrupt or a system reset.

Brown-out resets are controlled with the **Power-On and Brown-Out Reset Control (PBORCTL)** register. The `BORIOR` bit in the **PBORCTL** register must be set for a brown-out condition to trigger a reset.

The brown-out reset is equivalent to an assertion of the external $\overline{\text{RST}}$ input and the reset is held active until the proper $V_{DD}$ level is restored. The **RESC** register can be examined in the reset interrupt handler to determine if a Brown-Out condition was the cause of the reset, thus allowing software to determine what actions are required to recover.

The internal Brown-Out Reset timing is shown in Figure 21-6 on page 774.

#### 5.2.2.5 Software Reset

Software can reset a specific peripheral or generate a reset to the entire system .

Peripherals can be individually reset by software via three registers that control reset signals to each peripheral (see the **SRCRn** registers). If the bit position corresponding to a peripheral is set and subsequently cleared, the peripheral is reset. The encoding of the reset registers is consistent with the encoding of the clock gating control for peripherals and on-chip functions (see “System Control” on page 182). Note that all reset signals for all clocks of the specified unit are asserted as a result of a software-initiated reset.

The entire system can be reset by software by setting the SYSRESETREQ bit in the Cortex-M3 Application Interrupt and Reset Control register resets the entire system including the core. The software-initiated system reset sequence is as follows:

1. A software system reset is initiated by writing the SYSRESETREQ bit in the ARM Cortex-M3 Application Interrupt and Reset Control register.
2. An internal reset is asserted.
3. The internal reset is deasserted and the controller loads from memory the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution.

The software-initiated system reset timing is shown in Figure 21-7 on page 774.

#### 5.2.2.6 Watchdog Timer Reset

The watchdog timer module's function is to prevent system hangs. The watchdog timer can be configured to generate an interrupt to the controller on its first time-out, and to generate a reset signal on its second time-out.

After the first time-out event, the 32-bit counter is reloaded with the value of the **Watchdog Timer Load (WDTLOAD)** register, and the timer resumes counting down from that value. If the timer counts down to its zero state again before the first time-out interrupt is cleared, and the reset signal has been enabled, the watchdog timer asserts its reset signal to the system. The watchdog timer reset sequence is as follows:

1. The watchdog timer times out for the second time without being serviced.
2. An internal reset is asserted.
3. The internal reset is released and the controller loads from memory the initial stack pointer, the initial program counter, the first instruction designated by the program counter, and begins execution.

The watchdog reset timing is shown in Figure 21-8 on page 774.

### 5.2.3 Non-Maskable Interrupt

The controller has two sources of non-maskable interrupt (NMI):

- The assertion of the NMI signal.
- A main oscillator verification error.

If both sources of NMI are enabled, software must check that the main oscillator verification is the cause of the interrupt in order to distinguish between the two sources.

#### 5.2.3.1 NMI Pin

The alternate function to GPIO port pin B7 is an NMI signal. The alternate function must be enabled in the GPIO for the signal to be used as an interrupt, as described in “General-Purpose Input/Outputs (GPIOs)” on page 351. Note that enabling the NMI alternate function requires the use of the GPIO lock and commit function just like the GPIO port pins associated with JTAG/SWD functionality. The active sense of the NMI signal is High; asserting the enabled NMI signal above $V_{IH}$ initiates the NMI interrupt sequence.

#### 5.2.3.2 Main Oscillator Verification Failure

The main oscillator verification circuit may generate a reset event, at which time a Power-on Reset is generated and control is transferred to the NMI handler. The NMI handler is used to address the main oscillator verification failure because the necessary code can be removed from the general reset handler, speeding up reset processing. The detection circuit is enabled using the CVAL bit in the **Main Oscillator Control (MOSCCTL)** register. The main oscillator verification error is indicated in the main oscillator fail status bit (MOSCFAIL) bit in the **Reset Cause (RESC)** register. The main oscillator verification circuit action is described in more detail in “Clock Control” on page 176.

### 5.2.4 Power Control

The Stellaris® microcontroller provides an integrated LDO regulator that is used to provide power to the majority of the controller's internal logic. For power reduction, the LDO regulator provides software a mechanism to adjust the regulated value, in small increments (VSTEP), over the range of 2.25 V to 2.75 V (inclusive)—or 2.5 V ± 10%. The adjustment is made by changing the value of the VADJ field in the **LDO Power Control (LDOPCTL)** register.

**Note:** On the printed circuit board, use the LDO output as the source of VDD25 input. Do not use an external regulator to supply the voltage to VDD25. In addition, the LDO requires decoupling capacitors. See “On-Chip Low Drop-Out (LDO) Regulator Characteristics” on page 767.

VDDA must be supplied with 3.3 V, or the microcontroller does not function properly. VDDA is the supply for all of the analog circuitry on the device, including the clock circuitry.

### 5.2.5 Clock Control

System control determines the control of clocks in this part.

#### 5.2.5.1 Fundamental Clock Sources

There are multiple clock sources for use in the device:

- **Internal Oscillator (IOSC).** The internal oscillator is an on-chip clock source. It does not require the use of any external components. The frequency of the internal oscillator is 12 MHz ± 30%. Applications that do not depend on accurate clock sources may use this clock source to reduce system cost. The internal oscillator is the clock source the device uses during and following POR. If the main oscillator is required, software must enable the main oscillator following reset and allow the main oscillator to stabilize before changing the clock reference.

- **Main Oscillator (MOSC).** The main oscillator provides a frequency-accurate clock source by one of two means: an external single-ended clock source is connected to the OSC0 input pin, or an external crystal is connected across the OSC0 input and OSC1 output pins. If the PLL is being used, the crystal value must be one of the supported frequencies between 3.579545 MHz through 16.384 MHz (inclusive). If the PLL is not being used, the crystal may be any one of the supported frequencies between 1 MHz and 16.384 MHz. The single-ended clock source range is from DC through the specified speed of the device. The supported crystals are listed in the XTAL bit field in the **RCC** register (see page 194). Note that the MOSC must have a clock source for the USB PLL.

- **Internal 30-kHz Oscillator.** The internal 30-kHz oscillator is similar to the internal oscillator, except that it provides an operational frequency of 30 kHz ± 50%. It is intended for use during Deep-Sleep power-saving modes. This power-savings mode benefits from reduced internal switching and also allows the main oscillator to be powered down.

- **External Real-Time Oscillator.** The external real-time oscillator provides a low-frequency, accurate clock reference. It is intended to provide the system with a real-time clock source. The real-time oscillator is part of the Hibernation Module (see “Hibernation Module” on page 238) and may also provide an accurate source of Deep-Sleep or Hibernate mode power savings.

The internal system clock (SysClk), is derived from any of the above sources plus two others: the output of the main internal PLL, and the internal oscillator divided by four (3 MHz ± 30%). The frequency of the PLL clock reference must be in the range of 3.579545 MHz to 16.384 MHz (inclusive). Table 5-3 on page 177 shows how the various clock sources can be used in a system.

**Table 5-3. Clock Source Options**

| Clock Source | Drive PLL? | | Used as SysClk? | |
|---|---|---|---|---|
| Internal Oscillator (12 MHz) | No | BYPASS = 1 | Yes | BYPASS = 1, OSCSRC = 0x1 |
| Internal Oscillator divide by 4 (3 MHz) | No | BYPASS = 1 | Yes | BYPASS = 1, OSCSRC = 0x2 |
| Main Oscillator | Yes | BYPASS = 0, OSCSRC = 0x0 | Yes | BYPASS = 1, OSCSRC = 0x0 |
| Internal 30-kHz Oscillator | No | BYPASS = 1 | Yes | BYPASS = 1, OSCSRC = 0x3 |
| External Real-Time Oscillator | No | BYPASS = 1 | Yes | BYPASS = 1, OSCSRC2 = 0x7 |

### 5.2.5.2 Clock Configuration

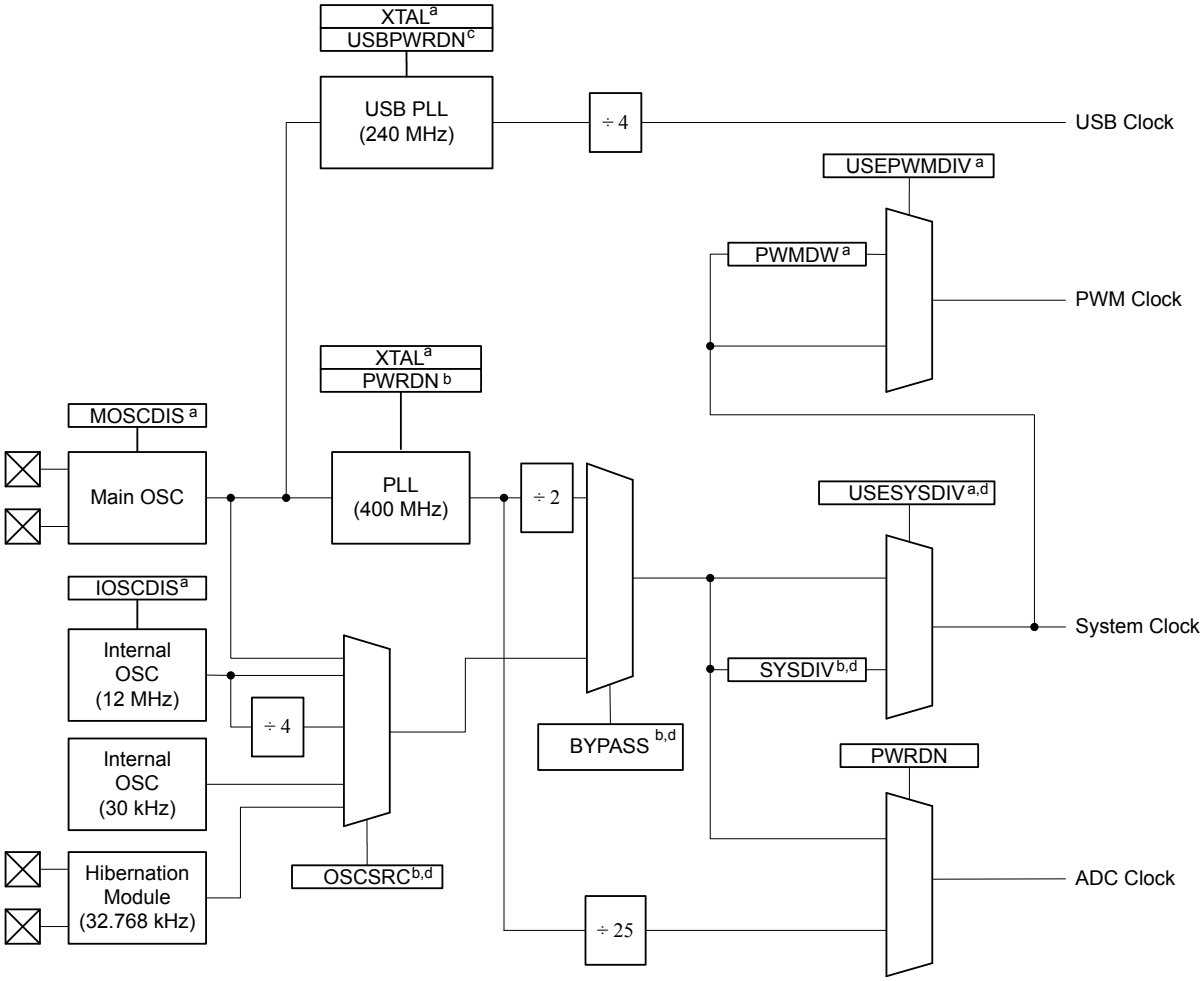
The **Run-Mode Clock Configuration (RCC)** and **Run-Mode Clock Configuration 2 (RCC2)** registers provide control for the system clock. The **RCC2** register is provided to extend fields that offer additional encodings over the **RCC** register. When used, the **RCC2** register field values are used by the logic over the corresponding field in the **RCC** register. In particular, **RCC2** provides for a larger assortment of clock configuration options. These registers control the following clock functionality:

- Source of clocks in sleep and deep-sleep modes
- System clock derived from PLL or other clock source
- Enabling/disabling of oscillators and PLL
- Clock divisors
- Crystal input selection

Figure 5-4 on page 178 shows the logic for the main clock tree. The peripheral blocks are driven by the system clock signal and can be individually enabled/disabled. The ADC clock signal is automatically divided down to 16 MHz for proper ADC operation.

**Note:** When the ADC module is in operation, the system clock must be at least 16 MHz. When the USB module is in operation, MOSC must be provided with a clock source, and the system clock must be at least 30 MHz.

**Figure 5-4. Main Clock Tree**




a. Control provided by RCC register bit/field.
b. Control provided by RCC register bit/field or RCC2 register bit/field, if overridden with RCC2 register bit USERCC2.
c. Control provided by RCC2 register bit/field.
d. Also may be controlled by DSLPCLKCFG when in deep sleep mode.

**Note:** The figure above shows all features available on all Stellaris® DustDevil-class devices. Not all peripherals may be available on this device.

In the **RCC** register, the SYSDIV field specifies which divisor is used to generate the system clock from either the PLL output or the oscillator source (depending on how the BYPASS bit in this register is configured). When using the PLL, the VCO frequency of 400 MHz is predivided by 2 before the divisor is applied. Table 5-4 shows how the SYSDIV encoding affects the system clock frequency, depending on whether the PLL is used (BYPASS=0) or another clock source is used (BYPASS=1). The divisor is equivalent to the SYSDIV encoding plus 1. For a list of possible clock sources, see Table 5-3 on page 177.

**Table 5-4. Possible System Clock Frequencies Using the SYSDIV Field**

| SYSDIV | Divisor | Frequency (BYPASS=0) | Frequency (BYPASS=1) | StellarisWare Parameter[a] |
|---|---|---|---|---|
| 0x0 | /1 | reserved | Clock source frequency/2 | SYSCTL_SYSDIV_1[b] |
| 0x1 | /2 | reserved | Clock source frequency/2 | SYSCTL_SYSDIV_2 |
| 0x2 | /3 | reserved | Clock source frequency/3 | SYSCTL_SYSDIV_3 |
| 0x3 | /4 | 50 MHz | Clock source frequency/4 | SYSCTL_SYSDIV_4 |
| 0x4 | /5 | 40 MHz | Clock source frequency/5 | SYSCTL_SYSDIV_5 |
| 0x5 | /6 | 33.33 MHz | Clock source frequency/6 | SYSCTL_SYSDIV_6 |
| 0x6 | /7 | 28.57 MHz | Clock source frequency/7 | SYSCTL_SYSDIV_7 |
| 0x7 | /8 | 25 MHz | Clock source frequency/8 | SYSCTL_SYSDIV_8 |
| 0x8 | /9 | 22.22 MHz | Clock source frequency/9 | SYSCTL_SYSDIV_9 |
| 0x9 | /10 | 20 MHz | Clock source frequency/10 | SYSCTL_SYSDIV_10 |
| 0xA | /11 | 18.18 MHz | Clock source frequency/11 | SYSCTL_SYSDIV_11 |
| 0xB | /12 | 16.67 MHz | Clock source frequency/12 | SYSCTL_SYSDIV_12 |
| 0xC | /13 | 15.38 MHz | Clock source frequency/13 | SYSCTL_SYSDIV_13 |
| 0xD | /14 | 14.29 MHz | Clock source frequency/14 | SYSCTL_SYSDIV_14 |
| 0xE | /15 | 13.33 MHz | Clock source frequency/15 | SYSCTL_SYSDIV_15 |
| 0xF | /16 | 12.5 MHz (default) | Clock source frequency/16 | SYSCTL_SYSDIV_16 |

a. This parameter is used in functions such as SysCtlClockSet() in the Stellaris Peripheral Driver Library.

b. SYSCTL_SYSDIV_1 does not set the USESYSDIV bit. As a result, using this parameter without enabling the PLL results in the system clock having the same frequency as the clock source.

The SYSDIV2 field in the **RCC2** register is 2 bits wider than the SYSDIV field in the **RCC** register so that additional larger divisors up to /64 are possible, allowing a lower system clock frequency for improved Deep Sleep power consumption. When using the PLL, the VCO frequency of 400 MHz is predivided by 2 before the divisor is applied. The divisor is equivalent to the SYSDIV2 encoding plus 1. Table 5-5 shows how the SYSDIV2 encoding affects the system clock frequency, depending on whether the PLL is used (BYPASS2=0) or another clock source is used (BYPASS2=1). For a list of possible clock sources, see Table 5-3 on page 177.

**Table 5-5. Examples of Possible System Clock Frequencies Using the SYSDIV2 Field**

| SYSDIV2 | Divisor | Frequency (BYPASS2=0) | Frequency (BYPASS2=1) | StellarisWare Parameter[a] |
|---|---|---|---|---|
| 0x00 | /1 | reserved | Clock source frequency/2 | SYSCTL_SYSDIV_1[b] |
| 0x01 | /2 | reserved | Clock source frequency/2 | SYSCTL_SYSDIV_2 |
| 0x02 | /3 | reserved | Clock source frequency/3 | SYSCTL_SYSDIV_3 |
| 0x03 | /4 | 50 MHz | Clock source frequency/4 | SYSCTL_SYSDIV_4 |
| 0x04 | /5 | 40 MHz | Clock source frequency/5 | SYSCTL_SYSDIV_5 |
| 0x05 | /6 | 33.33 MHz | Clock source frequency/6 | SYSCTL_SYSDIV_6 |
| 0x06 | /7 | 28.57 MHz | Clock source frequency/7 | SYSCTL_SYSDIV_7 |
| 0x07 | /8 | 25 MHz | Clock source frequency/8 | SYSCTL_SYSDIV_8 |
| 0x08 | /9 | 22.22 MHz | Clock source frequency/9 | SYSCTL_SYSDIV_9 |
| 0x09 | /10 | 20 MHz | Clock source frequency/10 | SYSCTL_SYSDIV_10 |
| ... | ... | ... | ... | ... |

**Table 5-5. Examples of Possible System Clock Frequencies Using the SYSDIV2 Field *(continued)***

| SYSDIV2 | Divisor | Frequency (BYPASS2=0) | Frequency (BYPASS2=1) | StellarisWare Parameter[a] |
|---|---|---|---|---|
| 0x3F | /64 | 3.125 MHz | Clock source frequency/64 | SYSCTL_SYSDIV_64 |

a. This parameter is used in functions such as SysCtlClockSet() in the Stellaris Peripheral Driver Library.

b. SYSCTL_SYSDIV_1 does not set the USESYSDIV bit. As a result, using this parameter without enabling the PLL results in the system clock having the same frequency as the clock source.

#### 5.2.5.3 Crystal Configuration for the Main Oscillator (MOSC)

The main oscillator supports the use of a select number of crystals. If the main oscillator is used by the PLL as a reference clock, the supported range of crystals is 3.579545 to 16.384 MHz, otherwise, the range of supported crystals is 1 to 16.384 MHz.

The XTAL bit in the **RCC** register (see page 194) describes the available crystal choices and default programming values.

Software configures the **RCC** register XTAL field with the crystal number. If the PLL is used in the design, the XTAL field value is internally translated to the PLL settings.

#### 5.2.5.4 Main PLL Frequency Configuration

The main PLL is disabled by default during power-on reset and is enabled later by software if required. Software specifies the output divisor to set the system clock frequency, and enables the main PLL to drive the output. The PLL operates at 400 MHz, but is divided by two prior to the application of the output divisor.

If the main oscillator provides the clock reference to the main PLL, the translation provided by hardware and used to program the PLL is available for software in the **XTAL to PLL Translation (PLLCFG)** register (see page 198). The internal translation provides a translation within ± 1% of the targeted PLL VCO frequency. Table 21-10 on page 770 shows the actual PLL frequency and error for a given crystal choice.

The Crystal Value field (XTAL) in the **Run-Mode Clock Configuration (RCC)** register (see page 194) describes the available crystal choices and default programming of the **PLLCFG** register. Any time the XTAL field changes, the new settings are translated and the internal PLL settings are updated.

To configure the external 32-kHz real-time oscillator as the PLL input reference, program the OSCRC2 field in the **Run-Mode Clock Configuration 2 (RCC2)** register to be 0x7.

#### 5.2.5.5 USB PLL Frequency Configuration

The USB PLL is disabled by default during power-on reset and is enabled later by software. The USB PLL must be enabled and running for proper USB function. The main oscillator is the only clock reference for the USB PLL. The USB PLL is enabled by clearing the USBPWRDN bit of the **RCC2** register. The XTAL bit field (Crystal Value) of the **RCC** register describes the available crystal choices. The main oscillator must be connected to one of the following crystal values in order to correctly generate the USB clock: 4, 5, 6, 8, 10, 12, or 16 MHz. Only these crystals provide the necessary USB PLL VCO frequency to conform with the USB timing specifications.

#### 5.2.5.6 PLL Modes

Both PLLs have two modes of operation: Normal and Power-Down

- Normal: The PLL multiplies the input clock reference and drives the output.

- Power-Down: Most of the PLL internal circuitry is disabled and the PLL does not drive the output.

The modes are programmed using the **RCC**/**RCC2** register fields (see page 194 and page 201).

#### 5.2.5.7 PLL Operation

If a PLL configuration is changed, the PLL output frequency is unstable until it reconverges (relocks) to the new setting. The time between the configuration change and relock is $T_{READY}$ (see Table 21-9 on page 770). During the relock time, the affected PLL is not usable as a clock reference.

Either PLL is changed by one of the following:

- Change to the XTAL value in the **RCC** register—writes of the same value do not cause a relock.
- Change in the PLL from Power-Down to Normal mode.

A counter is defined to measure the $T_{READY}$ requirement. The counter is clocked by the main oscillator. The range of the main oscillator has been taken into account and the down counter is set to 0x1200 (that is, ~600 µs at an 8.192 MHz external oscillator clock). When the XTAL value is greater than 0x0f, the down counter is set to 0x2400 to maintain the required lock time on higher frequency crystal inputs. Hardware is provided to keep the PLL from being used as a system clock until the $T_{READY}$ condition is met after one of the two changes above. It is the user's responsibility to have a stable clock source (like the main oscillator) before the **RCC**/**RCC2** register is switched to use the PLL.

If the main PLL is enabled and the system clock is switched to use the PLL in one step, the system control hardware continues to clock the controller from the oscillator selected by the **RCC**/**RCC2** register until the main PLL is stable ($T_{READY}$ time met), after which it changes to the PLL. Software can use many methods to ensure that the system is clocked from the main PLL, including periodically polling the PLLLRIS bit in the **Raw Interrupt Status (RIS)** register, and enabling the PLL Lock interrupt.

The USB PLL is not protected during the lock time ($T_{READY}$) and software should ensure that the USB PLL has locked before using the interface. Software can use many methods to ensure the $T_{READY}$ period has passed, including periodically polling the USBPLLLRIS bit in the **Raw Interrupt Status (RIS)** register, and enabling the USB PLL Lock interrupt.

#### 5.2.5.8 Main Oscillator Verification Circuit

A circuit is added to ensure that the main oscillator is running at the appropriate frequency. The circuit monitors the main oscillator frequency and signals if the frequency is outside of the allowable band of attached crystals.

The detection circuit is enabled using the CVAL bit in the **Main Oscillator Control (MOSCCTL)** register. If this circuit is enabled and detects an error, the following sequence is performed by the hardware:

1. The MOSCFAIL bit in the **Reset Cause (RESC)** register is set.
2. If the internal oscillator (IOSC) is disabled, it is enabled.
3. The system clock is switched from the main oscillator to the IOSC.
4. An internal power-on reset is initiated that lasts for 32 IOSC periods.
5. Reset is de-asserted and the processor is directed to the NMI handler during the reset sequence.

### 5.2.6 System Control

For power-savings purposes, the **RCGCn** , **SCGCn** , and **DCGCn** registers control the clock gating logic for each peripheral or block in the system while the controller is in Run, Sleep, and Deep-Sleep mode, respectively.

There are four levels of operation for the device defined as:

- **Run Mode.** In Run mode, the controller actively executes code. Run mode provides normal operation of the processor and all of the peripherals that are currently enabled by the **RCGCn** registers. The system clock can be any of the available clock sources including the PLL.

- **Sleep Mode.** In Sleep mode, the clock frequency of the active peripherals is unchanged, but the processor and the memory subsystem are not clocked and therefore no longer execute code. Sleep mode is entered by the Cortex-M3 core executing a `WFI(Wait for Interrupt)` instruction. Any properly configured interrupt event in the system will bring the processor back into Run mode. See “Power Management” on page 90 for more details.

  Peripherals are clocked that are enabled in the **SCGCn** register when auto-clock gating is enabled (see the **RCC** register) or the **RCGCn** register when the auto-clock gating is disabled. The system clock has the same source and frequency as that during Run mode.

- **Deep-Sleep Mode.** In Deep-Sleep mode, the clock frequency of the active peripherals may change (depending on the Run mode clock configuration) in addition to the processor clock being stopped. An interrupt returns the device to Run mode from one of the sleep modes; the sleep modes are entered on request from the code. Deep-Sleep mode is entered by first writing the Deep Sleep Enable bit in the ARM Cortex-M3 NVIC system control register and then executing a `WFI` instruction. Any properly configured interrupt event in the system will bring the processor back into Run mode. See “Power Management” on page 90 for more details.

  The Cortex-M3 processor core and the memory subsystem are not clocked. Peripherals are clocked that are enabled in the **DCGCn** register when auto-clock gating is enabled (see the **RCC** register) or the **RCGCn** register when auto-clock gating is disabled. The system clock source is the main oscillator by default or the internal oscillator specified in the **DSLPCLKCFG** register if one is enabled. When the **DSLPCLKCFG** register is used, the internal oscillator is powered up, if necessary, and the main oscillator is powered down. If the PLL is running at the time of the WFI instruction, hardware will power the PLL down and override the `SYSDIV` field of the active **RCC**/**RCC2** register, to be determined by the `DSDIVORIDE` setting in the **DSLPCLKCFG** register, up to /16 or /64 respectively. When the Deep-Sleep exit event occurs, hardware brings the system clock back to the source and frequency it had at the onset of Deep-Sleep mode before enabling the clocks that had been stopped during the Deep-Sleep duration.

- **Hibernate Mode.** In this mode, the power supplies are turned off to the main part of the device and only the Hibernation module's circuitry is active. An external wake event or RTC event is required to bring the device back to Run mode. The Cortex-M3 processor and peripherals outside of the Hibernation module see a normal "power on" sequence and the processor starts running code. It can determine that it has been restarted from Hibernate mode by inspecting the Hibernation module registers.

**Caution – If the Cortex-M3 Debug Access Port (DAP) has been enabled, and the device wakes from a low power sleep or deep-sleep mode, the core may start executing code before all clocks to peripherals have been restored to their run mode configuration. The DAP is usually enabled by software tools accessing the JTAG or SWD interface when debugging or flash programming. If this condition occurs, a Hard Fault is triggered when software accesses a peripheral with an invalid clock.**

**A software delay loop can be used at the beginning of the interrupt routine that is used to wake up a system from a WFI (Wait For Interrupt) instruction. This stalls the execution of any code that accesses a peripheral register that might cause a fault. This loop can be removed for production software as the DAP is most likely not enabled during normal execution.**

**Because the DAP is disabled by default (power on reset), the user can also power-cycle the device. The DAP is not enabled unless it is enabled through the JTAG or SWD interface.**

## 5.3 Initialization and Configuration

The PLL is configured using direct register writes to the **RCC**/**RCC2** register. If the **RCC2** register is being used, the USERCC2 bit must be set and the appropriate **RCC2** bit/field is used. The steps required to successfully change the PLL-based system clock are:

1. Bypass the PLL and system clock divider by setting the BYPASS bit and clearing the USESYS bit in the **RCC** register. This configures the system to run off a “raw” clock source and allows for the new PLL configuration to be validated before switching the system clock to the PLL.
2. Select the crystal value (XTAL) and oscillator source (OSCSRC), and clear the PWRDN bit in **RCC**/**RCC2**. Setting the XTAL field automatically pulls valid PLL configuration data for the appropriate crystal, and clearing the PWRDN bit powers and enables the PLL and its output.
3. Select the desired system divider (SYSDIV) in **RCC**/**RCC2** and set the USESYS bit in **RCC**. The SYSDIV field determines the system frequency for the microcontroller.
4. Wait for the PLL to lock by polling the PLLLRIS bit in the **Raw Interrupt Status (RIS)** register.
5. Enable use of the PLL by clearing the BYPASS bit in **RCC**/**RCC2**.

## 5.4 Register Map

Table 5-6 on page 183 lists the System Control registers, grouped by function. The offset listed is a hexadecimal increment to the register's address, relative to the System Control base address of 0x400F.E000.

**Note:** Spaces in the System Control register space that are not used are reserved for future or internal use. Software should not modify any reserved memory address.

**Note:** Additional Flash and ROM registers defined in the System Control register space are described in the “Internal Memory” on page 260.

**Table 5-6. System Control Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | DID0 | RO | - | Device Identification 0 | 186 |
| 0x004 | DID1 | RO | - | Device Identification 1 | 205 |
| 0x008 | DC0 | RO | 0x007F.003F | Device Capabilities 0 | 207 |

**Table 5-6. System Control Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x010 | DC1 | RO | 0x0101.32FF | Device Capabilities 1 | 208 |
| 0x014 | DC2 | RO | 0x0007.5013 | Device Capabilities 2 | 210 |
| 0x018 | DC3 | RO | 0x9F3F.0000 | Device Capabilities 3 | 211 |
| 0x01C | DC4 | RO | 0x0000.301F | Device Capabilities 4 | 212 |
| 0x020 | DC5 | RO | 0x0000.0000 | Device Capabilities 5 | 213 |
| 0x024 | DC6 | RO | 0x0000.0002 | Device Capabilities 6 | 214 |
| 0x028 | DC7 | RO | 0x40C0.0F3F | Device Capabilities 7 | 215 |
| 0x030 | PBORCTL | R/W | 0x0000.7FFD | Brown-Out Reset Control | 188 |
| 0x034 | LDOPCTL | R/W | 0x0000.0000 | LDO Power Control | 189 |
| 0x040 | SRCR0 | R/W | 0x00000000 | Software Reset Control 0 | 235 |
| 0x044 | SRCR1 | R/W | 0x00000000 | Software Reset Control 1 | 236 |
| 0x048 | SRCR2 | R/W | 0x00000000 | Software Reset Control 2 | 237 |
| 0x050 | RIS | RO | 0x0000.0000 | Raw Interrupt Status | 190 |
| 0x054 | IMC | R/W | 0x0000.0000 | Interrupt Mask Control | 191 |
| 0x058 | MISC | R/W1C | 0x0000.0000 | Masked Interrupt Status and Clear | 192 |
| 0x05C | RESC | R/W | - | Reset Cause | 193 |
| 0x060 | RCC | R/W | 0x0780.3AD1 | Run-Mode Clock Configuration | 194 |
| 0x064 | PLLCFG | RO | - | XTAL to PLL Translation | 198 |
| 0x06C | GPIOHBCTL | R/W | 0x0000.0000 | GPIO High-Performance Bus Control | 199 |
| 0x070 | RCC2 | R/W | 0x0780.6810 | Run-Mode Clock Configuration 2 | 201 |
| 0x07C | MOSCCTL | R/W | 0x0000.0000 | Main Oscillator Control | 203 |
| 0x100 | RCGC0 | R/W | 0x00000040 | Run Mode Clock Gating Control Register 0 | 217 |
| 0x104 | RCGC1 | R/W | 0x00000000 | Run Mode Clock Gating Control Register 1 | 223 |
| 0x108 | RCGC2 | R/W | 0x00000000 | Run Mode Clock Gating Control Register 2 | 229 |
| 0x110 | SCGC0 | R/W | 0x00000040 | Sleep Mode Clock Gating Control Register 0 | 219 |
| 0x114 | SCGC1 | R/W | 0x00000000 | Sleep Mode Clock Gating Control Register 1 | 225 |
| 0x118 | SCGC2 | R/W | 0x00000000 | Sleep Mode Clock Gating Control Register 2 | 231 |
| 0x120 | DCGC0 | R/W | 0x00000040 | Deep Sleep Mode Clock Gating Control Register 0 | 221 |
| 0x124 | DCGC1 | R/W | 0x00000000 | Deep Sleep Mode Clock Gating Control Register 1 | 227 |
| 0x128 | DCGC2 | R/W | 0x00000000 | Deep Sleep Mode Clock Gating Control Register 2 | 233 |
| 0x144 | DSLPCLKCFG | R/W | 0x0780.0000 | Deep Sleep Clock Configuration | 204 |

## 5.5 Register Descriptions

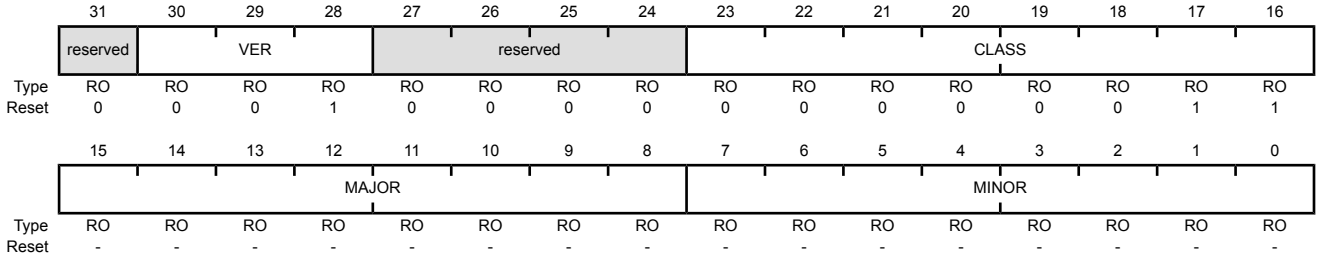
All addresses given are relative to the System Control base address of 0x400F.E000.

## Register 1: Device Identification 0 (DID0), offset 0x000

This register identifies the version of the microcontroller. Each microcontroller is uniquely identified by the combined values of the CLASS field in the **DID0** register and the PARTNO field in the **DID1** register.

Device Identification 0 (DID0)

Base 0x400F.E000
Offset 0x000
Type RO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 30:28 | VER | RO | 0x1 | DID0 Version<br>This field defines the **DID0** register format version. The version number is numeric. The value of the VER field is encoded as follows:<br>Value Description<br>0x1 Second version of the **DID0** register format. |
| 27:24 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:16 | CLASS | RO | 0x3 | Device Class<br>The CLASS field value identifies the internal design from which all mask sets are generated for all devices in a particular product line. The CLASS field value is changed for new product lines, for changes in fab process (for example, a remap or shrink), or any case where the MAJOR or MINOR fields require differentiation from prior devices. The value of the CLASS field is encoded as follows (all other encodings are reserved):<br>Value Description<br>0x3 Stellaris® DustDevil-class devices |

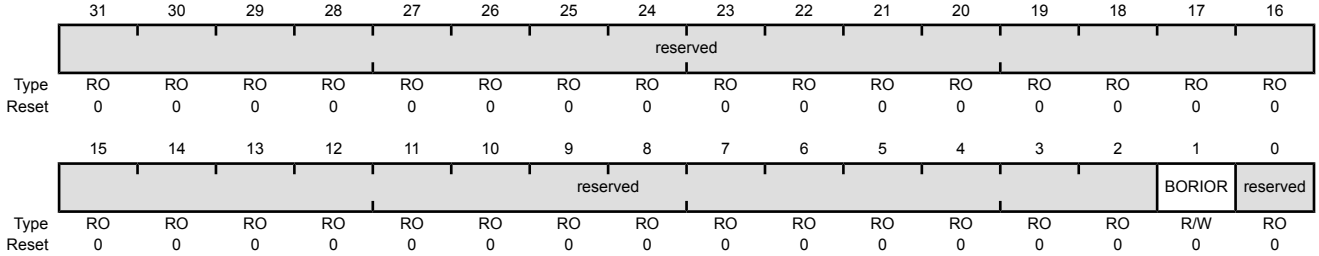
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:8 | MAJOR | RO | - | Major Revision<br>This field specifies the major revision number of the device. The major revision reflects changes to base layers of the design. The major revision number is indicated in the part number as a letter (A for first revision, B for second, and so on). This field is encoded as follows:<br><br>Value Description<br>0x0 Revision A (initial device)<br>0x1 Revision B (first base layer revision)<br>0x2 Revision C (second base layer revision)<br><br>and so on. |
| 7:0 | MINOR | RO | - | Minor Revision<br>This field specifies the minor revision number of the device. The minor revision reflects changes to the metal layers of the design. The MINOR field value is reset when the MAJOR field is changed. This field is numeric and is encoded as follows:<br><br>Value Description<br>0x0 Initial device, or a major revision update.<br>0x1 First metal layer change.<br>0x2 Second metal layer change.<br><br>and so on. |

## Register 2: Brown-Out Reset Control (PBORCTL), offset 0x030

This register is responsible for controlling reset conditions after initial power-on reset.

Brown-Out Reset Control (PBORCTL)

Base 0x400F.E000
Offset 0x030
Type R/W, reset 0x0000.7FFD



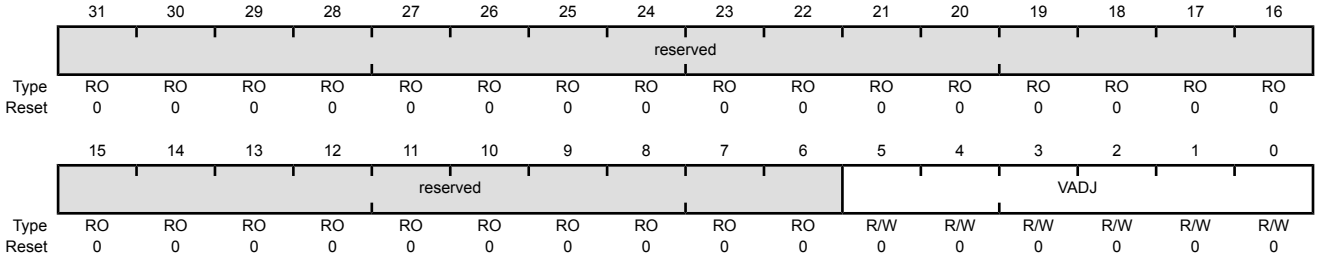

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | BORIOR | R/W | 0 | BOR Interrupt or Reset<br>This bit controls how a BOR event is signaled to the controller. If set, a reset is signaled. Otherwise, an interrupt is signaled. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 3: LDO Power Control (LDOPCTL), offset 0x034

The VADJ field in this register adjusts the on-chip output voltage ($V_{OUT}$).

LDO Power Control (LDOPCTL)

Base 0x400F.E000
Offset 0x034
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:0 | VADJ | R/W | 0x0 | LDO Output Voltage<br>This field sets the on-chip output voltage. The programming values for the VADJ field are provided below. |

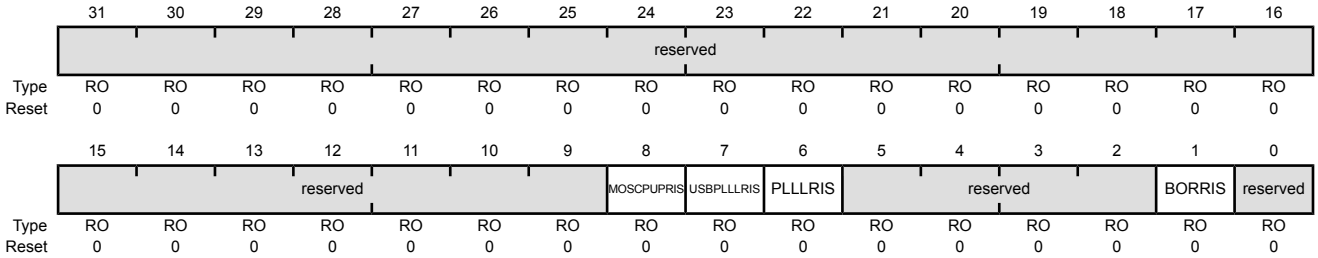
| Value | $V_{OUT}$ (V) |
|---|---|
| 0x00 | 2.50 |
| 0x01 | 2.45 |
| 0x02 | 2.40 |
| 0x03 | 2.35 |
| 0x04 | 2.30 |
| 0x05 | 2.25 |
| 0x06-0x3F | Reserved |
| 0x1B | 2.75 |
| 0x1C | 2.70 |
| 0x1D | 2.65 |
| 0x1E | 2.60 |
| 0x1F | 2.55 |

## Register 4: Raw Interrupt Status (RIS), offset 0x050

Central location for system control raw interrupts. These are set and cleared by hardware.

Raw Interrupt Status (RIS)

Base 0x400F.E000
Offset 0x050
Type RO, reset 0x0000.0000



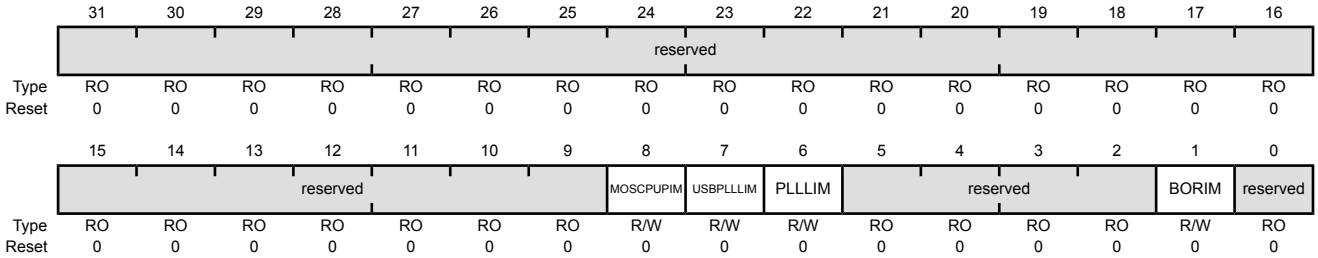

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:9 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 8 | MOSCPUPRIS | RO | 0 | MOSC Power Up Raw Interrupt Status<br>This bit is set when the PLL $T_{MOSCPUP}$ Timer asserts. |
| 7 | USBPLLLRIS | RO | 0 | USB PLL Lock Raw Interrupt Status<br>This bit is set when the USB PLL $T_{USBREADY}$ Timer asserts. |
| 6 | PLLLRIS | RO | 0 | PLL Lock Raw Interrupt Status<br>This bit is set when the PLL $T_{READY}$ Timer asserts. |
| 5:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | BORRIS | RO | 0 | Brown-Out Reset Raw Interrupt Status<br>This bit is the raw interrupt status for any brown-out conditions. If set, a brown-out condition is currently active. This is an unregistered signal from the brown-out detection circuit. An interrupt is reported if the `BORIM` bit in the **IMC** register is set and the `BORIOR` bit in the **PBORCTL** register is cleared. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 5: Interrupt Mask Control (IMC), offset 0x054

Central location for system control interrupt masks.

Interrupt Mask Control (IMC)

Base 0x400F.E000
Offset 0x054
Type R/W, reset 0x0000.0000



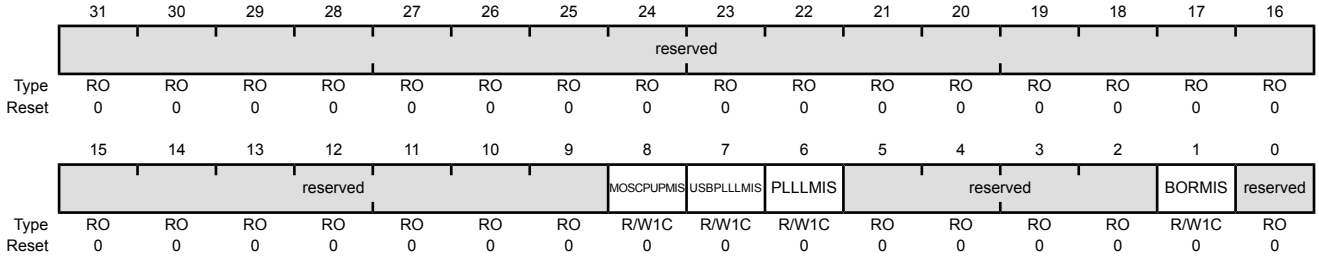

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:9 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 8 | MOSCPUPIM | R/W | 0 | MOSC Power Up Interrupt Mask<br>This bit specifies whether a MOSC power up intterupt is promoted to a controller interrupt. If set, an interrupt is generated if `MOSCPUPRIS` in **RIS** is set; otherwise, an interrupt is not generated. |
| 7 | USBPLLLIM | R/W | 0 | USB PLL Lock Interrupt Mask<br>This bit specifies whether a USB PLL Lock interrupt is promoted to a controller interrupt. If set, an interrupt is generated if `USBPLLLRIS` in **RIS** is set; otherwise, an interrupt is not generated. |
| 6 | PLLLIM | R/W | 0 | PLL Lock Interrupt Mask<br>This bit specifies whether a PLL Lock interrupt is promoted to a controller interrupt. If set, an interrupt is generated if `PLLLRIS` in **RIS** is set; otherwise, an interrupt is not generated. |
| 5:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | BORIM | R/W | 0 | Brown-Out Reset Interrupt Mask<br>This bit specifies whether a brown-out condition is promoted to a controller interrupt. If set, an interrupt is generated if `BORRIS` is set; otherwise, an interrupt is not generated. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 6: Masked Interrupt Status and Clear (MISC), offset 0x058

On a read, this register gives the current masked status value of the corresponding interrupt. All of the bits are R/W1C and this action also clears the corresponding raw interrupt bit in the **RIS** register (see page 190).

Masked Interrupt Status and Clear (MISC)

Base 0x400F.E000
Offset 0x058
Type R/W1C, reset 0x0000.0000



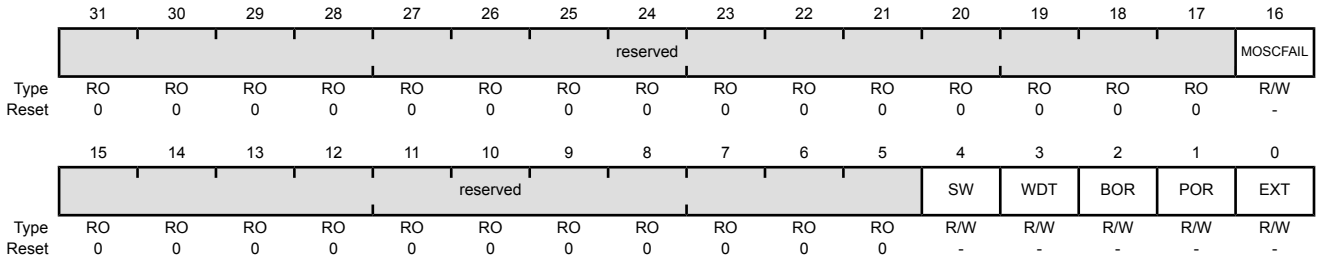

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:9 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 8 | MOSCPUPMIS | R/W1C | 0 | MOSC Power Up Masked Interrupt Status<br>This bit is set when the $T_{MOSCPUP}$ timer asserts. The interrupt is cleared by writing a 1 to this bit. |
| 7 | USBPLLLMIS | R/W1C | 0 | USB PLL Lock Masked Interrupt Status<br>This bit is set when the USB PLL $T_{USBREADY}$ timer asserts. The interrupt is cleared by writing a 1 to this bit. |
| 6 | PLLLMIS | R/W1C | 0 | PLL Lock Masked Interrupt Status<br>This bit is set when the PLL $T_{READY}$ timer asserts. The interrupt is cleared by writing a 1 to this bit. |
| 5:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | BORMIS | R/W1C | 0 | BOR Masked Interrupt Status<br>The `BORMIS` is simply the `BORRIS` ANDed with the mask value, `BORIM`. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 7: Reset Cause (RESC), offset 0x05C

This register is set with the reset cause after reset. The bits in this register are sticky and maintain their state across multiple reset sequences, except when a power- on reset or an external reset is the cause, in which case, all bits other than POR or EXT in the **RESC** register are cleared.

Reset Cause (RESC)

Base 0x400F.E000
Offset 0x05C
Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | MOSCFAIL | R/W | - | MOSC Failure Reset<br>When set, indicates the MOSC circuit was enable for clock validation and failed. This generated a reset event. |
| 15:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SW | R/W | - | Software Reset<br>When set, indicates a software reset is the cause of the reset event. |
| 3 | WDT | R/W | - | Watchdog Timer Reset<br>When set, indicates a watchdog reset is the cause of the reset event. |
| 2 | BOR | R/W | - | Brown-Out Reset<br>When set, indicates a brown-out reset is the cause of the reset event. |
| 1 | POR | R/W | - | Power-On Reset<br>When set, indicates a power-on reset is the cause of the reset event. |
| 0 | EXT | R/W | - | External Reset<br>When set, indicates an external reset ($\overline{\text{RST}}$ assertion) is the cause of the reset event. |

## Register 8: Run-Mode Clock Configuration (RCC), offset 0x060

This register is defined to provide source control and frequency speed.

Run-Mode Clock Configuration (RCC)

Base 0x400F.E000
Offset 0x060
Type R/W, reset 0x0780.3AD1

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | ACG | SYSDIV | | | | USESYSDIV | reserved | | | | | |
| Type | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | PWRDN | reserved | BYPASS | XTAL | | | | | OSCSRC | | reserved | | IOSCDIS | MOSCDIS |
| Type | RO | RO | R/W | RO | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | RO | RO | R/W | R/W |
| Reset | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:28 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 27 | ACG | R/W | 0 | Auto Clock Gating<br>This bit specifies whether the system uses the **Sleep-Mode Clock Gating Control (SCGCn)** registers and **Deep-Sleep-Mode Clock Gating Control (DCGCn)** registers if the controller enters a Sleep or Deep-Sleep mode (respectively). If set, the **SCGCn** or **DCGCn** registers are used to control the clocks distributed to the peripherals when the controller is in a sleep mode. Otherwise, the **Run-Mode Clock Gating Control (RCGCn)** registers are used when the controller enters a sleep mode.<br>The **RCGCn** registers are always used to control the clocks in Run mode.<br>This allows peripherals to consume less power when the controller is in a sleep mode and the peripheral is unused. |
| 26:23 | SYSDIV | R/W | 0xF | System Clock Divisor<br>Specifies which divisor is used to generate the system clock from either the PLL output or the oscillator source (depending on how the BYPASS bit in this register is configured). See Table 5-4 on page 179 for bit encodings.<br>If the SYSDIV value is less than MINSYSDIV (see page 208), and the PLL is being used, then the MINSYSDIV value is used as the divisor.<br>If the PLL is not being used, the SYSDIV value can be less than MINSYSDIV. |
| 22 | USESYSDIV | R/W | 0 | Enable System Clock Divider<br>Use the system clock divider as the source for the system clock. The system clock divider is forced to be used when the PLL is selected as the source.<br>If the USERCC2 bit in the **RCC2** register is set, then the SYSDIV2 field in the **RCC2** register is used as the system clock divider rather than the SYSDIV field in this register. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 21:14 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 13 | PWRDN | R/W | 1 | PLL Power Down<br>This bit connects to the PLL PWRDN input. The reset value of 1 powers down the PLL. |
| 12 | reserved | RO | 1 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 11 | BYPASS | R/W | 1 | PLL Bypass<br>Chooses whether the system clock is derived from the PLL output or the OSC source. If set, the clock that drives the system is the OSC source. Otherwise, the clock that drives the system is the PLL output clock divided by the system divider.<br>See Table 5-4 on page 179 for programming guidelines.<br>**Note:** The ADC must be clocked from the PLL or directly from a 14-MHz to 18-MHz clock source to operate properly. While the ADC works in a 14-18 MHz range, to maintain a 1 M sample/second rate, the ADC must be provided a 16-MHz clock source. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 10:6 | XTAL | R/W | 0xB | Crystal Value<br>This field specifies the crystal value attached to the main oscillator. The encoding for this field is provided below. Depending on the crystal used, the PLL frequency may not be exactly 400 MHz (see Table 21-10 on page 770 for more information).<br>Frequencies that may be used with the USB interface are indicated in the table. To function within the clocking requirements of the USB specification, a crystal of 4, 5, 6, 8, 10, 12, or 16 MHz must be used. |

| Value | Crystal Frequency (MHz) Not Using the PLL | Crystal Frequency (MHz) Using the PLL |
|---|---|---|
| 0x00 | 1.000 | reserved |
| 0x01 | 1.8432 | reserved |
| 0x02 | 2.000 | reserved |
| 0x03 | 2.4576 | reserved |
| 0x04 | 3.579545 MHz | |
| 0x05 | 3.6864 MHz | |
| 0x06 | 4 MHz (USB) | |
| 0x07 | 4.096 MHz | |
| 0x08 | 4.9152 MHz | |
| 0x09 | 5 MHz (USB) | |
| 0x0A | 5.12 MHz | |
| 0x0B | 6 MHz (reset value)(USB) | |
| 0x0C | 6.144 MHz | |
| 0x0D | 7.3728 MHz | |
| 0x0E | 8 MHz (USB) | |
| 0x0F | 8.192 MHz | |
| 0x10 | 10.0 MHz (USB) | |
| 0x11 | 12.0 MHz (USB) | |
| 0x12 | 12.288 MHz | |
| 0x13 | 13.56 MHz | |
| 0x14 | 14.31818 MHz | |
| 0x15 | 16.0 MHz (USB) | |
| 0x16 | 16.384 MHz | |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 5:4 | OSCSRC | R/W | 0x1 | Oscillator Source<br>Selects the input source for the OSC. The values are:<br><br>Value Input Source<br>0x0 MOSC<br>Main oscillator<br>0x1 IOSC<br>Internal oscillator (default)<br>0x2 IOSC/4<br>Internal oscillator / 4<br>0x3 30 kHz<br>30-KHz internal oscillator<br><br>For additional oscillator sources, see the **RCC2** register. |
| 3:2 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | IOSCDIS | R/W | 0 | Internal Oscillator Disable<br>0: Internal oscillator (IOSC) is enabled.<br>1: Internal oscillator is disabled. |
| 0 | MOSCDIS | R/W | 1 | Main Oscillator Disable<br>0: Main oscillator is enabled .<br>1: Main oscillator is disabled (default). |

## Register 9: XTAL to PLL Translation (PLLCFG), offset 0x064

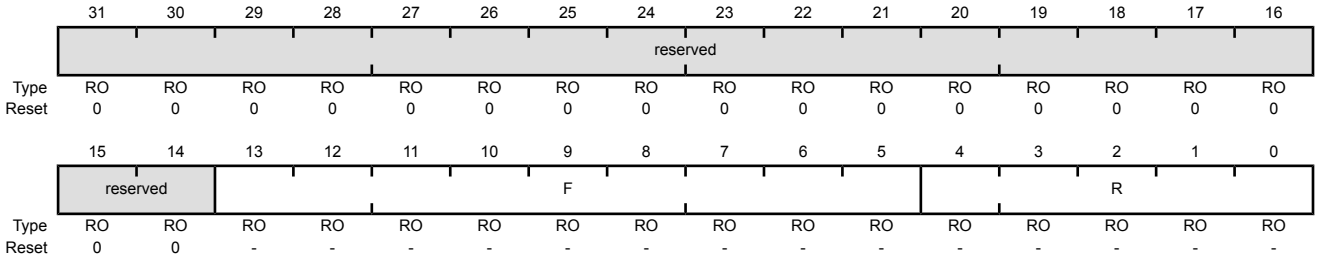
This register provides a means of translating external crystal frequencies into the appropriate PLL settings. This register is initialized during the reset sequence and updated anytime that the XTAL field changes in the **Run-Mode Clock Configuration (RCC)** register (see page 194).

The PLL frequency is calculated using the **PLLCFG** field values, as follows:

```
PLLFreq = OSCFreq * F / (R + 1)
```

XTAL to PLL Translation (PLLCFG)

Base 0x400F.E000
Offset 0x064
Type RO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:14 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 13:5 | F | RO | - | PLL F Value<br>This field specifies the value supplied to the PLL's F input. |
| 4:0 | R | RO | - | PLL R Value<br>This field specifies the value supplied to the PLL's R input. |

## Register 10: GPIO High-Performance Bus Control (GPIOHBCTL), offset 0x06C

This register controls which internal bus is used to access each GPIO port. When a bit is clear, the corresponding GPIO port is accessed across the legacy Advanced Peripheral Bus (APB) bus and through the APB memory aperture. When a bit is set, the corresponding port is accessed across the Advanced High-Performance Bus (AHB) bus and through the AHB memory aperture. Each GPIO port can be individually configured to use AHB or APB, but may be accessed only through one aperture. The AHB bus provides better back-to-back access performance than the APB bus. The address aperture in the memory map changes for the ports that are enabled for AHB access (see Table 9-6 on page 360).

GPIO High-Performance Bus Control (GPIOHBCTL)

Base 0x400F.E000
Offset 0x06C
Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | PORTE | PORTD | PORTC | PORTB | PORTA |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0x0000.0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | PORTE | R/W | 0 | Port E Advanced High-Performance Bus<br>This bit defines the memory aperture for Port E.<br><br>Value Description<br>1 Advanced High-Performance Bus (AHB)<br>0 Advanced Peripheral Bus (APB). This bus is the legacy bus. |
| 3 | PORTD | R/W | 0 | Port D Advanced High-Performance Bus<br>This bit defines the memory aperture for Port D.<br><br>Value Description<br>1 Advanced High-Performance Bus (AHB)<br>0 Advanced Peripheral Bus (APB). This bus is the legacy bus. |
| 2 | PORTC | R/W | 0 | Port C Advanced High-Performance Bus<br>This bit defines the memory aperture for Port C.<br><br>Value Description<br>1 Advanced High-Performance Bus (AHB)<br>0 Advanced Peripheral Bus (APB). This bus is the legacy bus. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | PORTB | R/W | 0 | Port B Advanced High-Performance Bus<br>This bit defines the memory aperture for Port B.<br><br>Value Description<br>1 Advanced High-Performance Bus (AHB)<br>0 Advanced Peripheral Bus (APB). This bus is the legacy bus. |
| 0 | PORTA | R/W | 0 | Port A Advanced High-Performance Bus<br>This bit defines the memory aperture for Port A.<br><br>Value Description<br>1 Advanced High-Performance Bus (AHB)<br>0 Advanced Peripheral Bus (APB). This bus is the legacy bus. |

## Register 11: Run-Mode Clock Configuration 2 (RCC2), offset 0x070

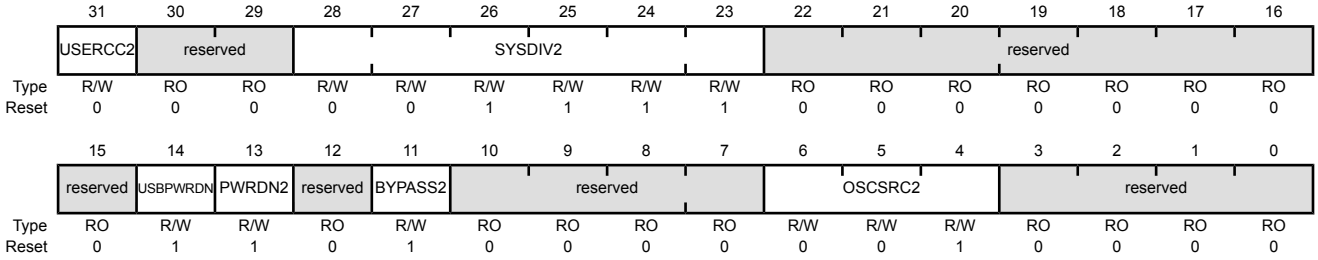
This register overrides the **RCC** equivalent register fields, as shown in Table 5-7, when the USERCC2 bit is set, allowing the extended capabilities of the **RCC2** register to be used while also providing a means to be backward-compatible to previous parts. Each **RCC2** field that supersedes an **RCC** field is located at the same LSB bit position; however, some **RCC2** fields are larger than the corresponding **RCC** field.

**Table 5-7. RCC2 Fields that Override RCC fields**

| RCC2 Field... | Overrides RCC Field |
|---|---|
| SYSDIV2, bits[28:23] | SYSDIV, bits[26:23] |
| PWRDN2, bit[13] | PWRDN, bit[13] |
| BYPASS2, bit[11] | BYPASS, bit[11] |
| OSCSRC2, bits[6:4] | OSCSRC, bits[5:4] |

Run-Mode Clock Configuration 2 (RCC2)

Base 0x400F.E000
Offset 0x070
Type R/W, reset 0x0780.6810




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | USERCC2 | R/W | 0 | Use RCC2<br>When set, overrides the **RCC** register fields. |
| 30:29 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 28:23 | SYSDIV2 | R/W | 0x0F | System Clock Divisor<br>Specifies which divisor is used to generate the system clock from either the PLL output or the oscillator source (depending on how the BYPASS2 bit is configured). SYSDIV2 is used for the divisor when both the USESYSDIV bit in the **RCC** register and the USERCC2 bit in this register are set. See Table 5-5 on page 179 for programming guidelines. |
| 22:15 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14 | USBPWRDN | R/W | 1 | Power-Down USB PLL<br>When set, powers down the USB PLL. |
| 13 | PWRDN2 | R/W | 1 | Power-Down PLL<br>When set, powers down the PLL. |

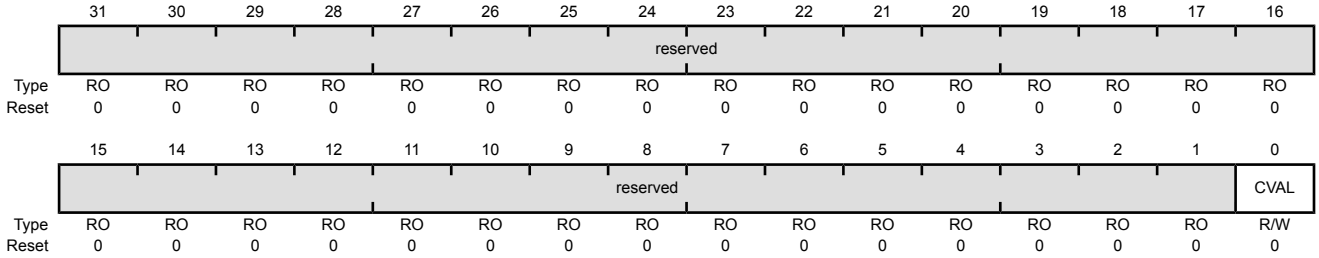
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 12 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 11 | BYPASS2 | R/W | 1 | Bypass PLL<br>When set, bypasses the PLL for the clock source.<br>See Table 5-5 on page 179 for programming guidelines. |
| 10:7 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:4 | OSCSRC2 | R/W | 0x1 | Oscillator Source<br>Selects the input source for the OSC. The values are:<br>Value Description<br>0x0 MOSC<br>Main oscillator<br>0x1 IOSC<br>Internal oscillator<br>0x2 IOSC/4<br>Internal oscillator / 4<br>0x3 30 kHz<br>30-kHz internal oscillator<br>0x4 Reserved<br>0x5 Reserved<br>0x6 Reserved<br>0x7 32 kHz<br>32.768-kHz external oscillator |
| 3:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 12: Main Oscillator Control (MOSCCTL), offset 0x07C

This register provides control over the features of the main oscillator, including the ability to enable the MOSC clock validation circuit. When enabled, this circuit monitors the energy on the MOSC pins to provide a Clock Valid signal. If the clock goes invalid after being enabled, the part does a hardware reset and reboots to the NMI handler.

Main Oscillator Control (MOSCCTL)

Base 0x400F.E000
Offset 0x07C
Type R/W, reset 0x0000.0000



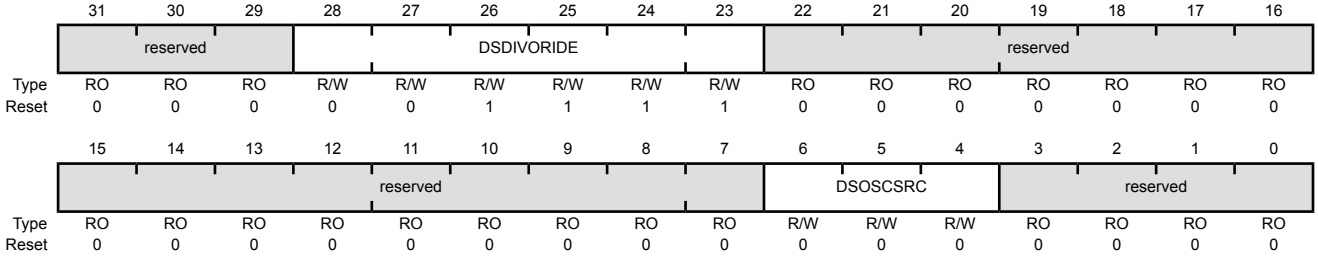

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | CVAL | R/W | 0 | Clock Validation for MOSC<br>When set, the monitor circuit is enabled. |

## Register 13: Deep Sleep Clock Configuration (DSLPCLKCFG), offset 0x144

This register provides configuration information for the hardware control of Deep Sleep Mode.

Deep Sleep Clock Configuration (DSLPCLKCFG)

Base 0x400F.E000
Offset 0x144
Type R/W, reset 0x0780.0000



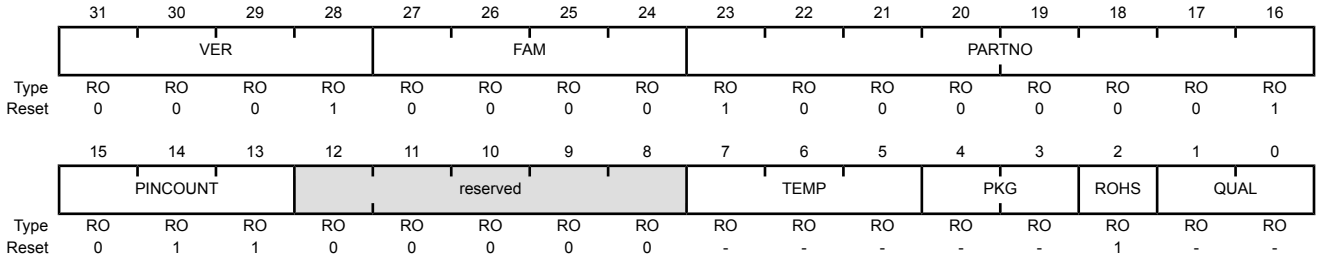

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:29 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 28:23 | DSDIVORIDE | R/W | 0x0F | Divider Field Override<br>6-bit system divider field to override when Deep-Sleep occurs with PLL running. |
| 22:7 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:4 | DSOSCSRC | R/W | 0x0 | Clock Source<br>Specifies the clock source during Deep-Sleep mode.<br>Value Description<br>0x0 MOSC<br>Use main oscillator as source.<br>0x1 IOSC<br>Use internal 12-MHz oscillator as source.<br>0x2 Reserved<br>0x3 30 kHz<br>Use 30-kHz internal oscillator as source.<br>0x4 Reserved<br>0x5 Reserved<br>0x6 Reserved<br>0x7 32 kHz<br>Use 32.768-kHz external oscillator as source. |
| 3:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 14: Device Identification 1 (DID1), offset 0x004

This register identifies the device family, part number, temperature range, pin count, and package type. Each microcontroller is uniquely identified by the combined values of the CLASS field in the **DID0** register and the PARTNO field in the **DID1** register.

Device Identification 1 (DID1)

Base 0x400F.E000
Offset 0x004
Type RO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:28 | VER | RO | 0x1 | DID1 Version<br>This field defines the **DID1** register format version. The version number is numeric. The value of the VER field is encoded as follows (all other encodings are reserved):<br><br>Value Description<br>0x1 Second version of the **DID1** register format. |
| 27:24 | FAM | RO | 0x0 | Family<br>This field provides the family identification of the device within the Luminary Micro product portfolio. The value is encoded as follows (all other encodings are reserved):<br><br>Value Description<br>0x0 Stellaris family of microcontollers, that is, all devices with external part numbers starting with LM3S. |
| 23:16 | PARTNO | RO | 0x81 | Part Number<br>This field provides the part number of the device within the family. The value is encoded as follows (all other encodings are reserved):<br><br>Value Description<br>0x81 LM3S5632 |
| 15:13 | PINCOUNT | RO | 0x3 | Package Pin Count<br>This field specifies the number of pins on the device package. The value is encoded as follows (all other encodings are reserved):<br><br>Value Description<br>0x3 64-pin package |

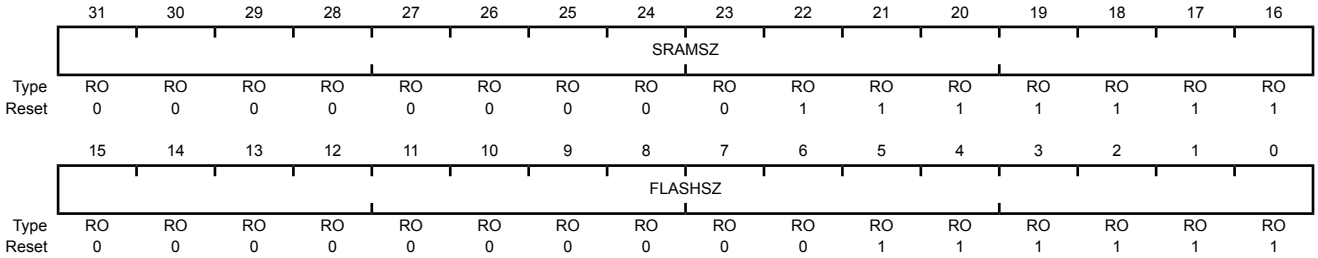
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 12:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:5 | TEMP | RO | - | Temperature Range<br>This field specifies the temperature rating of the device. The value is encoded as follows (all other encodings are reserved):<br>Value Description<br>0x0 Commercial temperature range (0°C to 70°C)<br>0x1 Industrial temperature range (-40°C to 85°C)<br>0x2 Extended temperature range (-40°C to 105°C) |
| 4:3 | PKG | RO | - | Package Type<br>This field specifies the package type. The value is encoded as follows (all other encodings are reserved):<br>Value Description<br>0x0 SOIC package<br>0x1 LQFP package<br>0x2 BGA package |
| 2 | ROHS | RO | 1 | RoHS-Compliance<br>This bit specifies whether the device is RoHS-compliant. A 1 indicates the part is RoHS-compliant. |
| 1:0 | QUAL | RO | - | Qualification Status<br>This field specifies the qualification status of the device. The value is encoded as follows (all other encodings are reserved):<br>Value Description<br>0x0 Engineering Sample (unqualified)<br>0x1 Pilot Production (unqualified)<br>0x2 Fully Qualified |

## Register 15: Device Capabilities 0 (DC0), offset 0x008

This register is predefined by the part and can be used to verify features.

Device Capabilities 0 (DC0)

Base 0x400F.E000
Offset 0x008
Type RO, reset 0x007F.003F



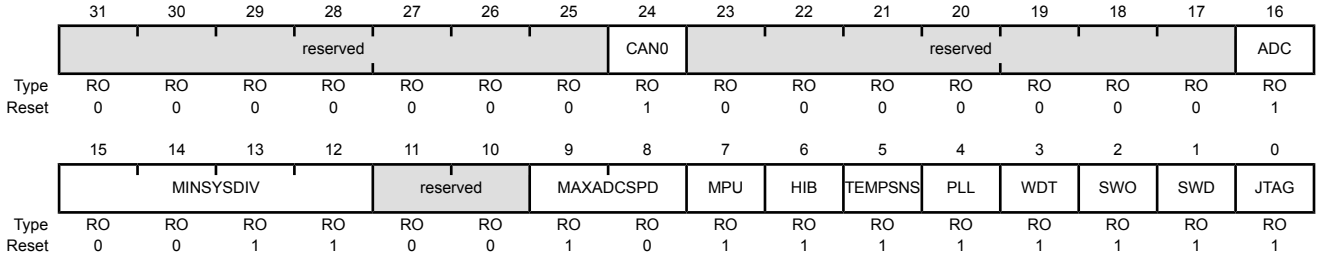

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | SRAMSZ | RO | 0x007F | SRAM Size<br>Indicates the size of the on-chip SRAM memory.<br><br>Value Description<br>0x007F 32 KB of SRAM |
| 15:0 | FLASHSZ | RO | 0x003F | Flash Size<br>Indicates the size of the on-chip flash memory.<br><br>Value Description<br>0x003F 128 KB of Flash |

## Register 16: Device Capabilities 1 (DC1), offset 0x010

This register is predefined by the part and can be used to verify features. The PWM, SARADC0, MAXADCSPD, WDT, SWO, SWD, and JTAG bits mask the **RCGC0**, **SCGC0**, and **DCGC0** registers. Other bits are passed as 0. MAXADCSPD is clipped to the maximum value specified in **DC1**.

Device Capabilities 1 (DC1)

Base 0x400F.E000
Offset 0x010
Type RO, reset 0x0101.32FF



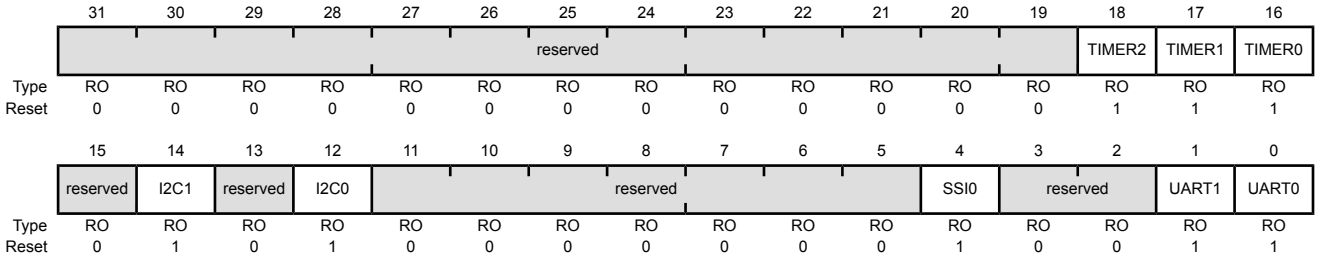

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:25 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 24 | CAN0 | RO | 1 | CAN Module 0 Present<br>When set, indicates that CAN unit 0 is present. |
| 23:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | ADC | RO | 1 | ADC Module Present. When set, indicates that the ADC module is present. |
| 15:12 | MINSYSDIV | RO | 0x3 | System Clock Divider. Minimum 4-bit divider value for system clock. The reset value is hardware-dependent. See the **RCC** register for how to change the system clock divisor using the SYSDIV bit.<br>Value Description<br>0x3 Specifies a 50-MHz CPU clock with a PLL divider of 4. |
| 11:10 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9:8 | MAXADCSPD | RO | 0x2 | Max ADC Speed. This field indicates the maximum rate at which the ADC samples data.<br>Value Description<br>0x2 500K samples/second |
| 7 | MPU | RO | 1 | MPU Present. When set, indicates that the Cortex-M3 Memory Protection Unit (MPU) module is present. See the "Cortex-M3 Peripherals" chapter in the Stellaris Data Sheet for details on the MPU. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | HIB | RO | 1 | Hibernation Module Present. When set, indicates that the Hibernation module is present. |
| 5 | TEMPSNS | RO | 1 | Temp Sensor Present. When set, indicates that the on-chip temperature sensor is present. |
| 4 | PLL | RO | 1 | PLL Present. When set, indicates that the on-chip Phase Locked Loop (PLL) is present. |
| 3 | WDT | RO | 1 | Watchdog Timer Present. When set, indicates that a watchdog timer is present. |
| 2 | SWO | RO | 1 | SWO Trace Port Present. When set, indicates that the Serial Wire Output (SWO) trace port is present. |
| 1 | SWD | RO | 1 | SWD Present. When set, indicates that the Serial Wire Debugger (SWD) is present. |
| 0 | JTAG | RO | 1 | JTAG Present. When set, indicates that the JTAG debugger interface is present. |

## Register 17: Device Capabilities 2 (DC2), offset 0x014

This register is predefined by the part and can be used to verify features.

Device Capabilities 2 (DC2)
Base 0x400F.E000
Offset 0x014
Type RO, reset 0x0007.5013



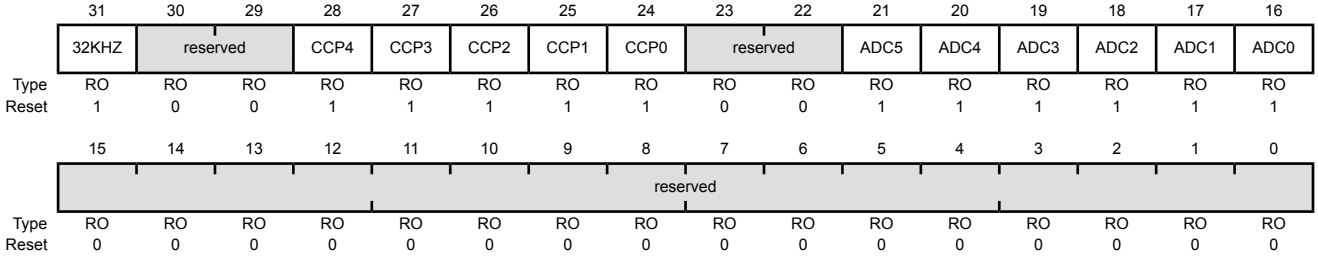

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | TIMER2 | RO | 1 | Timer 2 Present. When set, indicates that General-Purpose Timer module 2 is present. |
| 17 | TIMER1 | RO | 1 | Timer 1 Present. When set, indicates that General-Purpose Timer module 1 is present. |
| 16 | TIMER0 | RO | 1 | Timer 0 Present. When set, indicates that General-Purpose Timer module 0 is present. |
| 15 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14 | I2C1 | RO | 1 | I2C Module 1 Present. When set, indicates that I2C module 1 is present. |
| 13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12 | I2C0 | RO | 1 | I2C Module 0 Present. When set, indicates that I2C module 0 is present. |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SSI0 | RO | 1 | SSI0 Present. When set, indicates that SSI module 0 is present. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | UART1 | RO | 1 | UART1 Present. When set, indicates that UART module 1 is present. |
| 0 | UART0 | RO | 1 | UART0 Present. When set, indicates that UART module 0 is present. |

## Register 18: Device Capabilities 3 (DC3), offset 0x018

This register is predefined by the part and can be used to verify features.

Device Capabilities 3 (DC3)

Base 0x400F.E000
Offset 0x018
Type RO, reset 0x9F3F.0000



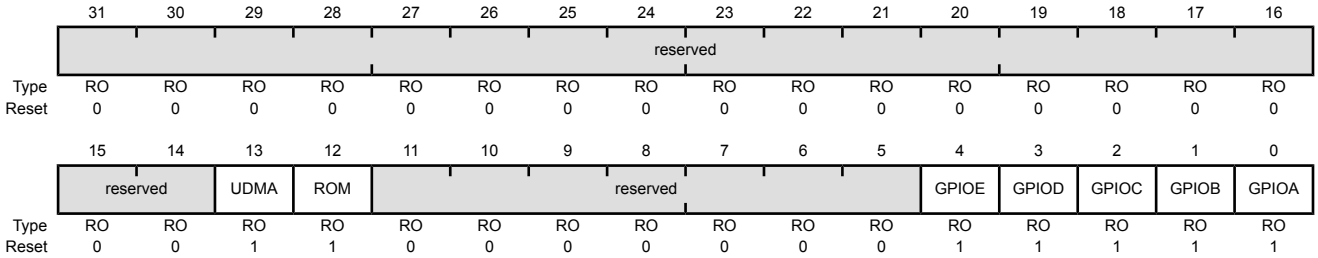

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | 32KHZ | RO | 1 | 32KHz Input Clock Available. When set, indicates an even CCP pin is present and can be used as a 32-KHz input clock. |
| 30:29 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 28 | CCP4 | RO | 1 | CCP4 Pin Present. When set, indicates that Capture/Compare/PWM pin 4 is present. |
| 27 | CCP3 | RO | 1 | CCP3 Pin Present. When set, indicates that Capture/Compare/PWM pin 3 is present. |
| 26 | CCP2 | RO | 1 | CCP2 Pin Present. When set, indicates that Capture/Compare/PWM pin 2 is present. |
| 25 | CCP1 | RO | 1 | CCP1 Pin Present. When set, indicates that Capture/Compare/PWM pin 1 is present. |
| 24 | CCP0 | RO | 1 | CCP0 Pin Present. When set, indicates that Capture/Compare/PWM pin 0 is present. |
| 23:22 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 21 | ADC5 | RO | 1 | ADC5 Pin Present. When set, indicates that ADC pin 5 is present. |
| 20 | ADC4 | RO | 1 | ADC4 Pin Present. When set, indicates that ADC pin 4 is present. |
| 19 | ADC3 | RO | 1 | ADC3 Pin Present. When set, indicates that ADC pin 3 is present. |
| 18 | ADC2 | RO | 1 | ADC2 Pin Present. When set, indicates that ADC pin 2 is present. |
| 17 | ADC1 | RO | 1 | ADC1 Pin Present. When set, indicates that ADC pin 1 is present. |
| 16 | ADC0 | RO | 1 | ADC0 Pin Present. When set, indicates that ADC pin 0 is present. |
| 15:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 19: Device Capabilities 4 (DC4), offset 0x01C

This register is predefined by the part and can be used to verify features.

Device Capabilities 4 (DC4)

Base 0x400F.E000
Offset 0x01C
Type RO, reset 0x0000.301F



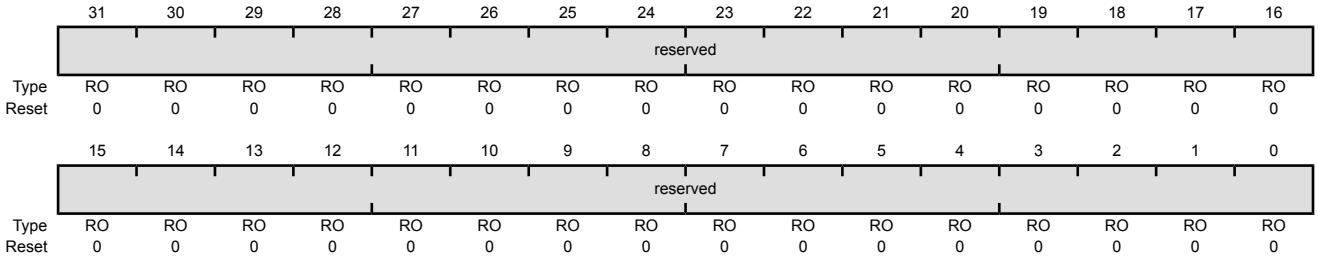

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:14 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 13 | UDMA | RO | 1 | Micro-DMA is present |
| 12 | ROM | RO | 1 | Internal Code ROM is present |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | GPIOE | RO | 1 | GPIO Port E Present. When set, indicates that GPIO Port E is present. |
| 3 | GPIOD | RO | 1 | GPIO Port D Present. When set, indicates that GPIO Port D is present. |
| 2 | GPIOC | RO | 1 | GPIO Port C Present. When set, indicates that GPIO Port C is present. |
| 1 | GPIOB | RO | 1 | GPIO Port B Present. When set, indicates that GPIO Port B is present. |
| 0 | GPIOA | RO | 1 | GPIO Port A Present. When set, indicates that GPIO Port A is present. |

## Register 20: Device Capabilities 5 (DC5), offset 0x020

This register is predefined by the part and can be used to verify features.

Device Capabilities 5 (DC5)

Base 0x400F.E000
Offset 0x020
Type RO, reset 0x0000.0000



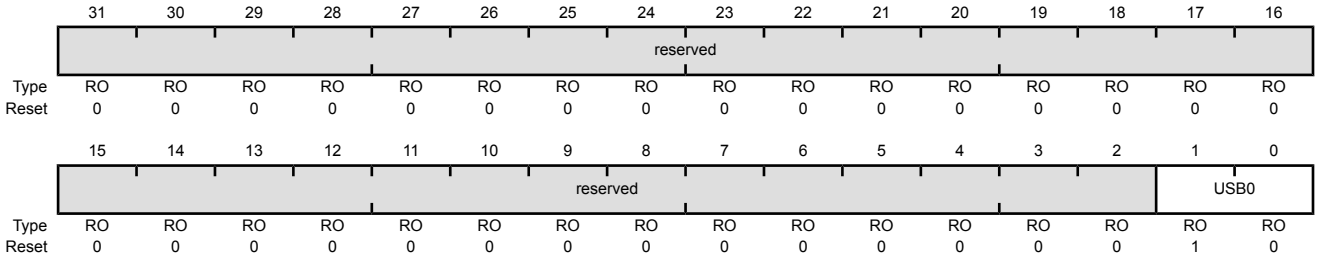

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 21: Device Capabilities 6 (DC6), offset 0x024

This register is predefined by the part and can be used to verify features.

Device Capabilities 6 (DC6)

Base 0x400F.E000
Offset 0x024
Type RO, reset 0x0000.0002



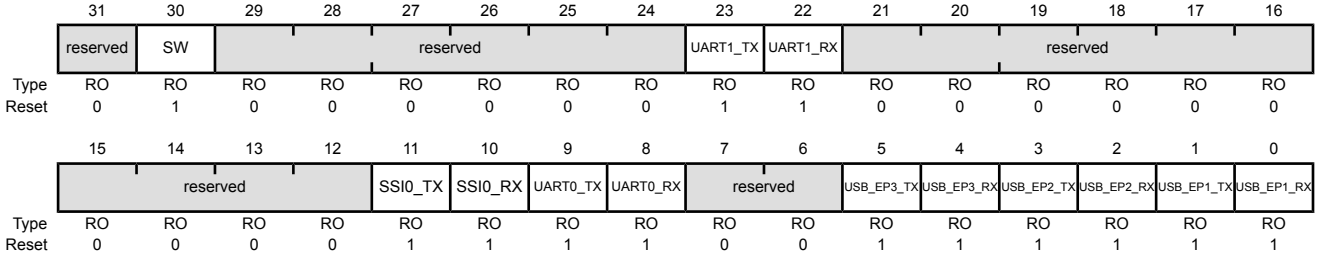

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1:0 | USB0 | RO | 0x2 | This specifies that USB0 is present and its capability<br><br>Value Description<br>0x2 USB is Device or Host. |

## Register 22: Device Capabilities 7 (DC7), offset 0x028

This register is predefined by the part and can be used to verify uDMA channel features.

Device Capabilities 7 (DC7)

Base 0x400F.E000
Offset 0x028
Type RO, reset 0x40C0.0F3F




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 30 | SW | RO | 1 | Software transfer on uDMA Ch30. When set, indicates uDMA channel 30 is available for software. |
| 29:24 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23 | UART1_TX | RO | 1 | UART1 TX on uDMA Ch23. When set, indicates uDMA channel 23 is available and connected to the transmit path of UART module 1. |
| 22 | UART1_RX | RO | 1 | UART1 RX on uDMA Ch22. When set, indicates uDMA channel 22 is available and connected to the receive path of UART module 1. |
| 21:12 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 11 | SSI0_TX | RO | 1 | SSI0 TX on uDMA Ch11. When set, indicates uDMA channel 11 is available and connected to the transmit path of SSI module 0. |
| 10 | SSI0_RX | RO | 1 | SSI0 RX on uDMA Ch10. When set, indicates uDMA channel 10 is available and connected to the receive path of SSI module 0. |
| 9 | UART0_TX | RO | 1 | UART0 TX on uDMA Ch9. When set, indicates uDMA channel 9 is available and connected to the transmit path of UART module 0. |
| 8 | UART0_RX | RO | 1 | UART0 RX on uDMA Ch8. When set, indicates uDMA channel 8 is available and connected to the receive path of UART module 0. |
| 7:6 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5 | USB_EP3_TX | RO | 1 | USB EP3 TX on uDMA Ch5. When set, indicates uDMA channel 5 is available and connected to the transmit path of USB endpoint 3. |

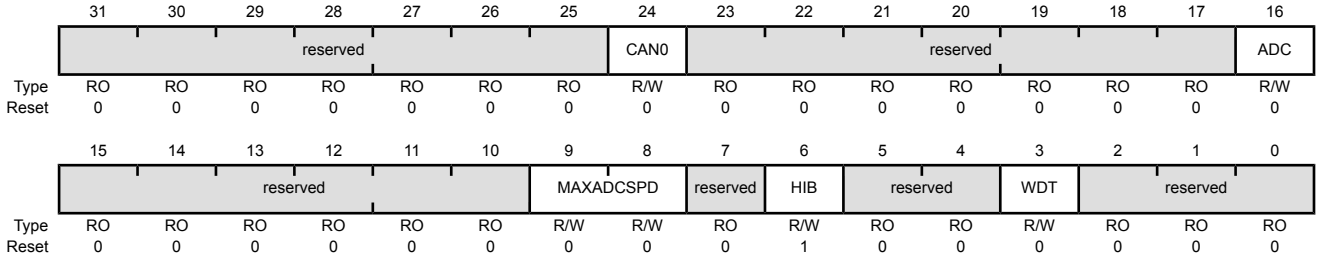
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | USB_EP3_RX | RO | 1 | USB EP3 RX on uDMA Ch4. When set, indicates uDMA channel 4 is available and connected to the receive path of USB endpoint 2. |
| 3 | USB_EP2_TX | RO | 1 | USB EP2 TX on uDMA Ch3. When set, indicates uDMA channel 3 is available and connected to the transmit path of USB endpoint 2. |
| 2 | USB_EP2_RX | RO | 1 | USB EP2 RX on uDMA Ch2. When set, indicates uDMA channel 1 is available and connected to the receive path of USB endpoint 2. |
| 1 | USB_EP1_TX | RO | 1 | USB EP1 TX on uDMA Ch1. When set, indicates uDMA channel 1 is available and connected to the transmit path of USB endpoint 1. |
| 0 | USB_EP1_RX | RO | 1 | USB EP1 RX on uDMA Ch0. When set, indicates uDMA channel 0 is available and connected to the receive path of USB endpoint 1. |

## Register 23: Run Mode Clock Gating Control Register 0 (RCGC0), offset 0x100

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC0** is the clock configuration register for running operation, **SCGC0** for Sleep operation, and **DCGC0** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Run Mode Clock Gating Control Register 0 (RCGC0)

Base 0x400F.E000
Offset 0x100
Type R/W, reset 0x00000040




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:25 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 24 | CAN0 | R/W | 0 | CAN0 Clock Gating Control. This bit controls the clock gating for CAN unit 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. |
| 23:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | ADC | R/W | 0 | ADC0 Clock Gating Control. This bit controls the clock gating for SAR ADC module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 15:10 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

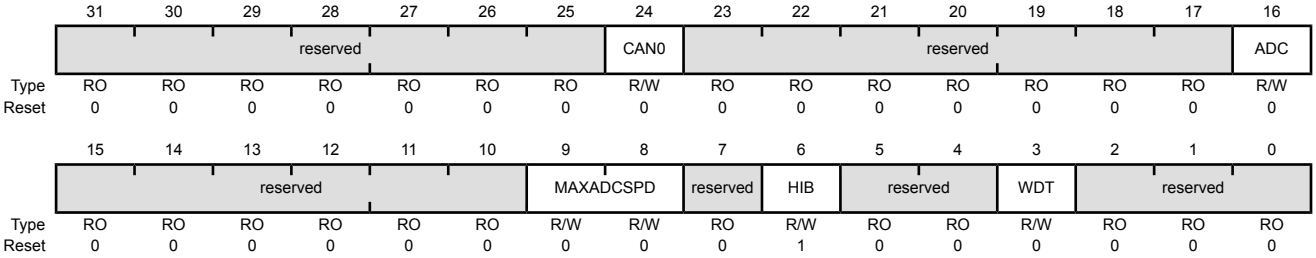
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 9:8 | MAXADCSPD | R/W | 0 | ADC Sample Speed. This field sets the rate at which the ADC samples data. You cannot set the rate higher than the maximum rate. You can set the sample rate by setting the MAXADCSPD bit as follows:<br><br>Value Description<br>0x2 500K samples/second<br>0x1 250K samples/second<br>0x0 125K samples/second |
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | HIB | R/W | 1 | HIB Clock Gating Control. This bit controls the clock gating for the Hibernation module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. |
| 5:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | WDT | R/W | 0 | WDT Clock Gating Control. This bit controls the clock gating for the WDT module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 2:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 24: Sleep Mode Clock Gating Control Register 0 (SCGC0), offset 0x110

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC0** is the clock configuration register for running operation, **SCGC0** for Sleep operation, and **DCGC0** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Sleep Mode Clock Gating Control Register 0 (SCGC0)

Base 0x400F.E000
Offset 0x110
Type R/W, reset 0x00000040




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:25 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 24 | CAN0 | R/W | 0 | CAN0 Clock Gating Control. This bit controls the clock gating for CAN unit 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. |
| 23:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | ADC | R/W | 0 | ADC0 Clock Gating Control. This bit controls the clock gating for general SAR ADC module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 15:10 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

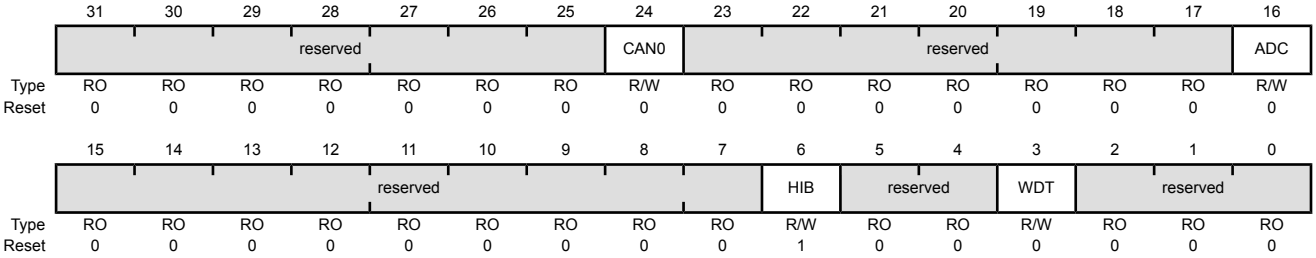
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 9:8 | MAXADCSPD | R/W | 0 | ADC Sample Speed. This field sets the rate at which the ADC samples data. You cannot set the rate higher than the maximum rate.You can set the sample rate by setting the MAXADCSPD bit as follows:<br><br>Value Description<br>0x2 500K samples/second<br>0x1 250K samples/second<br>0x0 125K samples/second |
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | HIB | R/W | 1 | HIB Clock Gating Control. This bit controls the clock gating for the Hibernation module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. |
| 5:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | WDT | R/W | 0 | WDT Clock Gating Control. This bit controls the clock gating for the WDT module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 2:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 25: Deep Sleep Mode Clock Gating Control Register 0 (DCGC0), offset 0x120

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC0** is the clock configuration register for running operation, **SCGC0** for Sleep operation, and **DCGC0** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Deep Sleep Mode Clock Gating Control Register 0 (DCGC0)

Base 0x400F.E000
Offset 0x120
Type R/W, reset 0x00000040




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:25 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 24 | CAN0 | R/W | 0 | CAN0 Clock Gating Control. This bit controls the clock gating for CAN unit 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. |
| 23:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | ADC | R/W | 0 | ADC0 Clock Gating Control. This bit controls the clock gating for general SAR ADC module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 15:7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | HIB | R/W | 1 | HIB Clock Gating Control. This bit controls the clock gating for the Hibernation module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. |
| 5:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | WDT | R/W | 0 | WDT Clock Gating Control. This bit controls the clock gating for the WDT module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 2:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 26: Run Mode Clock Gating Control Register 1 (RCGC1), offset 0x104

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC1** is the clock configuration register for running operation, **SCGC1** for Sleep operation, and **DCGC1** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Run Mode Clock Gating Control Register 1 (RCGC1)

Base 0x400F.E000
Offset 0x104
Type R/W, reset 0x00000000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | I2C1 | reserved | I2C0 | reserved | | | | | | | SSI0 | reserved | | UART1 | UART0 |
| Type | RO | R/W | RO | R/W | RO | RO | RO | RO | RO | RO | RO | R/W | RO | RO | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | TIMER2 | R/W | 0 | Timer 2 Clock Gating Control. This bit controls the clock gating for General-Purpose Timer module 2. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 17 | TIMER1 | R/W | 0 | Timer 1 Clock Gating Control. This bit controls the clock gating for General-Purpose Timer module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 16 | TIMER0 | R/W | 0 | Timer 0 Clock Gating Control. This bit controls the clock gating for General-Purpose Timer module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 15 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14 | I2C1 | R/W | 0 | I2C1 Clock Gating Control. This bit controls the clock gating for I2C module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

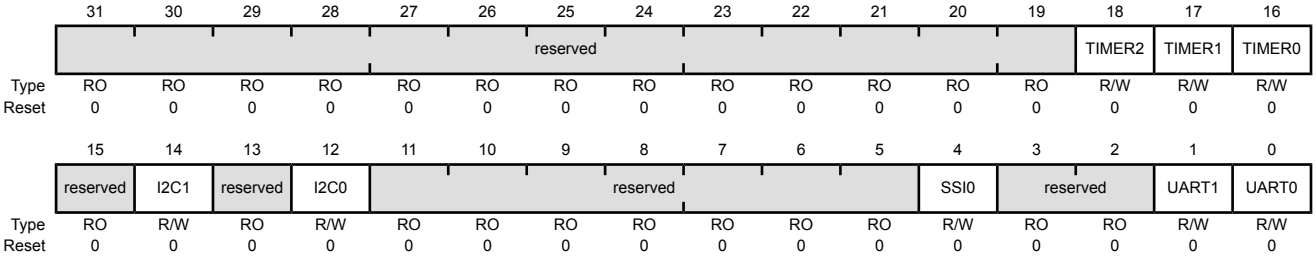
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12 | I2C0 | R/W | 0 | I2C0 Clock Gating Control. This bit controls the clock gating for I2C module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SSI0 | R/W | 0 | SSI0 Clock Gating Control. This bit controls the clock gating for SSI module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | UART1 | R/W | 0 | UART1 Clock Gating Control. This bit controls the clock gating for UART module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | UART0 | R/W | 0 | UART0 Clock Gating Control. This bit controls the clock gating for UART module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 27: Sleep Mode Clock Gating Control Register 1 (SCGC1), offset 0x114

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC1** is the clock configuration register for running operation, **SCGC1** for Sleep operation, and **DCGC1** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Sleep Mode Clock Gating Control Register 1 (SCGC1)

Base 0x400F.E000
Offset 0x114
Type R/W, reset 0x00000000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | TIMER2 | R/W | 0 | Timer 2 Clock Gating Control. This bit controls the clock gating for General-Purpose Timer module 2. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 17 | TIMER1 | R/W | 0 | Timer 1 Clock Gating Control. This bit controls the clock gating for General-Purpose Timer module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 16 | TIMER0 | R/W | 0 | Timer 0 Clock Gating Control. This bit controls the clock gating for General-Purpose Timer module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 15 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14 | I2C1 | R/W | 0 | I2C1 Clock Gating Control. This bit controls the clock gating for I2C module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12 | I2C0 | R/W | 0 | I2C0 Clock Gating Control. This bit controls the clock gating for I2C module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SSI0 | R/W | 0 | SSI0 Clock Gating Control. This bit controls the clock gating for SSI module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | UART1 | R/W | 0 | UART1 Clock Gating Control. This bit controls the clock gating for UART module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | UART0 | R/W | 0 | UART0 Clock Gating Control. This bit controls the clock gating for UART module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 28: Deep Sleep Mode Clock Gating Control Register 1 (DCGC1), offset 0x124

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC1** is the clock configuration register for running operation, **SCGC1** for Sleep operation, and **DCGC1** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Deep Sleep Mode Clock Gating Control Register 1 (DCGC1)

Base 0x400F.E000
Offset 0x124
Type R/W, reset 0x00000000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | I2C1 | reserved | I2C0 | reserved | | | | | | | SSI0 | reserved | | UART1 | UART0 |
| Type | RO | R/W | RO | R/W | RO | RO | RO | RO | RO | RO | RO | R/W | RO | RO | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | TIMER2 | R/W | 0 | Timer 2 Clock Gating Control. This bit controls the clock gating for General-Purpose Timer module 2. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 17 | TIMER1 | R/W | 0 | Timer 1 Clock Gating Control. This bit controls the clock gating for General-Purpose Timer module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 16 | TIMER0 | R/W | 0 | Timer 0 Clock Gating Control. This bit controls the clock gating for General-Purpose Timer module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 15 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14 | I2C1 | R/W | 0 | I2C1 Clock Gating Control. This bit controls the clock gating for I2C module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12 | I2C0 | R/W | 0 | I2C0 Clock Gating Control. This bit controls the clock gating for I2C module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SSI0 | R/W | 0 | SSI0 Clock Gating Control. This bit controls the clock gating for SSI module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | UART1 | R/W | 0 | UART1 Clock Gating Control. This bit controls the clock gating for UART module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | UART0 | R/W | 0 | UART0 Clock Gating Control. This bit controls the clock gating for UART module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 29: Run Mode Clock Gating Control Register 2 (RCGC2), offset 0x108

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC2** is the clock configuration register for running operation, **SCGC2** for Sleep operation, and **DCGC2** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Run Mode Clock Gating Control Register 2 (RCGC2)

Base 0x400F.E000
Offset 0x108
Type R/W, reset 0x00000000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | USB0 |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | UDMA | reserved | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| Type | RO | RO | R/W | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | USB0 | R/W | 0 | USB0 Clock Gating Control. This bit controls the clock gating for Port H. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 15:14 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 13 | UDMA | R/W | 0 | UDMA Clock Gating Control. This bit controls the clock gating for Port H. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 12:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | GPIOE | R/W | 0 | Port E Clock Gating Control. This bit controls the clock gating for Port E. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

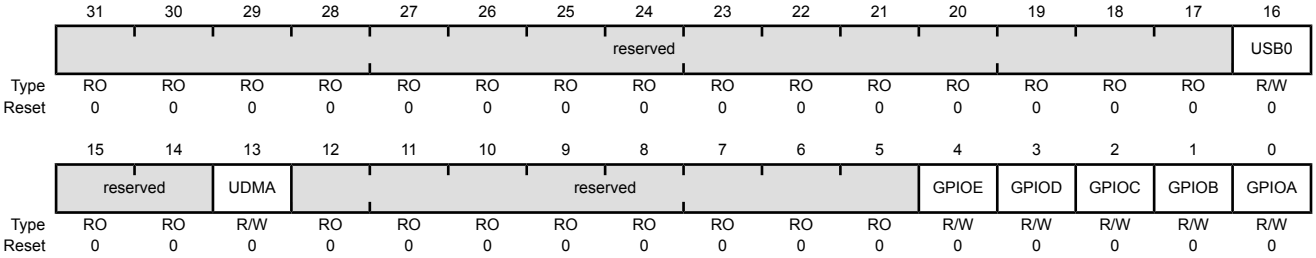
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | GPIOD | R/W | 0 | Port D Clock Gating Control. This bit controls the clock gating for Port D. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 2 | GPIOC | R/W | 0 | Port C Clock Gating Control. This bit controls the clock gating for Port C. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 1 | GPIOB | R/W | 0 | Port B Clock Gating Control. This bit controls the clock gating for Port B. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | GPIOA | R/W | 0 | Port A Clock Gating Control. This bit controls the clock gating for Port A. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 30: Sleep Mode Clock Gating Control Register 2 (SCGC2), offset 0x118

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC2** is the clock configuration register for running operation, **SCGC2** for Sleep operation, and **DCGC2** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Sleep Mode Clock Gating Control Register 2 (SCGC2)

Base 0x400F.E000
Offset 0x118
Type R/W, reset 0x00000000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | USB0 | R/W | 0 | USB0 Clock Gating Control. This bit controls the clock gating for Port H. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 15:14 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 13 | UDMA | R/W | 0 | UDMA Clock Gating Control. This bit controls the clock gating for Port H. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 12:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | GPIOE | R/W | 0 | Port E Clock Gating Control. This bit controls the clock gating for Port E. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

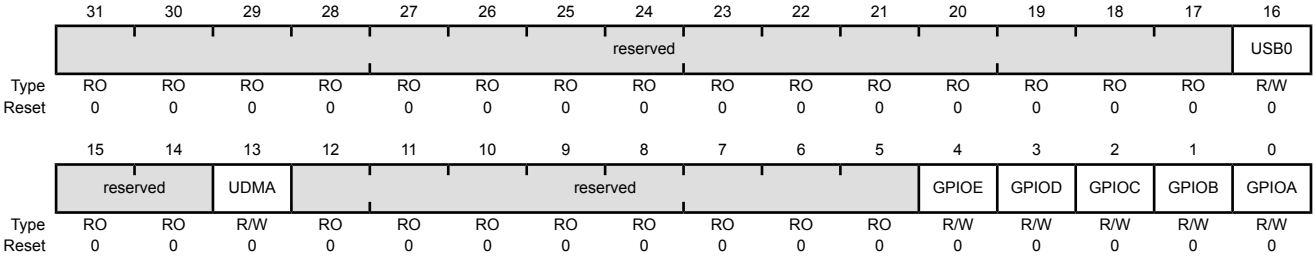
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | GPIOD | R/W | 0 | Port D Clock Gating Control. This bit controls the clock gating for Port D. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 2 | GPIOC | R/W | 0 | Port C Clock Gating Control. This bit controls the clock gating for Port C. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 1 | GPIOB | R/W | 0 | Port B Clock Gating Control. This bit controls the clock gating for Port B. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | GPIOA | R/W | 0 | Port A Clock Gating Control. This bit controls the clock gating for Port A. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 31: Deep Sleep Mode Clock Gating Control Register 2 (DCGC2), offset 0x128

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC2** is the clock configuration register for running operation, **SCGC2** for Sleep operation, and **DCGC2** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Deep Sleep Mode Clock Gating Control Register 2 (DCGC2)

Base 0x400F.E000
Offset 0x128
Type R/W, reset 0x00000000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | USB0 | R/W | 0 | USB0 Clock Gating Control. This bit controls the clock gating for Port H. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 15:14 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 13 | UDMA | R/W | 0 | UDMA Clock Gating Control. This bit controls the clock gating for Port H. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 12:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | GPIOE | R/W | 0 | Port E Clock Gating Control. This bit controls the clock gating for Port E. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

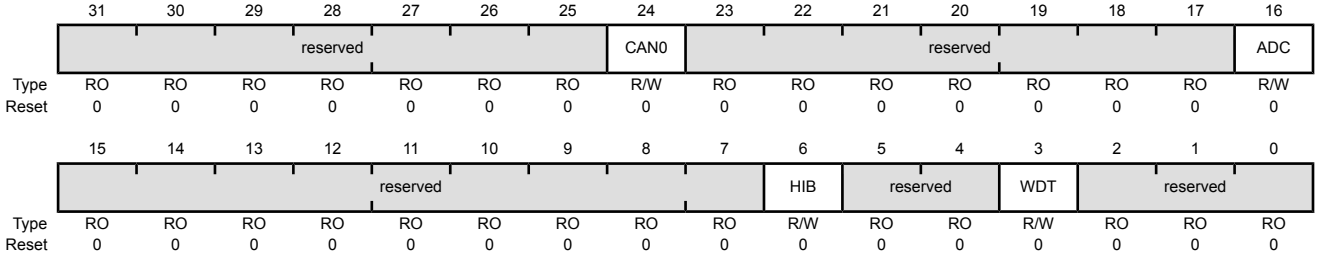
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | GPIOD | R/W | 0 | Port D Clock Gating Control. This bit controls the clock gating for Port D. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 2 | GPIOC | R/W | 0 | Port C Clock Gating Control. This bit controls the clock gating for Port C. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 1 | GPIOB | R/W | 0 | Port B Clock Gating Control. This bit controls the clock gating for Port B. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | GPIOA | R/W | 0 | Port A Clock Gating Control. This bit controls the clock gating for Port A. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 32: Software Reset Control 0 (SRCR0), offset 0x040

Writes to this register are masked by the bits in the **Device Capabilities 1 (DC1)** register.

Software Reset Control 0 (SRCR0)

Base 0x400F.E000
Offset 0x040
Type R/W, reset 0x00000000



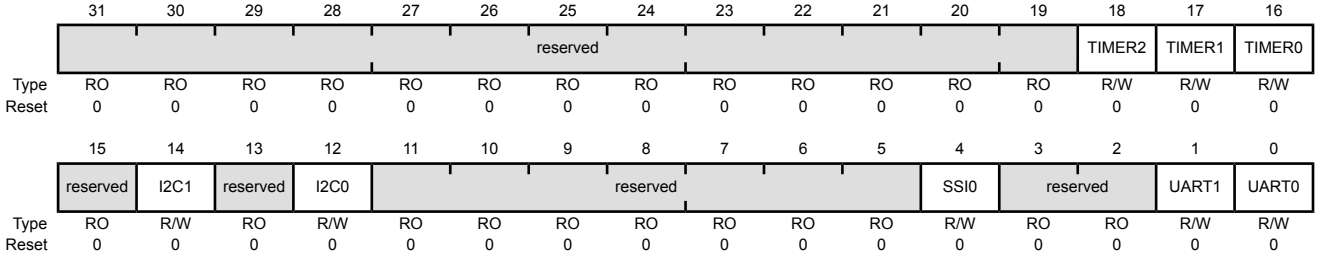

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:25 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 24 | CAN0 | R/W | 0 | CAN0 Reset Control. Reset control for CAN unit 0. |
| 23:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | ADC | R/W | 0 | ADC0 Reset Control. Reset control for SAR ADC module 0. |
| 15:7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | HIB | R/W | 0 | HIB Reset Control. Reset control for the Hibernation module. |
| 5:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | WDT | R/W | 0 | WDT Reset Control. Reset control for Watchdog unit. |
| 2:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 33: Software Reset Control 1 (SRCR1), offset 0x044

Writes to this register are masked by the bits in the **Device Capabilities 2 (DC2)** register.

Software Reset Control 1 (SRCR1)

Base 0x400F.E000
Offset 0x044
Type R/W, reset 0x00000000



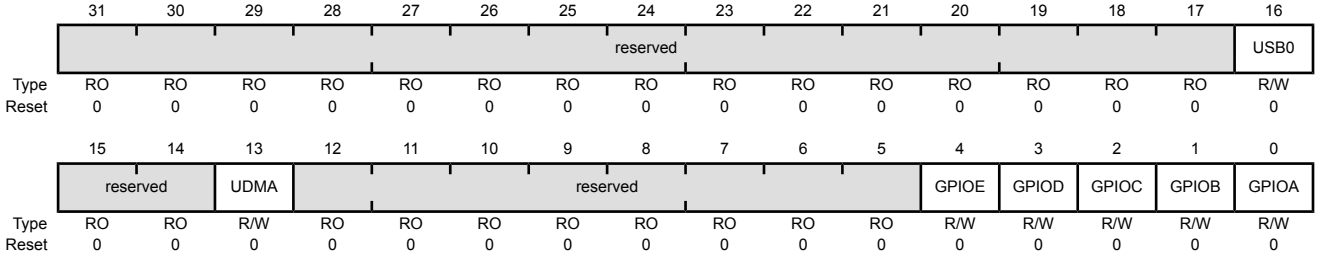

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | TIMER2 | R/W | 0 | Timer 2 Reset Control. Reset control for General-Purpose Timer module 2. |
| 17 | TIMER1 | R/W | 0 | Timer 1 Reset Control. Reset control for General-Purpose Timer module 1. |
| 16 | TIMER0 | R/W | 0 | Timer 0 Reset Control. Reset control for General-Purpose Timer module 0. |
| 15 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14 | I2C1 | R/W | 0 | I2C1 Reset Control. Reset control for I2C unit 1. |
| 13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12 | I2C0 | R/W | 0 | I2C0 Reset Control. Reset control for I2C unit 0. |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SSI0 | R/W | 0 | SSI0 Reset Control. Reset control for SSI unit 0. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | UART1 | R/W | 0 | UART1 Reset Control. Reset control for UART unit 1. |
| 0 | UART0 | R/W | 0 | UART0 Reset Control. Reset control for UART unit 0. |

## Register 34: Software Reset Control 2 (SRCR2), offset 0x048

Writes to this register are masked by the bits in the **Device Capabilities 4 (DC4)** register.

Software Reset Control 2 (SRCR2)

Base 0x400F.E000
Offset 0x048
Type R/W, reset 0x00000000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | USB0 | R/W | 0 | USB0 Reset Control. Reset control for USB unit 0. |
| 15:14 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 13 | UDMA | R/W | 0 | UDMA Reset Control. Reset control for uDMA unit. |
| 12:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | GPIOE | R/W | 0 | Port E Reset Control. Reset control for GPIO Port E. |
| 3 | GPIOD | R/W | 0 | Port D Reset Control. Reset control for GPIO Port D. |
| 2 | GPIOC | R/W | 0 | Port C Reset Control. Reset control for GPIO Port C. |
| 1 | GPIOB | R/W | 0 | Port B Reset Control. Reset control for GPIO Port B. |
| 0 | GPIOA | R/W | 0 | Port A Reset Control. Reset control for GPIO Port A. |

# 6 Hibernation Module

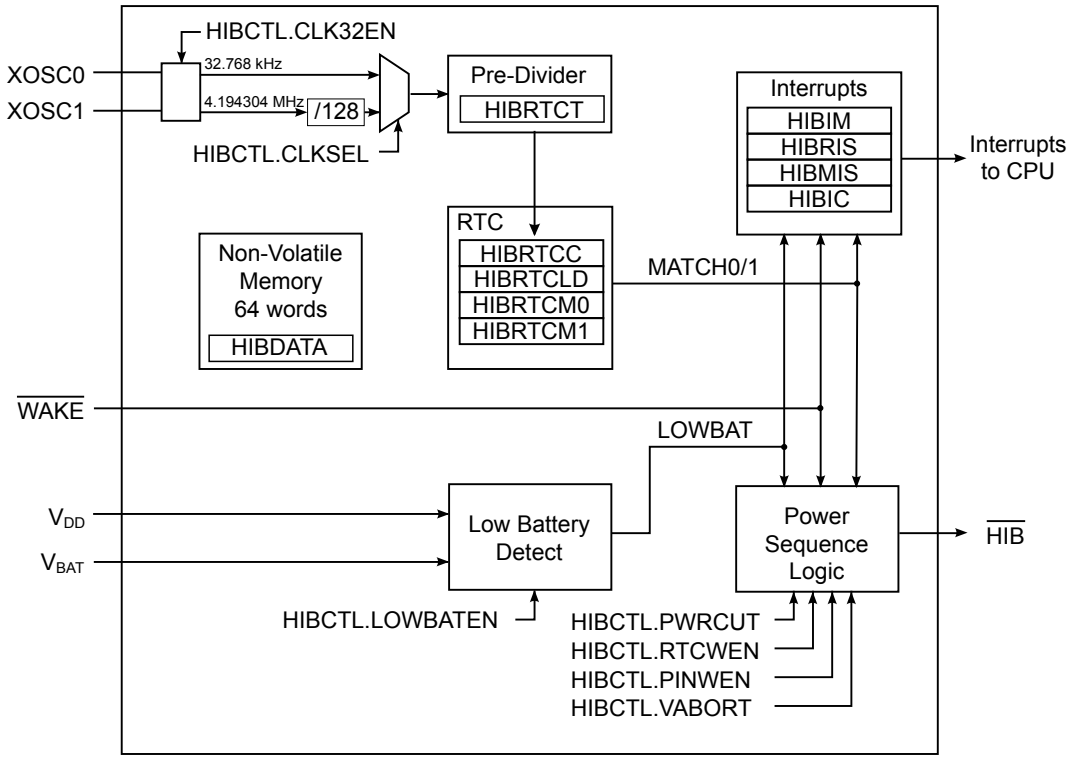
The Hibernation Module manages removal and restoration of power to provide a means for reducing power consumption. When the processor and peripherals are idle, power can be completely removed with only the Hibernation module remaining powered. Power can be restored based on an external signal, or at a certain time using the built-in Real-Time Clock (RTC). The Hibernation module can be independently supplied from a battery or an auxiliary power supply.

The Hibernation module has the following features:

- System power control using discrete external regulator
- Dedicated pin for waking from an external signal
- Low-battery detection, signaling, and interrupt generation
- 32-bit real-time clock (RTC)
- Two 32-bit RTC match registers for timed wake-up and interrupt generation
- Clock source from a 32.768-kHz external oscillator or a 4.194304-MHz crystal
- RTC predivider trim for making fine adjustments to the clock rate
- 64 32-bit words of non-volatile memory
- Programmable interrupts for RTC match, external wake, and low battery events

## 6.1 Block Diagram

**Figure 6-1. Hibernation Module Block Diagram**




## 6.2 Signal Description

Table 6-1 on page 239 lists the external signals of the Hibernation module and describes the function of each. These signals have dedicated functions and are not alternate functions for any GPIO signals.

**Table 6-1. Hibernate Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| HIB | 33 | O | OD | An output that indicates the processor is in Hibernate mode. |
| VBAT | 37 | - | Power | Power source for the Hibernation module. It is normally connected to the positive terminal of a battery and serves as the battery backup/Hibernation module power-source supply. |
| WAKE | 32 | I | TTL | An external input that brings the processor out of Hibernate mode when asserted. |
| XOSC0 | 34 | I | Analog | Hibernation module oscillator crystal input or an external clock reference input. Note that this is either a crystal or a 32.768-kHz oscillator for the Hibernation module RTC. |
| XOSC1 | 35 | O | Analog | Hibernation module oscillator crystal output. Leave unconnected when using a single-ended clock source. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 6.3 Functional Description

The Hibernation module controls the power to the processor with an enable signal (HIB) that signals an external voltage regulator to turn off.

The Hibernation module power source is determined dynamically. The supply voltage of the Hibernation module is the larger of the main voltage source ($V_{DD}$) or the battery/auxilliary voltage source ($V_{BAT}$). A voting circuit indicates the larger and an internal power switch selects the appropriate voltage source. The Hibernation module also has a separate clock source to maintain a real-time clock (RTC). Once in hibernation, the module signals an external voltage regulator to turn back on the power when an external pin (WAKE) is asserted, or when the internal RTC reaches a certain value. The Hibernation module can also detect when the battery voltage is low, and optionally prevent hibernation when this occurs.

When waking from hibernation, the HIB signal is deasserted. The return of $V_{DD}$ causes a POR to be executed. The time from when the WAKE signal is asserted to when code begins execution is equal to the wake-up time ($t_{WAKE\_TO\_HIB}$) plus the power-on reset time ($T_{IRPOR}$).

### 6.3.1 Register Access Timing

Because the Hibernation module has an independent clocking domain, certain registers must be written only with a timing gap between accesses. The delay time is $t_{HIB\_REG\_WRITE}$, therefore software must guarantee that a delay of $t_{HIB\_REG\_WRITE}$ is inserted between back-to-back writes to certain Hibernation registers, or between a write followed by a read to those same registers. There is no restriction on timing for back-to-back reads from the Hibernation module. Software may make use of the WRC bit in the **HIBCTL** register to ensure that the required timing gap has elapsed. This bit is cleared on a write operation and set once the write completes, indicating to software that another write or read may be started safely. Software should poll **HIBCTL** for WRC=1 prior to accessing any affected register. The following registers are subject to this timing restriction:

- **Hibernation RTC Counter (HIBRTCC)**
- **Hibernation RTC Match 0 (HIBRTCM0)**
- **Hibernation RTC Match 1 (HIBRTCM1)**
- **Hibernation RTC Load (HIBRTCLD)**
- **Hibernation RTC Trim (HIBRTCT)**
- **Hibernation Data (HIBDATA)**

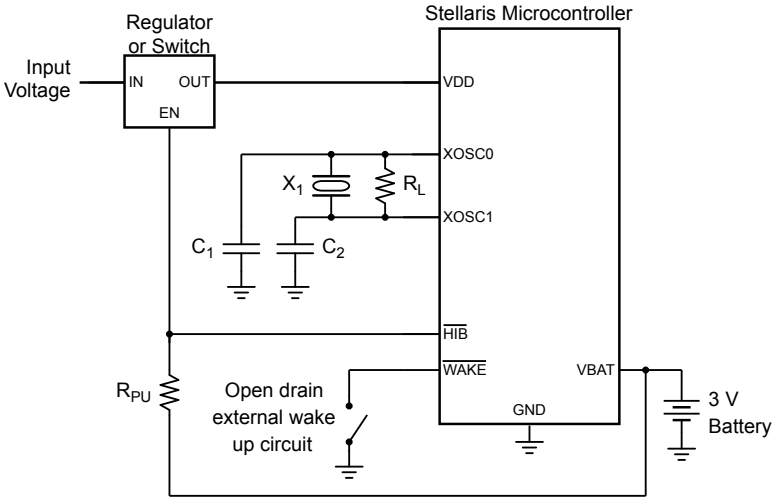
### 6.3.2 Clock Source

The Hibernation module must be clocked by an external source, even if the RTC feature is not used. An external oscillator or crystal can be used for this purpose. To use a crystal, a 4.194304-MHz crystal is connected to the XOSC0 and XOSC1 pins. This clock signal is divided by 128 internally to produce the 32.768-kHz clock reference. For an alternate clock source, a 32.768-kHz oscillator can be connected to the XOSC0 pin. See Figure 6-2 on page 241 and Figure 6-3 on page 241. Note that these diagrams only show the connection to the Hibernation pins and not to the full system. See “Hibernation Module” on page 775 for specific values.

The clock source is enabled by setting the CLK32EN bit of the **HIBCTL** register. The type of clock source is selected by setting the CLKSEL bit to 0 for a 4.194304-MHz clock source, and to 1 for a 32.768-kHz clock source. If the bit is set to 0, the 4.194304-MHz input clock is divided by 128,

resulting in a 32.768-kHz clock source. If a crystal is used for the clock source, the software must leave a delay of $t_{XOSC\_SETTLE}$ after setting the CLK32EN bit and before any other accesses to the Hibernation module registers. The delay allows the crystal to power up and stabilize. If an oscillator is used for the clock source, no delay is needed.

**Figure 6-2. Clock Source Using Crystal**




**Note:** $X_1$ = Crystal frequency is $f_{XOSC\_XTAL}$.

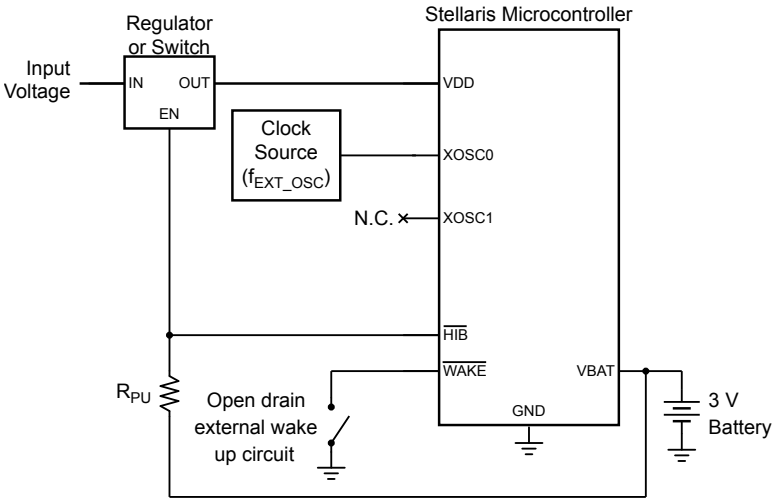
$C_{1,2}$ = Capacitor value derived from crystal vendor load capacitance specifications.

$R_L$ = Load resistor is $R_{XOSC\_LOAD}$.

$R_{PU}$ = Pull-up resistor (1 M½).

See “Hibernation Module” on page 775 for specific parameter values.

**Figure 6-3. Clock Source Using Dedicated Oscillator**




**Note:** $R_{PU}$ = Pull-up resistor (1 M½).

### 6.3.3 Battery Management

The Hibernation module can be independently powered by a battery or an auxiliary power source. The module can monitor the voltage level of the battery and detect when the voltage drops below $V_{LOWBAT}$. When this happens, an interrupt can be generated. The module can also be configured so that it will not go into Hibernate mode if the battery voltage drops below this threshold. Battery voltage is not measured while in Hibernate mode.

**Important:** System level factors may affect the accuracy of the low battery detect circuit. The designer should consider battery type, discharge characteristics, and a test load during battery voltage measurements.

Note that the Hibernation module draws power from whichever source ($V_{BAT}$ or $V_{DD}$) has the higher voltage. Therefore, it is important to design the circuit to ensure that $V_{DD}$ is higher that $V_{BAT}$ under nominal conditions or else the Hibernation module draws power from the battery even when $V_{DD}$ is available.

The Hibernation module can be configured to detect a low battery condition by setting the LOWBATEN bit of the **HIBCTL** register. In this configuration, the LOWBAT bit of the **HIBRIS** register will be set when the battery level is low. If the VABORT bit is also set, then the module is prevented from entering Hibernation mode when a low battery is detected. The module can also be configured to generate an interrupt for the low-battery condition (see “Interrupts and Status” on page 243).

### 6.3.4 Real-Time Clock

The Hibernation module includes a 32-bit counter that increments once per second with a proper clock source and configuration (see “Clock Source” on page 240). The 32.768-kHz clock signal is fed into a predivider register which counts down the 32.768-kHz clock ticks to achieve a once per second clock rate for the RTC. The rate can be adjusted to compensate for inaccuracies in the clock source by using the predivider trim register, **HIBRTCT**. This register has a nominal value of 0x7FFF, and is used for one second out of every 64 seconds to divide the input clock. This allows the software to make fine corrections to the clock rate by adjusting the predivider trim register up or down from 0x7FFF. The predivider trim should be adjusted up from 0x7FFF in order to slow down the RTC rate, and down from 0x7FFF in order to speed up the RTC rate.

The Hibernation module includes two 32-bit match registers that are compared to the value of the RTC counter. The match registers can be used to wake the processor from hibernation mode, or to generate an interrupt to the processor if it is not in hibernation.

The RTC must be enabled with the RTCEN bit of the **HIBCTL** register. The value of the RTC can be set at any time by writing to the **HIBRTCLD** register. The predivider trim can be adjusted by reading and writing the **HIBRTCT** register. The predivider uses this register once every 64 seconds to adjust the clock rate. The two match registers can be set by writing to the **HIBRTCM0** and **HIBRTCM1** registers. The RTC can be configured to generate interrupts by using the interrupt registers (see “Interrupts and Status” on page 243). As long as the RTC is enabled and a valid $V_{BAT}$ is present, the RTC continues counting, regardless of whether $V_{DD}$ is present or if the part is in hibernation.

### 6.3.5 Battery-Backed Memory

The Hibernation module contains 64 32-bit words of memory which are retained during hibernation. This memory is powered from the battery or auxiliary power supply during hibernation. The processor software can save state information in this memory prior to hibernation, and can then recover the state upon waking. The battery-backed memory can be accessed through the **HIBDATA** registers.

### 6.3.6 Power Control

**Important:** The Hibernation Module requires special system implementation considerations when using $\overline{\text{HIB}}$ to control power, as it is intended to power-down all other sections of its host device. All system signals and power supplies that connect to the chip must be driven to 0 $V_{DC}$ or powered down with the same regulator controlled by $\overline{\text{HIB}}$. See “Hibernation Module” on page 775 for more details.

The Hibernation module controls power to the microcontroller through the use of the $\overline{\text{HIB}}$ pin. This pin is intended to be connected to the enable signal of the external regulator(s) providing 3.3 V and/or 2.5 V to the microcontroller. When the $\overline{\text{HIB}}$ signal is asserted by the Hibernation module, the external regulator is turned off and no longer powers the system. The Hibernation module remains powered from the $V_{BAT}$ supply (which could be a battery or an auxiliary power source) until a Wake event. Power to the device is restored by deasserting the $\overline{\text{HIB}}$ signal, which causes the external regulator to turn power back on to the chip.

### 6.3.7 Initiating Hibernate

Hibernation mode is initiated by the microcontroller setting the HIBREQ bit of the **HIBCTL** register. Prior to doing this, a wake-up condition must be configured, either from the external WAKE pin, or by using an RTC match.

The Hibernation module is configured to wake from the external $\overline{\text{WAKE}}$ pin by setting the PINWEN bit of the **HIBCTL** register. It is configured to wake from RTC match by setting the RTCWEN bit. Either one or both of these bits can be set prior to going into hibernation. The $\overline{\text{WAKE}}$ pin includes a weak internal pull-up. Note that both the HIB and $\overline{\text{WAKE}}$ pins use the Hibernation module's internal power supply as the logic 1 reference.

When the Hibernation module wakes, the microcontroller will see a normal power-on reset. Software can detect that the power-on was due to a wake from hibernation by examining the raw interrupt status register (see “Interrupts and Status” on page 243) and by looking for state data in the battery-backed memory (see “Battery-Backed Memory” on page 242).

When the $\overline{\text{HIB}}$ signal deasserts, enabling the external regulator, the external regulator must reach the operating voltage within $t_{HIB\_TO\_VDD}$.

### 6.3.8 Interrupts and Status

The Hibernation module can generate interrupts when the following conditions occur:

- Assertion of $\overline{\text{WAKE}}$ pin
- RTC match
- Low battery detected

All of the interrupts are ORed together before being sent to the interrupt controller, so the Hibernate module can only generate a single interrupt request to the controller at any given time. The software interrupt handler can service multiple interrupt events by reading the **HIBMIS** register. Software can also read the status of the Hibernation module at any time by reading the **HIBRIS** register which shows all of the pending events. This register can be used at power-on to see if a wake condition is pending, which indicates to the software that a hibernation wake occurred.

The events that can trigger an interrupt are configured by setting the appropriate bits in the **HIBIM** register. Pending interrupts can be cleared by writing the corresponding bit in the **HIBIC** register.

## 6.4 Initialization and Configuration

The Hibernation module can be set in several different configurations. The following sections show the recommended programming sequence for various scenarios. The examples below assume that a 32.768-kHz oscillator is used, and thus always show bit 2 (CLKSEL) of the **HIBCTL** register set to 1. If a 4.194304-MHz crystal is used instead, then the CLKSEL bit remains cleared. Because the Hibernation module runs at 32.768 kHz and is asynchronous to the rest of the system, software must allow a delay of $t_{HIB\_REG\_WRITE}$ after writes to certain registers (see "Register Access Timing" on page 240). The registers that require a delay are listed in a note in "Register Map" on page 245 as well as in each register description.

### 6.4.1 Initialization

The Hibernation module clock source must be enabled first, even if the RTC feature is not used. If a 4.194304-MHz crystal is used, perform the following steps:

1. Write 0x40 to the **HIBCTL** register at offset 0x10 to enable the crystal and select the divide-by-128 input path.
2. Wait for a time of $t_{XOSC\_SETTLE}$ for the crystal to power up and stabilize before performing any other operations with the Hibernation module.

If a 32.678-kHz oscillator is used, then perform the following steps:

1. Write 0x44 to the **HIBCTL** register at offset 0x10 to enable the oscillator input.
2. No delay is necessary.

The above is only necessary when the entire system is initialized for the first time. If the processor is powered due to a wake from hibernation, then the Hibernation module has already been powered up and the above steps are not necessary. The software can detect that the Hibernation module and clock are already powered by examining the CLK32EN bit of the **HIBCTL** register.

### 6.4.2 RTC Match Functionality (No Hibernation)

Use the following steps to implement the RTC match functionality of the Hibernation module:

1. Write the required RTC match value to one of the **HIBRTCMn** registers at offset 0x004 or 0x008.
2. Write the required RTC load value to the **HIBRTCLD** register at offset 0x00C.
3. Set the required RTC match interrupt mask in the RTCALT0 and RTCALT1 bits (bits 1:0) in the **HIBIM** register at offset 0x014.
4. Write 0x0000.0041 to the **HIBCTL** register at offset 0x010 to enable the RTC to begin counting.

### 6.4.3 RTC Match/Wake-Up from Hibernation

Use the following steps to implement the RTC match and wake-up functionality of the Hibernation module:

1. Write the required RTC match value to the **HIBRTCMn** registers at offset 0x004 or 0x008.
2. Write the required RTC load value to the **HIBRTCLD** register at offset 0x00C.
3. Write any data to be retained during power cut to the **HIBDATA** register at offsets 0x030-0x12C.

4. Set the RTC Match Wake-Up and start the hibernation sequence by writing 0x0000.004F to the **HIBCTL** register at offset 0x010.

### 6.4.4 External Wake-Up from Hibernation

Use the following steps to implement the Hibernation module with the external WAKE pin as the wake-up source for the microcontroller:

1. Write any data to be retained during power cut to the **HIBDATA** register at offsets 0x030-0x12C.
2. Enable the external wake and start the hibernation sequence by writing 0x0000.0056 to the **HIBCTL** register at offset 0x010.

### 6.4.5 RTC/External Wake-Up from Hibernation

1. Write the required RTC match value to the **HIBRTCMn** registers at offset 0x004 or 0x008.
2. Write the required RTC load value to the **HIBRTCLD** register at offset 0x00C.
3. Write any data to be retained during power cut to the **HIBDATA** register at offsets 0x030-0x12C.
4. Set the RTC Match/External Wake-Up and start the hibernation sequence by writing 0x0000.005F to the **HIBCTL** register at offset 0x010.

## 6.5 Register Map

Table 6-2 on page 245 lists the Hibernation registers. All addresses given are relative to the Hibernation Module base address at 0x400F.C000. Note that the Hibernation module clock must be enabled before the registers can be programmed (see page 217). There must be a delay of 3 system clocks after the Hibernation module clock is enabled before any Hibernation module registers are accessed.

**Note:** **HIBRTCC**, **HIBRTCM0**, **HIBRTCM1**, **HIBRTCLD**, **HIBRTCT**, and **HIBDATA** are on the Hibernation module clock domain and and have special timing requirements. Software should make use of the WRC bit in the **HIBCTL** register to ensure that the required timing gap has elapsed. See “Register Access Timing” on page 240.

**Important:** The Hibernation module registers are reset under two conditions:

1. A system reset when the RTCEN and the PINWEN bits in the **HIBCTL** register are both cleared.
2. A cold POR, when both the $V_{DD}$ and $V_{BAT}$ supplies are removed.

Any other reset condition is ignored by the Hibernation module.

**Table 6-2. Hibernation Module Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | HIBRTCC | RO | 0x0000.0000 | Hibernation RTC Counter | 247 |
| 0x004 | HIBRTCM0 | R/W | 0xFFFF.FFFF | Hibernation RTC Match 0 | 248 |
| 0x008 | HIBRTCM1 | R/W | 0xFFFF.FFFF | Hibernation RTC Match 1 | 249 |
| 0x00C | HIBRTCLD | R/W | 0xFFFF.FFFF | Hibernation RTC Load | 250 |

**Table 6-2. Hibernation Module Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x010 | HIBCTL | R/W | 0x8000.0000 | Hibernation Control | 251 |
| 0x014 | HIBIM | R/W | 0x0000.0000 | Hibernation Interrupt Mask | 254 |
| 0x018 | HIBRIS | RO | 0x0000.0000 | Hibernation Raw Interrupt Status | 255 |
| 0x01C | HIBMIS | RO | 0x0000.0000 | Hibernation Masked Interrupt Status | 256 |
| 0x020 | HIBIC | R/W1C | 0x0000.0000 | Hibernation Interrupt Clear | 257 |
| 0x024 | HIBRTCT | R/W | 0x0000.7FFF | Hibernation RTC Trim | 258 |
| 0x030-0x12C | HIBDATA | R/W | - | Hibernation Data | 259 |

## 6.6 Register Descriptions

The remainder of this section lists and describes the Hibernation module registers, in numerical order by address offset.

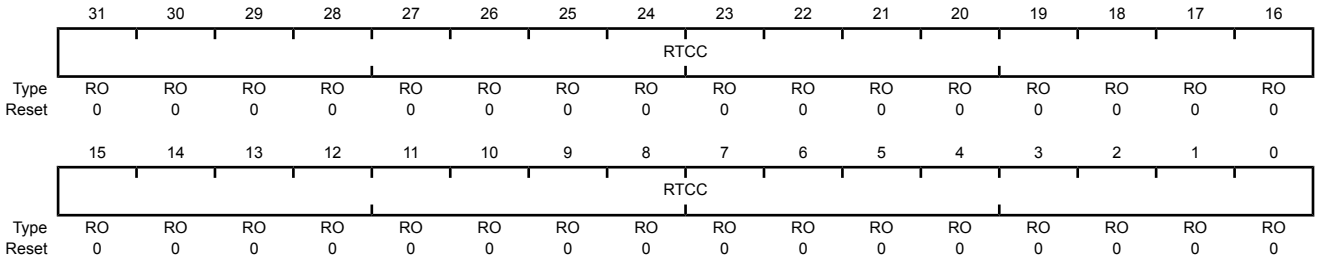
## Register 1: Hibernation RTC Counter (HIBRTCC), offset 0x000

This register is the current 32-bit value of the RTC counter.

**Note:** **HIBRTCC**, **HIBRTCM0**, **HIBRTCM1**, **HIBRTCLD**, **HIBRTCT**, and **HIBDATA** are on the Hibernation module clock domain and and have special timing requirements. Software should make use of the WRC bit in the **HIBCTL** register to ensure that the required timing gap has elapsed. See “Register Access Timing” on page 240.

Hibernation RTC Counter (HIBRTCC)

Base 0x400F.C000
Offset 0x000
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | RTCC | RO | 0x0000.0000 | RTC Counter<br>A read returns the 32-bit counter value. This register is read-only. To change the value, use the **HIBRTCLD** register. |

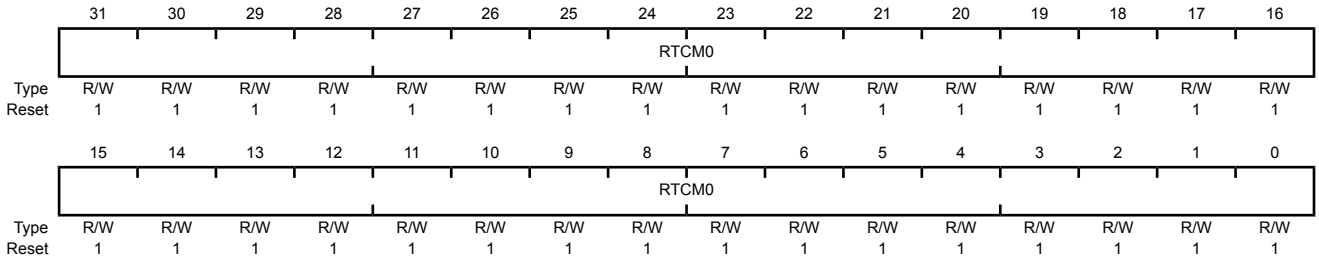
## Register 2: Hibernation RTC Match 0 (HIBRTCM0), offset 0x004

This register is the 32-bit match 0 register for the RTC counter.

**Note:** **HIBRTCC**, **HIBRTCM0**, **HIBRTCM1**, **HIBRTCLD**, **HIBRTCT**, and **HIBDATA** are on the Hibernation module clock domain and and have special timing requirements. Software should make use of the WRC bit in the **HIBCTL** register to ensure that the required timing gap has elapsed. See “Register Access Timing” on page 240.

Hibernation RTC Match 0 (HIBRTCM0)

Base 0x400F.C000
Offset 0x004
Type R/W, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | RTCM0 | R/W | 0xFFFF.FFFF | RTC Match 0<br>A write loads the value into the RTC match register.<br>A read returns the current match value. |

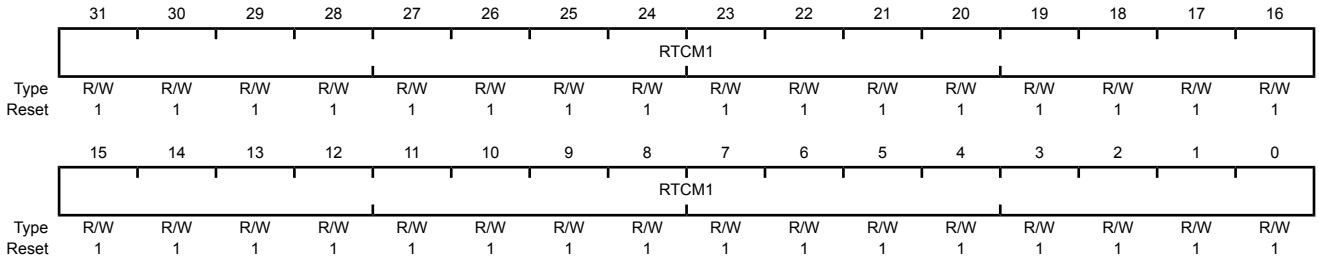
## Register 3: Hibernation RTC Match 1 (HIBRTCM1), offset 0x008

This register is the 32-bit match 1 register for the RTC counter.

**Note:** **HIBRTCC**, **HIBRTCM0**, **HIBRTCM1**, **HIBRTCLD**, **HIBRTCT**, and **HIBDATA** are on the Hibernation module clock domain and and have special timing requirements. Software should make use of the WRC bit in the **HIBCTL** register to ensure that the required timing gap has elapsed. See “Register Access Timing” on page 240.

Hibernation RTC Match 1 (HIBRTCM1)

Base 0x400F.C000
Offset 0x008
Type R/W, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | RTCM1 | R/W | 0xFFFF.FFFF | RTC Match 1<br>A write loads the value into the RTC match register.<br>A read returns the current match value. |

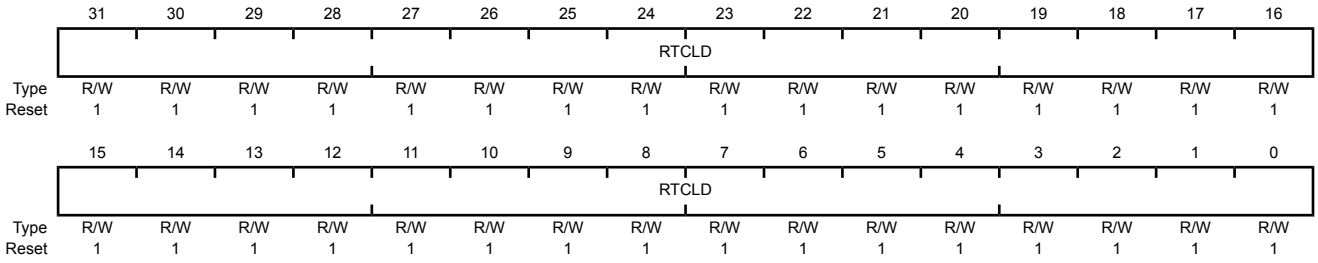
## Register 4: Hibernation RTC Load (HIBRTCLD), offset 0x00C

This register is the 32-bit value loaded into the RTC counter.

**Note:** **HIBRTCC**, **HIBRTCM0**, **HIBRTCM1**, **HIBRTCLD**, **HIBRTCT**, and **HIBDATA** are on the Hibernation module clock domain and and have special timing requirements. Software should make use of the WRC bit in the **HIBCTL** register to ensure that the required timing gap has elapsed. See “Register Access Timing” on page 240.

Hibernation RTC Load (HIBRTCLD)

Base 0x400F.C000
Offset 0x00C
Type R/W, reset 0xFFFF.FFFF



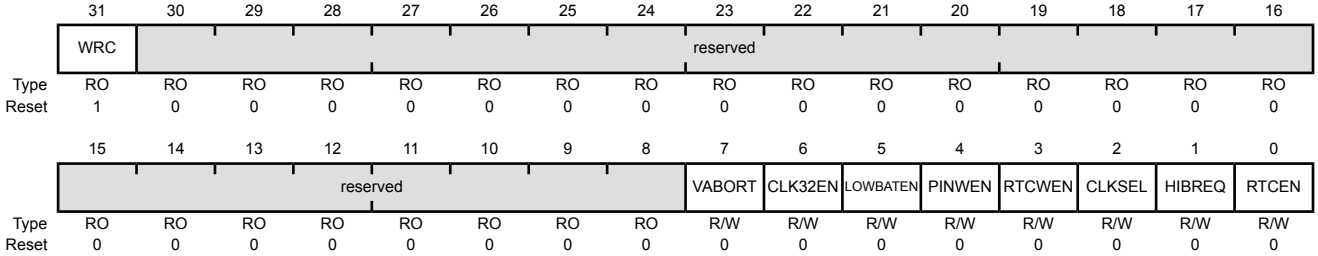

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | RTCLD | R/W | 0xFFFF.FFFF | RTC Load<br>A write loads the current value into the RTC counter (RTCC).<br>A read returns the 32-bit load value. |

## Register 5: Hibernation Control (HIBCTL), offset 0x010

This register is the control register for the Hibernation module.

Hibernation Control (HIBCTL)

Base 0x400F.C000
Offset 0x010
Type R/W, reset 0x8000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | WRC | RO | 1 | Write Complete/Capable<br>This bit indicates whether the hibernation module can receive a write operation.<br><br>Value Description<br>0 The interface is processing a prior write and is busy. Any write operation that is attempted while WRC is 0 results in undetermined behavior.<br>1 The interface is ready to accept a write.<br><br>Software must poll this bit between write requests and defer writes until WRC=1 to ensure proper operation.<br>This difference may be exploited by software at reset time to detect which method of programming is appropriate: 0 = software delay loops required; 1 = WRC paced available.<br>The bit name WRC means "Write Complete," which is the normal use of the bit (between write accesses). However, because the bit is set out-of-reset, the name can also mean "Write Capable" which simply indicates that the interface may be written to by software. This meaning also has more meaning to the out-of-reset sense. |
| 30:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7 | VABORT | R/W | 0 | Power Cut Abort Enable<br><br>Value Description<br>0 Power cut occurs during a low-battery alert.<br>1 Power cut is aborted. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | CLK32EN | R/W | 0 | Clocking Enable<br><br>Value: 0 = Disabled; 1 = Enabled<br><br>This bit must be enabled to use the Hibernation module. If a crystal is used, then software should wait 20 ms after setting this bit to allow the crystal to power up and stabilize. |
| 5 | LOWBATEN | R/W | 0 | Low Battery Monitoring Enable<br><br>Value: 0 = Disabled; 1 = Enabled<br><br>When set, low battery voltage detection is enabled ($V_{BAT} < V_{LOWBAT}$). |
| 4 | PINWEN | R/W | 0 | External $\overline{\text{WAKE}}$ Pin Enable<br><br>Value: 0 = Disabled; 1 = Enabled<br><br>When set, an external event on the $\overline{\text{WAKE}}$ pin will re-power the device. |
| 3 | RTCWEN | R/W | 0 | RTC Wake-up Enable<br><br>Value: 0 = Disabled; 1 = Enabled<br><br>When set, an RTC match event (RTCM0 or RTCM1) will re-power the device based on the RTC counter value matching the corresponding match register 0 or 1. |
| 2 | CLKSEL | R/W | 0 | Hibernation Module Clock Select<br><br>Value: 0 = Use Divide by 128 output. Use this value for a 4.194304-MHz crystal.; 1 = Use raw output. Use this value for a 32.768-kHz oscillator. |
| 1 | HIBREQ | R/W | 0 | Hibernation Request<br><br>Value: 0 = Disabled; 1 = Hibernation initiated<br><br>After a wake-up event, this bit is cleared by hardware. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | RTCEN | R/W | 0 | RTC Timer Enable |

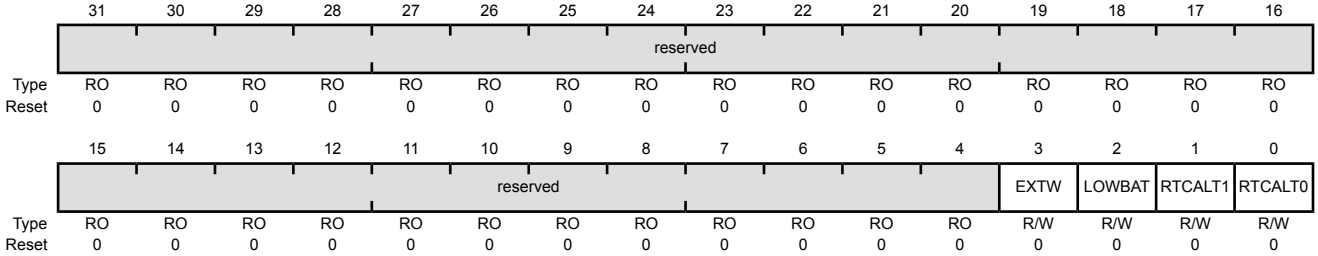
| Value | Description |
|---|---|
| 0 | Disabled |
| 1 | Enabled |

## Register 6: Hibernation Interrupt Mask (HIBIM), offset 0x014

This register is the interrupt mask register for the Hibernation module interrupt sources.

Hibernation Interrupt Mask (HIBIM)

Base 0x400F.C000
Offset 0x014
Type R/W, reset 0x0000.0000



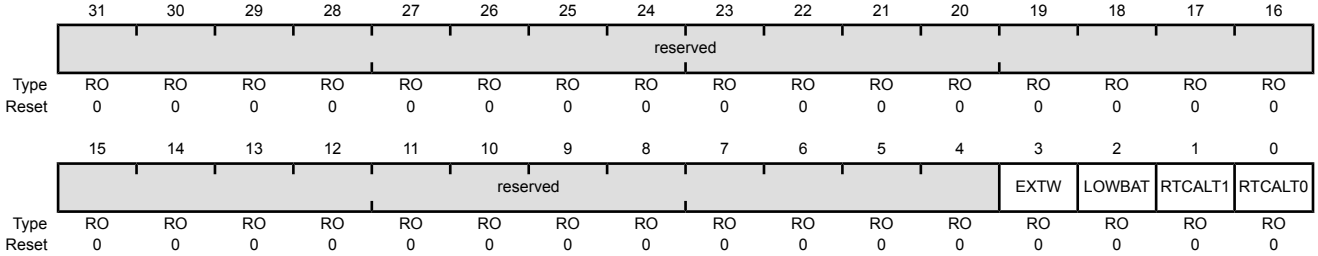

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x000.0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EXTW | R/W | 0 | External Wake-Up Interrupt Mask<br>Value: 0 = Masked; 1 = Unmasked |
| 2 | LOWBAT | R/W | 0 | Low Battery Voltage Interrupt Mask<br>Value: 0 = Masked; 1 = Unmasked |
| 1 | RTCALT1 | R/W | 0 | RTC Alert1 Interrupt Mask<br>Value: 0 = Masked; 1 = Unmasked |
| 0 | RTCALT0 | R/W | 0 | RTC Alert0 Interrupt Mask<br>Value: 0 = Masked; 1 = Unmasked |

## Register 7: Hibernation Raw Interrupt Status (HIBRIS), offset 0x018

This register is the raw interrupt status for the Hibernation module interrupt sources.

Hibernation Raw Interrupt Status (HIBRIS)

Base 0x400F.C000
Offset 0x018
Type RO, reset 0x0000.0000



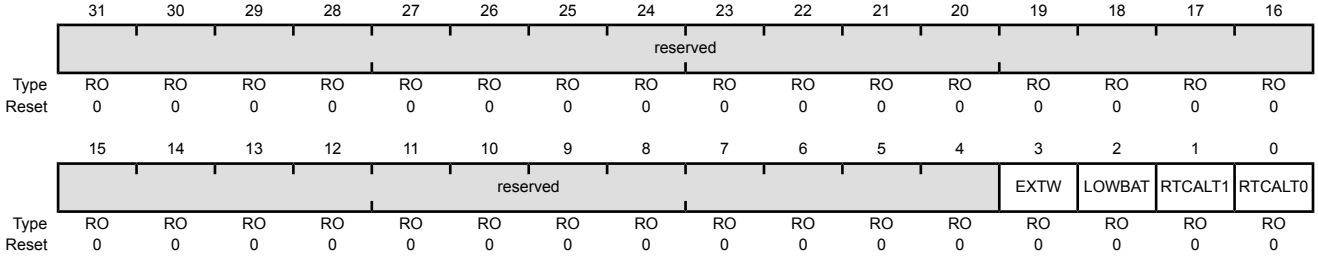

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x000.0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EXTW | RO | 0 | External Wake-Up Raw Interrupt Status |
| 2 | LOWBAT | RO | 0 | Low Battery Voltage Raw Interrupt Status |
| 1 | RTCALT1 | RO | 0 | RTC Alert1 Raw Interrupt Status |
| 0 | RTCALT0 | RO | 0 | RTC Alert0 Raw Interrupt Status |

## Register 8: Hibernation Masked Interrupt Status (HIBMIS), offset 0x01C

This register is the masked interrupt status for the Hibernation module interrupt sources.

Hibernation Masked Interrupt Status (HIBMIS)

Base 0x400F.C000
Offset 0x01C
Type RO, reset 0x0000.0000



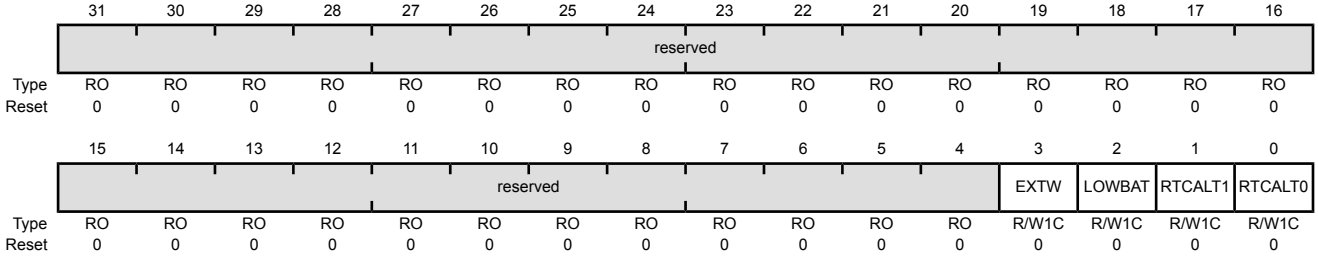

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x000.0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EXTW | RO | 0 | External Wake-Up Masked Interrupt Status |
| 2 | LOWBAT | RO | 0 | Low Battery Voltage Masked Interrupt Status |
| 1 | RTCALT1 | RO | 0 | RTC Alert1 Masked Interrupt Status |
| 0 | RTCALT0 | RO | 0 | RTC Alert0 Masked Interrupt Status |

## Register 9: Hibernation Interrupt Clear (HIBIC), offset 0x020

This register is the interrupt write-one-to-clear register for the Hibernation module interrupt sources.

Hibernation Interrupt Clear (HIBIC)

Base 0x400F.C000
Offset 0x020
Type R/W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x000.0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EXTW | R/W1C | 0 | External Wake-Up Masked Interrupt Clear<br>Reads return an indeterminate value. |
| 2 | LOWBAT | R/W1C | 0 | Low Battery Voltage Masked Interrupt Clear<br>Reads return an indeterminate value. |
| 1 | RTCALT1 | R/W1C | 0 | RTC Alert1 Masked Interrupt Clear<br>Reads return an indeterminate value. |
| 0 | RTCALT0 | R/W1C | 0 | RTC Alert0 Masked Interrupt Clear<br>Reads return an indeterminate value. |

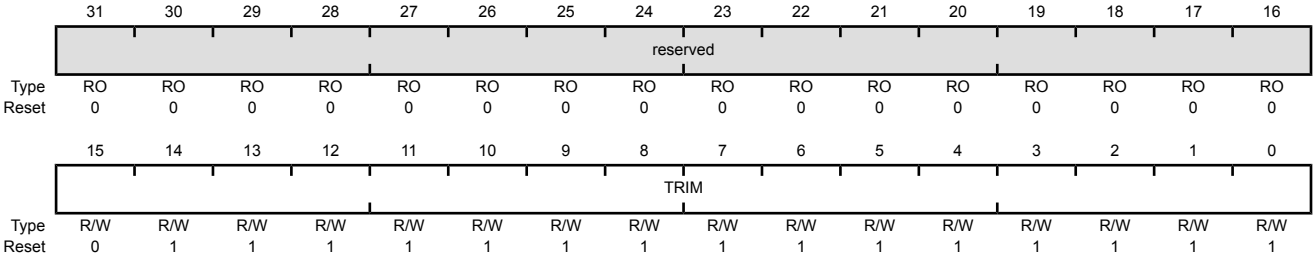
## Register 10: Hibernation RTC Trim (HIBRTCT), offset 0x024

This register contains the value that is used to trim the RTC clock predivider. It represents the computed underflow value that is used during the trim cycle. It is represented as 0x7FFF ± N clock cycles.

**Note:** **HIBRTCC**, **HIBRTCM0**, **HIBRTCM1**, **HIBRTCLD**, **HIBRTCT**, and **HIBDATA** are on the Hibernation module clock domain and and have special timing requirements. Software should make use of the WRC bit in the **HIBCTL** register to ensure that the required timing gap has elapsed. See “Register Access Timing” on page 240.

Hibernation RTC Trim (HIBRTCT)

Base 0x400F.C000
Offset 0x024
Type R/W, reset 0x0000.7FFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | TRIM | R/W | 0x7FFF | RTC Trim Value<br>This value is loaded into the RTC predivider every 64 seconds. It is used to adjust the RTC rate to account for drift and inaccuracy in the clock source. The compensation is made by software by adjusting the default value of 0x7FFF up or down. |

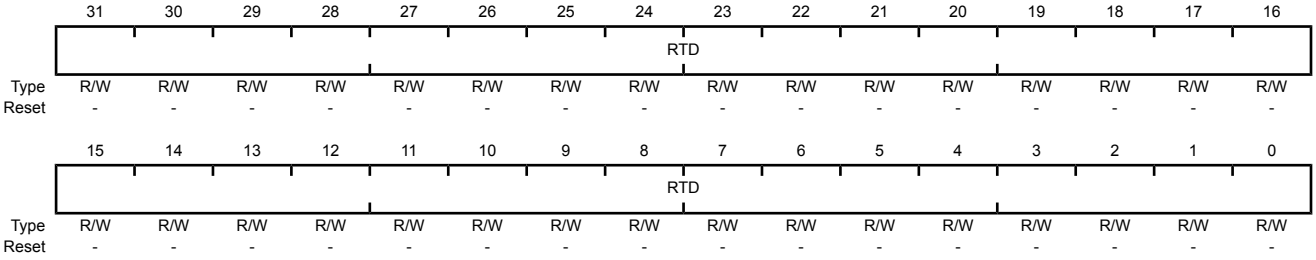
## Register 11: Hibernation Data (HIBDATA), offset 0x030-0x12C

This address space is implemented as a 64x32-bit memory (256 bytes). It can be loaded by the system processor in order to store state information and does not lose power during a power-cut operation as long as a battery is present.

**Note:** **HIBRTCC**, **HIBRTCM0**, **HIBRTCM1**, **HIBRTCLD**, **HIBRTCT**, and **HIBDATA** are on the Hibernation module clock domain and and have special timing requirements. Software should make use of the WRC bit in the **HIBCTL** register to ensure that the required timing gap has elapsed. See “Register Access Timing” on page 240.

Hibernation Data (HIBDATA)

Base 0x400F.C000
Offset 0x030-0x12C
Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | RTD | R/W | - | Hibernation Module NV Registers[63:0] |

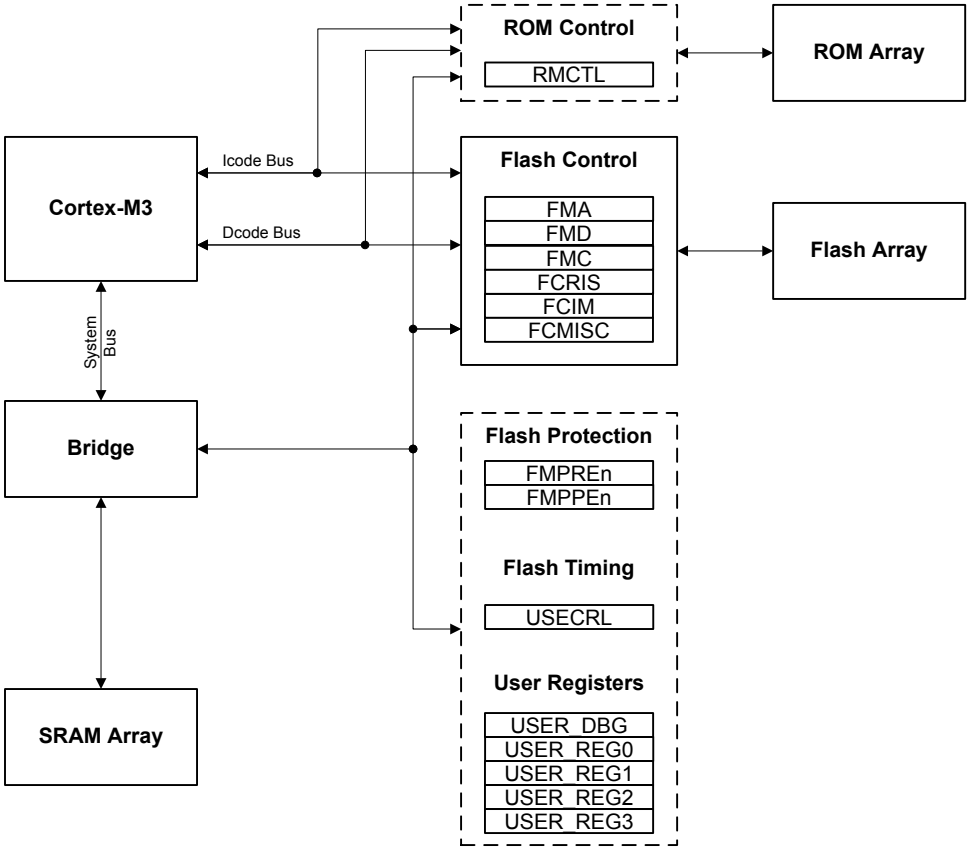
# 7 Internal Memory

The LM3S5632 microcontroller comes with 32 KB of bit-banded SRAM and 128 KB of flash memory. The flash controller provides a user-friendly interface, making flash programming a simple task. Flash protection can be applied to the flash memory on a 2-KB block basis.

## 7.1 Block Diagram

Figure 7-1 on page 260 illustrates the Flash functions. The dashed boxes in the figure indicate registers residing in the System Control module rather than the Flash Control module.

**Figure 7-1. Flash Block Diagram**




## 7.2 Functional Description

This section describes the functionality of the SRAM, ROM, and Flash memories.

### 7.2.1 SRAM Memory

**Note:** The SRAM memory is implemented using two 32-bit wide SRAM banks (separate SRAM arrays). The banks are partitioned so that one bank contains all even words (the even bank) and the other contains all odd words (the odd bank). A write access that is followed immediately by a read access to the same bank will incur a stall of a single clock cycle. However, a write to one bank followed by a read of the other bank can occur in successive clock cycles without incurring any delay.

The internal SRAM of the Stellaris® devices is located at address 0x2000.0000 of the device memory map. To reduce the number of time consuming read-modify-write (RMW) operations, ARM has introduced *bit-banding* technology in the Cortex-M3 processor. With a bit-band-enabled processor, certain regions in the memory map (SRAM and peripheral space) can use address aliases to access individual bits in a single, atomic operation.

The bit-band alias is calculated by using the formula:

```
bit-band alias = bit-band base + (byte offset * 32) + (bit number * 4)
```

For example, if bit 3 at address 0x2000.1000 is to be modified, the bit-band alias is calculated as:

```
0x2200.0000 + (0x1000 * 32) + (3 * 4) = 0x2202.000C
```

With the alias address calculated, an instruction performing a read/write to address 0x2202.000C allows direct access to only bit 3 of the byte at address 0x2000.1000.

For details about bit-banding, see “Bit-Banding” on page 76.

### 7.2.2 ROM Memory

The ROM of the Stellaris device is located at address 0x0100.0000 of the device memory map and contains the following components:

- Stellaris Boot Loader and vector table (see “Boot Loader” on page 781)
- Stellaris Peripheral Driver Library (DriverLib) release for product-specific peripherals and interfaces (see “ROM DriverLib Functions” on page 786)

### 7.2.3 Flash Memory

The flash is organized as a set of 1-KB blocks that can be individually erased. Erasing a block causes the entire contents of the block to be reset to all 1s. An individual 32-bit word can be programmed to change bits that are currently 1 to a 0. These blocks are paired into a set of 2-KB blocks that can be individually protected. The protection allows blocks to be marked as read-only or execute-only, providing different levels of code protection. Read-only blocks cannot be erased or programmed, protecting the contents of those blocks from being modified. Execute-only blocks cannot be erased or programmed, and can only be read by the controller instruction fetch mechanism, protecting the contents of those blocks from being read by either the controller or by a debugger.

#### 7.2.3.1 Flash Memory Timing

The timing for the flash is automatically handled by the flash controller. However, in order to do so, it must know the clock rate of the system in order to time its internal signals properly. The number of clock cycles per microsecond must be provided to the flash controller for it to accomplish this timing. It is software's responsibility to keep the flash controller updated with this information via the **USec Reload (USECRL)** register.

On reset, the **USECRL** register is loaded with a value that configures the flash timing so that it works with the maximum clock rate of the part. If software changes the system operating frequency, the new operating frequency minus 1 (in MHz) must be loaded into **USECRL** before any flash modifications are attempted. For example, if the device is operating at a speed of 20 MHz, a value of 0x13 (20-1) must be written to the **USECRL** register.

### 7.2.3.2 Flash Memory Protection

The user is provided two forms of flash protection per 2-KB flash blocks in two pairs of 32-bit wide registers. The protection policy for each form is controlled by individual bits (per policy per block) in the **FMPPEn** and **FMPREn** registers.

- **Flash Memory Protection Program Enable (FMPPEn)**: If set, the block may be programmed (written) or erased. If cleared, the block may not be changed.
- **Flash Memory Protection Read Enable (FMPREn)**: If a bit is set, the corresponding block may be executed or read by software or debuggers. If a bit is cleared, the corresponding block may only be executed, and contents of the memory block are prohibited from being read as data.

The policies may be combined as shown in Table 7-1 on page 262.

**Table 7-1. Flash Protection Policy Combinations**

| FMPPEn | FMPREn | Protection |
|---|---|---|
| 0 | 0 | Execute-only protection. The block may only be executed and may not be written or erased. This mode is used to protect code. |
| 1 | 0 | The block may be written, erased or executed, but not read. This combination is unlikely to be used. |
| 0 | 1 | Read-only protection. The block may be read or executed but may not be written or erased. This mode is used to lock the block from further modification while allowing any read or execute access. |
| 1 | 1 | No protection. The block may be written, erased, executed or read. |

A Flash memory access that attempts to read a read-protected block (**FMPREn** bit is set) is prohibited and generates a bus fault. A Flash memory access that attempts to program or erase a program-protected block (**FMPPEn** bit is set) is prohibited and can optionally generate an interrupt (by setting the AMASK bit in the **Flash Controller Interrupt Mask (FCIM)** register) to alert software developers of poorly behaving software during the development and debug phases.

The factory settings for the **FMPREn** and **FMPPEn** registers are a value of 1 for all implemented banks. These settings create a policy of open access and programmability. The register bits may be changed by clearing the specific register bit. The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. The changes are committed using the **Flash Memory Control (FMC)** register. Details on programming these bits are discussed in “Nonvolatile Register Programming” on page 264.

### 7.2.3.3 Execute-Only Protection

Execute-only protection prevents both modification and visibility to a protected flash block. This mode is intended to be used in situations where a device requires debug capability, yet portions of the application space must be protected from external access. An example of this is a company who wishes to sell Stellaris devices with their proprietary software pre-programmed, yet allow the end user to add custom code to an unprotected region of the flash (such as a motor control module with a customizable motor configuration section in flash).

Literal data introduces a complication to the protection mechanism. When C code is compiled and linked, literal data (constants, and so on) is typically placed in the text section, between functions, by the compiler. The literal data is accessed at run time through the use of the LDR instruction, which loads the data from memory using a PC-relative memory address. The execution of the LDR instruction generates a read transaction across the Cortex-M3's DCode bus, which is subject to the

execute-only protection mechanism. If the accessed block is marked as execute only, the transaction is blocked, and the processor is prevented from loading the constant data and, therefore, inhibiting correct execution. Therefore, using execute-only protection requires that literal data be handled differently. There are three ways to address this:

1. Use a compiler that allows literal data to be collected into a separate section that is put into one or more read-enabled flash blocks. Note that the LDR instruction may use a PC-relative address—in which case the literal pool cannot be located outside the span of the offset—or the software may reserve a register to point to the base address of the literal pool and the LDR offset is relative to the beginning of the pool.
2. Use a compiler that generates literal data from arithmetic instruction immediate data and subsequent computation.
3. Use method 1 or 2, but in assembly language, if the compiler does not support either method.

#### 7.2.3.4 Read-Only Protection

Read-only protection prevents the contents of the flash block from being re-programmed, while still allowing the content to be read by processor or the debug interface. Note that if a **FMPREn** bit is cleared, all read accesses to the Flash memory block are disallowed, including any data accesses. Care must be taken not to store required data in a Flash memory block that has the associated **FMPREn** bit cleared.

The read-only mode does not prevent read access to the stored program, but it does provide protection against accidental (or malicious) erasure or programming. Read-only is especially useful for utilities like the boot loader when the debug interface is permanently disabled. In such combinations, the boot loader, which provides access control to the Flash memory, is protected from being erased or modified.

#### 7.2.3.5 Permanently Disabling Debug

For extremely sensitive applications, the debug interface to the processor and peripherals can be permanently disabled, blocking all accesses to the device through the JTAG or SWD interfaces. With the debug interface disabled, it is still possible to perform standard IEEE instructions (such as boundary scan operations), but access to the processor and peripherals is blocked.

The DBG0 and DBG1 bits of the **User Debug (USER_DBG)** register control whether the debug interface is turned on or off.

The debug interface should not be permanently disabled without providing some mechanism—such as the boot loader—to provide customer-installable updates or bug fixes. Disabling the debug interface is permanent and cannot be reversed.

#### 7.2.3.6 Interrupts

The Flash memory controller can generate interrupts when the following conditions are observed:

- Programming Interrupt - signals when a program or erase action is complete.
- Access Interrupt - signals when a program or erase action has been attempted on a 2-kB block of memory that is protected by its corresponding **FMPPEn** bit.

The interrupt events that can trigger a controller-level interrupt are defined in the **Flash Controller Masked Interrupt Status (FCMIS)** register (see page 273) by setting the corresponding MASK bits. If interrupts are not used, the raw interrupt status is always visible via the **Flash Controller Raw Interrupt Status (FCRIS)** register (see page 272).

Interrupts are always cleared (for both the **FCMIS** and **FCRIS** registers) by writing a 1 to the corresponding bit in the **Flash Controller Masked Interrupt Status and Clear (FCMISC)** register (see page 274).

# 7.3 Flash Memory Initialization and Configuration

## 7.3.1 Flash Programming

The Stellaris devices provide a user-friendly interface for flash programming. All erase/program operations are handled via three registers: **FMA**, **FMD**, and **FMC**.

During a Flash memory operation (write, page erase, or mass erase) access to the Flash memory is inhibited. As a result, instruction and literal fetches are held off until the Flash memory operation is complete. If instruction execution is required during a Flash memory operation, the code that is executing must be placed in SRAM and executed from there while the flash operation is in progress.

### 7.3.1.1 To program a 32-bit word

1. Write source data to the **FMD** register.
2. Write the target address to the **FMA** register.
3. Write the flash write key and the WRITE bit (a value of 0xA442.0001) to the **FMC** register.
4. Poll the **FMC** register until the WRITE bit is cleared.

### 7.3.1.2 To perform an erase of a 1-KB page

1. Write the page address to the **FMA** register.
2. Write the flash write key and the ERASE bit (a value of 0xA442.0002) to the **FMC** register.
3. Poll the **FMC** register until the ERASE bit is cleared.

### 7.3.1.3 To perform a mass erase of the flash

1. Write the flash write key and the MERASE bit (a value of 0xA442.0004) to the **FMC** register.
2. Poll the **FMC** register until the MERASE bit is cleared.

## 7.3.2 Nonvolatile Register Programming

**Note:** The **USER_DBG** register requires a POR before the committed changes take effect.

This section discusses how to update registers that are resident within the Flash memory itself. These registers exist in a separate space from the main Flash memory array and are not affected by an ERASE or MASS ERASE operation. The bits in these registers can be changed from 1 to 0 with a write operation. Prior to being committed, the register contents are unaffected by any reset condition except power-on reset, which returns the register contents to the original value. By committing the register values using the COMT bit in the **FMC** register, the register contents become nonvolatile and are therefore retained following power cycling. Once the register contents are committed, the only way to restore the factory default values is to perform the sequence described in the section called “Recovering a "Locked" Device” on page 164.

With the exception of the **USER_DBG** register, the settings in these registers can be tested before committing them to Flash memory. For the **USER_DBG** register, the data to be written is loaded

into the **FMD** register before it is committed. The **FMD** register is read only and does not allow the **USER_DBG** operation to be tried before committing it to nonvolatile memory.

**Important:** The Flash memory resident registers can only have bits changed from 1 to 0 by user programming and can only be committed once. After being committed, these registers can only be restored to their factory default values only by performing the sequence described in the section called “Recovering a "Locked" Device” on page 164. The mass erase of the main Flash memory array caused by the sequence is performed prior to restoring these registers.

In addition, the **USER_REG0**, **USER_REG1**, **USER_REG2**, **USER_REG3**, and **USER_DBG** registers each use bit 31 (NW) to indicate that they have not been committed and bits in the register may be changed from 1 to 0. Table 7-2 on page 265 provides the **FMA** address required for commitment of each of the registers and the source of the data to be written when the **FMC** register is written with a value of 0xA442.0008. After writing the COMT bit, the user may poll the **FMC** register to wait for the commit operation to complete.

**Table 7-2. User-Programmable Flash Memory Resident Registers**

| Register to be Committed | FMA Value | Data Source |
|---|---|---|
| FMPRE0 | 0x0000.0000 | FMPRE0 |
| FMPRE1 | 0x0000.0002 | FMPRE1 |
| FMPPE0 | 0x0000.0001 | FMPPE0 |
| FMPPE1 | 0x0000.0003 | FMPPE1 |
| USER_REG0 | 0x8000.0000 | USER_REG0 |
| USER_REG1 | 0x8000.0001 | USER_REG1 |
| USER_REG2 | 0x8000.0002 | USER_REG2 |
| USER_REG3 | 0x8000.0003 | USER_REG3 |
| USER_DBG | 0x7510.0000 | FMD |

## 7.4 Register Map

Table 7-3 on page 265 lists the ROM Controller register and the Flash memory and control registers. The offset listed is a hexadecimal increment to the register's address. The **FMA**, **FMD**, **FMC**, **FCRIS**, **FCIM**, and **FCMISC** register offsets are relative to the Flash memory control base address of 0x400F.D000. The ROM and Flash memory protection register offsets are relative to the System Control base address of 0x400F.E000.

**Table 7-3. Flash Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| **ROM Registers (System Control Offset)** | | | | | |
| 0x0F0 | RMCTL | R/W1C | - | ROM Control | 267 |
| **Flash Memory Control Registers (Flash Control Offset)** | | | | | |
| 0x000 | FMA | R/W | 0x0000.0000 | Flash Memory Address | 268 |
| 0x004 | FMD | R/W | 0x0000.0000 | Flash Memory Data | 269 |
| 0x008 | FMC | R/W | 0x0000.0000 | Flash Memory Control | 270 |

**Table 7-3. Flash Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x00C | FCRIS | RO | 0x0000.0000 | Flash Controller Raw Interrupt Status | 272 |
| 0x010 | FCIM | R/W | 0x0000.0000 | Flash Controller Interrupt Mask | 273 |
| 0x014 | FCMISC | R/W1C | 0x0000.0000 | Flash Controller Masked Interrupt Status and Clear | 274 |
| **Flash Memory Protection Registers (System Control Offset)** | | | | | |
| 0x130 | FMPRE0 | R/W | 0xFFFF.FFFF | Flash Memory Protection Read Enable 0 | 277 |
| 0x200 | FMPRE0 | R/W | 0xFFFF.FFFF | Flash Memory Protection Read Enable 0 | 277 |
| 0x134 | FMPPE0 | R/W | 0xFFFF.FFFF | Flash Memory Protection Program Enable 0 | 278 |
| 0x400 | FMPPE0 | R/W | 0xFFFF.FFFF | Flash Memory Protection Program Enable 0 | 278 |
| 0x140 | USECRL | R/W | 0x31 | USec Reload | 276 |
| 0x1D0 | USER_DBG | R/W | 0xFFFF.FFFE | User Debug | 279 |
| 0x1E0 | USER_REG0 | R/W | 0xFFFF.FFFF | User Register 0 | 280 |
| 0x1E4 | USER_REG1 | R/W | 0xFFFF.FFFF | User Register 1 | 281 |
| 0x1E8 | USER_REG2 | R/W | 0xFFFF.FFFF | User Register 2 | 282 |
| 0x1EC | USER_REG3 | R/W | 0xFFFF.FFFF | User Register 3 | 283 |
| 0x204 | FMPRE1 | R/W | 0xFFFF.FFFF | Flash Memory Protection Read Enable 1 | 284 |
| 0x208 | FMPRE2 | R/W | 0x0000.0000 | Flash Memory Protection Read Enable 2 | 285 |
| 0x20C | FMPRE3 | R/W | 0x0000.0000 | Flash Memory Protection Read Enable 3 | 286 |
| 0x404 | FMPPE1 | R/W | 0xFFFF.FFFF | Flash Memory Protection Program Enable 1 | 287 |
| 0x408 | FMPPE2 | R/W | 0x0000.0000 | Flash Memory Protection Program Enable 2 | 288 |
| 0x40C | FMPPE3 | R/W | 0x0000.0000 | Flash Memory Protection Program Enable 3 | 289 |

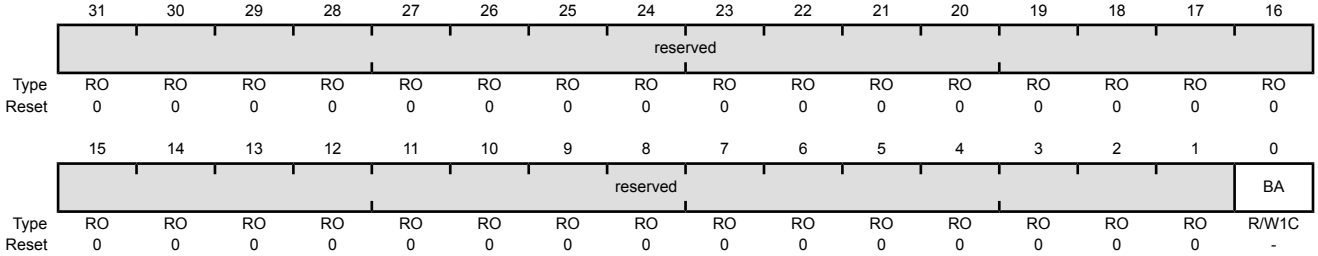
## 7.5 ROM Register Descriptions (System Control Offset)

This section lists and describes the ROM Controller registers, in numerical order by address offset. Registers in this section are relative to the System Control base address of 0x400F.E000.

### Register 1: ROM Control (RMCTL), offset 0x0F0

This register provides control of the ROM controller state.

ROM Control (RMCTL)
Base 0x400F.E000
Offset 0x0F0
Type R/W1C, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | BA | R/W1C | - | Boot Alias<br>■ The device has ROM.<br>■ The first two words of the Flash memory contain 0xFFFF.FFFF.<br>This bit is cleared by writing a 1 to this bit position.<br>When the BA bit is set, the boot alias is in effect and the ROM appears at address 0x0. When the BA bit is clear, the Flash appears at address 0x0. |

## 7.6 Flash Register Descriptions (Flash Control Offset)

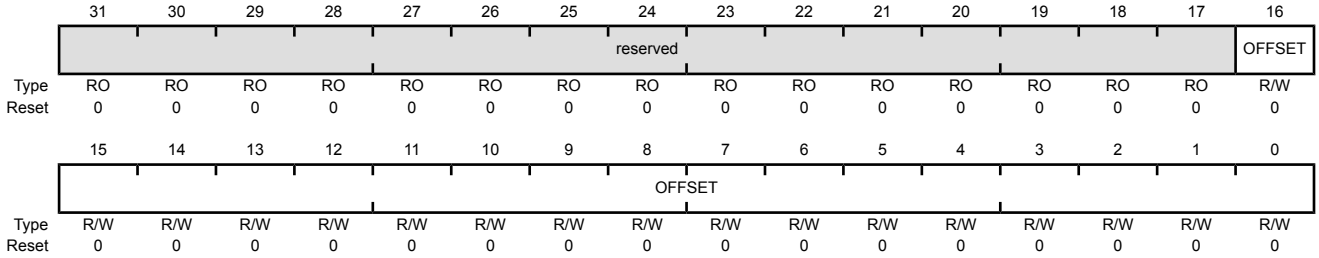
This section lists and describes the Flash Memory registers, in numerical order by address offset. Registers in this section are relative to the Flash control base address of 0x400F.D000.

## Register 2: Flash Memory Address (FMA), offset 0x000

During a write operation, this register contains a 4-byte-aligned address and specifies where the data is written. During erase operations, this register contains a 1 KB-aligned address and specifies which page is erased. Note that the alignment requirements must be met by software or the results of the operation are unpredictable.

Flash Memory Address (FMA)

Base 0x400F.D000
Offset 0x000
Type R/W, reset 0x0000.0000



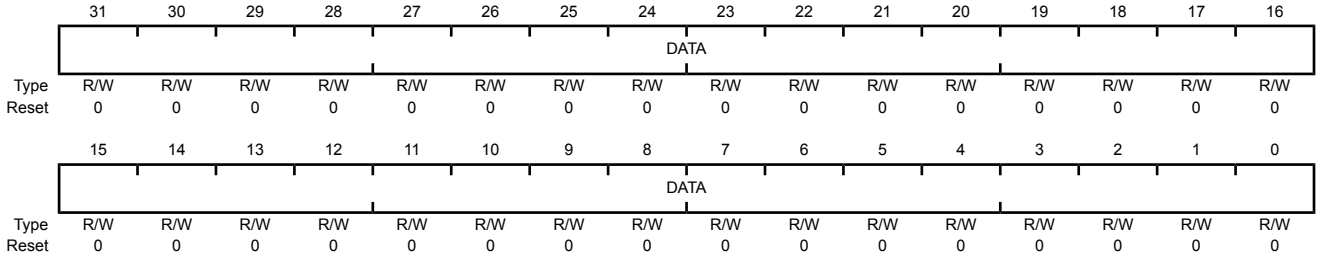

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16:0 | OFFSET | R/W | 0x0 | Address Offset<br>Address offset in flash where operation is performed, except for nonvolatile registers (see “Nonvolatile Register Programming” on page 264 for details on values for this field). |

## Register 3: Flash Memory Data (FMD), offset 0x004

This register contains the data to be written during the programming cycle or read during the read cycle. Note that the contents of this register are undefined for a read access of an execute-only block. This register is not used during the erase cycles.

Flash Memory Data (FMD)

Base 0x400F.D000
Offset 0x004
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R/W | 0x0 | Data Value<br>Data value for write operation. |

## Register 4: Flash Memory Control (FMC), offset 0x008

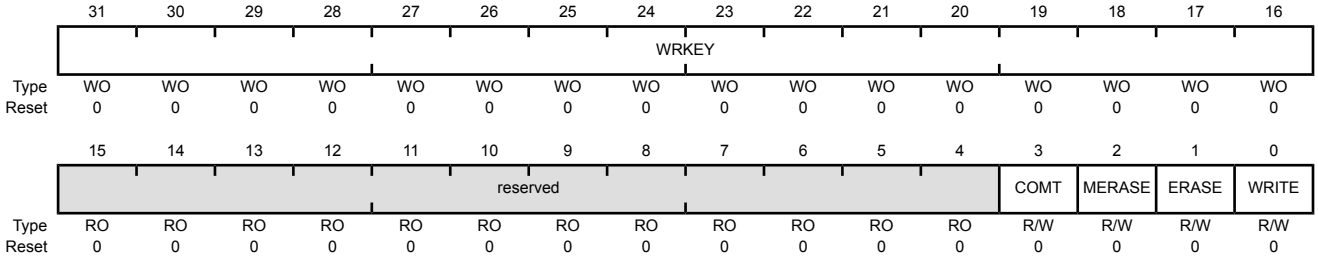
When this register is written, the flash controller initiates the appropriate access cycle for the location specified by the **Flash Memory Address (FMA)** register (see page 268). If the access is a write access, the data contained in the **Flash Memory Data (FMD)** register (see page 269) is written.

This is the final register written and initiates the memory operation. There are four control bits in the lower byte of this register that, when set, initiate the memory operation. The most used of these register bits are the ERASE and WRITE bits.

It is a programming error to write multiple control bits and the results of such an operation are unpredictable.

Flash Memory Control (FMC)

Base 0x400F.D000
Offset 0x008
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | WRKEY | WO | 0x0 | Flash Write Key<br>This field contains a write key, which is used to minimize the incidence of accidental flash writes. The value 0xA442 must be written into this field for a write to occur. Writes to the **FMC** register without this WRKEY value are ignored. A read of this field returns the value 0. |
| 15:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | COMT | R/W | 0 | Commit Register Value<br>Commit (write) of register value to nonvolatile storage. A write of 0 has no effect on the state of this bit.<br>If read, the state of the previous commit access is provided. If the previous commit access is complete, a 0 is returned; otherwise, if the commit access is not complete, a 1 is returned.<br>This can take up to 50 µs. |
| 2 | MERASE | R/W | 0 | Mass Erase Flash Memory<br>If this bit is set, the flash main memory of the device is all erased. A write of 0 has no effect on the state of this bit.<br>If read, the state of the previous mass erase access is provided. If the previous mass erase access is complete, a 0 is returned; otherwise, if the previous mass erase access is not complete, a 1 is returned.<br>This can take up to 250 ms. |

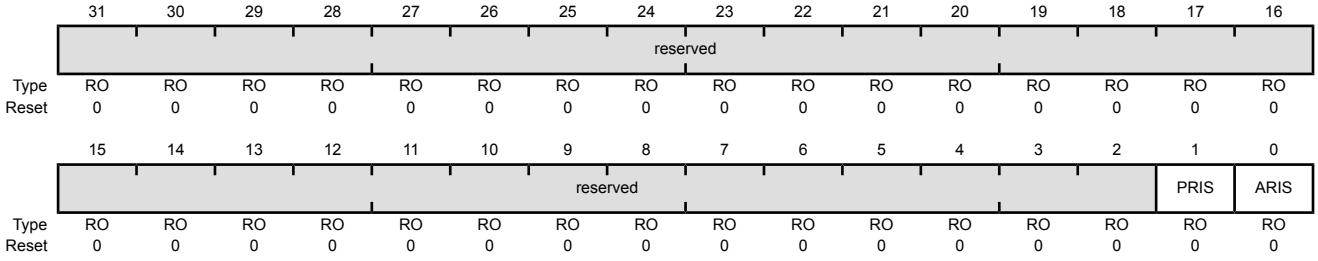
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | ERASE | R/W | 0 | Erase a Page of Flash Memory<br>If this bit is set, the page of flash main memory as specified by the contents of **FMA** is erased. A write of 0 has no effect on the state of this bit.<br>If read, the state of the previous erase access is provided. If the previous erase access is complete, a 0 is returned; otherwise, if the previous erase access is not complete, a 1 is returned.<br>This can take up to 25 ms. |
| 0 | WRITE | R/W | 0 | Write a Word into Flash Memory<br>If this bit is set, the data stored in **FMD** is written into the location as specified by the contents of **FMA**. A write of 0 has no effect on the state of this bit.<br>If read, the state of the previous write update is provided. If the previous write access is complete, a 0 is returned; otherwise, if the write access is not complete, a 1 is returned.<br>This can take up to 50 µs. |

## Register 5: Flash Controller Raw Interrupt Status (FCRIS), offset 0x00C

This register indicates that the flash controller has an interrupt condition. An interrupt is only signaled if the corresponding **FCIM** register bit is set.

Flash Controller Raw Interrupt Status (FCRIS)

Base 0x400F.D000
Offset 0x00C
Type RO, reset 0x0000.0000



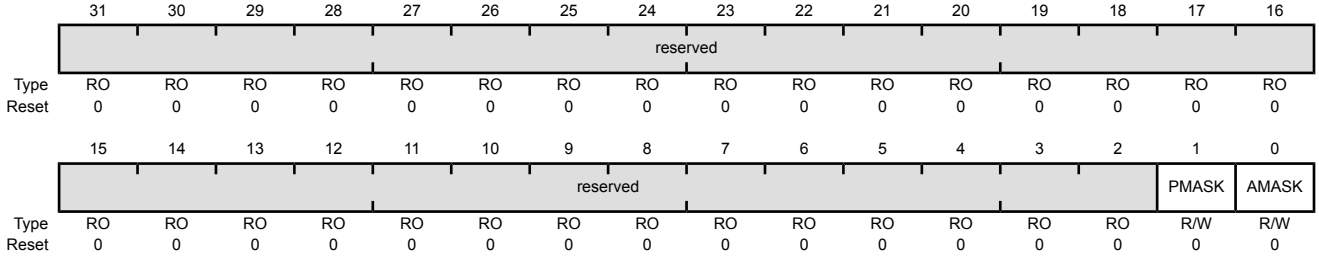

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | PRIS | RO | 0 | Programming Raw Interrupt Status<br>This bit provides status on programming cycles which are write or erase actions generated through the **FMC** register bits (see page 270).<br>Value Description<br>1 The programming cycle has completed.<br>0 The programming cycle has not completed.<br>This status is sent to the interrupt controller when the PMASK bit in the **FCIM** register is set.<br>This bit is cleared by writing a 1 to the PMISC bit in the **FCMISC** register. |
| 0 | ARIS | RO | 0 | Access Raw Interrupt Status<br>Value Description<br>1 A program or erase action was attempted on a block of Flash memory that contradicts the protection policy for that block as set in the **FMPPEn** registers.<br>0 No access has tried to improperly program or erase the Flash memory.<br>This status is sent to the interrupt controller when the AMASK bit in the **FCIM** register is set.<br>This bit is cleared by writing a 1 to the AMISC bit in the **FCMISC** register. |

## Register 6: Flash Controller Interrupt Mask (FCIM), offset 0x010

This register controls whether the flash controller generates interrupts to the controller.

Flash Controller Interrupt Mask (FCIM)

Base 0x400F.D000
Offset 0x010
Type R/W, reset 0x0000.0000



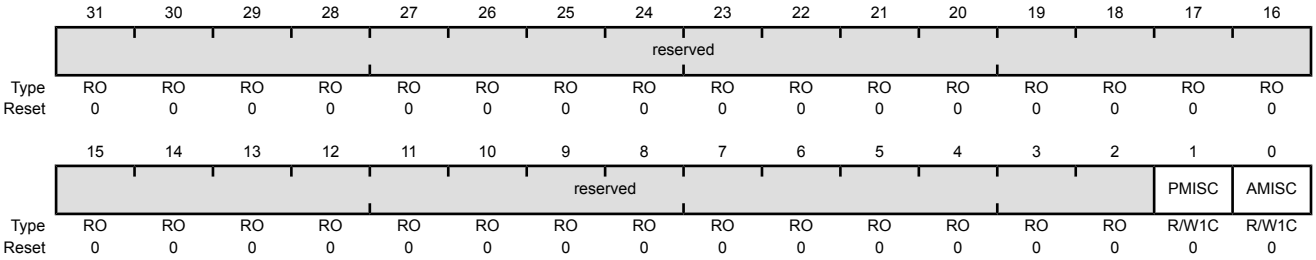

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | PMASK | R/W | 0 | Programming Interrupt Mask<br>This bit controls the reporting of the programming raw interrupt status to the interrupt controller.<br><br>Value Description<br>1 An interrupt is sent to the interrupt controller when the PRIS bit is set.<br>0 The PRIS interrupt is suppressed and not sent to the interrupt controller. |
| 0 | AMASK | R/W | 0 | Access Interrupt Mask<br>This bit controls the reporting of the access raw interrupt status to the interrupt controller.<br><br>Value Description<br>1 An interrupt is sent to the interrupt controller when the ARIS bit is set.<br>0 The ARIS interrupt is suppressed and not sent to the interrupt controller. |

## Register 7: Flash Controller Masked Interrupt Status and Clear (FCMISC), offset 0x014

This register provides two functions. First, it reports the cause of an interrupt by indicating which interrupt source or sources are signalling the interrupt. Second, it serves as the method to clear the interrupt reporting.

Flash Controller Masked Interrupt Status and Clear (FCMISC)

Base 0x400F.D000
Offset 0x014
Type R/W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | PMISC | R/W1C | 0 | Programming Masked Interrupt Status and Clear<br><br>Value Description<br>1 When read, a 1 indicates that an unmasked interrupt was signaled because a programming cycle completed.<br>Writing a 1 to this bit clears `PMISC` and also the `PRIS` bit in the **FCRIS** register (see page 272).<br>0 When read, a 0 indicates that a programming cycle complete interrupt has not occurred.<br>A write of 0 has no effect on the state of this bit. |
| 0 | AMISC | R/W1C | 0 | Access Masked Interrupt Status and Clear<br><br>Value Description<br>1 When read, a 1 indicates that an unmasked interrupt was signaled because a program or erase action was attempted on a block of Flash memory that contradicts the protection policy for that block as set in the **FMPPEn** registers.<br>Writing a 1 to this bit clears `AMISC` and also the `ARIS` bit in the **FCRIS** register (see page 272).<br>0 When read, a 0 indicates that no improper accesses have occurred.<br>A write of 0 has no effect on the state of this bit. |

## 7.7 Flash Register Descriptions (System Control Offset)

The remainder of this section lists and describes the Flash Memory registers, in numerical order by address offset. Registers in this section are relative to the System Control base address of 0x400F.E000.

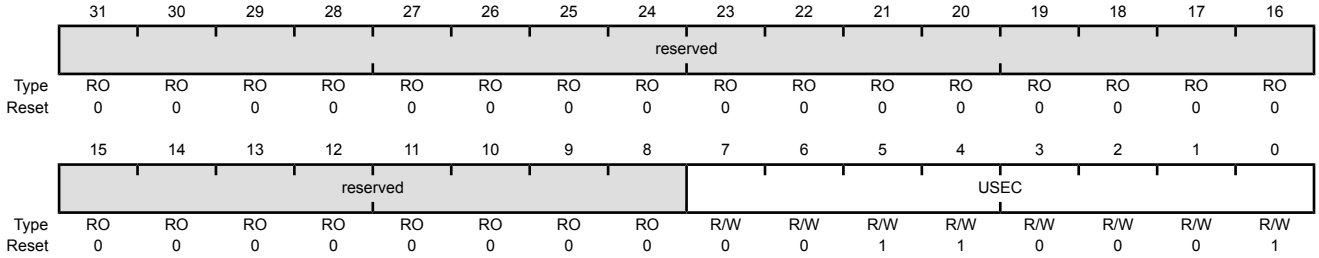
## Register 8: USec Reload (USECRL), offset 0x140

**Note:** Offset is relative to System Control base address of 0x400F.E000

This register is provided as a means of creating a 1-µs tick divider reload value for the flash controller. The internal flash has specific minimum and maximum requirements on the length of time the high voltage write pulse can be applied. It is required that this register contain the operating frequency (in MHz -1) whenever the flash is being erased or programmed. The user is required to change this value if the clocking conditions are changed for a flash erase/program operation.

USec Reload (USECRL)

Base 0x400F.E000
Offset 0x140
Type R/W, reset 0x31




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | USEC | R/W | 0x31 | Microsecond Reload Value<br>MHz -1 of the controller clock when the flash is being erased or programmed.<br>If the maximum system frequency is being used, USEC should be set to 0x31 (50 MHz) whenever the flash is being erased or programmed. |

## Register 9: Flash Memory Protection Read Enable 0 (FMPRE0), offset 0x130 and 0x200

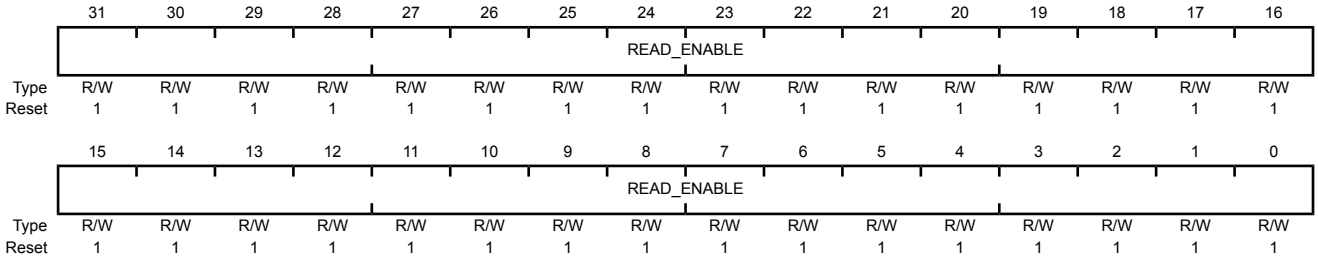
**Note:** This register is aliased for backwards compatability.

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register stores the read-only protection bits for each 2-KB flash block (**FMPPEn** stores the execute-only bits). Flash memory up to a total of 64 KB is controlled by this register. Other **FMPREn** registers (if any) provide protection for other 64K blocks. This register is loaded during the power-on reset sequence. The factory settings for the **FMPREn** and **FMPPEn** registers are a value of 1 for all implemented banks. This achieves a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection bit from a 1 to a 0 (and may NOT change a 0 to a 1). The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. The reset value shown only applies to power-on reset; any other type of reset does not affect this register. Once committed, the only way to restore the factory default value of this register is to perform the "Recover Locked Device" sequence detailed in the JTAG chapter. For additional information, see the "Flash Memory Protection" section.

Flash Memory Protection Read Enable 0 (FMPRE0)

Base 0x400F.E000
Offset 0x130 and 0x200
Type R/W, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | READ_ENABLE | R/W | 0xFFFFFFFF | Flash Read Enable<br>Configures 2-KB flash blocks to be read only. The policies may be combined as shown in the table “Flash Protection Policy Combinations”.<br><br>Value Description<br>0xFFFFFFFF Bits [31:0] each enable protection on a 2-KB block of Flash memory up to the total of 64 KB. |

## Register 10: Flash Memory Protection Program Enable 0 (FMPPE0), offset 0x134 and 0x400

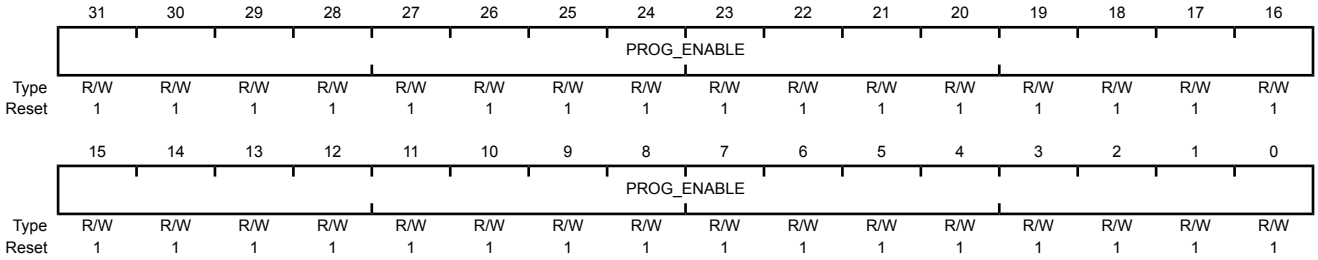
**Note:** This register is aliased for backwards compatability.

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register stores the execute-only protection bits for each 2-KB flash block (**FMPREn** stores the execute-only bits). Flash memory up to a total of 64 KB is controlled by this register. Other **FMPPEn** registers (if any) provide protection for other 64K blocks. This register is loaded during the power-on reset sequence. The factory settings for the **FMPREn** and **FMPPEn** registers are a value of 1 for all implemented banks. This achieves a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection bit from a 1 to a 0 (and may NOT change a 0 to a 1). The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. The reset value shown only applies to power-on reset; any other type of reset does not affect this register. Once committed, the only way to restore the factory default value of this register is to perform the "Recover Locked Device" sequence detailed in the JTAG chapter. For additional information, see the "Flash Memory Protection" section.

Flash Memory Protection Program Enable 0 (FMPPE0)

Base 0x400F.E000
Offset 0x134 and 0x400
Type R/W, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | PROG_ENABLE | R/W | 0xFFFFFFFF | Flash Programming Enable<br>Configures 2-KB flash blocks to be execute only. The policies may be combined as shown in the table “Flash Protection Policy Combinations”.<br><br>Value Description<br>0xFFFFFFFF Bits [31:0] each enable protection on a 2-KB block of Flash memory up to the total of 64 KB. |

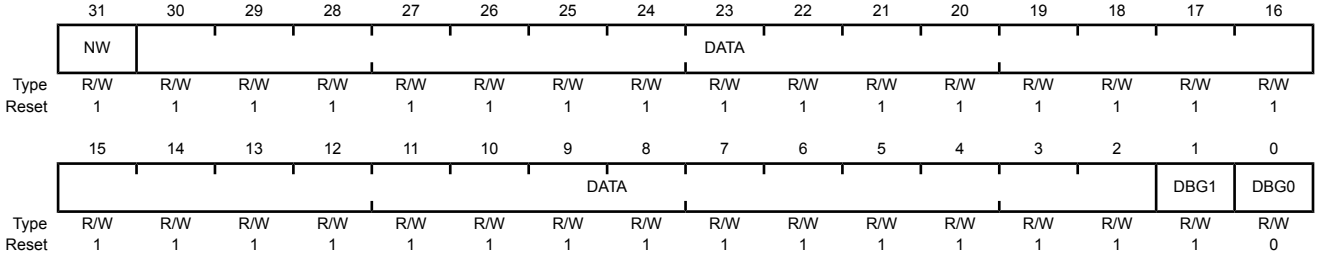
## Register 11: User Debug (USER_DBG), offset 0x1D0

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register provides a write-once mechanism to disable external debugger access to the device in addition to 27 additional bits of user-defined data. The DBG0 bit (bit 0) is set to 0 from the factory and the DBG1 bit (bit 1) is set to 1, which enables external debuggers. Changing the DBG1 bit to 0 disables any external debugger access to the device permanently, starting with the next power-up cycle of the device. The NW bit (bit 31) indicates that the register has not yet been committed and is controlled through hardware to ensure that the register is only committed once. Prior to being committed, bits can only be changed from 1 to 0. The reset value shown only applies to power-on reset; any other type of reset does not affect this register. Once commited, the value of this register can never be restored to the factory default value.

User Debug (USER_DBG)

Base 0x400F.E000
Offset 0x1D0
Type R/W, reset 0xFFFF.FFFE




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | NW | R/W | 1 | User Debug Not Written<br>When set, this bit indicates that this 32-bit register has not been committed. When clear, this bit specifies that this register has been committed and may not be committed again. |
| 30:2 | DATA | R/W | 0x1FFFFFFF | User Data<br>Contains the user data value. This field is initialized to all 1s and can only be committed once. |
| 1 | DBG1 | R/W | 1 | Debug Control 1<br>The DBG1 bit must be 1 and DBG0 must be 0 for debug to be available. |
| 0 | DBG0 | R/W | 0 | Debug Control 0<br>The DBG1 bit must be 1 and DBG0 must be 0 for debug to be available. |

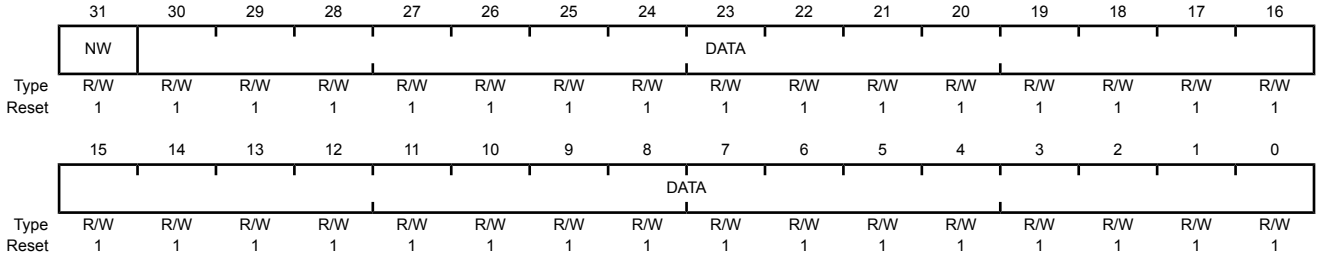
## Register 12: User Register 0 (USER_REG0), offset 0x1E0

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register provides 31 bits of user-defined data that is non-volatile and can only be written once. Bit 31 indicates that the register is available to be written and is controlled through hardware to ensure that the register is only written once. The write-once characteristics of this register are useful for keeping static information like communication addresses that need to be unique per part and would otherwise require an external EEPROM or other non-volatile device. Once commited, the value of this register can never be restored to the factory default value.

User Register 0 (USER_REG0)

Base 0x400F.E000
Offset 0x1E0
Type R/W, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | NW | R/W | 1 | Not Written<br>When set, this bit indicates that this 32-bit register has not been committed. When clear, this bit specifies that this register has been committed and may not be committed again. |
| 30:0 | DATA | R/W | 0x7FFFFFFF | User Data<br>Contains the user data value. This field is initialized to all 1s and can only be committed once. |

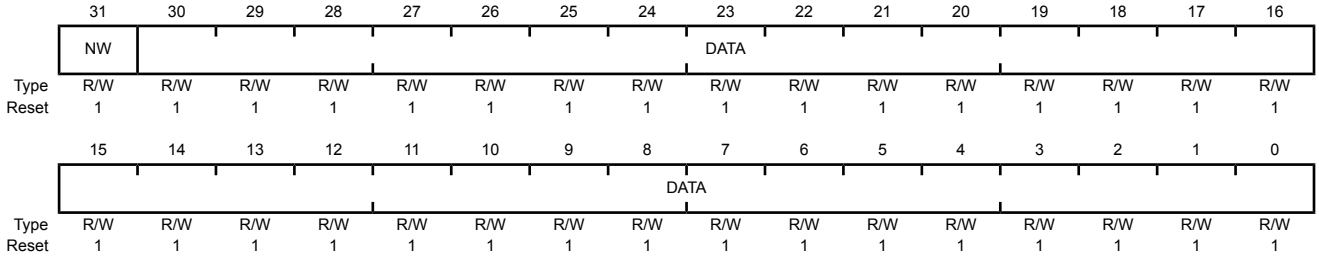
## Register 13: User Register 1 (USER_REG1), offset 0x1E4

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register provides 31 bits of user-defined data that is non-volatile and can only be written once. Bit 31 indicates that the register is available to be written and is controlled through hardware to ensure that the register is only written once. The write-once characteristics of this register are useful for keeping static information like communication addresses that need to be unique per part and would otherwise require an external EEPROM or other non-volatile device. Once commited, the value of this register can never be restored to the factory default value.

User Register 1 (USER_REG1)

Base 0x400F.E000
Offset 0x1E4
Type R/W, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | NW | R/W | 1 | Not Written<br>When set, this bit indicates that this 32-bit register has not been committed. When clear, this bit specifies that this register has been committed and may not be committed again. |
| 30:0 | DATA | R/W | 0x7FFFFFFF | User Data<br>Contains the user data value. This field is initialized to all 1s and can only be committed once. |

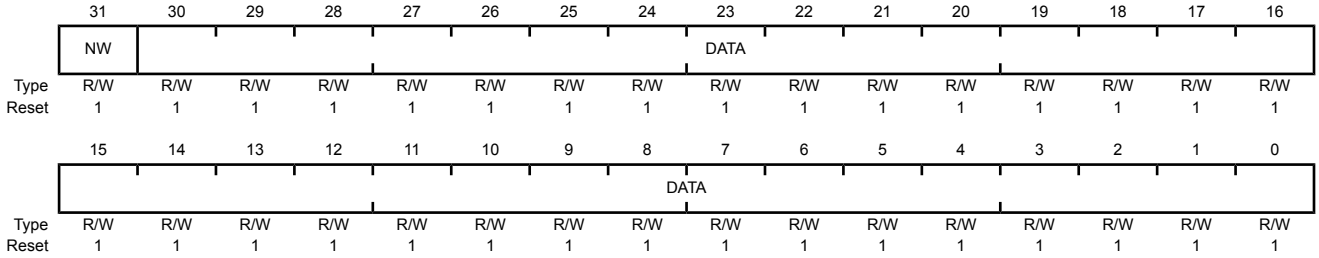
## Register 14: User Register 2 (USER_REG2), offset 0x1E8

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register provides 31 bits of user-defined data that is non-volatile and can only be written once. Bit 31 indicates that the register is available to be written and is controlled through hardware to ensure that the register is only written once. The write-once characteristics of this register are useful for keeping static information like communication addresses that need to be unique per part and would otherwise require an external EEPROM or other non-volatile device. Once commited, the value of this register can never be restored to the factory default value.

User Register 2 (USER_REG2)

Base 0x400F.E000
Offset 0x1E8
Type R/W, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | NW | R/W | 1 | Not Written<br>When set, this bit indicates that this 32-bit register has not been committed. When clear, this bit specifies that this register has been committed and may not be committed again. |
| 30:0 | DATA | R/W | 0x7FFFFFFF | User Data<br>Contains the user data value. This field is initialized to all 1s and can only be committed once. |

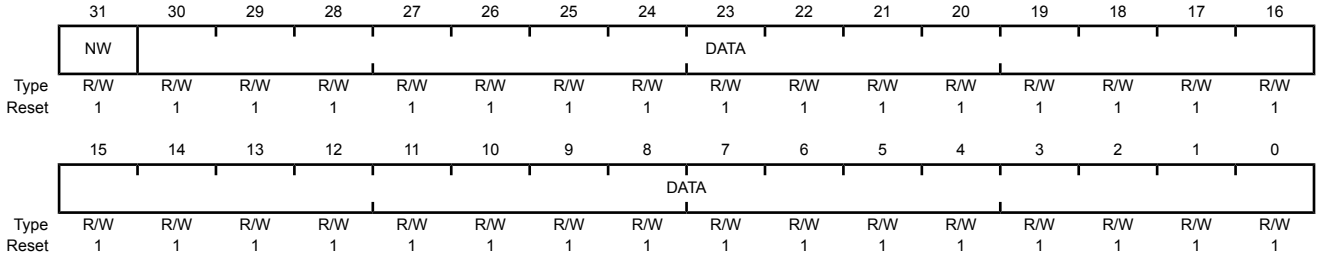
## Register 15: User Register 3 (USER_REG3), offset 0x1EC

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register provides 31 bits of user-defined data that is non-volatile and can only be written once. Bit 31 indicates that the register is available to be written and is controlled through hardware to ensure that the register is only written once. The write-once characteristics of this register are useful for keeping static information like communication addresses that need to be unique per part and would otherwise require an external EEPROM or other non-volatile device. Once commited, the value of this register can never be restored to the factory default value.

User Register 3 (USER_REG3)

Base 0x400F.E000
Offset 0x1EC
Type R/W, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | NW | R/W | 1 | Not Written<br>When set, this bit indicates that this 32-bit register has not been committed. When clear, this bit specifies that this register has been committed and may not be committed again. |
| 30:0 | DATA | R/W | 0x7FFFFFFF | User Data<br>Contains the user data value. This field is initialized to all 1s and can only be committed once. |

## Register 16: Flash Memory Protection Read Enable 1 (FMPRE1), offset 0x204

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register stores the read-only protection bits for each 2-KB flash block (**FMPPEn** stores the execute-only bits). Flash memory up to a total of 64 KB is controlled by this register. Other **FMPREn** registers (if any) provide protection for other 64K blocks. This register is loaded during the power-on reset sequence. The factory settings for the **FMPREn** and **FMPPEn** registers are a value of 1 for all implemented banks. This achieves a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection bit from a 1 to a 0 (and may NOT change a 0 to a 1). The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. The reset value shown only applies to power-on reset; any other type of reset does not affect this register. Once committed, the only way to restore the factory default value of this register is to perform the "Recover Locked Device" sequence detailed in the JTAG chapter. If the Flash memory size on the device is less than 64 KB, this register usually reads as zeroes, but software should not rely on these bits to be zero. For additional information, see the "Flash Memory Protection" section.

Flash Memory Protection Read Enable 1 (FMPRE1)

Base 0x400F.E000
Offset 0x204
Type R/W, reset 0xFFFF.FFFF

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | READ_ENABLE | | | | | | | | | | | | | | | |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | READ_ENABLE | | | | | | | | | | | | | | | |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | READ_ENABLE | R/W | 0xFFFFFFFF | Flash Read Enable<br>Configures 2-KB flash blocks to be read only. The policies may be combined as shown in the table “Flash Protection Policy Combinations”.<br><br>Value: 0xFFFFFFFF<br>Description: Bits [31:0] each enable protection on a 2-KB block of Flash memory in memory range from 65 to 128 KB. |

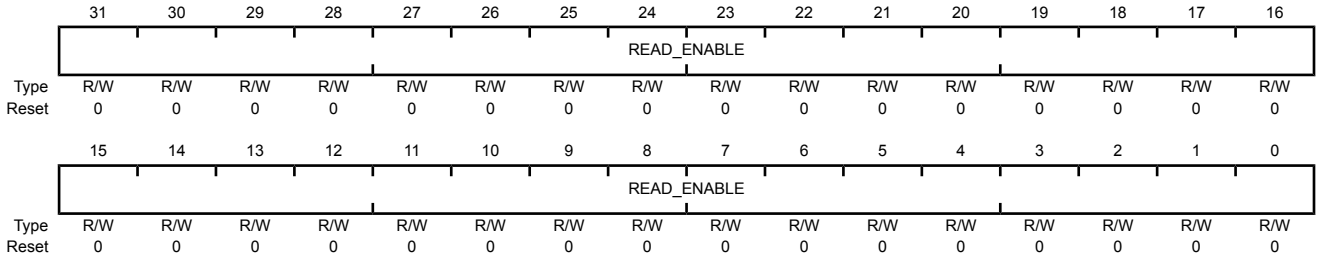
## Register 17: Flash Memory Protection Read Enable 2 (FMPRE2), offset 0x208

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register stores the read-only protection bits for each 2-KB flash block (**FMPPEn** stores the execute-only bits). Flash memory up to a total of 64 KB is controlled by this register. Other **FMPREn** registers (if any) provide protection for other 64K blocks. This register is loaded during the power-on reset sequence. The factory settings for the **FMPREn** and **FMPPEn** registers are a value of 1 for all implemented banks. This achieves a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection bit from a 1 to a 0 (and may NOT change a 0 to a 1). The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. The reset value shown only applies to power-on reset; any other type of reset does not affect this register. Once committed, the only way to restore the factory default value of this register is to perform the "Recover Locked Device" sequence detailed in the JTAG chapter. If the Flash memory size on the device is less than 128 KB, this register usually reads as zeroes, but software should not rely on these bits to be zero. For additional information, see the "Flash Memory Protection" section.

Flash Memory Protection Read Enable 2 (FMPRE2)

Base 0x400F.E000
Offset 0x208
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | READ_ENABLE | R/W | 0x00000000 | Flash Read Enable<br>Configures 2-KB flash blocks to be read only. The policies may be combined as shown in the table “Flash Protection Policy Combinations”.<br><br>Value Description<br>0x00000000 Bits [31:0] each enable protection on a 2-KB block of Flash memory in the range from 129 to 192 KB. |

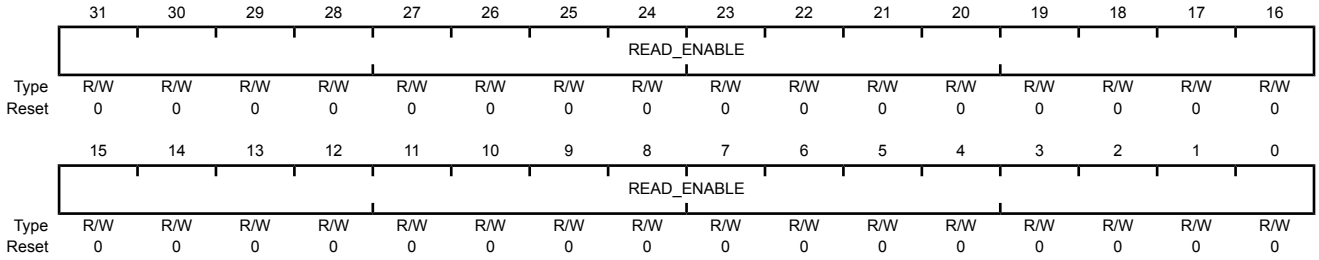
## Register 18: Flash Memory Protection Read Enable 3 (FMPRE3), offset 0x20C

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register stores the read-only protection bits for each 2-KB flash block (**FMPPEn** stores the execute-only bits). Flash memory up to a total of 64 KB is controlled by this register. Other **FMPREn** registers (if any) provide protection for other 64K blocks. This register is loaded during the power-on reset sequence. The factory settings for the **FMPREn** and **FMPPEn** registers are a value of 1 for all implemented banks. This achieves a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection bit from a 1 to a 0 (and may NOT change a 0 to a 1). The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. The reset value shown only applies to power-on reset; any other type of reset does not affect this register. Once committed, the only way to restore the factory default value of this register is to perform the "Recover Locked Device" sequence detailed in the JTAG chapter. If the Flash memory size on the device is less than 192 KB, this register usually reads as zeroes, but software should not rely on these bits to be zero. For additional information, see the "Flash Memory Protection" section.

Flash Memory Protection Read Enable 3 (FMPRE3)

Base 0x400F.E000
Offset 0x20C
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | READ_ENABLE | R/W | 0x00000000 | Flash Read Enable<br>Configures 2-KB flash blocks to be read only. The policies may be combined as shown in the table “Flash Protection Policy Combinations”.<br><br>Value: 0x00000000 — Description: Bits [31:0] each enable protection on a 2-KB block of Flash memory in the range from 193 to 256 KB. |

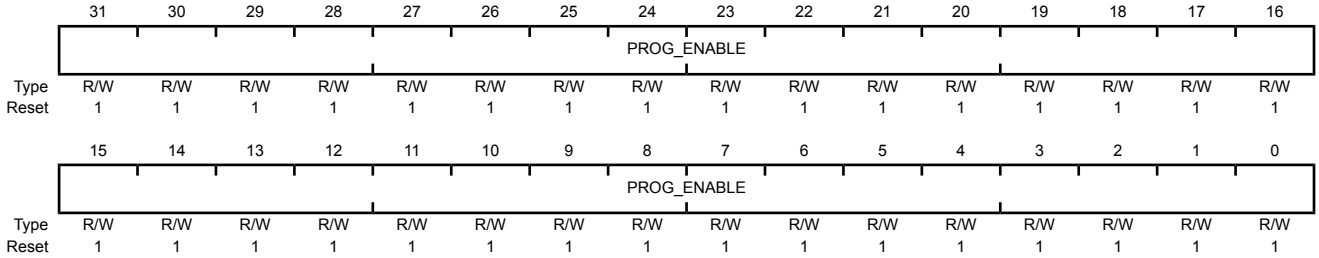
## Register 19: Flash Memory Protection Program Enable 1 (FMPPE1), offset 0x404

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register stores the execute-only protection bits for each 2-KB flash block (**FMPREn** stores the execute-only bits). Flash memory up to a total of 64 KB is controlled by this register. Other **FMPPEn** registers (if any) provide protection for other 64K blocks. This register is loaded during the power-on reset sequence. The factory settings for the **FMPREn** and **FMPPEn** registers are a value of 1 for all implemented banks. This achieves a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection bit from a 1 to a 0 (and may NOT change a 0 to a 1). The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. The reset value shown only applies to power-on reset; any other type of reset does not affect this register. Once committed, the only way to restore the factory default value of this register is to perform the "Recover Locked Device" sequence detailed in the JTAG chapter. If the Flash memory size on the device is less than 64 KB, this register usually reads as zeroes, but software should not rely on these bits to be zero. For additional information, see the "Flash Memory Protection" section.

Flash Memory Protection Program Enable 1 (FMPPE1)

Base 0x400F.E000
Offset 0x404
Type R/W, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | PROG_ENABLE | R/W | 0xFFFFFFFF | Flash Programming Enable<br>Configures 2-KB flash blocks to be execute only. The policies may be combined as shown in the table “Flash Protection Policy Combinations”.<br><br>Value Description<br>0xFFFFFFFF Bits [31:0] each enable protection on a 2-KB block of Flash memory in memory range from 65 to 128 KB. |

## Register 20: Flash Memory Protection Program Enable 2 (FMPPE2), offset 0x408

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register stores the execute-only protection bits for each 2-KB flash block (**FMPREn** stores the execute-only bits). Flash memory up to a total of 64 KB is controlled by this register. Other **FMPPEn** registers (if any) provide protection for other 64K blocks. This register is loaded during the power-on reset sequence. The factory settings for the **FMPREn** and **FMPPEn** registers are a value of 1 for all implemented banks. This achieves a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection bit from a 1 to a 0 (and may NOT change a 0 to a 1). The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. The reset value shown only applies to power-on reset; any other type of reset does not affect this register. Once committed, the only way to restore the factory default value of this register is to perform the "Recover Locked Device" sequence detailed in the JTAG chapter. If the Flash memory size on the device is less than 128 KB, this register usually reads as zeroes, but software should not rely on these bits to be zero. For additional information, see the "Flash Memory Protection" section.

Flash Memory Protection Program Enable 2 (FMPPE2)

Base 0x400F.E000
Offset 0x408
Type R/W, reset 0x0000.0000

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

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | PROG_ENABLE | R/W | 0x00000000 | Flash Programming Enable<br>Configures 2-KB flash blocks to be execute only. The policies may be combined as shown in the table “Flash Protection Policy Combinations”.<br><br>Value Description<br>0x00000000 Bits [31:0] each enable protection on a 2-KB block of Flash memory in the range from 129 to 192 KB. |

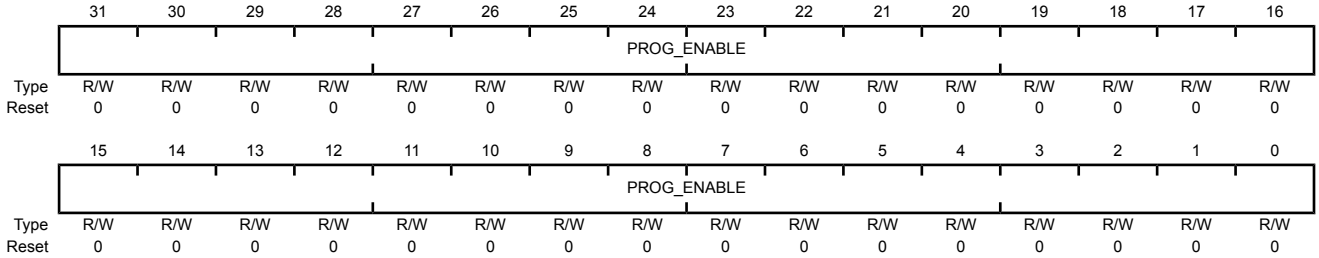
## Register 21: Flash Memory Protection Program Enable 3 (FMPPE3), offset 0x40C

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register stores the execute-only protection bits for each 2-KB flash block (**FMPREn** stores the execute-only bits). Flash memory up to a total of 64 KB is controlled by this register. Other **FMPPEn** registers (if any) provide protection for other 64K blocks. This register is loaded during the power-on reset sequence. The factory settings for the **FMPREn** and **FMPPEn** registers are a value of 1 for all implemented banks. This achieves a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection bit from a 1 to a 0 (and may NOT change a 0 to a 1). The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. The reset value shown only applies to power-on reset; any other type of reset does not affect this register. Once committed, the only way to restore the factory default value of this register is to perform the "Recover Locked Device" sequence detailed in the JTAG chapter. If the Flash memory size on the device is less than 192 KB, this register usually reads as zeroes, but software should not rely on these bits to be zero. For additional information, see the "Flash Memory Protection" section.

Flash Memory Protection Program Enable 3 (FMPPE3)

Base 0x400F.E000
Offset 0x40C
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | PROG_ENABLE | R/W | 0x00000000 | Flash Programming Enable<br>Configures 2-KB flash blocks to be execute only. The policies may be combined as shown in the table “Flash Protection Policy Combinations”.<br><br>Value Description<br>0x00000000 Bits [31:0] each enable protection on a 2-KB block of Flash memory in the range from 193 to 256 KB. |

# 8 Micro Direct Memory Access (µDMA)

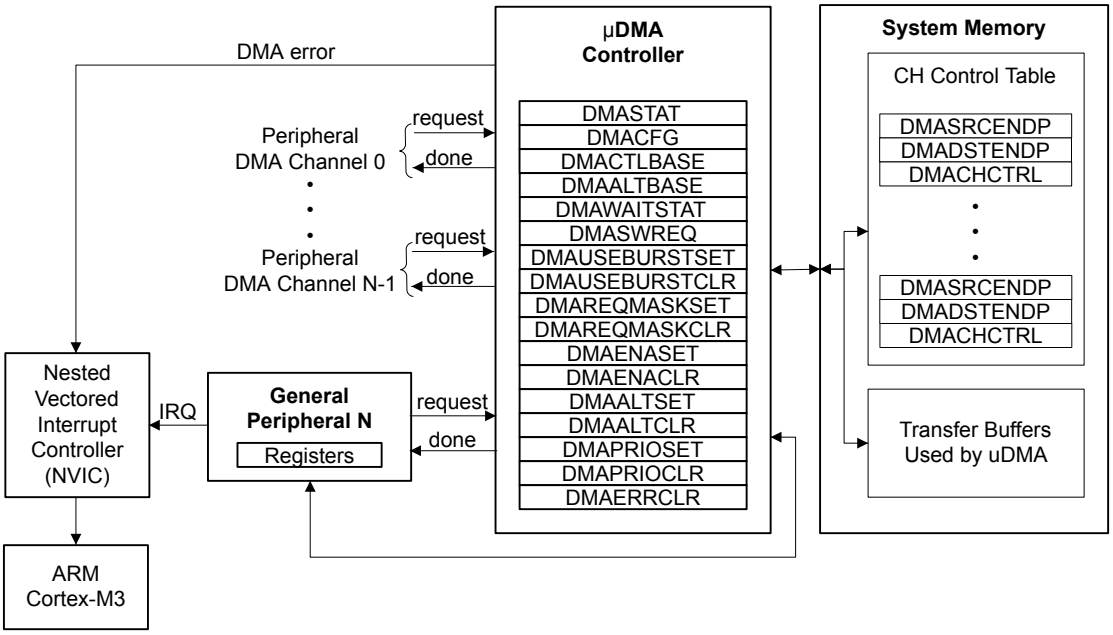
The LM3S5632 microcontroller includes a Direct Memory Access (DMA) controller, known as micro-DMA (µDMA). The µDMA controller provides a way to offload data transfer tasks from the Cortex-M3 processor, allowing for more efficient use of the processor and the expanded available bus bandwidth. The µDMA controller can perform transfers between memory and peripherals. It has dedicated channels for each supported peripheral and can be programmed to automatically perform transfers between peripherals and memory as the peripheral is ready to transfer more data. The µDMA controller also supports sophisticated transfer modes such as ping-pong and scatter-gather, which allows the processor to set up a list of transfer tasks for the controller.

The µDMA controller has the following features:

- ARM PrimeCell® 32-channel configurable µDMA controller
- Support for multiple transfer modes
  - Basic, for simple transfer scenarios
  - Ping-pong, for continuous data flow to/from peripherals
  - Scatter-gather, from a programmable list of up to 256 arbitrary transfers initiated from a single request
- Dedicated channels for supported peripherals
- One channel each for receive and transmit path for bidirectional peripherals
- Dedicated channel for software-initiated transfers
- Independently configured and operated channels
- Per-channel configurable bus arbitration scheme
- Two levels of priority
- Design optimizations for improved bus access performance between µDMA controller and the processor core
  - µDMA controller access is subordinate to core access
  - RAM striping
  - Peripheral bus segmentation
- Data sizes of 8, 16, and 32 bits
- Source and destination address increment size of byte, half-word, word, or no increment
- Maskable device requests
- Optional software initiated requests for any channel
- Interrupt on transfer completion, with a separate interrupt per channel

## 8.1 Block Diagram

**Figure 8-1. µDMA Block Diagram**




## 8.2 Functional Description

The µDMA controller is a flexible and highly configurable DMA controller designed to work effeciently with the microcontroller's Cortex-M3 processor core. It supports multiple data sizes and address increment schemes, multiple levels of priority among DMA channels, and several transfer modes to allow for sophisticated programmed data transfers. The DMA controller's usage of the bus is always subordinate to the processor core, and so it will never hold up a bus transaction by the processor. Because the µDMA controller is only using otherwise-idle bus cycles, the data transfer bandwidth it provides is essentially free, with no impact on the rest of the system. The bus architecture has been optimized to greatly reduce contention between the processor core and the µDMA controller, thus improving performance. The optimizations include RAM striping and peripheral bus segmentation, which in many cases allows both the processor core and the µDMA controller to access the bus and perform simultaneous data transfers.

Each peripheral function that is supported has a dedicated channel on the µDMA controller that can be configured independently.

The µDMA controller makes use of a unique configuration method by using channel control structures that are maintained in system memory by the processor. While simple transfer modes are supported, it is also possible to build up sophisticated "task" lists in memory that allow the controller to perform arbitrary-sized transfers to and from arbitrary locations as part of a single transfer request. The controller also supports the use of ping-pong buffering to accomodate constant streaming of data to or from a peripheral.

Each channel also has a configurable arbitration size. The arbitration size is the number of items that will be transferred in a burst before the controller rearbitrates for channel priority. Using the arbitration size, it is possible to control exactly how many items are transferred to or from a peripheral each time it makes a DMA service request.

### 8.2.1 Channel Assigments

µDMA channels 0-31 are assigned to peripherals according to the following table.

**Note:** Channels that are not listed in the table may be assigned to peripherals in the future. However, they are currently available for software use.

**Table 8-1. DMA Channel Assignments**

| DMA Channel | Peripheral Assigned |
|---|---|
| 0 | USB Endpoint 1 Receive |
| 1 | USB Endpoint 1 Transmit |
| 2 | USB Endpoint 2 Receive |
| 3 | USB Endpoint 2 Transmit |
| 4 | USB Endpoint 3 Receive |
| 5 | USB Endpoint 3 Transmit |
| 8 | UART0 Receive |
| 9 | UART0 Transmit |
| 10 | SSI0 Receive |
| 11 | SSI0 Transmit |
| 22 | UART1 Receive |
| 23 | UART1 Transmit |
| 30 | Dedicated for software use |

### 8.2.2 Priority

The µDMA controller assigns priority to each channel based on the channel number and the priority level bit for the channel. Channel number 0 has the highest priority and as the channel number increases, the priority of a channel decreases. Each channel has a priority level bit to provide two levels of priority: default priority and high priority. If the priority level bit is set, then that channel has higher priority than all other channels at default priority. If multiple channels are set for high priority, then the channel number is used to determine relative priority among all the high priority channels.

The priority bit for a channel can be set using the **DMA Channel Priority Set (DMAPRIOSET)** register, and cleared with the **DMA Channel Priority Clear (DMAPRIOCLR)** register.

### 8.2.3 Arbitration Size

When a µDMA channel requests a transfer, the µDMA controller arbitrates between all the channels making a request and services the DMA channel with the highest priority. Once a transfer begins, it continues for a selectable number of transfers before rearbitrating among the requesting channels again. The arbitration size can be configured for each channel, ranging from 1 to 1024 item transfers. After the µDMA controller transfers the number of items specified by the arbitration size, it then checks among all the channels making a request and services the channel with the highest priority.

If a lower priority DMA channel uses a large arbitration size, the latency for higher priority channels will be increased because the µDMA controller will complete the lower priority burst before checking for higher priority requests. Therefore, lower priority channels should not use a large arbitration size for best response on high priority channels.

The arbitration size can also be thought of as a burst size. It is the maximum number of items that will be transferred at any one time in a burst. Here, the term arbitration refers to determination of DMA channel priority, not arbitration for the bus. When the µDMA controller arbitrates for the bus,

the processor always takes priority. Furthermore, the µDMA controller will be held off whenever the processor needs to perform a bus transaction on the same bus, even in the middle of a burst transfer.

### 8.2.4 Request Types

The µDMA controller responds to two types of requests from a peripheral: single or burst. Each peripheral may support either or both types of requests. A single request means that the peripheral is ready to transfer one item, while a burst request means that the peripheral is ready to transfer multiple items.

The µDMA controller responds differently depending on whether the peripheral is making a single request or a burst request. If both are asserted and the µDMA channel has been set up for a burst transfer, then the burst request takes precedence. See Table 8-2 on page 293, which shows how each peripheral supports the two request types.

**Table 8-2. Request Type Support**

| Peripheral | Single Request Signal | Burst Request Signal |
|---|---|---|
| USB TX | None | FIFO TXRDY |
| USB RX | None | FIFO RXRDY |
| UART TX | TX FIFO Not Full | TX FIFO Level (configurable) |
| UART RX | RX FIFO Not Empty | RX FIFO Level (configurable) |
| SSI TX | TX FIFO Not Full | TX FIFO Level (fixed at 4) |
| SSI RX | RX FIFO Not Empty | RX FIFO Level (fixed at 4) |

#### 8.2.4.1 Single Request

When a single request is detected, and not a burst request, the µDMA controller will transfer one item, and then stop and wait for another request.

#### 8.2.4.2 Burst Request

When a burst request is detected, the µDMA controller will transfer the number of items that is the lesser of the arbitration size or the number of items remaining in the transfer. Therefore, the arbitration size should be the same as the number of data items that the peripheral can accomodate when making a burst request. For example, the UART will generate a burst request based on the FIFO trigger level. In this case, the arbitration size should be set to the amount of data that the FIFO can transfer when the trigger level is reached.

It may be desirable to use only burst transfers and not allow single transfers. For example, perhaps the nature of the data is such that it only makes sense when transferred together as a single unit rather than one piece at a time. The single request can be disabled by using the **DMA Channel Useburst Set (DMAUSEBURSTSET)** register. By setting the bit for a channel in this register, the µDMA controller will only respond to burst requests for that channel.

### 8.2.5 Channel Configuration

The µDMA controller uses an area of system memory to store a set of channel control structures in a table. The control table may have one or two entries for each DMA channel. Each entry in the table structure contains source and destination pointers, transfer size, and transfer mode. The control table can be located anywhere in system memory, but it must be contiguous and aligned on a 1024-byte boundary.

Table 8-3 on page 294 shows the layout in memory of the channel control table. Each channel may have one or two control structures in the contol table: a primary control structure and an optional

alternate control structure. The table is organized so that all of the primary entries are in the first half of the table and all the alternate structures are in the second half of the table. The primary entry is used for simple transfer modes where transfers can be reconfigured and restarted after each transfer is complete. In this case, the alternate control structures are not used and therefore only the first half of the table needs to be allocated in memory. The second half of the control table is not needed and that memory can be used for something else. If a more complex transfer mode is used such as ping-pong or scatter-gather, then the alternate control structure is also used and memory space should be allocated for the entire table.

Any unused memory in the control table may be used by the application. This includes the control structures for any channels that are unused by the application as well as the unused control word for each channel.

**Table 8-3. Control Structure Memory Map**

| Offset | Channel |
|---|---|
| 0x0 | 0, Primary |
| 0x10 | 1, Primary |
| ... | ... |
| 0x1F0 | 31, Primary |
| 0x200 | 0, Alternate |
| 0x210 | 1, Alternate |
| ... | ... |
| 0x3F0 | 31, Alternate |

Table 8-4 on page 294 shows an individual control structure entry in the control table. Each entry has a source and destination *end* pointer. These pointers point to the ending address of the transfer and are inclusive. If the source or destination is non-incrementing (as for a peripheral register), then the pointer should point to the transfer address.

**Table 8-4. Channel Control Structure**

| Offset | Description |
|---|---|
| 0x000 | Source End Pointer |
| 0x004 | Destination End Pointer |
| 0x008 | Control Word |
| 0x00C | Unused |

The remaining part of the control structure is the control word. The control word contains the following fields:

- Source and destination data sizes
- Source and destination address increment size
- Number of transfers before bus arbitration
- Total number of items to transfer
- Useburst flag
- Transfer mode

The control word and each field are described in detail in “µDMA Channel Control Structure” on page 311. The µDMA controller updates the transfer size and transfer mode fields as the transfer is performed. At the end of a transfer, the transfer size will indicate 0, and the transfer mode will indicate "stopped". Since the control word is modified by the µDMA controller, it must be reconfigured before each new transfer. The source and destination end pointers are not modified so they can be left unchanged if the source or destination addresses remain the same.

Prior to starting a transfer, a µDMA channel must be enabled by setting the appropriate bit in the **DMA Channel Enable Set ((DMAENASET)** register. A channel can be disabled by setting the channel bit in the **DMA Channel Enable Clear (DMAENACLR)** register. At the end of a complete DMA transfer, the controller will automatically disable the channel.

### 8.2.6 Transfer Modes

The µDMA controller supports several transfer modes. Two of the modes support simple one-time transfers. There are several complex modes that are meant to support a continuous flow of data.

#### 8.2.6.1 Stop Mode

While Stop is not actually a transfer mode, it is a valid value for the mode field of the control word. When the mode field has this value, the µDMA controller will not perform a transfer and will disable the channel if it is enabled. At the end of a transfer, the µDMA controller will update the control word to set the mode to Stop.

#### 8.2.6.2 Basic Mode

In Basic mode, the µDMA controller will perform transfers as long as there are more items to transfer and a transfer request is present. This mode is used with peripherals that assert a DMA request signal whenever the peripheral is ready for a data transfer. Basic mode should not be used in any situation where the request is momentary but the entire transfer should be completed. For example, for a software initiated transfer, the request is momentary, and if Basic mode is used then only one item will be transferred on a software request.

When all of the items have been transferred using Basic mode, the µDMA controller will set the mode for that channel to Stop.

#### 8.2.6.3 Auto Mode

Auto mode is similar to Basic mode, except that once a transfer request is received the transfer will run to completion, even if the DMA request is removed. This mode is suitable for software-triggered transfers. Generally, you would not use Auto mode with a peripheral.

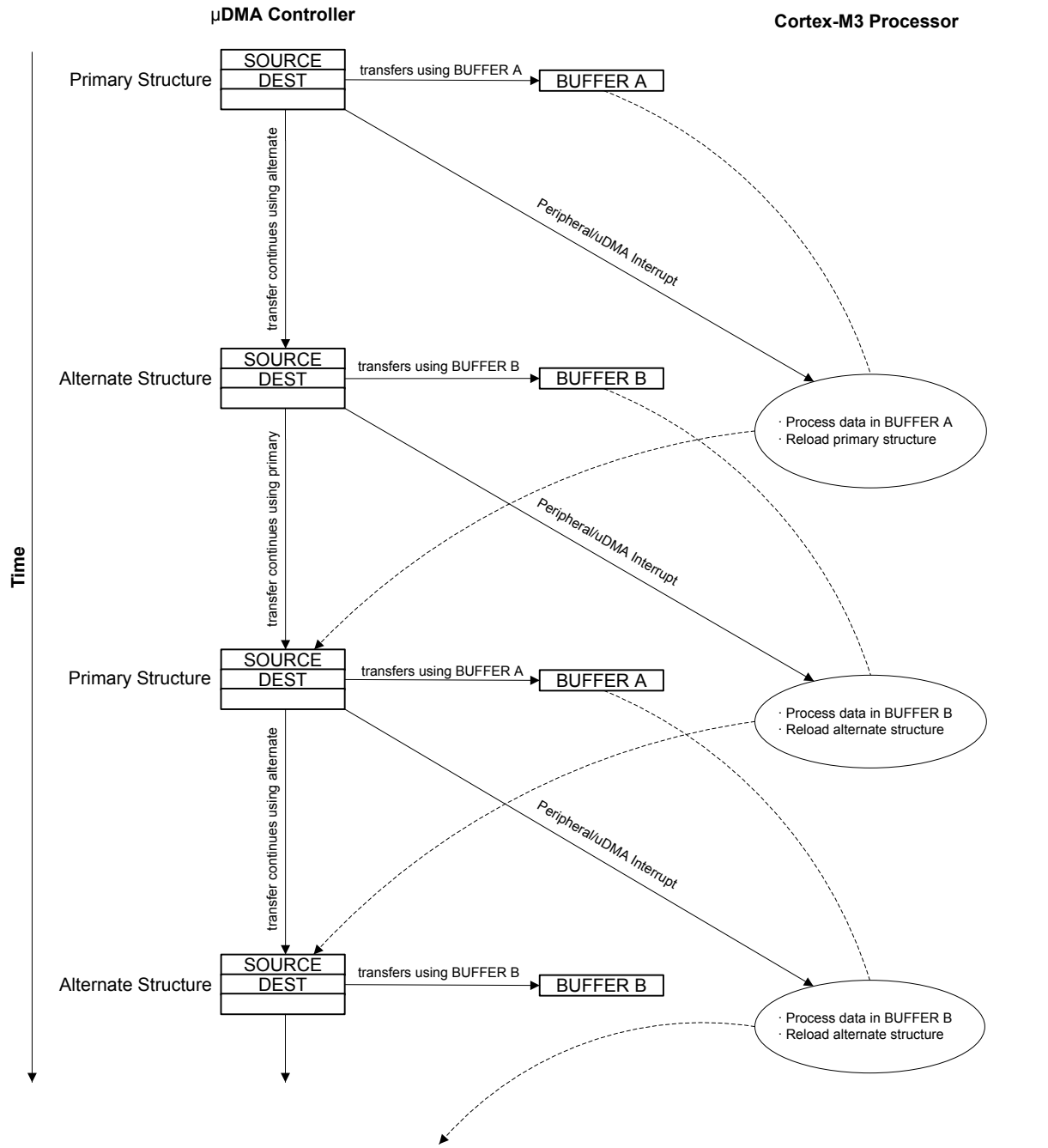
When all the items have been transferred using Auto mode, the µDMA controller will set the mode for that channel to Stop.

#### 8.2.6.4 Ping-Pong

Ping-Pong mode is used to support a continuous data flow to or from a peripheral. To use Ping-Pong mode, both the primary and alternate data structures are used. Both are set up by the processor for data transfer between memory and a peripheral. Then the transfer is started using the primary control structure. When the transfer using the primary control structure is complete, the µDMA controller will then read the alternate control structure for that channel to continue the transfer. Each time this happens, an interrupt is generated and the processor can reload the control structure for the just-completed transfer. Data flow can continue indefinitely this way, using the primary and alternate control structures to switch back and forth between buffers as the data flows to or from the peripheral.

Refer to Figure 8-2 on page 296 for an example showing operation in Ping-Pong mode.

**Figure 8-2. Example of Ping-Pong DMA Transaction**




### 8.2.6.5 Memory Scatter-Gather

Memory Scatter-Gather mode is a complex mode used when data needs to be transferred to or from varied locations in memory instead of a set of contiguous locations in a memory buffer. For example, a gather DMA operation could be used to selectively read the payload of several stored packets of a communication protocol, and store them together in sequence in a memory buffer.

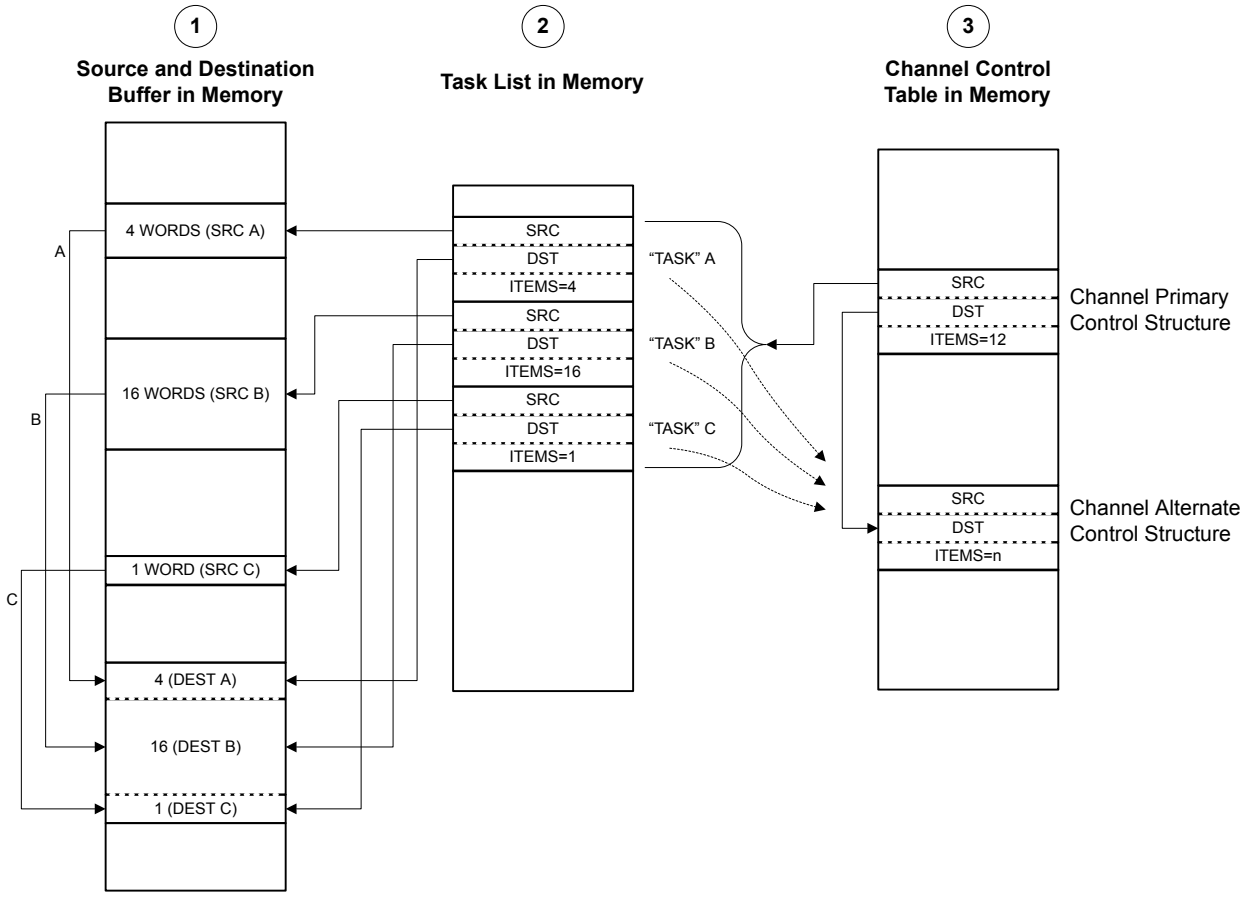
In Memory Scatter-Gather mode, the primary control structure is used to program the alternate control structure from a table in memory. The table is set up by the processor software and contains a list of control structures, each containing the source and destination end pointers, and the control word for a specific transfer. The mode of each control word must be set to Scatter-Gather mode. Each entry in the table is copied in turn to the alternate structure where it is then executed. The µDMA controller alternates between using the primary control structure to copy the next transfer instruction from the list, and then executing the new transfer instruction. The end of the list is marked by setting the control word for the last entry to use Basic transfer mode. Once the last transfer is performed using Basic mode, the µDMA controller will stop. A completion interrupt will only be generated after the last transfer. It is possible to loop the list by having the last entry copy the primary control structure to point back to the beginning of the list (or to a new list). It is also possible to trigger a set of other channels to perform a transfer, either directly by programming a write to the software trigger for another channel, or indirectly by causing a peripheral action that will result in a µDMA request.

By programming the µDMA controller using this method, a set of up to 256 arbitrary transfers can be performed based on a single DMA request.

Refer to Figure 8-3 on page 298 and Figure 8-4 on page 299, which show an example of operation in Memory Scatter-Gather mode. This example shows a *gather* operation, where data in three separate buffers in memory will be copied together into one buffer. Figure 8-3 on page 298 shows how the application sets up a µDMA *task list* in memory that is used by the controller to perform three sets of copy operations from different locations in memory. The primary control structure for the channel that will be used for the operation is configured to copy from the task list to the alternate control structure.

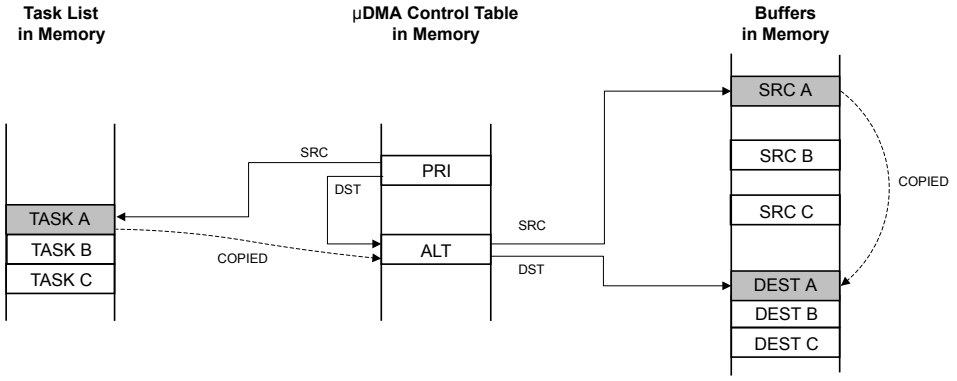
Figure 8-4 on page 299 shows the sequence as the µDMA controller peforms the three sets of copy operations. First, using the primary control structure, the µDMA controller loads the alternate control structure with task A. It then peforms the copy operation specified by task A, copying the data from the source buffer A to the destination buffer. Next, the µDMA controller again uses the primary control structure to load task B into the alternate control structure, and then performs the B operation with the alternate control structure. The process is repeated for task C.

**Figure 8-3. Memory Scatter-Gather, Setup and Configuration**



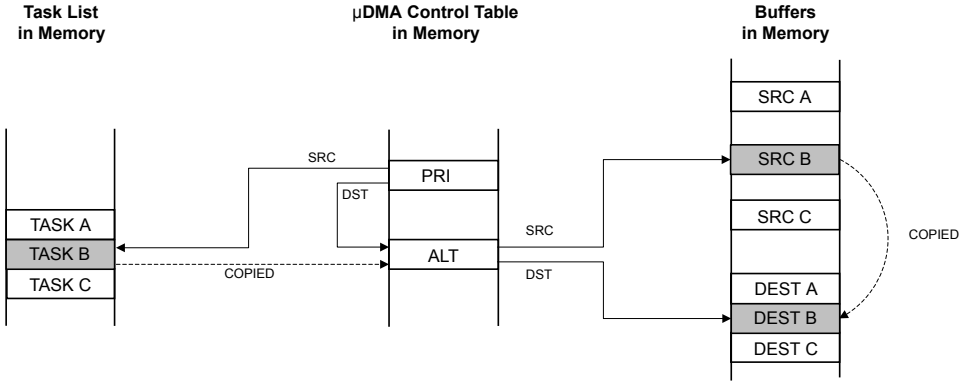

NOTES:
1. Application has a need to copy data items from three separate location in memory into one combined buffer.
2. Application sets up uDMA "task list" in memory, which contains the pointers and control configuration for three uDMA copy "tasks."
3. Application sets up the channel primary control structure to copy each task configuration, one at a time, to the alternate control structure, where it will be executed by the uDMA controller.

## Figure 8-4. Memory Scatter-Gather, µDMA Copy Sequence



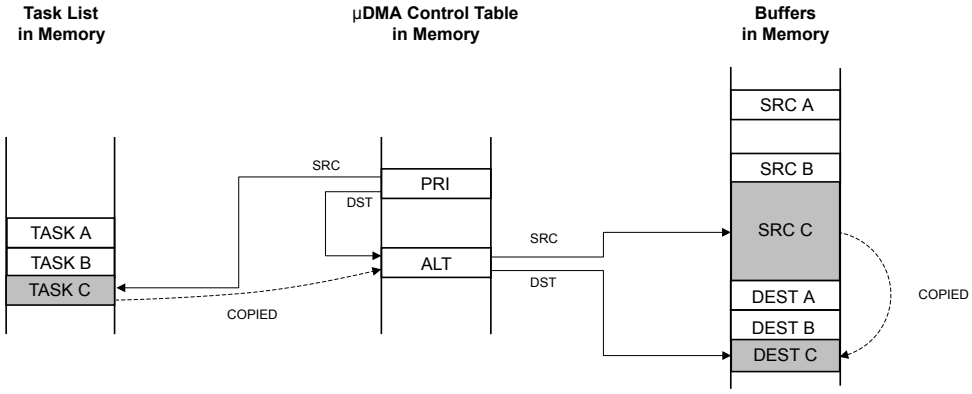

Using the channel's primary control structure, the µDMA controller copies task A configuration to the channel's alternate control structure.

Then, using the channel's alternate control structure, the µDMA controller copies data from the source buffer A to the destination buffer.




Using the channel's primary control structure, the µDMA controller copies task B configuration to the channel's alternate control structure.

Then, using the channel's alternate control structure, the µDMA controller copies data from the source buffer B to the destination buffer.




Using the channel's primary control structure, the µDMA controller copies task C configuration to the channel's alternate control structure.

Then, using the channel's alternate control structure, the µDMA controller copies data from the source buffer C to the destination buffer.

#### 8.2.6.6 Peripheral Scatter-Gather

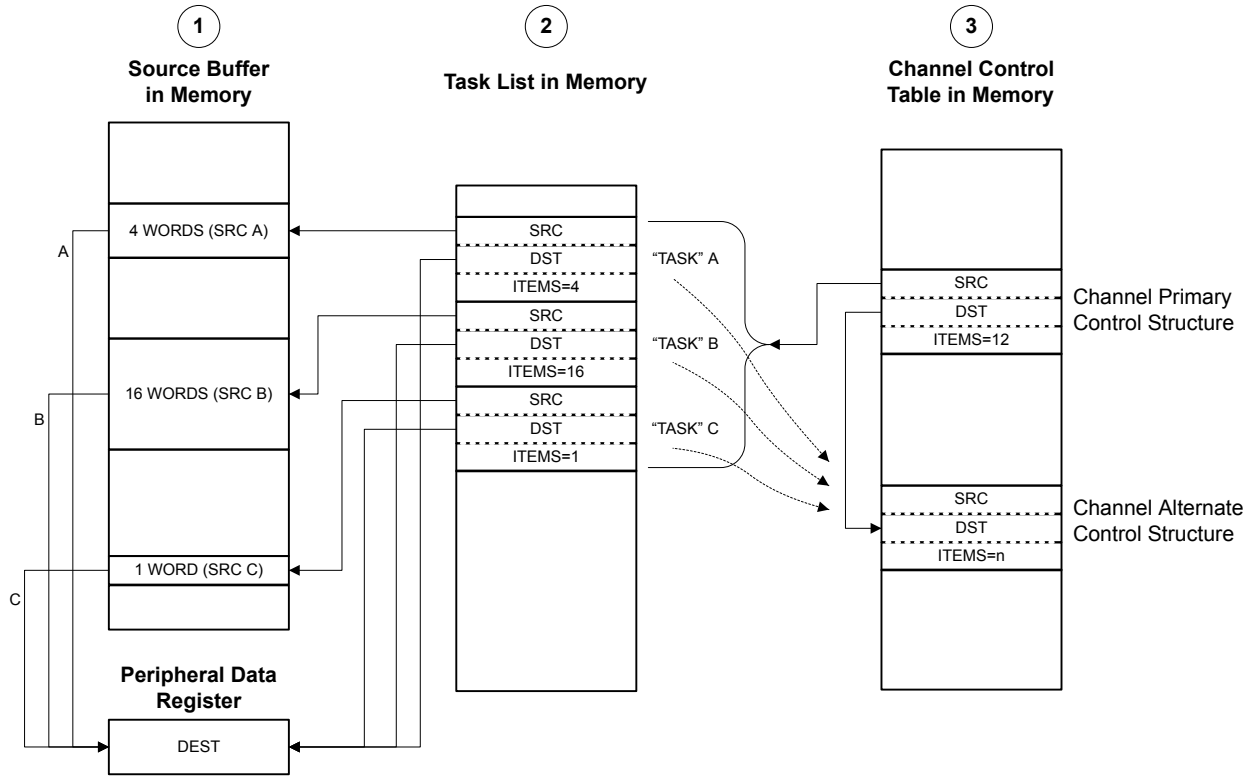
Peripheral Scatter-Gather mode is very similar to Memory Scatter-Gather, except that the transfers are controlled by a peripheral making a DMA request. Upon detecting a DMA request from the peripheral, the µDMA controller will use the primary control structure to copy one entry from the list to the alternate control structure, and then perform the transfer. At the end of this transfer, the next transfer will only be started if the peripheral again asserts a DMA request. The µDMA controller will continue to perform transfers from the list only when the peripheral is making a request, until the last transfer is complete. A completion interrupt will only be generated after the last transfer.

By programming the µDMA controller using this method, data can be transferred to or from a peripheral from a set of arbitrary locations whenever the peripheral is ready to transfer data.

Refer to Figure 8-5 on page 301 and Figure 8-6 on page 302, which show an example of operation in Peripheral Scatter-Gather mode. This example shows a gather operation, where data from three separate buffers in memory will be copied to a single peripheral data register. Figure 8-5 on page 301 shows how the application sets up a µDMA task list in memory that is used by the controller to perform three sets of copy operations from different locations in memory. The primary control structure for the channel that will be used for the operation is configured to copy from the task list to the alternate control structure.

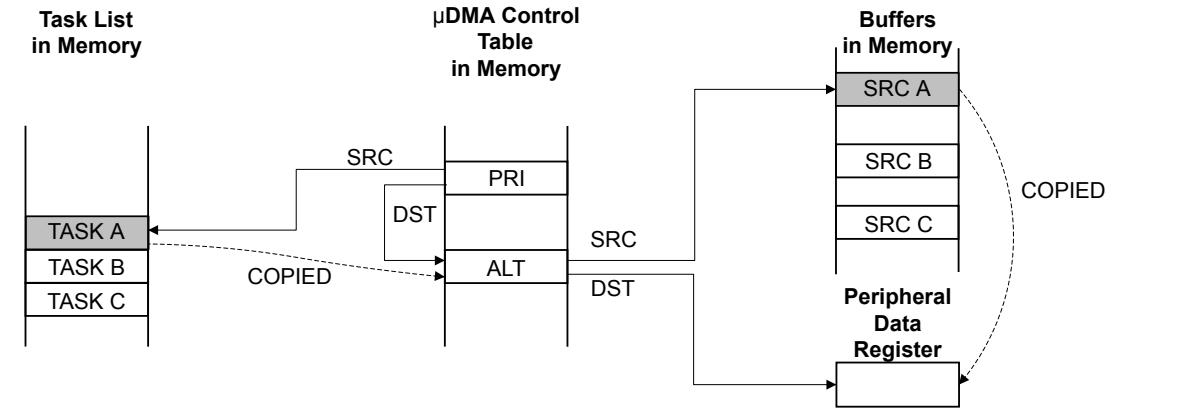
Figure 8-6 on page 302 shows the sequence as the µDMA controller peforms the three sets of copy operations. First, using the primary control structure, the µDMA controller loads the alternate control structure with task A. It then peforms the copy operation specified by task A, copying the data from the source buffer A to the peripheral data register. Next, the µDMA controller again uses the primary control structure to load task B into the alternate control structure, and then performs the B operation with the alternate control structure. The process is repeated for task C.

**Figure 8-5. Peripheral Scatter-Gather, Setup and Configuration**



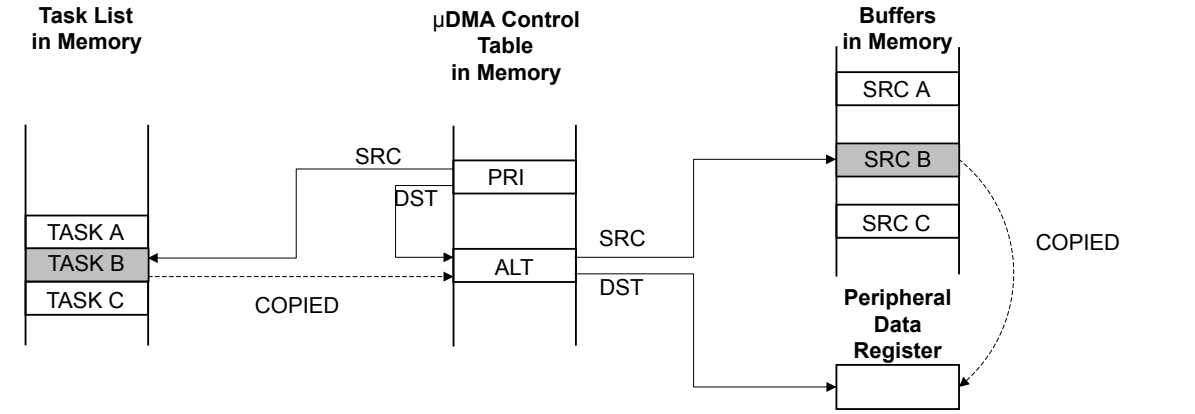

NOTES:
1. Application has a need to copy data items from three separate location in memory into a peripheral data register.
2. Application sets up µDMA "task list" in memory, which contains the pointers and control configuration for three uDMA copy "tasks."
3. Application sets up the channel primary control structure to copy each task configuration, one at a time, to the alternate control structure, where it will be executed by the µDMA controller.

**Figure 8-6. Peripheral Scatter-Gather, µDMA Copy Sequence**



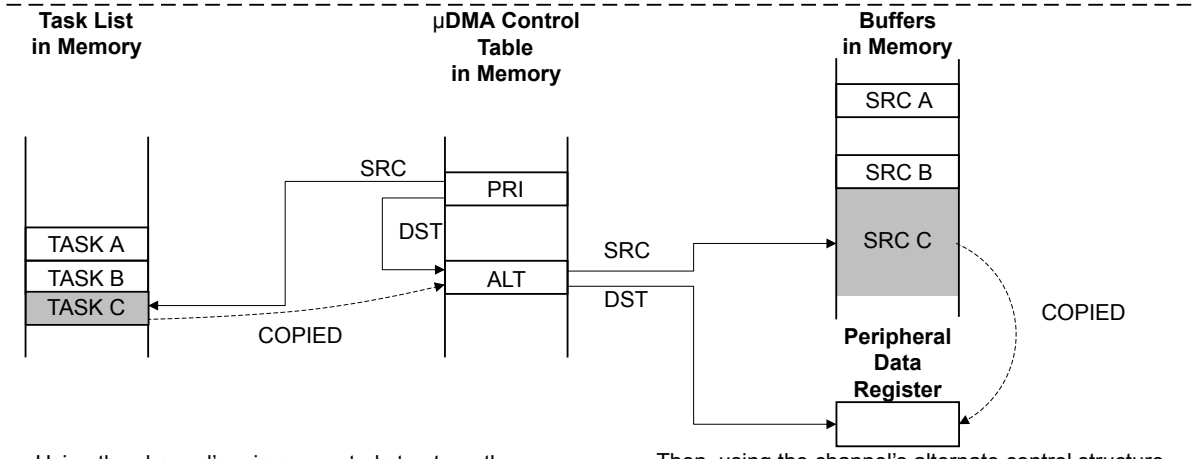

Using the channel's primary control structure, the µDMA controller copies task A configuration to the channel's alternate control structure.

Then, using the channel's alternate control structure, the µDMA controller copies data from the source buffer A to the peripheral data register.




Using the channel's primary control structure, the µDMA controller copies task B configuration to the channel's alternate control structure.

Then, using the channel's alternate control structure, the µDMA controller copies data from the source buffer B to the peripheral data register.




Using the channel's primary control structure, the µDMA controller copies task C configuration to the channel's alternate control structure.

Then, using the channel's alternate control structure, the µDMA controller copies data from the source buffer C to the peripheral data register.

### 8.2.7 Transfer Size and Increment

The µDMA controller supports transfer data sizes of 8, 16, or 32 bits. The source and destination data size must be the same for any given transfer. The source and destination address can be auto-incremented by bytes, half-words, or words, or can be set to no increment. The source and destination address increment values can be set independently, and it is not necessary for the address increment to match the data size as long as the increment is the same or larger than the data size. For example, it is possible to perform a transfer using 8-bit data size, but using an address increment of full words (4 bytes). The data to be transferred must be aligned in memory according to the data size (8, 16, or 32 bits).

Table 8-5 on page 303 shows the configuration to read from a peripheral that supplies 8-bit data.

**Table 8-5. µDMA Read Example: 8-Bit Peripheral**

| Field | Configuration |
|---|---|
| Source data size | 8 bits |
| Destination data size | 8 bits |
| Source address increment | No increment |
| Destination address increment | Byte |
| Source end pointer | Peripheral read FIFO register |
| Destination end pointer | End of the data buffer in memory |

### 8.2.8 Peripheral Interface

Each peripheral that supports µDMA has a DMA single request and/or burst request signal that is asserted when the device is ready to transfer data. The request signal can be disabled or enabled by using the **DMA Channel Request Mask Set (DMAREQMASKSET)** and **DMA Channel Request Mask Clear (DMAREQMASKCLR)** registers. The DMA request signal is disabled, or masked, when the channel request mask bit is set. When the request is not masked, the DMA channel is configured correctly and enabled, and the peripheral asserts the DMA request signal, the µDMA controller will begin the transfer.

When a DMA transfer is complete, the µDMA controller asserts a DMA Done signal, which is routed through the interrupt vector of the peripheral. Therefore, if DMA is used to transfer data for a peripheral and interrupts are used, then the interrupt handler for that peripheral must be designed to handle the µDMA transfer completion interrupt. When DMA is enabled for a peripheral, the µDMA controller will mask the normal interrupts for a peripheral. This means that when a large amount of data is transferred using DMA, instead of receiving multiple interrupts from the peripheral as data flows, the processor will only receive one interrupt when the transfer is complete.

The interrupt request from the µDMA controller is automatically cleared when the interrupt handler is activated.

### 8.2.9 Software Request

There is a dedicated µDMA channel for software-initiated transfers. This channel also has a dedicated interrupt to signal completion of a DMA transfer. A transfer is initiated by software by first configuring and enabling the transfer, and then issuing a software request using the **DMA Channel Software Request (DMASWREQ)** register. For software-based transfers, the Auto transfer mode should be used.

It is possible to initiate a transfer on any channel using the **DMASWREQ** register. If a request is initiated by software using a peripheral DMA channel, then the completion interrupt will occur on the interrupt vector for the peripheral instead of the software interrupt vector. This means that any

channel may be used for software requests as long as the corresponding peripheral is not using µDMA.

### 8.2.10 Interrupts and Errors

When a DMA transfer is complete, the µDMA controller will generate a completion interrupt on the interrupt vector of the peripheral. If the transfer uses the software DMA channel, then the completion interrupt will occur on the dedicated software DMA interrupt vector.

If the µDMA controller encounters a bus or memory protection error as it attempts to perform a data transfer, it will disable the DMA channel that caused the error, and generate an interrupt on the µDMA Error interrupt vector. The processor can read the **DMA Bus Error Clear (DMAERRCLR)** register to determine if an error is pending. The ERRCLR bit will be set if an error occurred. The error can be cleared by writing a 1 to the ERRCLR bit.

If the peripheral generates an error that causes an interrupt, the interrupt will be generated on the interrupt vector for that peripheral. This is the same whether or not µDMA is being used with the peripheral.

Table 8-6 on page 304 shows the dedicated interrupt assignments for the µDMA controller.

**Table 8-6. µDMA Interrupt Assignments**

| Interrupt | Assignment |
|---|---|
| 46 | µDMA Software Channel Transfer |
| 47 | µDMA Error |

## 8.3 Initialization and Configuration

### 8.3.1 Module Initialization

Before the µDMA controller can be used, it must be enabled in the System Control block and in the peripheral. The location of the channel control structure must also be programmed.

The following steps should be performed one time during system initialization:

1. The µDMA peripheral must be enabled in the System Control block. To do this, set the UDMA bit of the System Control **RCGC2** register.
2. Enable the µDMA controller by setting the MASTEREN bit of the **DMA Configuration (DMACFG)** register.
3. Program the location of the channel control table by writing the base address of the table to the **DMA Channel Control Base Pointer (DMACTLBASE)** register. The base address must be aligned on a 1024-byte boundary.

### 8.3.2 Configuring a Memory-to-Memory Transfer

µDMA channel 30 is dedicated for software-initiated transfers. However, any channel can be used for software-initiated, memory-to-memory transfer if the associated peripheral is not being used.

#### 8.3.2.1 Configure the Channel Attributes

First, configure the channel attributes:

1. Set bit 30 of the **DMA Channel Priority Set (DMAPRIOSET)** or **DMA Channel Priority Clear (DMAPRIOCLR)** registers to set the channel to High priority or Default priority.

2. Set bit 30 of the **DMA Channel Primary Alternate Clear (DMAALTCLR)** register to select the primary channel control structure for this transfer.

3. Set bit 30 of the **DMA Channel Useburst Clear (DMAUSEBURSTCLR)** register to allow the µDMA controller to respond to single and burst requests.

4. Set bit 30 of the **DMA Channel Request Mask Clear (DMAREQMASKCLR)** register to allow the µDMA controller to recognize requests for this channel.

#### 8.3.2.2 Configure the Channel Control Structure

Now the channel control structure must be configured.

This example will transfer 256 32-bit words from one memory buffer to another. Channel 30 is used for a software transfer, and the control structure for channel 30 is at offset 0x1E0 of the channel control table. The channel control structure for channel 30 is located at the offsets shown in Table 8-7 on page 305.

**Table 8-7. Channel Control Structure Offsets for Channel 30**

| Offset | Description |
|---|---|
| Control Table Base + 0x1E0 | Channel 30 Source End Pointer |
| Control Table Base + 0x1E4 | Channel 30 Destination End Pointer |
| Control Table Base + 0x1E8 | Channel 30 Control Word |

***Configure the Source and Destination***

The source and destination end pointers must be set to the last address for the transfer (inclusive).

1. Set the source end pointer at offset 0x1E0 to the address of the source buffer + 0x3FC.

2. Set the destination end pointer at offset 0x1E4 to the address of the destination buffer + 0x3FC.

The control word at offset 0x1E8 must be programmed according to Table 8-8 on page 305.

**Table 8-8. Channel Control Word Configuration for Memory Transfer Example**

| Field in DMACHCTL | Bits | Value | Description |
|---|---|---|---|
| DSTINC | 31:30 | 2 | 32-bit destination address increment |
| DSTSIZE | 29:28 | 2 | 32-bit destination data size |
| SRCINC | 27:26 | 2 | 32-bit source address increment |
| SRCSIZE | 25:24 | 2 | 32-bit source data size |
| reserved | 23:18 | 0 | Reserved |
| ARBSIZE | 17:14 | 3 | Arbitrates after 8 transfers |
| XFERSIZE | 13:4 | 255 | Transfer 256 items |
| NXTUSEBURST | 3 | 0 | N/A for this transfer type |
| XFERMODE | 2:0 | 2 | Use Auto-request transfer mode |

#### 8.3.2.3 Start the Transfer

Now the channel is configured and is ready to start.

1. Enable the channel by setting bit 30 of the **DMA Channel Enable Set (DMAENASET)** register.

2. Issue a transfer request by setting bit 30 of the **DMA Channel Software Request (DMASWREQ)** register.

The DMA transfer will now take place. If the interrupt is enabled, then the processor will be notified by interrupt when the transfer is complete. If needed, the status can be checked by reading bit 30 of the **DMAENASET** register. This bit will be automatically cleared when the transfer is complete. The status can also be checked by reading the XFERMODE field of the channel control word at offset 0x1E8. This field will automatically be set to 0 at the end of the transfer.

## 8.3.3 Configuring a Peripheral for Simple Transmit

This example will set up the µDMA controller to transmit a buffer of data to a peripheral. The peripheral has a transmit FIFO with a trigger level of 4. The example peripheral will use µDMA channel 7.

### 8.3.3.1 Configure the Channel Attributes

First, configure the channel attributes:

1. Set bit 7 of the **DMA Channel Priority Set (DMAPRIOSET)** or **DMA Channel Priority Clear (DMAPRIOCLR)** registers to set the channel to High priority or Default priority.
2. Set bit 7 of the **DMA Channel Primary Alternate Clear (DMAALTCLR)** register to select the primary channel control structure for this transfer.
3. Set bit 7 of the **DMA Channel Useburst Clear (DMAUSEBURSTCLR)** register to allow the µDMA controller to respond to single and burst requests.
4. Set bit 7 of the **DMA Channel Request Mask Clear (DMAREQMASKCLR)** register to allow the µDMA controller to recognize requests for this channel.

### 8.3.3.2 Configure the Channel Control Structure

Now the channel control structure must be configured. This example will transfer 64 8-bit bytes from a memory buffer to the peripheral's transmit FIFO register. This example uses µDMA channel 7, and the control structure for channel 7 is at offset 0x070 of the channel control table. The channel control structure for channel 7 is located at the offsets shown in Table 8-9 on page 306.

**Table 8-9. Channel Control Structure Offsets for Channel 7**

| Offset | Description |
|---|---|
| Control Table Base + 0x070 | Channel 7 Source End Pointer |
| Control Table Base + 0x074 | Channel 7 Destination End Pointer |
| Control Table Base + 0x078 | Channel 7 Control Word |

#### *Configure the Source and Destination*

The source and destination end pointers must be set to the last address for the transfer (inclusive). Since the peripheral pointer does not change, it simply points to the peripheral's data register.

1. Set the source end pointer at offset 0x070 to the address of the source buffer + 0x3F.
2. Set the destination end pointer at offset 0x074 to the address of the peripheral's transmit FIFO register.

The control word at offset 0x078 must be programmed according to Table 8-10 on page 307.

**Table 8-10. Channel Control Word Configuration for Peripheral Transmit Example**

| Field in DMACHCTL | Bits | Value | Description |
|---|---|---|---|
| DSTINC | 31:30 | 3 | Destination address does not increment |
| DSTSIZE | 29:28 | 0 | 8-bit destination data size |
| SRCINC | 27:26 | 0 | 8-bit source address increment |
| SRCSIZE | 25:24 | 0 | 8-bit source data size |
| reserved | 23:18 | 0 | Reserved |
| ARBSIZE | 17:14 | 2 | Arbitrates after 4 transfers |
| XFERSIZE | 13:4 | 63 | Transfer 64 items |
| NXTUSEBURST | 3 | 0 | N/A for this transfer type |
| XFERMODE | 2:0 | 1 | Use Basic transfer mode |

**Note:** In this example, it is not important if the peripheral makes a single request or a burst request. Since the peripheral has a FIFO that will trigger at a level of 4, the arbitration size is set to 4. If the peripheral does make a burst request, then 4 bytes will be transferred, which is what the FIFO can accomodate. If the peripheral makes a single request (if there is any space in the FIFO), then one byte will be transferred at a time. If it is important to the application that transfers only be made in bursts, then the channel useburst SET[n] bit should be set by writing a 1 to bit 7 of the **DMA Channel Useburst Set (DMAUSEBURSTSET)** register.

#### 8.3.3.3 Start the Transfer

Now the channel is configured and is ready to start.

1. Enable the channel by setting bit 7 of the **DMA Channel Enable Set (DMAENASET)** register.

The µDMA controller is now configured for transfer on channel 7. The controller will make transfers to the peripheral whenever the peripheral asserts a DMA request. The transfers will continue until the entire buffer of 64 bytes has been transferred. When that happens, the µDMA controller will disable the channel and set the XFERMODE field of the channel control word to 0 (Stopped). The status of the transfer can be checked by reading bit 7 of the **DMA Channel Enable Set (DMAENASET)** register. This bit will be automatically cleared when the transfer is complete. The status can also be checked by reading the XFERMODE field of the channel control word at offset 0x078. This field will automatically be set to 0 at the end of the transfer.

If peripheral interrupts were enabled, then the peripheral interrupt handler would receive an interrupt when the entire transfer was complete.

### 8.3.4 Configuring a Peripheral for Ping-Pong Receive

This example will set up the µDMA controller to continuously receive 8-bit data from a peripheral into a pair of 64 byte buffers. The peripheral has a receive FIFO with a trigger level of 8. The example peripheral will use µDMA channel 8.

#### 8.3.4.1 Configure the Channel Attributes

First, configure the channel attributes:

1. Set bit 8 of the **DMA Channel Priority Set (DMAPRIOSET)** or **DMA Channel Priority Clear (DMAPRIOCLR)** registers to set the channel to High priority or Default priority.

2. Set bit 8 of the **DMA Channel Primary Alternate Clear (DMAALTCLR)** register to select the primary channel control structure for this transfer.

3. Set bit 8 of the **DMA Channel Useburst Clear (DMAUSEBURSTCLR)** register to allow the µDMA controller to respond to single and burst requests.

4. Set bit 8 of the **DMA Channel Request Mask Clear (DMAREQMASKCLR)** register to allow the µDMA controller to recognize requests for this channel.

#### 8.3.4.2 Configure the Channel Control Structure

Now the channel control structure must be configured. This example will transfer 8-bit bytes from the peripheral's receive FIFO register into two memory buffers of 64 bytes each. As data is received, when one buffer is full, the µDMA controller switches to use the other.

To use Ping-Pong buffering, both primary and alternate channel control structures must be used. The primary control structure for channel 8 is at offset 0x080 of the channel control table, and the alternate channel control structure is at offset 0x280. The channel control structures for channel 8 are located at the offsets shown in Table 8-11 on page 308.

**Table 8-11. Primary and Alternate Channel Control Structure Offsets for Channel 8**

| Offset | Description |
|---|---|
| Control Table Base + 0x080 | Channel 8 Primary Source End Pointer |
| Control Table Base + 0x084 | Channel 8 Primary Destination End Pointer |
| Control Table Base + 0x088 | Channel 8 Primary Control Word |
| Control Table Base + 0x280 | Channel 8 Alternate Source End Pointer |
| Control Table Base + 0x284 | Channel 8 Alternate Destination End Pointer |
| Control Table Base + 0x288 | Channel 8 Alternate Control Word |

##### *Configure the Source and Destination*

The source and destination end pointers must be set to the last address for the transfer (inclusive). Since the peripheral pointer does not change, it simply points to the peripheral's data register. Both the primary and alternate sets of pointers must be configured.

1. Set the primary source end pointer at offset 0x080 to the address of the peripheral's receive buffer.

2. Set the primary destination end pointer at offset 0x084 to the address of ping-pong buffer A + 0x3F.

3. Set the alternate source end pointer at offset 0x280 to the address of the peripheral's receive buffer.

4. Set the alternate destination end pointer at offset 0x284 to the address of ping-pong buffer B + 0x3F.

The primary control word at offset 0x088, and the alternate control word at offset 0x288 must be programmed according to Table 8-10 on page 307. Both control words are initially programmed the same way.

1. Program the primary channel control word at offset 0x088 according to Table 8-12 on page 309.

2. Program the alternate channel control word at offset 0x288 according to Table 8-12 on page 309.

**Table 8-12. Channel Control Word Configuration for Peripheral Ping-Pong Receive Example**

| Field in DMACHCTL | Bits | Value | Description |
|---|---|---|---|
| DSTINC | 31:30 | 0 | 8-bit destination address increment |
| DSTSIZE | 29:28 | 0 | 8-bit destination data size |
| SRCINC | 27:26 | 3 | Source address does not increment |
| SRCSIZE | 25:24 | 0 | 8-bit source data size |
| reserved | 23:18 | 0 | Reserved |
| ARBSIZE | 17:14 | 3 | Arbitrates after 8 transfers |
| XFERSIZE | 13:4 | 63 | Transfer 64 items |
| NXTUSEBURST | 3 | 0 | N/A for this transfer type |
| XFERMODE | 2:0 | 3 | Use Ping-Pong transfer mode |

**Note:** In this example, it is not important if the peripheral makes a single request or a burst request. Since the peripheral has a FIFO that will trigger at a level of 8, the arbitration size is set to 8. If the peripheral does make a burst request, then 8 bytes will be transferred, which is what the FIFO can accomodate. If the peripheral makes a single request (if there is any data in the FIFO), then one byte will be transferred at a time. If it is important to the application that transfers only be made in bursts, then the channel useburst SET[n] bit should be set by writing a 1 to bit 8 of the **DMA Channel Useburst Set (DMAUSEBURSTSET)** register.

#### 8.3.4.3 Configure the Peripheral Interrupt

In order to use µDMA Ping-Pong mode, it is best to use an interrupt handler. (It is also possible to use ping-pong mode without interrupts by polling). The interrupt handler will be triggered after each buffer is complete.

1. Configure and enable an interrupt handler for the peripheral.

#### 8.3.4.4 Enable the µDMA Channel

Now the channel is configured and is ready to start.

1. Enable the channel by setting bit 8 of the **DMA Channel Enable Set (DMAENASET)** register.

#### 8.3.4.5 Process Interrupts

The µDMA controller is now configured and enabled for transfer on channel 8. When the peripheral asserts the DMA request signal, the µDMA controller will make transfers into buffer A using the primary channel control structure. When the primary transfer to buffer A is complete, it will switch to the alternate channel control structure and make transfers into buffer B. At the same time, the primary channel control word mode field will be set to indicate Stopped, and an interrupt will be triggered.

When an interrupt is triggered, the interrupt handler must determine which buffer is complete and process the data, or set a flag that the data needs to be processed by non-interrupt buffer processing code. Then the next buffer transfer must be set up.

In the interrupt handler:

1. Read the primary channel control word at offset 0x088 and check the XFERMODE field. If the field is 0, this means buffer A is complete. If buffer A is complete, then:

**a.** Process the newly received data in buffer A, or signal the buffer processing code that buffer A has data available.

**b.** Reprogram the primary channel control word at offset 0x88 according to Table 8-12 on page 309.

**2.** Read the alternate channel control word at offset 0x288 and check the XFERMODE field. If the field is 0, this means buffer B is complete. If buffer B is complete, then:

**a.** Process the newly received data in buffer B, or signal the buffer processing code that buffer B has data available.

**b.** Reprogram the alternate channel control word at offset 0x288 according to Table 8-12 on page 309.

## 8.4 Register Map

Table 8-13 on page 310 lists the µDMA channel control structures and registers. The channel control structure shows the layout of one entry in the channel control table. The channel control table is located in system memory, and the location is determined by the application, that is, the base address is n/a (not applicable). In the table below, the offset for the channel control structures is the offset from the entry in the channel control table. See “Channel Configuration” on page 293 and Table 8-3 on page 294 for a description of how the entries in the channel control table are located in memory. The µDMA register addresses are given as a hexadecimal increment, relative to the µDMA base address of 0x400F.F000. Note that the µDMA module clock must be enabled before the registers can be programmed (see page 229). There must be a delay of 3 system clocks after the µDMA module clock is enabled before any µDMA module registers are accessed.

**Table 8-13. µDMA Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| **µDMA Channel Control Structure** | | | | | |
| 0x000 | DMASRCENDP | R/W | - | DMA Channel Source Address End Pointer | 312 |
| 0x004 | DMADSTENDP | R/W | - | DMA Channel Destination Address End Pointer | 313 |
| 0x008 | DMACHCTL | R/W | - | DMA Channel Control Word | 314 |
| **µDMA Registers** | | | | | |
| 0x000 | DMASTAT | RO | 0x001F.0000 | DMA Status | 318 |
| 0x004 | DMACFG | WO | - | DMA Configuration | 320 |
| 0x008 | DMACTLBASE | R/W | 0x0000.0000 | DMA Channel Control Base Pointer | 321 |
| 0x00C | DMAALTBASE | RO | 0x0000.0200 | DMA Alternate Channel Control Base Pointer | 322 |
| 0x010 | DMAWAITSTAT | RO | 0x0000.0000 | DMA Channel Wait on Request Status | 323 |
| 0x014 | DMASWREQ | WO | - | DMA Channel Software Request | 324 |
| 0x018 | DMAUSEBURSTSET | R/W | 0x0000.0000 | DMA Channel Useburst Set | 325 |
| 0x01C | DMAUSEBURSTCLR | WO | - | DMA Channel Useburst Clear | 327 |
| 0x020 | DMAREQMASKSET | R/W | 0x0000.0000 | DMA Channel Request Mask Set | 328 |

**Table 8-13. µDMA Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x024 | DMAREQMASKCLR | WO | - | DMA Channel Request Mask Clear | 330 |
| 0x028 | DMAENASET | R/W | 0x0000.0000 | DMA Channel Enable Set | 331 |
| 0x02C | DMAENACLR | WO | - | DMA Channel Enable Clear | 333 |
| 0x030 | DMAALTSET | R/W | 0x0000.0000 | DMA Channel Primary Alternate Set | 334 |
| 0x034 | DMAALTCLR | WO | - | DMA Channel Primary Alternate Clear | 336 |
| 0x038 | DMAPRIOSET | R/W | 0x0000.0000 | DMA Channel Priority Set | 337 |
| 0x03C | DMAPRIOCLR | WO | - | DMA Channel Priority Clear | 339 |
| 0x04C | DMAERRCLR | R/W | 0x0000.0000 | DMA Bus Error Clear | 340 |
| 0xFD0 | DMAPeriphID4 | RO | 0x0000.0004 | DMA Peripheral Identification 4 | 346 |
| 0xFE0 | DMAPeriphID0 | RO | 0x0000.0030 | DMA Peripheral Identification 0 | 342 |
| 0xFE4 | DMAPeriphID1 | RO | 0x0000.00B2 | DMA Peripheral Identification 1 | 343 |
| 0xFE8 | DMAPeriphID2 | RO | 0x0000.000B | DMA Peripheral Identification 2 | 344 |
| 0xFEC | DMAPeriphID3 | RO | 0x0000.0000 | DMA Peripheral Identification 3 | 345 |
| 0xFF0 | DMAPCellID0 | RO | 0x0000.000D | DMA PrimeCell Identification 0 | 347 |
| 0xFF4 | DMAPCellID1 | RO | 0x0000.00F0 | DMA PrimeCell Identification 1 | 348 |
| 0xFF8 | DMAPCellID2 | RO | 0x0000.0005 | DMA PrimeCell Identification 2 | 349 |
| 0xFFC | DMAPCellID3 | RO | 0x0000.00B1 | DMA PrimeCell Identification 3 | 350 |

## 8.5 µDMA Channel Control Structure

The µDMA Channel Control Structure holds the DMA transfer settings for a DMA channel. Each channel has two control structures, which are located in a table in system memory. Refer to “Channel Configuration” on page 293 for an explanation of the Channel Control Table and the Channel Control Structure.

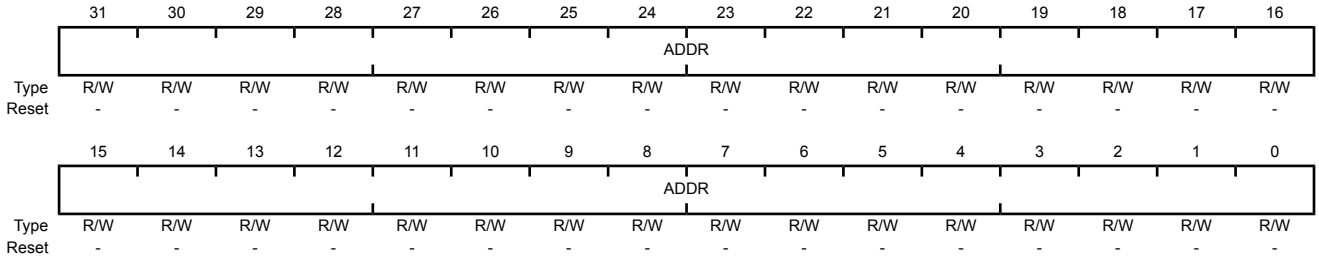
The channel control structure is one entry in the channel control table. There is a primary and alternate structure for each channel. The primary control structures are located at offsets 0x0, 0x10, 0x20 and so on. The alternate control structures are located at offsets 0x200, 0x210, 0x220, and so on.

## Register 1: DMA Channel Source Address End Pointer (DMASRCENDP), offset 0x000

**DMA Channel Source Address End Pointer (DMASRCENDP)** is part of the Channel Control Structure, and is used to specify the source address for a DMA transfer.

DMA Channel Source Address End Pointer (DMASRCENDP)

Base n/a
Offset 0x000
Type R/W, reset -



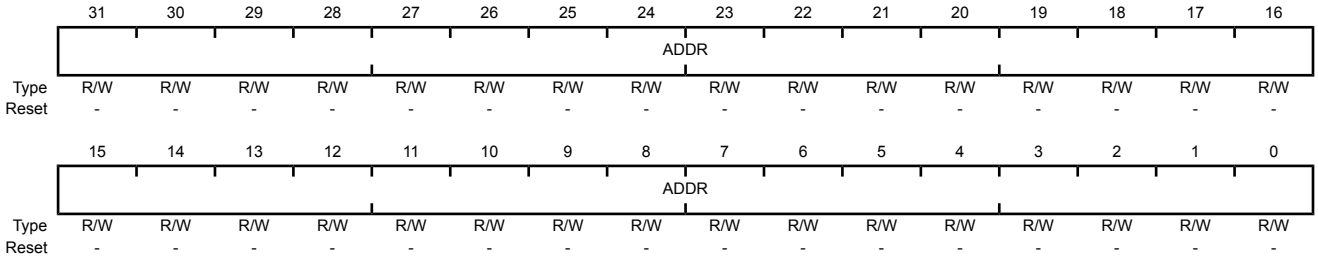

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | ADDR | R/W | - | Source Address End Pointer<br>Points to the last address of the DMA transfer source (inclusive). If the source address is not incrementing, then this points at the source location itself (such as a peripheral data register). |

## Register 2: DMA Channel Destination Address End Pointer (DMADSTENDP), offset 0x004

**DMA Channel Destination Address End Pointer (DMADSTENDP)** is part of the Channel Control Structure, and is used to specify the destination address for a DMA transfer.

DMA Channel Destination Address End Pointer (DMADSTENDP)

Base n/a
Offset 0x004
Type R/W, reset -



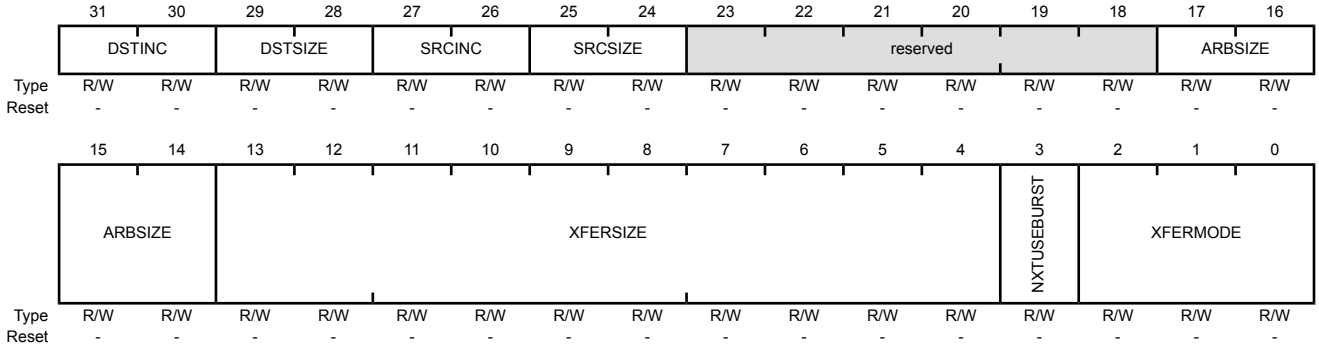

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | ADDR | R/W | - | Destination Address End Pointer<br>Points to the last address of the DMA transfer destination (inclusive). If the destination address is not incrementing, then this points at the destination location itself (such as a peripheral data register). |

## Register 3: DMA Channel Control Word (DMACHCTL), offset 0x008

**DMA Channel Control Word (DMACHCTL)** is part of the Channel Control Structure, and is used to specify parameters of a DMA transfer.

DMA Channel Control Word (DMACHCTL)

Base n/a
Offset 0x008
Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | DSTINC | R/W | - | Destination Address Increment<br>Sets the bits to control the destination address increment.<br>The address increment value must be equal or greater than the value of the destination size (DSTSIZE).<br>Value Description<br>0x0 Byte<br>Increment by 8-bit locations.<br>0x1 Half-word<br>Increment by 16-bit locations.<br>0x2 Word<br>Increment by 32-bit locations.<br>0x3 No increment<br>Address remains set to the value of the Destination Address End Pointer (DMADSTENDP) for the channel. |
| 29:28 | DSTSIZE | R/W | - | Destination Data Size<br>Sets the destination item data size.<br>**Note:** You must set DSTSIZE to be the same as SRCSIZE.<br>Value Description<br>0x0 Byte<br>8-bit data size.<br>0x1 Half-word<br>16-bit data size.<br>0x2 Word<br>32-bit data size.<br>0x3 Reserved |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 27:26 | SRCINC | R/W | - | Source Address Increment<br>Sets the bits to control the source address increment.<br>The address increment value must be equal or greater than the value of the source size (SRCSIZE).<br>Value Description<br>0x0 Byte<br>Increment by 8-bit locations.<br>0x1 Half-word<br>Increment by 16-bit locations.<br>0x2 Word<br>Increment by 32-bit locations.<br>0x3 No increment<br>Address remains set to the value of the Source Address End Pointer (DMASRCENDP) for the channel. |
| 25:24 | SRCSIZE | R/W | - | Source Data Size<br>Sets the source item data size.<br>**Note:** You must set DSTSIZE to be the same as SRCSIZE.<br>Value Description<br>0x0 Byte<br>8-bit data size.<br>0x1 Half-word<br>16-bit data size.<br>0x2 Word<br>32-bit data size.<br>0x3 Reserved |
| 23:18 | reserved | R/W | - | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 17:14 | ARBSIZE | R/W | - | Arbitration Size<br>Sets the number of DMA transfers that can occur before the controller re-arbitrates. The possible arbitration rate settings represent powers of 2 and are shown below.<br>Value: Description<br>0x0: 1 Transfer<br>Arbitrates after each DMA transfer.<br>0x1: 2 Transfers<br>0x2: 4 Transfers<br>0x3: 8 Transfers<br>0x4: 16 Transfers<br>0x5: 32 Transfers<br>0x6: 64 Transfers<br>0x7: 128 Transfers<br>0x8: 256 Transfers<br>0x9: 512 Transfers<br>0xA-0xF: 1024 Transfers<br>This means that no arbitration occurs during the DMA transfer because the maximum transfer size is 1024. |
| 13:4 | XFERSIZE | R/W | - | Transfer Size (minus 1)<br>Sets the total number of items to transfer. The value of this field is 1 less than the number to transfer (value 0 means transfer 1 item). The maximum value for this 10-bit field is 1023 which represents a transfer size of 1024 items.<br>The transfer size is the number of items, not the number of bytes. If the data size is 32 bits, then this value is the number of 32-bit words to transfer.<br>The controller updates this field immediately prior to it entering the arbitration process, so it contains the number of outstanding DMA items that are necessary to complete the DMA cycle. |
| 3 | NXTUSEBURST | R/W | - | Next Useburst<br>Controls whether the useburst SET[n] bit is automatically set for the last transfer of a peripheral scatter-gather operation. Normally, for the last transfer, if the number of remaining items to transfer is less than the arbitration size, the controller will use single transfers to complete the transaction. If this bit is set, then the controller will only use a burst transfer to complete the last transfer. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2:0 | XFERMODE | R/W | - | DMA Transfer Mode<br>Since this register is in system RAM, it has no reset value. Therefore, this field should be initialized to 0 before the channel is enabled.<br>The operating mode of the DMA cycle. Refer to "Transfer Modes" on page 295 for a detailed explanation of transfer modes.<br>Value Description<br>0x0 Stop<br>Channel is stopped, or configuration data is invalid.<br>0x1 Basic<br>The controller must receive a new request, prior to it entering the arbitration process, to enable the DMA cycle to complete.<br>0x2 Auto-Request<br>The initial request (software- or peripheral-initiated) is sufficient to complete the entire transfer of XFERSIZE items without any further requests.<br>0x3 Ping-Pong<br>The controller performs a DMA cycle using one of the channel control structures. After the DMA cycle completes, it performs a DMA cycle using the other channel control structure. After the next DMA cycle completes (and provided that the host processor has updated the original channel control data structure), it performs a DMA cycle using the original channel control data structure. The controller continues to perform DMA cycles until it either reads an invalid data structure or the host processor changes this field to 0x1 or 0x2. See "Ping-Pong" on page 295.<br>0x4 Memory Scatter-Gather<br>When the controller operates in Memory Scatter-Gather mode, you must only use this value in the primary channel control data structure. See "Memory Scatter-Gather" on page 296.<br>0x5 Alternate Memory Scatter-Gather<br>When the controller operates in Memory Scatter-Gather mode, you must only use this value in the alternate channel control data structure.<br>0x6 Peripheral Scatter-Gather<br>When the controller operates in Peripheral Scatter-Gather mode, you must only use this value in the primary channel control data structure. See "Peripheral Scatter-Gather" on page 300.<br>0x7 Alternate Peripheral Scatter-Gather<br>When the controller operates in Peripheral Scatter-Gather mode, you must only use this value in the alternate channel control data structure. |

## 8.6 µDMA Register Descriptions

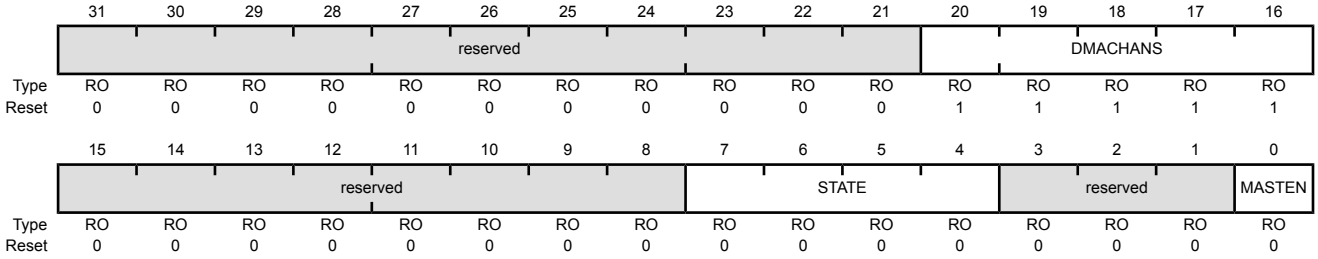
The register addresses given are relative to the µDMA base address of 0x400F.F000.

## Register 4: DMA Status (DMASTAT), offset 0x000

The **DMA Status (DMASTAT)** register returns the status of the controller. You cannot read this register when the controller is in the reset state.

DMA Status (DMASTAT)

Base 0x400F.F000
Offset 0x000
Type RO, reset 0x001F.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:21 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 20:16 | DMACHANS | RO | 0x1F | Available DMA Channels Minus 1<br>This bit contains a value equal to the number of DMA channels the controller is configured to use, minus one. That is, 32 DMA channels. |
| 15:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

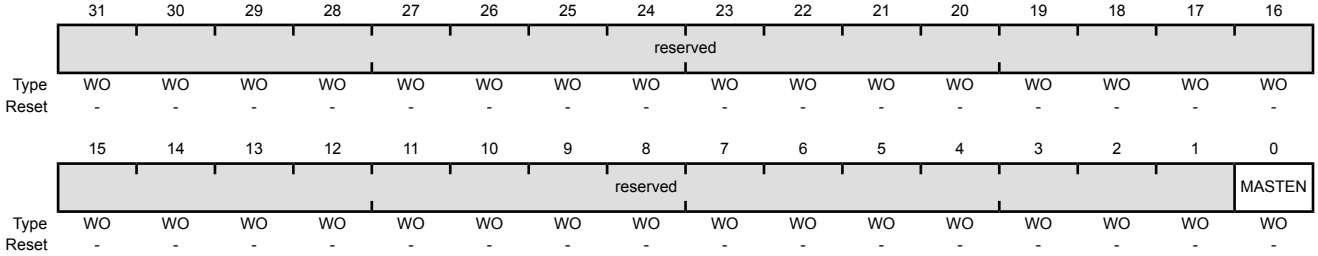
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:4 | STATE | RO | 0x00 | Control State Machine State<br>Current state of the control state machine. State can be one of the following.<br><br>Value — Description<br>0x0 — Idle<br>0x1 — Read Chan Control Data<br>Reading channel controller data.<br>0x2 — Read Source End Ptr<br>Reading source end pointer.<br>0x3 — Read Dest End Ptr<br>Reading destination end pointer.<br>0x4 — Read Source Data<br>Reading source data.<br>0x5 — Write Dest Data<br>Writing destination data.<br>0x6 — Wait for Req Clear<br>Waiting for DMA request to clear.<br>0x7 — Write Chan Control Data<br>Writing channel controller data.<br>0x8 — Stalled<br>0x9 — Done<br>0xA-0xF — Undefined |
| 3:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | MASTEN | RO | 0x00 | Master Enable<br>Returns status of the controller.<br><br>Value — Description<br>0 — Disabled<br>1 — Enabled |

## Register 5: DMA Configuration (DMACFG), offset 0x004

The **DMACFG** register controls the configuration of the controller.

DMA Configuration (DMACFG)

Base 0x400F.F000
Offset 0x004
Type WO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | WO | - | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | MASTEN | WO | - | Controller Master Enable<br>Enables the controller.<br>Value Description<br>0 Disables<br>1 Enables |

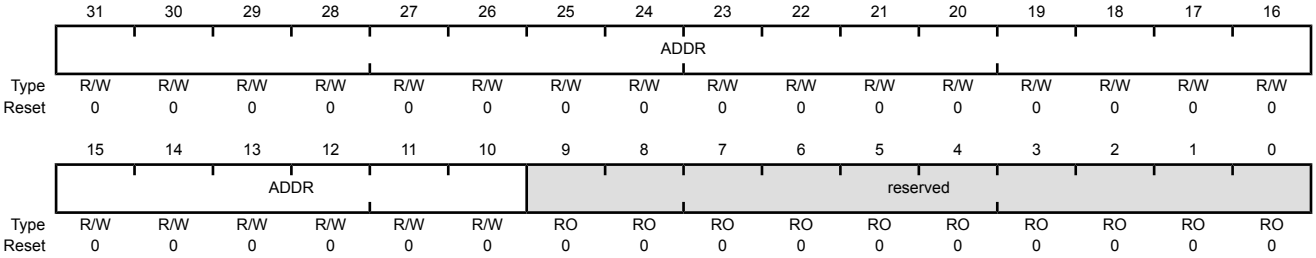
## Register 6: DMA Channel Control Base Pointer (DMACTLBASE), offset 0x008

The **DMACTLBASE** register must be configured so that the base pointer points to a location in system memory.

The amount of system memory that you must assign to the controller depends on the number of DMA channels used and whether you configure it to use the alternate channel control data structure. See “Channel Configuration” on page 293 for details about the Channel Control Table. The base address must be aligned on a 1024-byte boundary. You cannot read this register when the controller is in the reset state.

DMA Channel Control Base Pointer (DMACTLBASE)

Base 0x400F.F000
Offset 0x008
Type R/W, reset 0x0000.0000



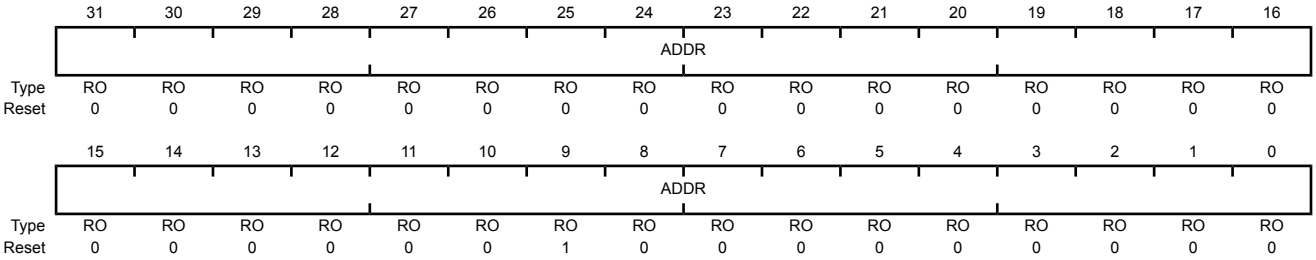

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | ADDR | R/W | 0x00 | Channel Control Base Address<br>Pointer to the base address of the channel control table. The base address must be 1024-byte aligned. |
| 9:0 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 7: DMA Alternate Channel Control Base Pointer (DMAALTBASE), offset 0x00C

The **DMAALTBASE** register returns the base address of the alternate channel control data. This register removes the necessity for application software to calculate the base address of the alternate channel control structures. You cannot read this register when the controller is in the reset state.

DMA Alternate Channel Control Base Pointer (DMAALTBASE)

Base 0x400F.F000
Offset 0x00C
Type RO, reset 0x0000.0200



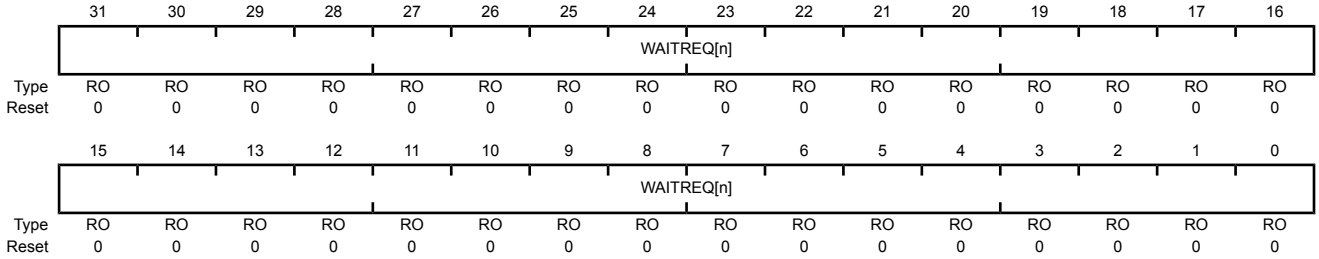

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | ADDR | RO | 0x200 | Alternate Channel Address Pointer<br>Provides the base address of the alternate channel control structures. |

## Register 8: DMA Channel Wait on Request Status (DMAWAITSTAT), offset 0x010

This read-only register indicates that the µDMA channel is waiting on a request. A peripheral can pull this Low to hold off the µDMA from performing a single request until the peripheral is ready for a burst request. The use of this feature is dependent on the design of the peripheral and is used to enhance performance of the µDMA with that peripheral. You cannot read this register when the controller is in the reset state.

DMA Channel Wait on Request Status (DMAWAITSTAT)

Base 0x400F.F000
Offset 0x010
Type RO, reset 0x0000.0000



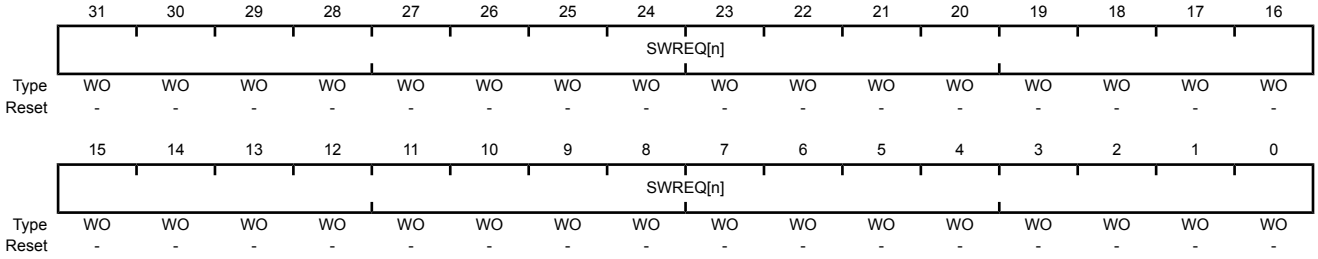

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WAITREQ[n] | RO | 0x00 | Channel [n] Wait Status<br>Channel wait on request status. For each channel 0 through 31, a 1 in the corresponding bit field indicates that the channel is waiting on a request. |

## Register 9: DMA Channel Software Request (DMASWREQ), offset 0x014

Each bit of the **DMASWREQ** register represents the corresponding DMA channel. When you set a bit, it generates a request for the specified DMA channel.

DMA Channel Software Request (DMASWREQ)

Base 0x400F.F000
Offset 0x014
Type WO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SWREQ[n] | WO | - | Channel [n] Software Request<br>For each channel 0 through 31, write a 1 to the corresponding bit field to generate a software DMA request for that DMA channel. Writing a 0 does not create a DMA request for the corresponding channel. |

## Register 10: DMA Channel Useburst Set (DMAUSEBURSTSET), offset 0x018

Each bit of the **DMAUSEBURSTSET** register represents the corresponding DMA channel. Writing a 1 disables the peripheral's single request input from generating requests, and therefore only the peripheral's burst request generates requests. Reading the register returns the status of useburst.

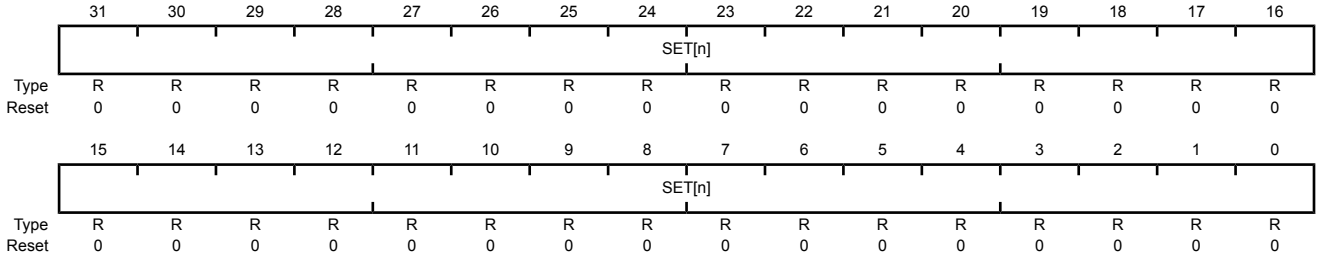
When there are fewer items remaining to transfer than the arbitration (burst) size, the controller automatically clears the useburst bit to 0. This enables the remaining items to transfer using single requests. This bit should not be set for a peripheral's channel that does not support the burst request model.

Refer to “Request Types” on page 293 for more details about request types.

### Reads

DMA Channel Useburst Set (DMAUSEBURSTSET)

Base 0x400F.F000
Offset 0x018
Type RO, reset 0x0000.0000

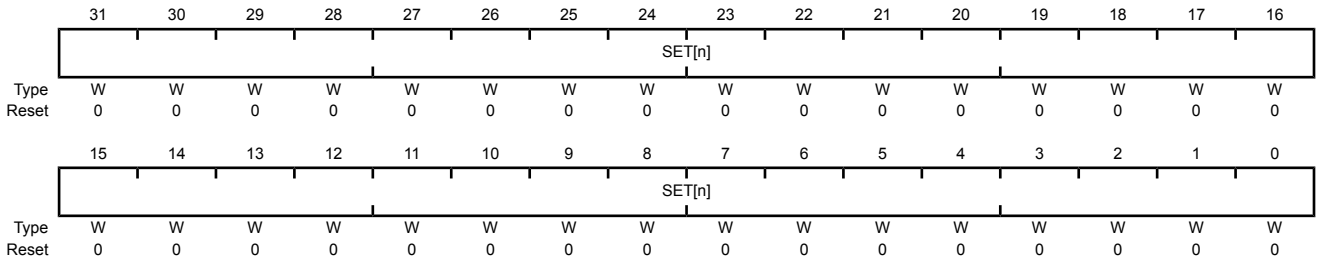



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SET[n] | R | 0x00 | Channel [n] Useburst Status<br>Returns the useburst status of channel [n].<br><br>Value Description<br>0 Single and Burst<br>DMA channel [n] responds to single or burst requests.<br>1 Burst Only<br>DMA channel [n] responds only to burst requests. |

### Writes

DMA Channel Useburst Set (DMAUSEBURSTSET)

Base 0x400F.F000
Offset 0x018
Type WO, reset 0x0000.0000



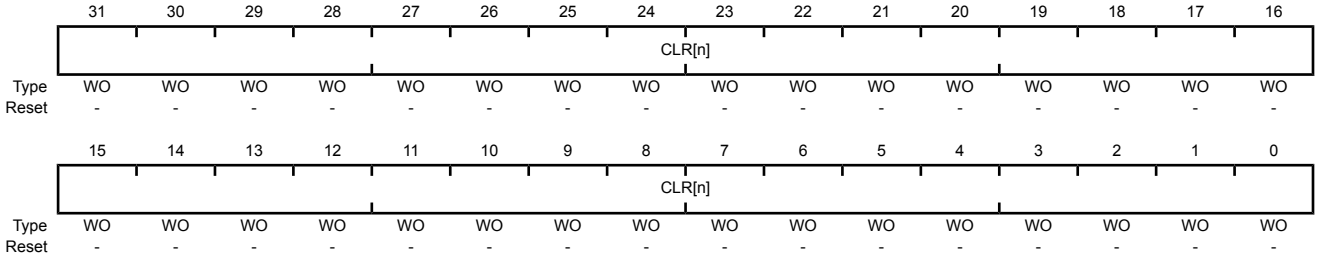
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SET[n] | W | 0x00 | Channel [n] Useburst Set<br>Sets useburst bit on channel [n].<br><br>Value Description<br>0 No Effect<br>Use the **DMAUSEBURSTCLR** register to clear bit [n] to 0.<br>1 Burst Only<br>DMA channel [n] responds only to burst requests. |

## Register 11: DMA Channel Useburst Clear (DMAUSEBURSTCLR), offset 0x01C

Each bit of the **DMAUSEBURSTCLR** register represents the corresponding DMA channel. Writing a 1 enables `dma_sreq[n]` to generate requests.

DMA Channel Useburst Clear (DMAUSEBURSTCLR)

Base 0x400F.F000
Offset 0x01C
Type WO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | CLR[n] | WO | - | Channel [n] Useburst Clear<br>Clears useburst bit on channel [n].<br><br>Value Description<br>0 No Effect<br>Use the **DMAUSEBURSTSET** to set bit [n] to 1.<br>1 Single and Burst<br>DMA channel [n] responds to single and burst requests. |

## Register 12: DMA Channel Request Mask Set (DMAREQMASKSET), offset 0x020

Each bit of the **DMAREQMASKSET** register represents the corresponding DMA channel. Writing a 1 disables DMA requests for the channel. Reading the register returns the request mask status. When a µDMA channel's request is masked, that means the peripheral can no longer request µDMA transfers. The channel can then be used for software-initiated transfers.

### Reads

DMA Channel Request Mask Set (DMAREQMASKSET)

Base 0x400F.F000
Offset 0x020
Type RO, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SET[n] | | | | | | | | | | | | | | | |
| Type | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SET[n] | | | | | | | | | | | | | | | |
| Type | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SET[n] | R | 0x00 | Channel [n] Request Mask Status<br>Returns the channel request mask status.<br><br>Value Description<br>0 Enabled<br>External requests are not masked for channel [n].<br>1 Masked<br>External requests are masked for channel [n]. |

### Writes

DMA Channel Request Mask Set (DMAREQMASKSET)

Base 0x400F.F000
Offset 0x020
Type WO, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SET[n] | | | | | | | | | | | | | | | |
| Type | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SET[n] | | | | | | | | | | | | | | | |
| Type | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

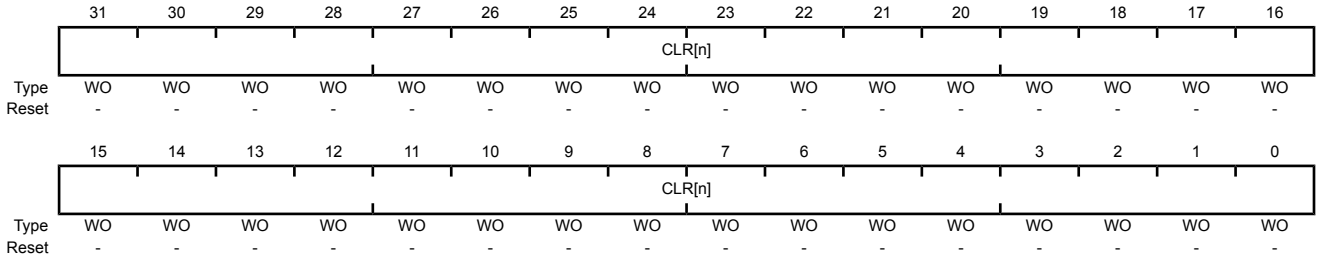
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SET[n] | W | 0x00 | Channel [n] Request Mask Set<br>Masks (disables) the corresponding channel [n] from generating DMA requests.<br><br>Value Description<br>0 No Effect<br>Use the **DMAREQMASKCLR** register to clear the request mask.<br>1 Masked<br>Masks (disables) DMA requests on channel [n]. |

## Register 13: DMA Channel Request Mask Clear (DMAREQMASKCLR), offset 0x024

Each bit of the **DMAREQMASKCLR** register represents the corresponding DMA channel. Writing a 1 clears the request mask for the channel, and enables the channel to receive DMA requests.

DMA Channel Request Mask Clear (DMAREQMASKCLR)

Base 0x400F.F000
Offset 0x024
Type WO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | CLR[n] | WO | - | Channel [n] Request Mask Clear<br>Set the appropriate bit to clear the DMA request mask for channel [n]. This will enable DMA requests for the channel.<br><br>Value Description<br>0 No Effect<br>Use the **DMAREQMASKSET** register to set the request mask.<br>1 Clear Mask<br>Clears the request mask for the DMA channel. This enables DMA requests for the channel. |

## Register 14: DMA Channel Enable Set (DMAENASET), offset 0x028

Each bit of the **DMAENASET** register represents the corresponding DMA channel. Writing a 1 enables the DMA channel. Reading the register returns the enable status of the channels. If a channel is enabled but the request mask is set (**DMAREQMASKSET**), then the channel can be used for software-initiated transfers.

### Reads

DMA Channel Enable Set (DMAENASET)

Base 0x400F.F000
Offset 0x028
Type RO, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SET[n] | | | | | | | | | | | | | | | |
| Type | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SET[n] | | | | | | | | | | | | | | | |
| Type | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SET[n] | R | 0x00 | Channel [n] Enable Status<br>Returns the enable status of the channels.<br><br>Value Description<br>0 Disabled<br>1 Enabled |

### Writes

DMA Channel Enable Set (DMAENASET)

Base 0x400F.F000
Offset 0x028
Type WO, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SET[n] | | | | | | | | | | | | | | | |
| Type | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SET[n] | | | | | | | | | | | | | | | |
| Type | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

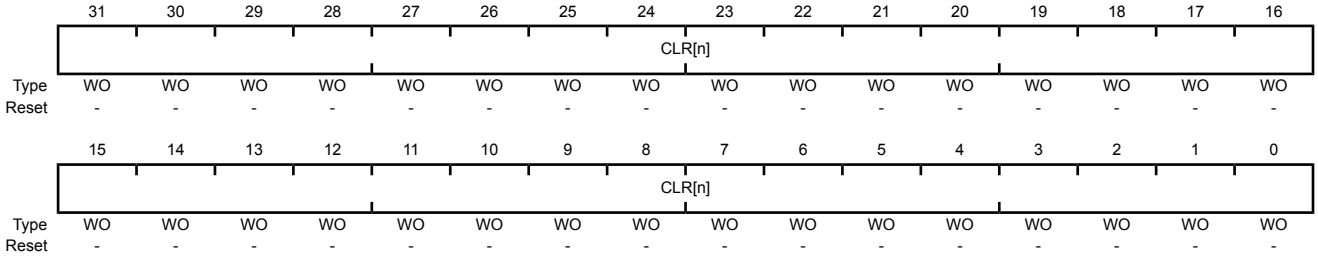
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SET[n] | W | 0x00 | Channel [n] Enable Set<br>Enables the corresponding channels.<br>**Note:** The controller disables a channel when it completes the DMA cycle.<br>Value Description<br>0 No Effect<br>Use the **DMAENACLR** register to disable a channel.<br>1 Enable<br>Enables channel [n]. |

## Register 15: DMA Channel Enable Clear (DMAENACLR), offset 0x02C

Each bit of the **DMAENACLR** register represents the corresponding DMA channel. Writing a 1 disables the specified DMA channel.

DMA Channel Enable Clear (DMAENACLR)

Base 0x400F.F000
Offset 0x02C
Type WO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | CLR[n] | WO | - | Clear Channel [n] Enable<br>Set the appropriate bit to disable the corresponding DMA channel.<br>**Note:** The controller disables a channel when it completes the DMA cycle.<br><br>Value Description<br>0 No Effect<br>Use the **DMAENASET** register to enable DMA channels.<br>1 Disable<br>Disables channel [n]. |

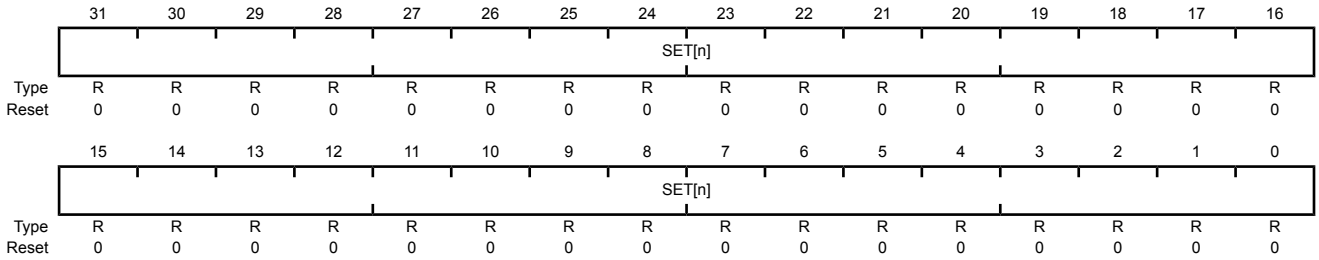
## Register 16: DMA Channel Primary Alternate Set (DMAALTSET), offset 0x030

Each bit of the **DMAALTSET** register represents the corresponding DMA channel. Writing a 1 configures the DMA channel to use the alternate control data structure. Reading the register returns the status of which control data structure is in use for the corresponding DMA channel.

### Reads

DMA Channel Primary Alternate Set (DMAALTSET)

Base 0x400F.F000
Offset 0x030
Type RO, reset 0x0000.0000



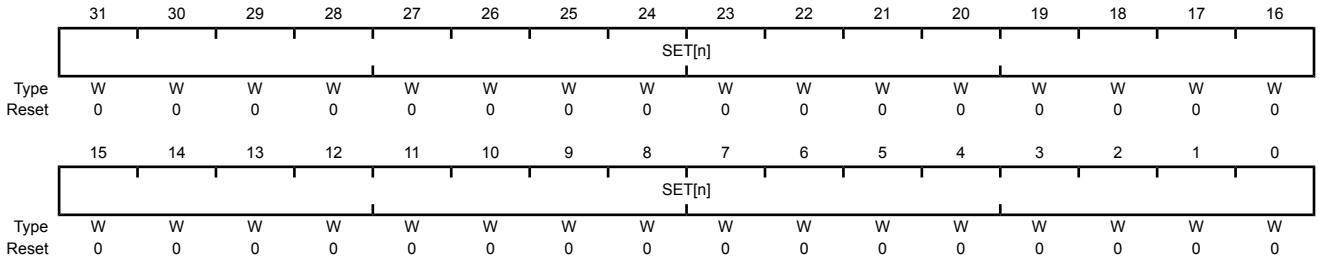

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SET[n] | R | 0x00 | Channel [n] Alternate Status<br>Returns the channel control data structure status. |

| Value | Description |
|---|---|
| 0 | Primary<br>DMA channel [n] is using the primary control structure. |
| 1 | Alternate<br>DMA channel [n] is using the alternate control structure. |

### Writes

DMA Channel Primary Alternate Set (DMAALTSET)

Base 0x400F.F000
Offset 0x030
Type WO, reset 0x0000.0000



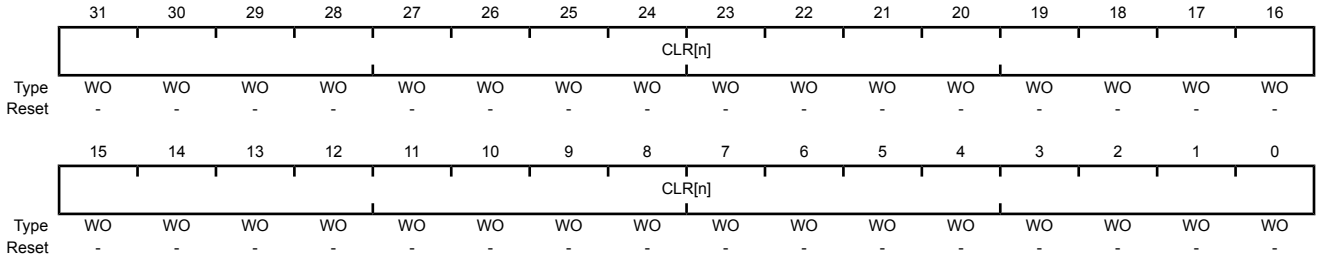
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SET[n] | W | 0x00 | Channel [n] Alternate Set<br>Selects the alternate channel control data structure for the corresponding DMA channel.<br>**Note:** For Ping-Pong and Scatter-Gather DMA cycle types, the controller automatically sets these bits to select the alternate channel control data structure.<br>Value Description<br>0 No Effect<br>Use the **DMAALTCLR** register to set bit [n] to 0.<br>1 Alternate<br>Selects the alternate control data structure for channel [n]. |

## Register 17: DMA Channel Primary Alternate Clear (DMAALTCLR), offset 0x034

Each bit of the **DMAALTCLR** register represents the corresponding DMA channel. Writing a 1 configures the DMA channel to use the primary control data structure.

DMA Channel Primary Alternate Clear (DMAALTCLR)

Base 0x400F.F000
Offset 0x034
Type WO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | CLR[n] | WO | - | Channel [n] Alternate Clear<br>Set the appropriate bit to select the primary control data structure for the corresponding DMA channel.<br>**Note:** For Ping-Pong and Scatter-Gather DMA cycle types, the controller sets these bits to select the primary channel control data structure.<br>Value Description<br>0 No Effect<br>Use the **DMAALTSET** register to select the alternate control data structure.<br>1 Primary<br>Selects the primary control data structure for channel [n]. |

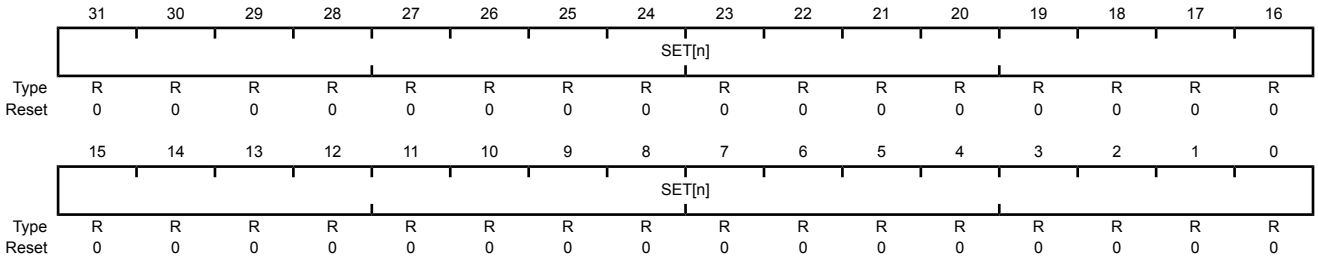
## Register 18: DMA Channel Priority Set (DMAPRIOSET), offset 0x038

Each bit of the the **DMAPRIOSET** register represents the corresponding DMA channel. Writing a 1 configures the DMA channel to have a high priority level. Reading the register returns the status of the channel priority mask.

### Reads

DMA Channel Priority Set (DMAPRIOSET)

Base 0x400F.F000
Offset 0x038
Type RO, reset 0x0000.0000

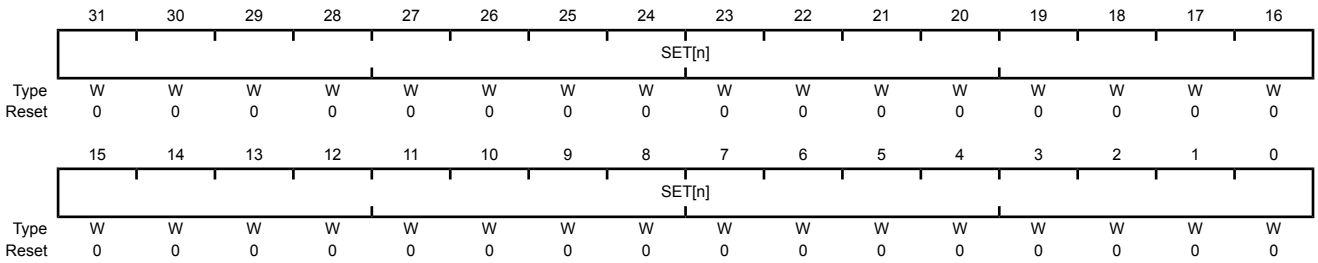



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SET[n] | R | 0x00 | Channel [n] Priority Status<br>Returns the channel priority status.<br><br>Value Description<br>0 Default Priority<br>DMA channel [n] is using the default priority level.<br>1 High Priority<br>DMA channel [n] is using a High Priority level. |

### Writes

DMA Channel Priority Set (DMAPRIOSET)

Base 0x400F.F000
Offset 0x038
Type WO, reset 0x0000.0000



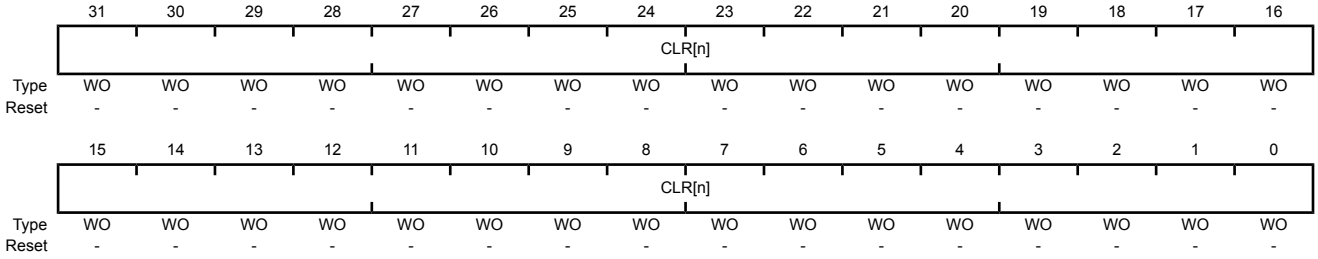
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SET[n] | W | 0x00 | Channel [n] Priority Set<br>Sets the channel priority to high.<br><br>Value Description<br>0 No Effect<br>Use the **DMAPRIOCLR** register to set channel [n] to the default priority level.<br>1 High Priority<br>Sets DMA channel [n] to a High Priority level. |

## Register 19: DMA Channel Priority Clear (DMAPRIOCLR), offset 0x03C

Each bit of the **DMAPRIOCLR** register represents the corresponding DMA channel. Writing a 1 configures the DMA channel to have the default priority level.

DMA Channel Priority Clear (DMAPRIOCLR)

Base 0x400F.F000
Offset 0x03C
Type WO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | CLR[n] | WO | - | Channel [n] Priority Clear<br>Set the appropriate bit to clear the high priority level for the specified DMA channel.<br><br>Value Description<br>0 No Effect<br>Use the **DMAPRIOSET** register to set channel [n] to the High priority level.<br>1 Default Priority<br>Sets DMA channel [n] to a Default priority level. |

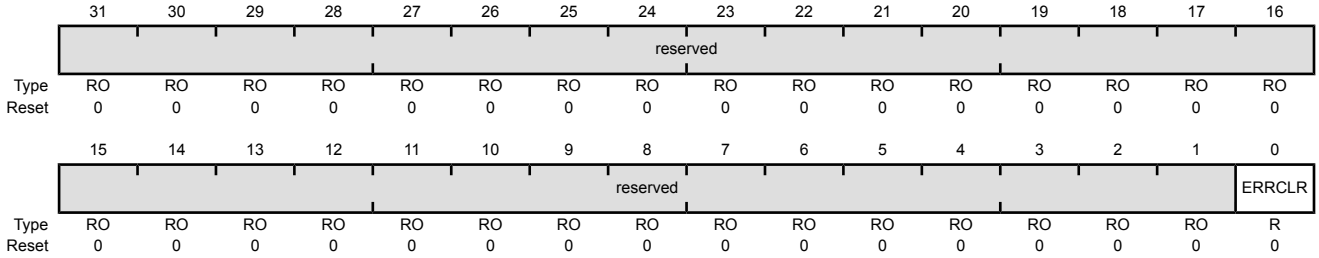
## Register 20: DMA Bus Error Clear (DMAERRCLR), offset 0x04C

The **DMAERRCLR** register is used to read and clear the DMA bus error status. The error status will be set if the µDMA controller encountered a bus error while performing a DMA transfer. If a bus error occurs on a channel, that channel will be automatically disabled by the µDMA controller. The other channels are unaffected.

### Reads

DMA Bus Error Clear (DMAERRCLR)

Base 0x400F.F000
Offset 0x04C
Type RO, reset 0x0000.0000

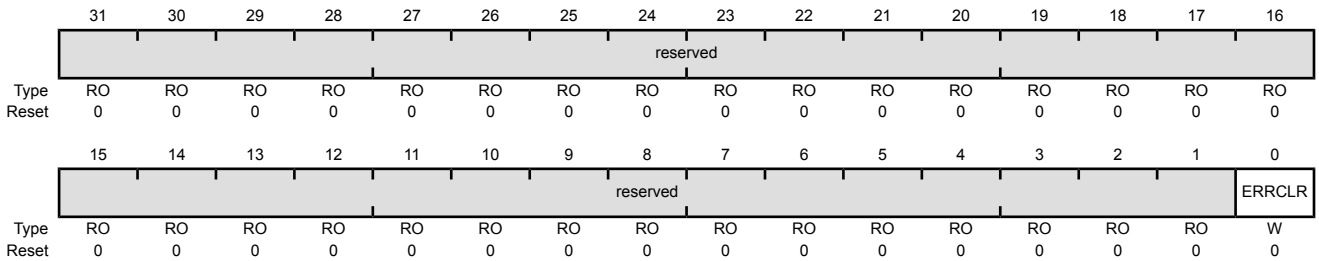



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | ERRCLR | R | 0 | DMA Bus Error Status<br>Value Description<br>0 Low<br>No bus error is pending.<br>1 High<br>Bus error is pending. |

### Writes

DMA Bus Error Clear (DMAERRCLR)

Base 0x400F.F000
Offset 0x04C
Type WO, reset 0x0000.0000



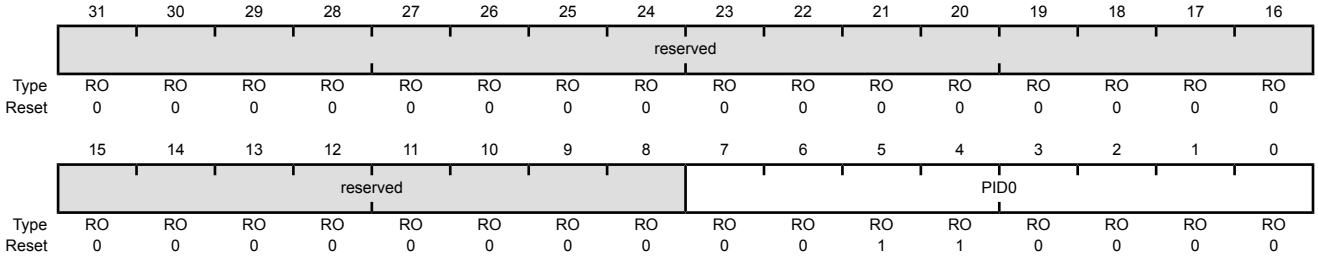
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | ERRCLR | W | 0 | DMA Bus Error Clear<br>Clears the bus error.<br><br>Value Description<br>0 No Effect<br>Bus error status is unchanged.<br>1 Clear<br>Clears a pending bus error. |

## Register 21: DMA Peripheral Identification 0 (DMAPeriphID0), offset 0xFE0

The **DMAPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

DMA Peripheral Identification 0 (DMAPeriphID0)

Base 0x400F.F000
Offset 0xFE0
Type RO, reset 0x0000.0030



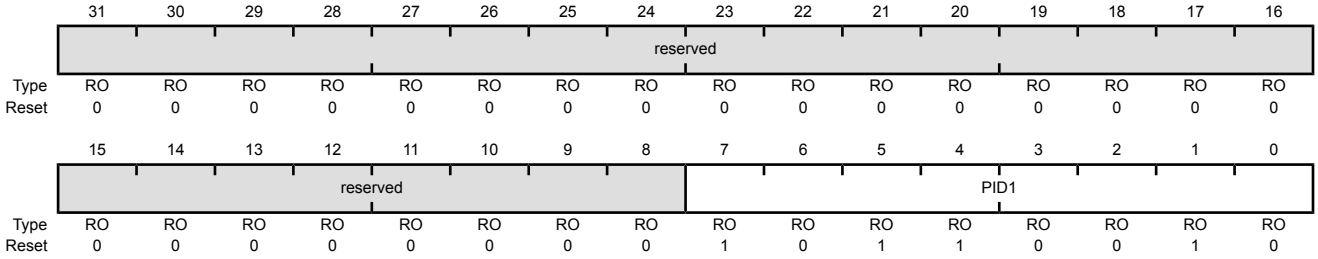

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID0 | RO | 0x30 | DMA Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

## Register 22: DMA Peripheral Identification 1 (DMAPeriphID1), offset 0xFE4

The **DMAPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

DMA Peripheral Identification 1 (DMAPeriphID1)

Base 0x400F.F000
Offset 0xFE4
Type RO, reset 0x0000.00B2



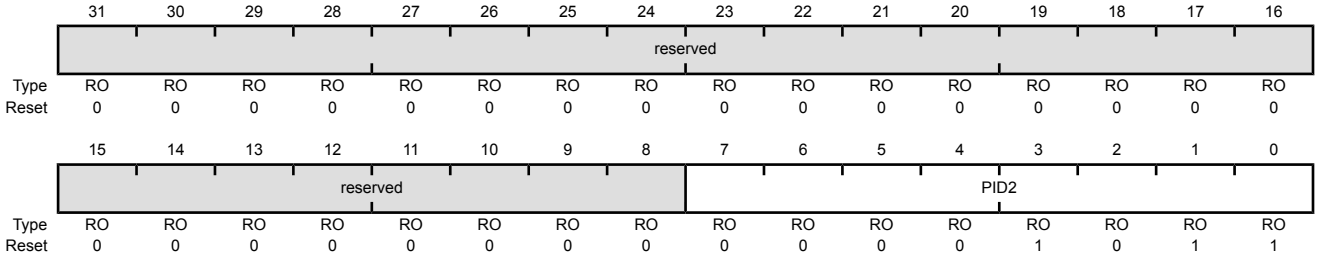

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID1 | RO | 0xB2 | DMA Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

## Register 23: DMA Peripheral Identification 2 (DMAPeriphID2), offset 0xFE8

The **DMAPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

DMA Peripheral Identification 2 (DMAPeriphID2)

Base 0x400F.F000
Offset 0xFE8
Type RO, reset 0x0000.000B



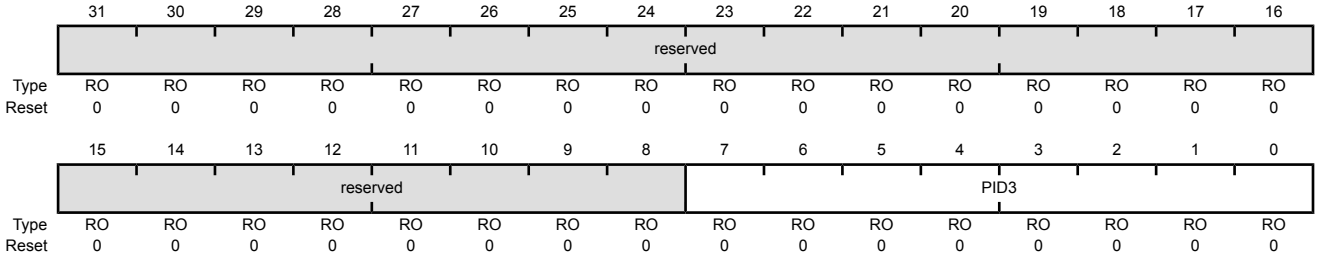

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID2 | RO | 0x0B | DMA Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 24: DMA Peripheral Identification 3 (DMAPeriphID3), offset 0xFEC

The **DMAPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

DMA Peripheral Identification 3 (DMAPeriphID3)

Base 0x400F.F000
Offset 0xFEC
Type RO, reset 0x0000.0000



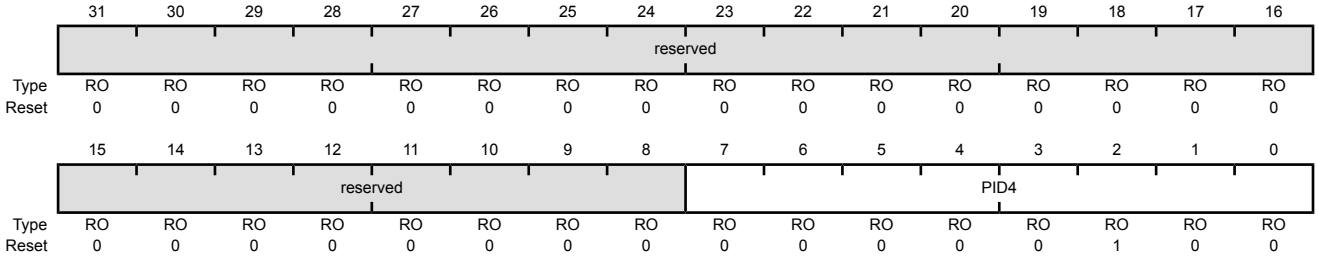

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID3 | RO | 0x00 | DMA Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

## Register 25: DMA Peripheral Identification 4 (DMAPeriphID4), offset 0xFD0

The **DMAPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

DMA Peripheral Identification 4 (DMAPeriphID4)

Base 0x400F.F000
Offset 0xFD0
Type RO, reset 0x0000.0004



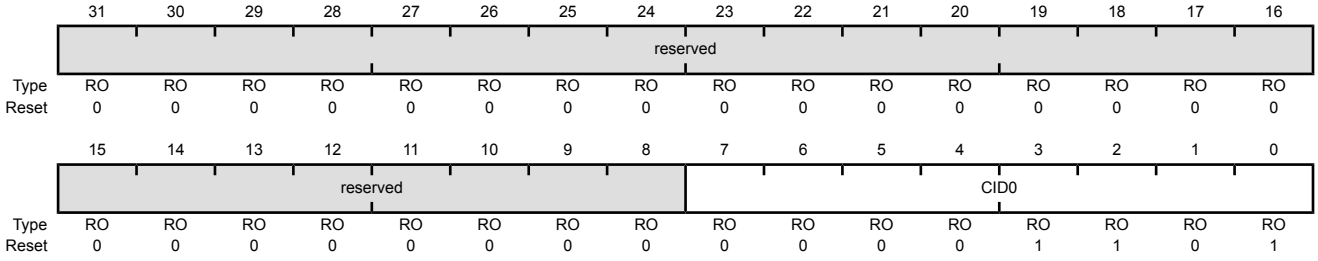

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID4 | RO | 0x04 | DMA Peripheral ID Register<br>Can be used by software to identify the presence of this peripheral. |

## Register 26: DMA PrimeCell Identification 0 (DMAPCellID0), offset 0xFF0

The **DMAPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

DMA PrimeCell Identification 0 (DMAPCellID0)

Base 0x400F.F000
Offset 0xFF0
Type RO, reset 0x0000.000D



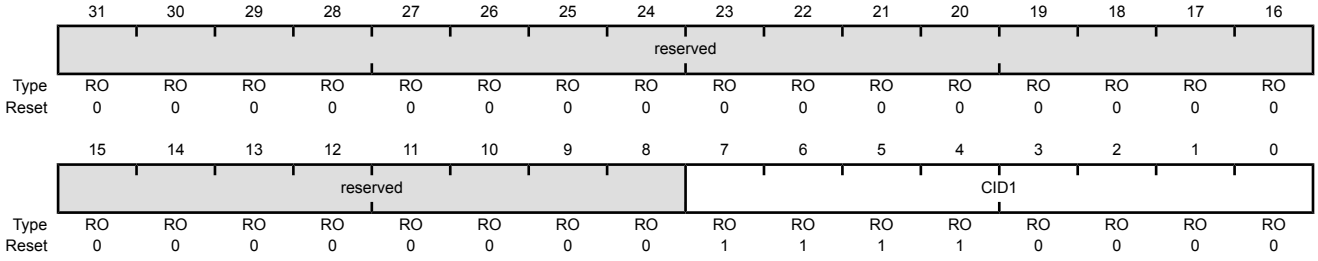

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID0 | RO | 0x0D | DMA PrimeCell ID Register[7:0]<br>Provides software a standard cross-peripheral identification system. |

## Register 27: DMA PrimeCell Identification 1 (DMAPCellID1), offset 0xFF4

The **DMAPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

DMA PrimeCell Identification 1 (DMAPCellID1)

Base 0x400F.F000
Offset 0xFF4
Type RO, reset 0x0000.00F0



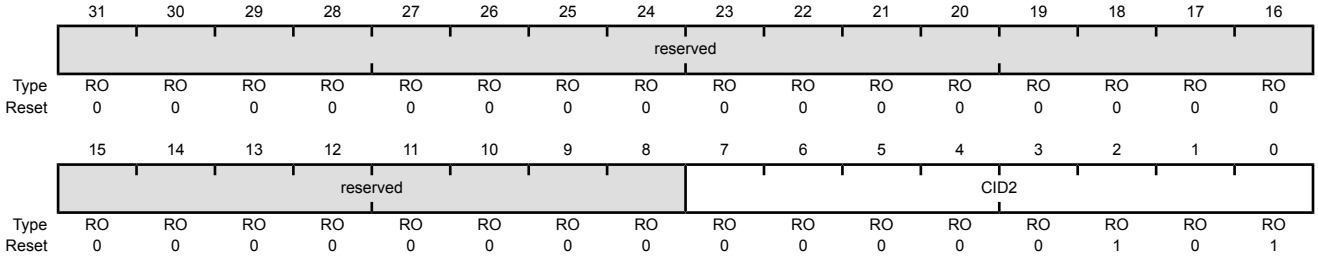

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID1 | RO | 0xF0 | DMA PrimeCell ID Register[15:8]<br>Provides software a standard cross-peripheral identification system. |

## Register 28: DMA PrimeCell Identification 2 (DMAPCellID2), offset 0xFF8

The **DMAPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

DMA PrimeCell Identification 2 (DMAPCellID2)

Base 0x400F.F000
Offset 0xFF8
Type RO, reset 0x0000.0005



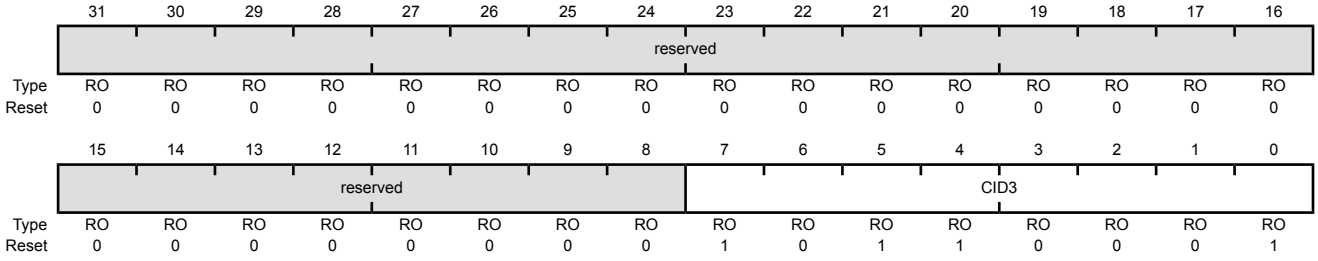

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID2 | RO | 0x05 | DMA PrimeCell ID Register[23:16]<br>Provides software a standard cross-peripheral identification system. |

## Register 29: DMA PrimeCell Identification 3 (DMAPCellID3), offset 0xFFC

The **DMAPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

DMA PrimeCell Identification 3 (DMAPCellID3)

Base 0x400F.F000
Offset 0xFFC
Type RO, reset 0x0000.00B1




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID3 | RO | 0xB1 | DMA PrimeCell ID Register[31:24]<br>Provides software a standard cross-peripheral identification system. |

# 9 General-Purpose Input/Outputs (GPIOs)

The GPIO module is composed of five physical GPIO blocks, each corresponding to an individual GPIO port (Port A, Port B, Port C, Port D, Port E). The GPIO module supports 1-33 programmable input/output pins, depending on the peripherals being used.

The GPIO module has the following features:

- 1-33 GPIOs, depending on configuration
- 5-V-tolerant in input configuration
- Two means of port access: either Advanced High-Performance Bus (AHB) with better back-to-back access performance, or the legacy Advanced Peripheral Bus (APB) for backwards-compatibility with existing code
- Fast toggle capable of a change every clock cycle for ports on AHB, every two clock cycles for ports on APB
- Programmable control for GPIO interrupts
  - Interrupt generation masking
  - Edge-triggered on rising, falling, or both
  - Level-sensitive on High or Low values
- Bit masking in both read and write operations through address lines
- Can initiate an ADC sample sequence
- Pins configured as digital inputs are Schmitt-triggered.
- Programmable control for GPIO pad configuration
  - Weak pull-up or pull-down resistors
  - 2-mA, 4-mA, and 8-mA pad drive for digital communication; up to four pads can be configured with an 18-mA pad drive for high-current applications
  - Slew rate control for the 8-mA drive
  - Open drain enables
  - Digital input enables

## 9.1 Signal Description

GPIO signals have alternate hardware functions. Table 9-3 on page 353 lists the GPIO pins and their analog and digital alternate functions. The AINx analog signals are not 5-V tolerant and go through an isolation circuit before reaching their circuitry. These signals are configured by clearing the corresponding DEN bit in the **GPIO Digital Enable (GPIODEN)** register and setting the corresponding AMSEL bit in the **GPIO Analog Mode Select (GPIOAMSEL)** register. Other analog signals are 5-V tolerant and are connected directly to their circuitry (). These signals are configured by clearing the DEN bit in the **GPIO Digital Enable (GPIODEN)** register. The digital alternate hardware functions

are enabled by setting the appropriate bit in the **GPIO Alternate Function Select (GPIOAFSEL)** and **GPIODEN** registers and configuring the PMCx bit field in the **GPIO Port Control (GPIOPCTL)** register to the numeric enoding shown in the table below. Note that each pin must be programmed individually; no type of grouping is implied by the columns in the table.

**Important:** All GPIO pins are configured as GPIOs and tri-stated by default (**GPIOAFSEL**=0, **GPIODEN**=0, **GPIOPDR**=0, **GPIOPUR**=0, and **GPIOPCTL**=0, with the exception of the four JTAG/SWD pins (shown in the table below). A Power-On-Reset ($\overline{POR}$) or asserting $\overline{RST}$ puts the pins back to their default state.

**Table 9-1. GPIO Pins With Non-Zero Reset Values**

| GPIO Pins | Default State | GPIOAFSEL | GPIODEN | GPIOPDR | GPIOPUR | GPIOPCTL |
|---|---|---|---|---|---|---|
| PA[1:0] | UART0 | 1 | 1 | 0 | 0 | 0x1 |
| PA[5:2] | SSI0 | 1 | 1 | 0 | 0 | 0x1 |
| PB[3:2] | $I^2C0$ | 1 | 1 | 0 | 0 | 0x1 |
| PC[3:0] | JTAG/SWD | 1 | 1 | 0 | 1 | 0x3 |

**Table 9-2. GPIO Pins and Alternate Functions (64LQFP)**

| IO | Pin Number | Multiplexed Function | Multiplexed Function |
|---|---|---|---|
| PA0 | 17 | U0Rx | |
| PA1 | 18 | U0Tx | |
| PA2 | 19 | SSI0Clk | |
| PA3 | 20 | SSI0Fss | |
| PA4 | 21 | SSI0Rx | |
| PA5 | 22 | SSI0Tx | |
| PA6 | 25 | I2C1SCL | |
| PA7 | 26 | I2C1SDA | |
| PB0 | 41 | U1Rx | |
| PB1 | 42 | U1Tx | |
| PB2 | 47 | I2C0SCL | |
| PB3 | 27 | I2C0SDA | |
| PB4 | 58 | | |
| PB5 | 57 | CCP0 | |
| PB6 | 56 | CCP1 | |
| PB7 | 55 | NMI | |
| PC0 | 52 | TCK | SWCLK |
| PC1 | 51 | TMS | SWDIO |
| PC2 | 50 | TDI | |
| PC3 | 49 | TDO | SWO |
| PC4 | 11 | CCP2 | |
| PC5 | 14 | USB0EPEN | |
| PC6 | 15 | USB0PFLT | |
| PC7 | 16 | CCP4 | |
| PD0 | 61 | CAN0Rx | |

**Table 9-2. GPIO Pins and Alternate Functions (64LQFP) *(continued)***

| IO | Pin Number | Multiplexed Function | Multiplexed Function |
|---|---|---|---|
| PD1 | 62 | CAN0Tx | |
| PD2 | 63 | ADC5 | |
| PD3 | 64 | ADC4 | |
| PE0 | 6 | ADC3 | |
| PE1 | 5 | ADC2 | |
| PE2 | 2 | ADC1 | |
| PE3 | 1 | ADC0 | |
| PE4 | 8 | CCP3 | |

**Table 9-3. GPIO Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| PA0 | 17 | I/O | TTL | GPIO port A bit 0. |
| PA1 | 18 | I/O | TTL | GPIO port A bit 1. |
| PA2 | 19 | I/O | TTL | GPIO port A bit 2. |
| PA3 | 20 | I/O | TTL | GPIO port A bit 3. |
| PA4 | 21 | I/O | TTL | GPIO port A bit 4. |
| PA5 | 22 | I/O | TTL | GPIO port A bit 5. |
| PA6 | 25 | I/O | TTL | GPIO port A bit 6. |
| PA7 | 26 | I/O | TTL | GPIO port A bit 7. |
| PB0 | 41 | I/O | TTL | GPIO port B bit 0. |
| PB1 | 42 | I/O | TTL | GPIO port B bit 1. |
| PB2 | 47 | I/O | TTL | GPIO port B bit 2. |
| PB3 | 27 | I/O | TTL | GPIO port B bit 3. |
| PB4 | 58 | I/O | TTL | GPIO port B bit 4. |
| PB5 | 57 | I/O | TTL | GPIO port B bit 5. |
| PB6 | 56 | I/O | TTL | GPIO port B bit 6. |
| PB7 | 55 | I/O | TTL | GPIO port B bit 7. |
| PC0 | 52 | I/O | TTL | GPIO port C bit 0. |
| PC1 | 51 | I/O | TTL | GPIO port C bit 1. |
| PC2 | 50 | I/O | TTL | GPIO port C bit 2. |
| PC3 | 49 | I/O | TTL | GPIO port C bit 3. |
| PC4 | 11 | I/O | TTL | GPIO port C bit 4. |
| PC5 | 14 | I/O | TTL | GPIO port C bit 5. |
| PC6 | 15 | I/O | TTL | GPIO port C bit 6. |
| PC7 | 16 | I/O | TTL | GPIO port C bit 7. |
| PD0 | 61 | I/O | TTL | GPIO port D bit 0. |
| PD1 | 62 | I/O | TTL | GPIO port D bit 1. |
| PD2 | 63 | I/O | TTL | GPIO port D bit 2. |
| PD3 | 64 | I/O | TTL | GPIO port D bit 3. |
| PE0 | 6 | I/O | TTL | GPIO port E bit 0. |
| PE1 | 5 | I/O | TTL | GPIO port E bit 1. |

**Table 9-3. GPIO Signals (64LQFP) *(continued)***

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| PE2 | 2 | I/O | TTL | GPIO port E bit 2. |
| PE3 | 1 | I/O | TTL | GPIO port E bit 3. |
| PE4 | 8 | I/O | TTL | GPIO port E bit 4. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 9.2 Functional Description

**Important:** All GPIO pins are tri-stated by default (**GPIOAFSEL**=0, **GPIODEN**=0, **GPIOPDR**=0, and **GPIOPUR**=0), with the exception of the four JTAG/SWD pins (PC[3:0]). The JTAG/SWD pins default to their JTAG/SWD functionality (**GPIOAFSEL**=1, **GPIODEN=1** and **GPIOPUR**=1). A Power-On-Reset ($\overline{POR}$) or asserting $\overline{RST}$ puts both groups of pins back to their default state.

Each GPIO port is a separate hardware instantiation of the same physical block(see Figure 9-1 on page 354 and Figure 9-2 on page 355). The LM3S5632 microcontroller contains five ports and thus five of these physical GPIO blocks.

**Figure 9-1. Digital I/O Pads**

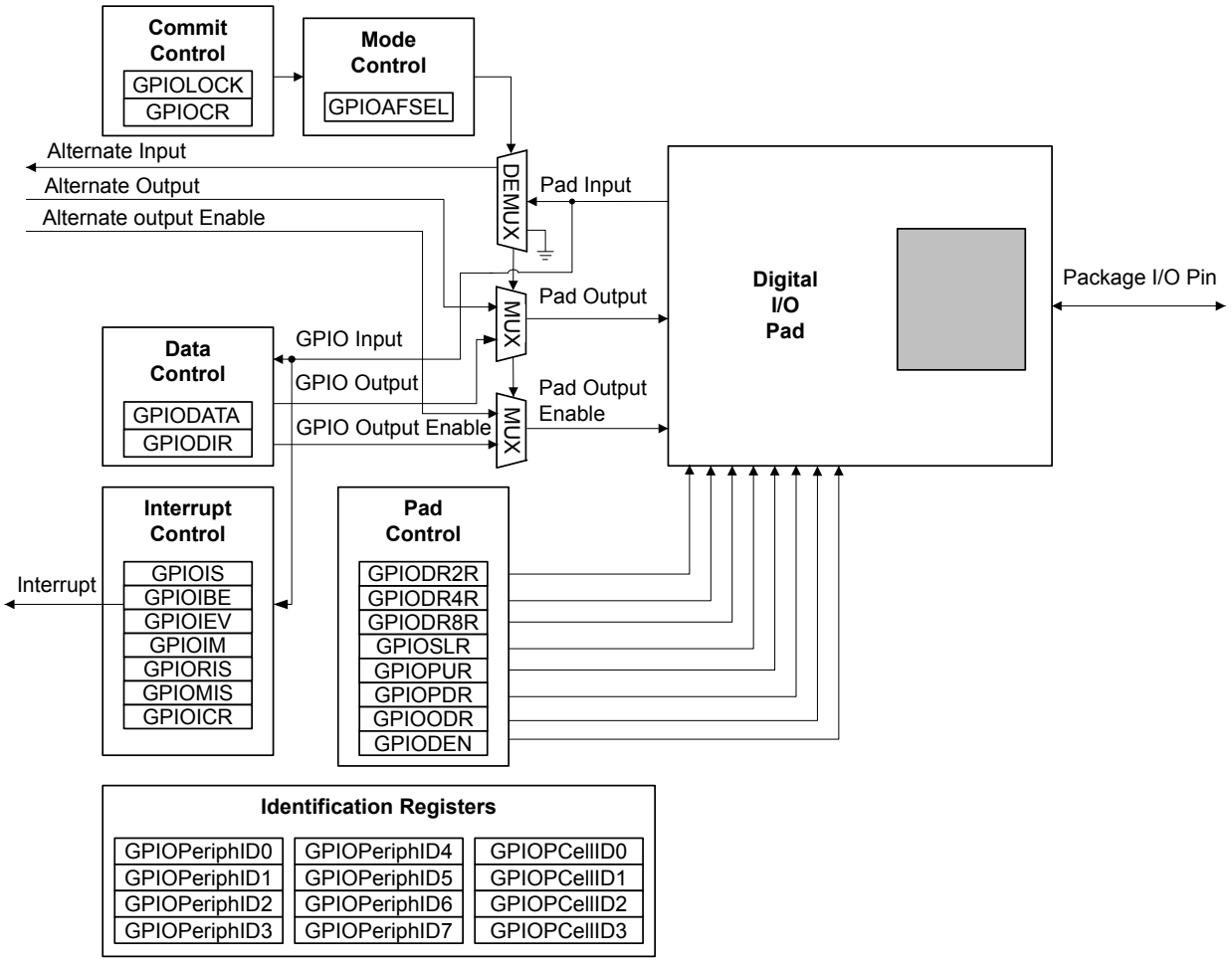
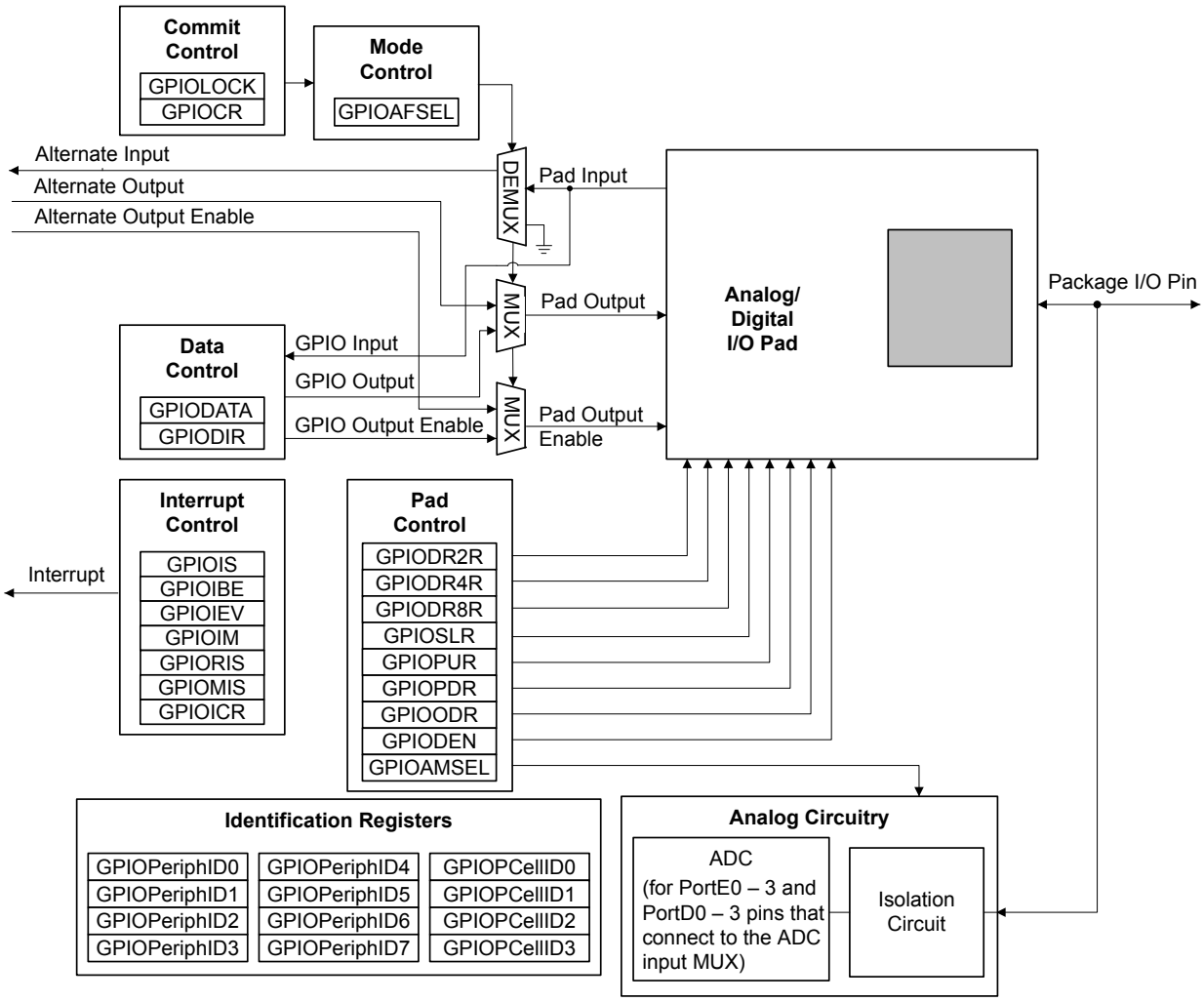


**Figure 9-2. Analog/Digital I/O Pads**




## 9.2.1 Data Control

The data control registers allow software to configure the operational modes of the GPIOs. The data direction register configures the GPIO as an input or an output while the data register either captures incoming data or drives it out to the pads.

### 9.2.1.1 Data Direction Operation

The **GPIO Direction (GPIODIR)** register (see page 363) is used to configure each individual pin as an input or output. When the data direction bit is set to 0, the GPIO is configured as an input and the corresponding data register bit will capture and store the value on the GPIO port. When the data direction bit is set to 1, the GPIO is configured as an output and the corresponding data register bit will be driven out on the GPIO port.

### 9.2.1.2 Data Register Operation

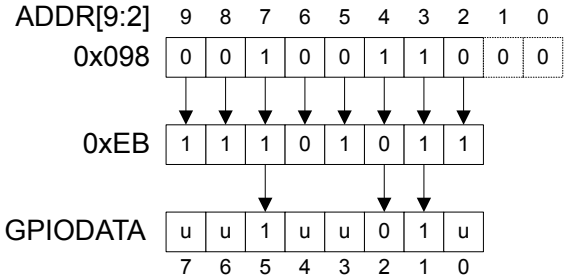
To aid in the efficiency of software, the GPIO ports allow for the modification of individual bits in the **GPIO Data (GPIODATA)** register (see page 362) by using bits [9:2] of the address bus as a mask. This allows software drivers to modify individual GPIO pins in a single instruction, without affecting

the state of the other pins. This is in contrast to the "typical" method of doing a read-modify-write operation to set or clear an individual GPIO pin. To accommodate this feature, the **GPIODATA** register covers 256 locations in the memory map.

During a write, if the address bit associated with that data bit is set to 1, the value of the **GPIODATA** register is altered. If it is cleared to 0, it is left unchanged.

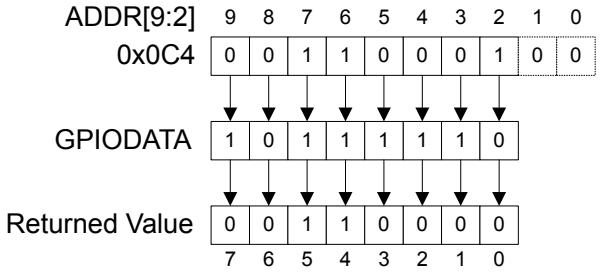
For example, writing a value of 0xEB to the address GPIODATA + 0x098 would yield as shown in Figure 9-3 on page 356, where u is data unchanged by the write.

**Figure 9-3. GPIODATA Write Example**




During a read, if the address bit associated with the data bit is set to 1, the value is read. If the address bit associated with the data bit is set to 0, it is read as a zero, regardless of its actual value. For example, reading address GPIODATA + 0x0C4 yields as shown in Figure 9-4 on page 356.

**Figure 9-4. GPIODATA Read Example**




## 9.2.2 Interrupt Control

The interrupt capabilities of each GPIO port are controlled by a set of seven registers. With these registers, it is possible to select the source of the interrupt, its polarity, and the edge properties. When one or more GPIO inputs cause an interrupt, a single interrupt output is sent to the interrupt controller for the entire GPIO port. For edge-triggered interrupts, software must clear the interrupt to enable any further interrupts. For a level-sensitive interrupt, it is assumed that the external source holds the level constant for the interrupt to be recognized by the controller.

Three registers are required to define the edge or sense that causes interrupts:

- **GPIO Interrupt Sense (GPIOIS)** register (see page 364)
- **GPIO Interrupt Both Edges (GPIOIBE)** register (see page 365)
- **GPIO Interrupt Event (GPIOIEV)** register (see page 366)

Interrupts are enabled/disabled via the **GPIO Interrupt Mask (GPIOIM)** register (see page 367).

When an interrupt condition occurs, the state of the interrupt signal can be viewed in two locations: the **GPIO Raw Interrupt Status (GPIORIS)** and **GPIO Masked Interrupt Status (GPIOMIS)** registers (see page 368 and page 369). As the name implies, the **GPIOMIS** register only shows interrupt conditions that are allowed to be passed to the controller. The **GPIORIS** register indicates that a GPIO pin meets the conditions for an interrupt, but has not necessarily been sent to the controller.

In addition to providing GPIO functionality, PB4 can also be used as an external trigger for the ADC. If PB4 is configured as a non-masked interrupt pin (the appropriate bit of GPIOIM is set to 1), not only is an interrupt for PortB generated, but an external trigger signal is sent to the ADC. If the **ADC Event Multiplexer Select (ADCEMUX)** register is configured to use the external trigger, an ADC conversion is initiated.

If no other PortB pins are being used to generate interrupts, the **Interrupt 0-31 Set Enable (EN0)** register can disable the PortB interrupts, and the ADC interrupt can be used to read back the converted data. Otherwise, the PortB interrupt handler needs to ignore and clear interrupts on PB4, and wait for the ADC interrupt or the ADC interrupt must be disabled in the **EN0** register and the PortB interrupt handler must poll the ADC registers until the conversion is completed. See page 110 for more information.

Interrupts are cleared by writing a 1 to the appropriate bit of the **GPIO Interrupt Clear (GPIOICR)** register (see page 370).

When programming the following interrupt control registers, the interrupts should be masked (**GPIOIM** set to 0). Writing any value to an interrupt control register (**GPIOIS**, **GPIOIBE**, or **GPIOIEV**) can generate a spurious interrupt if the corresponding bits are enabled.

### 9.2.3 Mode Control

The GPIO pins can be controlled by either hardware or software. When hardware control is enabled via the **GPIO Alternate Function Select (GPIOAFSEL)** register (see page 371), the pin state is controlled by its alternate function (that is, the peripheral). Software control corresponds to GPIO mode, where the **GPIODATA** register is used to read/write the corresponding pins.

**Note:** If any pin is to be used as an ADC input, the appropriate bit in **GPIOAMSEL** must be written to 1 to disable the analog isolation circuit.

### 9.2.4 Commit Control

The GPIO commit control registers provide a layer of protection against accidental programming of critical hardware peripherals. Protection is currently provided for the NMI pin (PB7) and the four JTAG/SWD pins (PC[3:0]). Writes to protected bits of the **GPIO Alternate Function Select (GPIOAFSEL)** register (see page 371), **GPIO Pull-Up Select (GPIOPUR)** register (see page 377), and **GPIO Digital Enable (GPIODEN)** register (see page 381) are not committed to storage unless the **GPIO Lock (GPIOLOCK)** register (see page 383) has been unlocked and the appropriate bits of the **GPIO Commit (GPIOCR)** register (see page 384) have been set to 1.

### 9.2.5 Pad Control

The pad control registers allow for GPIO pad configuration by software based on the application requirements. The pad control registers include the **GPIODR2R**, **GPIODR4R**, **GPIODR8R**, **GPIOODR**, **GPIOPUR**, **GPIOPDR**, **GPIOSLR**, and **GPIODEN** registers. These registers control drive strength, open-drain configuration, pull-up and pull-down resistors, slew-rate control and digital enable.

For special high-current applications, the GPIO output buffers may be used with the following restrictions. With the GPIO pins configured as 8-mA output drivers, a total of four GPIO outputs may be used to sink current loads up to 18 mA each. At 18-mA sink current loading, the $V_{OL}$ value is specified as 1.2 V. The high-current GPIO package pins must be selected such that there are only

a maximum of two per side of the physical package with the total number of high-current GPIO outputs not exceeding four for the entire package.

### 9.2.6 Identification

The identification registers configured at reset allow software to detect and identify the module as a GPIO block. The identification registers include the **GPIOPeriphID0**-**GPIOPeriphID7** registers as well as the **GPIOPCellID0**-**GPIOPCellID3** registers.

## 9.3 Initialization and Configuration

The GPIO modules may be accessed via two different memory apertures. The legacy aperture, the Advanced Peripheral Bus (APB), is backwards-compatible with previous Stellaris® parts. The other aperture, the Advanced High-Performance Bus (AHB), offers the same register map but provides better back-to-back access performance than the APB bus. These apertures are mutually exclusive. The aperture enabled for a given GPIO port is controlled by the appropriate bit in the **GPIOHBCTL** register (see page 199).

To use the GPIO, the peripheral clock must be enabled by setting the appropriate GPIO Port bit field (`GPIOn`) in the **RCGC2** register.

On reset, all GPIO pins (except for the four JTAG pins) are configured out of reset to be undriven (tristate): **GPIOAFSEL**=0, **GPIODEN**=0, **GPIOPDR**=0, and **GPIOPUR**=0. Table 9-4 on page 358 shows all possible configurations of the GPIO pads and the control register settings required to achieve them. Table 9-5 on page 359 shows how a rising edge interrupt would be configured for pin 2 of a GPIO port.

**Table 9-4. GPIO Pad Configuration Examples**

| Configuration | GPIO Register Bit Value[a] | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | AFSEL | DIR | ODR | DEN | PUR | PDR | DR2R | DR4R | DR8R | SLR |
| Digital Input (GPIO) | 0 | 0 | 0 | 1 | ? | ? | X | X | X | X |
| Digital Output (GPIO) | 0 | 1 | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Open Drain Output (GPIO) | 0 | 1 | 1 | 1 | X | X | ? | ? | ? | ? |
| Open Drain Input/Output ($I^2C$) | 1 | X | 1 | 1 | X | X | ? | ? | ? | ? |
| Digital Input (Timer CCP) | 1 | X | 0 | 1 | ? | ? | X | X | X | X |
| Digital Output (Timer PWM) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Digital Input/Output (SSI) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Digital Input/Output (UART) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |

a. X=Ignored (don't care bit)

?=Can be either 0 or 1, depending on the configuration

**Table 9-5. GPIO Interrupt Configuration Example**

| Register | Desired Interrupt Event Trigger | Pin 2 Bit Value[a] | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| GPIOIS | 0=edge<br>1=level | X | X | X | X | X | 0 | X | X |
| GPIOIBE | 0=single edge<br>1=both edges | X | X | X | X | X | 0 | X | X |
| GPIOIEV | 0=Low level, or negative edge<br>1=High level, or positive edge | X | X | X | X | X | 1 | X | X |
| GPIOIM | 0=masked<br>1=not masked | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

a. X=Ignored (don't care bit)

## 9.4 Register Map

Table 9-6 on page 360 lists the GPIO registers. Each GPIO port can be accessed through one of two bus apertures. The legacy aperture, the Advanced Peripheral Bus (APB), is backwards-compatible with previous Stellaris parts. The other aperture, the Advanced High-Performance Bus (AHB), offers the same register map but provides better back-to-back access performance than the APB bus. The offset listed is a hexadecimal increment to the register's address, relative to that GPIO port's base address:

- GPIO Port A (APB): 0x4000.4000
- GPIO Port A (AHB): 0x4005.8000
- GPIO Port B (APB): 0x4000.5000
- GPIO Port B (AHB): 0x4005.9000
- GPIO Port C (APB): 0x4000.6000
- GPIO Port C (AHB): 0x4005.A000
- GPIO Port D (APB): 0x4000.7000
- GPIO Port D (AHB): 0x4005.B000
- GPIO Port E (APB): 0x4002.4000
- GPIO Port E (AHB): 0x4005.C000

Note that the GPIO module clock must be enabled before the registers can be programmed (see page 229). There must be a delay of 3 system clocks after the GPIO module clock is enabled before any GPIO module registers are accessed.

**Important:** The GPIO registers in this chapter are duplicated in each GPIO block; however, depending on the block, all eight bits may not be connected to a GPIO pad. In those cases, writing to those unconnected bits has no effect, and reading those unconnected bits returns no meaningful data.

**Note:** The default reset value for the **GPIOAFSEL**, **GPIOPUR**, and **GPIODEN** registers are 0x0000.0000 for all GPIO pins, with the exception of the four JTAG/SWD pins (PC[3:0]).

These four pins default to JTAG/SWD functionality. Because of this, the default reset value of these registers for Port C is 0x0000.000F.

The default register type for the **GPIOCR** register is RO for all GPIO pins with the exception of the NMI pin and the four JTAG/SWD pins (PB7 and PC[3:0]). These five pins are currently the only GPIOs that are protected by the **GPIOCR** register. Because of this, the register type for GPIO Port B7 and GPIO Port C[3:0] is R/W.

The default reset value for the **GPIOCR** register is 0x0000.00FF for all GPIO pins, with the exception of the NMI pin and the four JTAG/SWD pins (PB7 and PC[3:0]). To ensure that the JTAG port is not accidentally programmed as a GPIO, these four pins default to non-committable. To ensure that the NMI pin is not accidentally programmed as the non-maskable interrupt pin, it defaults to non-committable. Because of this, the default reset value of **GPIOCR** for GPIO Port B is 0x0000.007F while the default reset value of GPIOCR for Port C is 0x0000.00F0.

**Table 9-6. GPIO Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | GPIODATA | R/W | 0x0000.0000 | GPIO Data | 362 |
| 0x400 | GPIODIR | R/W | 0x0000.0000 | GPIO Direction | 363 |
| 0x404 | GPIOIS | R/W | 0x0000.0000 | GPIO Interrupt Sense | 364 |
| 0x408 | GPIOIBE | R/W | 0x0000.0000 | GPIO Interrupt Both Edges | 365 |
| 0x40C | GPIOIEV | R/W | 0x0000.0000 | GPIO Interrupt Event | 366 |
| 0x410 | GPIOIM | R/W | 0x0000.0000 | GPIO Interrupt Mask | 367 |
| 0x414 | GPIORIS | RO | 0x0000.0000 | GPIO Raw Interrupt Status | 368 |
| 0x418 | GPIOMIS | RO | 0x0000.0000 | GPIO Masked Interrupt Status | 369 |
| 0x41C | GPIOICR | W1C | 0x0000.0000 | GPIO Interrupt Clear | 370 |
| 0x420 | GPIOAFSEL | R/W | - | GPIO Alternate Function Select | 371 |
| 0x500 | GPIODR2R | R/W | 0x0000.00FF | GPIO 2-mA Drive Select | 373 |
| 0x504 | GPIODR4R | R/W | 0x0000.0000 | GPIO 4-mA Drive Select | 374 |
| 0x508 | GPIODR8R | R/W | 0x0000.0000 | GPIO 8-mA Drive Select | 375 |
| 0x50C | GPIOODR | R/W | 0x0000.0000 | GPIO Open Drain Select | 376 |
| 0x510 | GPIOPUR | R/W | - | GPIO Pull-Up Select | 377 |
| 0x514 | GPIOPDR | R/W | 0x0000.0000 | GPIO Pull-Down Select | 379 |
| 0x518 | GPIOSLR | R/W | 0x0000.0000 | GPIO Slew Rate Control Select | 380 |
| 0x51C | GPIODEN | R/W | - | GPIO Digital Enable | 381 |
| 0x520 | GPIOLOCK | R/W | 0x0000.0001 | GPIO Lock | 383 |
| 0x524 | GPIOCR | - | - | GPIO Commit | 384 |
| 0x528 | GPIOAMSEL | R/W | 0x0000.0000 | GPIO Analog Mode Select | 386 |
| 0xFD0 | GPIOPeriphID4 | RO | 0x0000.0000 | GPIO Peripheral Identification 4 | 387 |

**Table 9-6. GPIO Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0xFD4 | GPIOPeriphID5 | RO | 0x0000.0000 | GPIO Peripheral Identification 5 | 388 |
| 0xFD8 | GPIOPeriphID6 | RO | 0x0000.0000 | GPIO Peripheral Identification 6 | 389 |
| 0xFDC | GPIOPeriphID7 | RO | 0x0000.0000 | GPIO Peripheral Identification 7 | 390 |
| 0xFE0 | GPIOPeriphID0 | RO | 0x0000.0061 | GPIO Peripheral Identification 0 | 391 |
| 0xFE4 | GPIOPeriphID1 | RO | 0x0000.0000 | GPIO Peripheral Identification 1 | 392 |
| 0xFE8 | GPIOPeriphID2 | RO | 0x0000.0018 | GPIO Peripheral Identification 2 | 393 |
| 0xFEC | GPIOPeriphID3 | RO | 0x0000.0001 | GPIO Peripheral Identification 3 | 394 |
| 0xFF0 | GPIOPCellID0 | RO | 0x0000.000D | GPIO PrimeCell Identification 0 | 395 |
| 0xFF4 | GPIOPCellID1 | RO | 0x0000.00F0 | GPIO PrimeCell Identification 1 | 396 |
| 0xFF8 | GPIOPCellID2 | RO | 0x0000.0005 | GPIO PrimeCell Identification 2 | 397 |
| 0xFFC | GPIOPCellID3 | RO | 0x0000.00B1 | GPIO PrimeCell Identification 3 | 398 |

## 9.5 Register Descriptions

The remainder of this section lists and describes the GPIO registers, in numerical order by address offset.

## Register 1: GPIO Data (GPIODATA), offset 0x000

The **GPIODATA** register is the data register. In software control mode, values written in the **GPIODATA** register are transferred onto the GPIO port pins if the respective pins have been configured as outputs through the **GPIO Direction (GPIODIR)** register (see page 363).

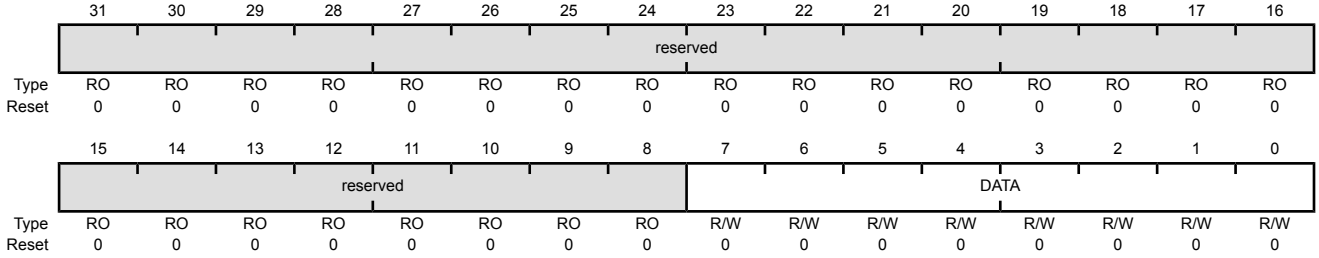
In order to write to **GPIODATA**, the corresponding bits in the mask, resulting from the address bus bits [9:2], must be High. Otherwise, the bit values remain unchanged by the write.

Similarly, the values read from this register are determined for each bit by the mask bit derived from the address used to access the data register, bits [9:2]. Bits that are 1 in the address mask cause the corresponding bits in **GPIODATA** to be read, and bits that are 0 in the address mask cause the corresponding bits in **GPIODATA** to be read as 0, regardless of their value.

A read from **GPIODATA** returns the last bit value written if the respective pins are configured as outputs, or it returns the value on the corresponding input pin when these are configured as inputs. All bits are cleared by a reset.

GPIO Data (GPIODATA)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x000
Type R/W, reset 0x0000.0000



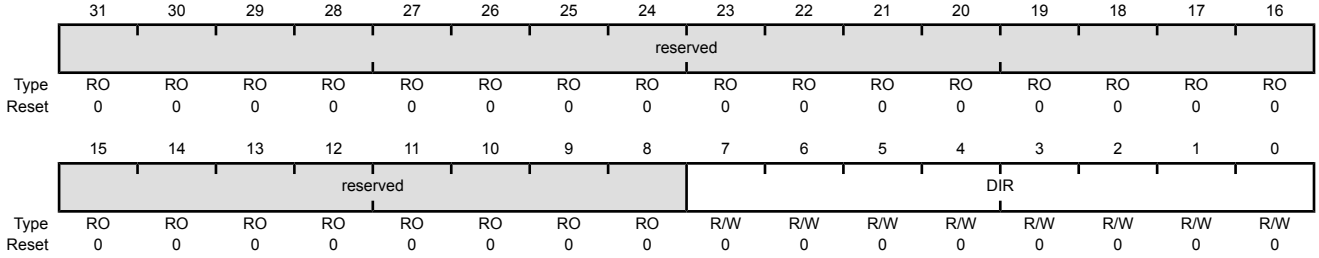

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DATA | R/W | 0x00 | GPIO Data<br>This register is virtually mapped to 256 locations in the address space. To facilitate the reading and writing of data to these registers by independent drivers, the data read from and the data written to the registers are masked by the eight address lines `ipaddr[9:2]`. Reads from this register return its current state. Writes to this register only affect bits that are not masked by `ipaddr[9:2]` and are configured as outputs. See “Data Register Operation” on page 355 for examples of reads and writes. |

## Register 2: GPIO Direction (GPIODIR), offset 0x400

The **GPIODIR** register is the data direction register. Bits set to 1 in the **GPIODIR** register configure the corresponding pin to be an output, while bits set to 0 configure the pins to be inputs. All bits are cleared by a reset, meaning all GPIO pins are inputs by default.

GPIO Direction (GPIODIR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x400
Type R/W, reset 0x0000.0000



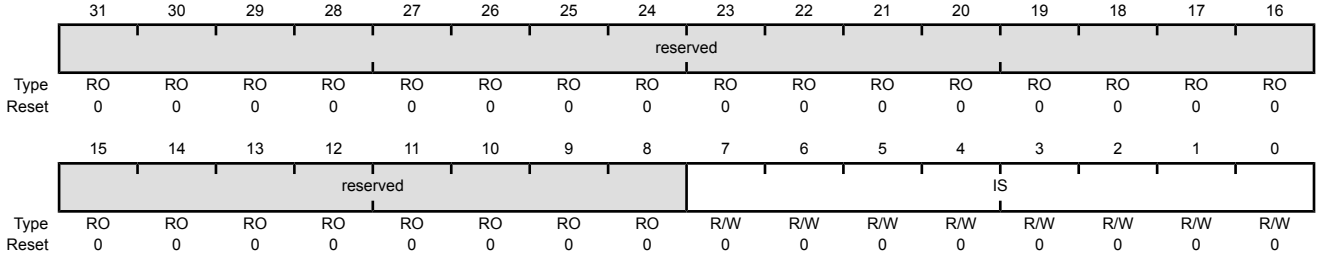

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DIR | R/W | 0x00 | GPIO Data Direction<br>The `DIR` values are defined as follows:<br><br>Value Description<br>0 Pins are inputs.<br>1 Pins are outputs. |

## Register 3: GPIO Interrupt Sense (GPIOIS), offset 0x404

The **GPIOIS** register is the interrupt sense register. Bits set to 1 in **GPIOIS** configure the corresponding pins to detect levels, while bits set to 0 configure the pins to detect edges. All bits are cleared by a reset.

GPIO Interrupt Sense (GPIOIS)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x404
Type R/W, reset 0x0000.0000



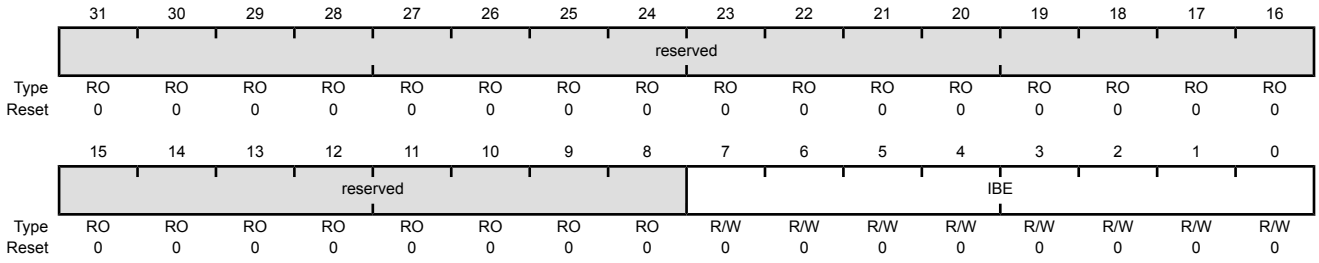

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | IS | R/W | 0x00 | GPIO Interrupt Sense<br>The `IS` values are defined as follows:<br><br>Value Description<br>0 Edge on corresponding pin is detected (edge-sensitive).<br>1 Level on corresponding pin is detected (level-sensitive). |

## Register 4: GPIO Interrupt Both Edges (GPIOIBE), offset 0x408

The **GPIOIBE** register is the interrupt both-edges register. When the corresponding bit in the **GPIO Interrupt Sense (GPIOIS)** register (see page 364) is set to detect edges, bits set to High in **GPIOIBE** configure the corresponding pin to detect both rising and falling edges, regardless of the corresponding bit in the **GPIO Interrupt Event (GPIOIEV)** register (see page 366). Clearing a bit configures the pin to be controlled by **GPIOIEV**. All bits are cleared by a reset.

GPIO Interrupt Both Edges (GPIOIBE)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x408
Type R/W, reset 0x0000.0000



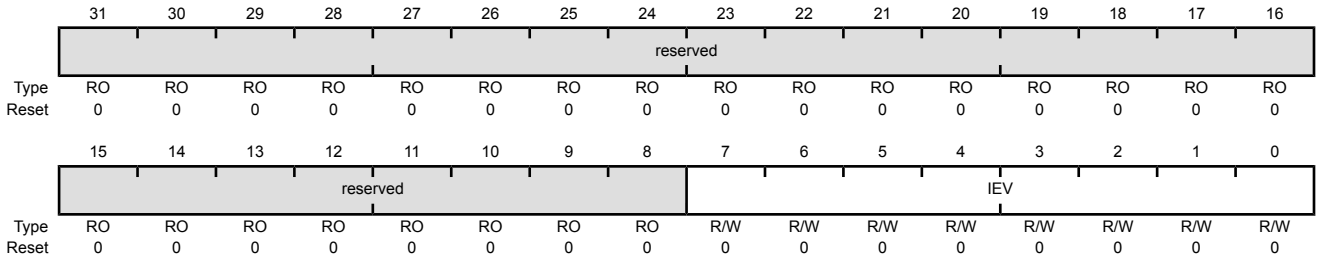

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | IBE | R/W | 0x00 | GPIO Interrupt Both Edges<br>The `IBE` values are defined as follows:<br><br>Value Description<br>0 Interrupt generation is controlled by the **GPIO Interrupt Event (GPIOIEV)** register (see page 366).<br>1 Both edges on the corresponding pin trigger an interrupt.<br><br>**Note:** Single edge is determined by the corresponding bit in **GPIOIEV**. |

## Register 5: GPIO Interrupt Event (GPIOIEV), offset 0x40C

The **GPIOIEV** register is the interrupt event register. Bits set to High in **GPIOIEV** configure the corresponding pin to detect rising edges or high levels, depending on the corresponding bit value in the **GPIO Interrupt Sense (GPIOIS)** register (see page 364). Clearing a bit configures the pin to detect falling edges or low levels, depending on the corresponding bit value in **GPIOIS**. All bits are cleared by a reset.

GPIO Interrupt Event (GPIOIEV)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x40C
Type R/W, reset 0x0000.0000



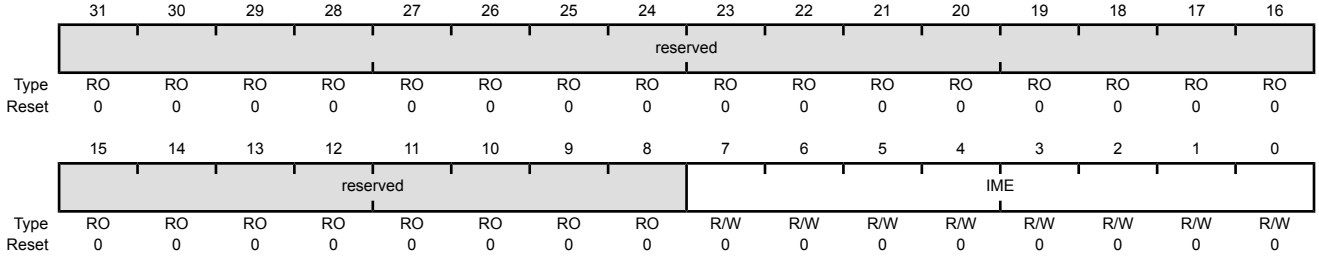

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | IEV | R/W | 0x00 | GPIO Interrupt Event<br>The `IEV` values are defined as follows:<br><br>Value Description<br>0 Falling edge or Low levels on corresponding pins trigger interrupts.<br>1 Rising edge or High levels on corresponding pins trigger interrupts. |

## Register 6: GPIO Interrupt Mask (GPIOIM), offset 0x410

The **GPIOIM** register is the interrupt mask register. Bits set to High in **GPIOIM** allow the corresponding pins to trigger their individual interrupts and the combined **GPIOINTR** line. Clearing a bit disables interrupt triggering on that pin. All bits are cleared by a reset.

GPIO Interrupt Mask (GPIOIM)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x410
Type R/W, reset 0x0000.0000



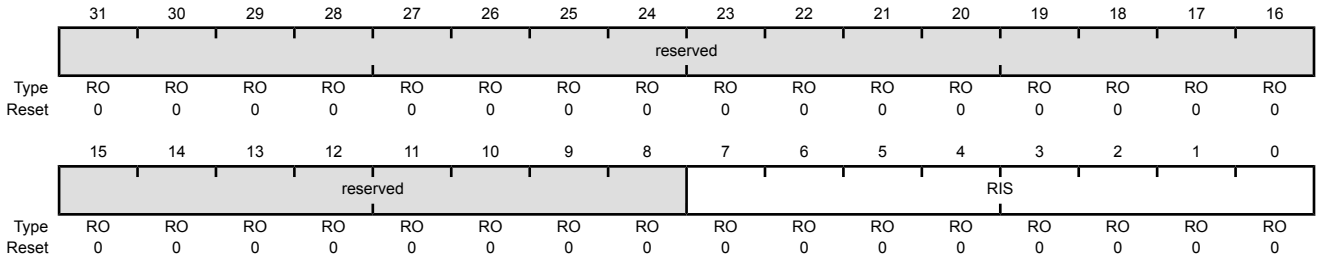

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | IME | R/W | 0x00 | GPIO Interrupt Mask Enable<br>The `IME` values are defined as follows:<br><br>Value Description<br>0 Corresponding pin interrupt is masked.<br>1 Corresponding pin interrupt is not masked. |

## Register 7: GPIO Raw Interrupt Status (GPIORIS), offset 0x414

The **GPIORIS** register is the raw interrupt status register. Bits read High in **GPIORIS** reflect the status of interrupt trigger conditions detected (raw, prior to masking), indicating that all the requirements have been met, before they are finally allowed to trigger by the **GPIO Interrupt Mask (GPIOIM)** register (see page 367). Bits read as zero indicate that corresponding input pins have not initiated an interrupt. All bits are cleared by a reset.

GPIO Raw Interrupt Status (GPIORIS)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x414
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | RIS | RO | 0x00 | GPIO Interrupt Raw Status<br>Reflects the status of interrupt trigger condition detection on pins (raw, prior to masking).<br>The `RIS` values are defined as follows:<br><br>Value Description<br>0 Corresponding pin interrupt requirements not met.<br>1 Corresponding pin interrupt has met requirements. |

## Register 8: GPIO Masked Interrupt Status (GPIOMIS), offset 0x418

The **GPIOMIS** register is the masked interrupt status register. Bits read High in **GPIOMIS** reflect the status of input lines triggering an interrupt. Bits read as Low indicate that either no interrupt has been generated, or the interrupt is masked.

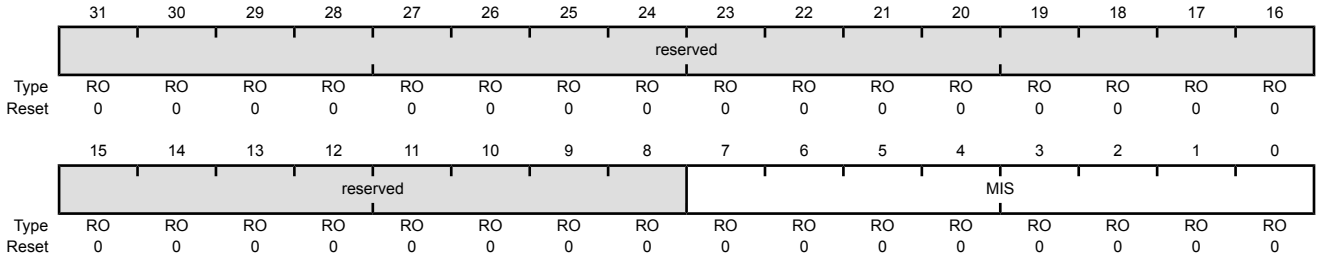
In addition to providing GPIO functionality, PB4 can also be used as an external trigger for the ADC. If PB4 is configured as a non-masked interrupt pin (the appropriate bit of GPIOIM is set to 1), not only is an interrupt for PortB generated, but an external trigger signal is sent to the ADC. If the **ADC Event Multiplexer Select (ADCEMUX)** register is configured to use the external trigger, an ADC conversion is initiated.

If no other PortB pins are being used to generate interrupts, the **Interrupt 0-31 Set Enable (EN0)** register can disable the PortB interrupts, and the ADC interrupt can be used to read back the converted data. Otherwise, the PortB interrupt handler needs to ignore and clear interrupts on PB4, and wait for the ADC interrupt or the ADC interrupt must be disabled in the **EN0** register and the PortB interrupt handler must poll the ADC registers until the conversion is completed. See page 110 for more information.

**GPIOMIS** is the state of the interrupt after masking.

GPIO Masked Interrupt Status (GPIOMIS)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x418
Type RO, reset 0x0000.0000



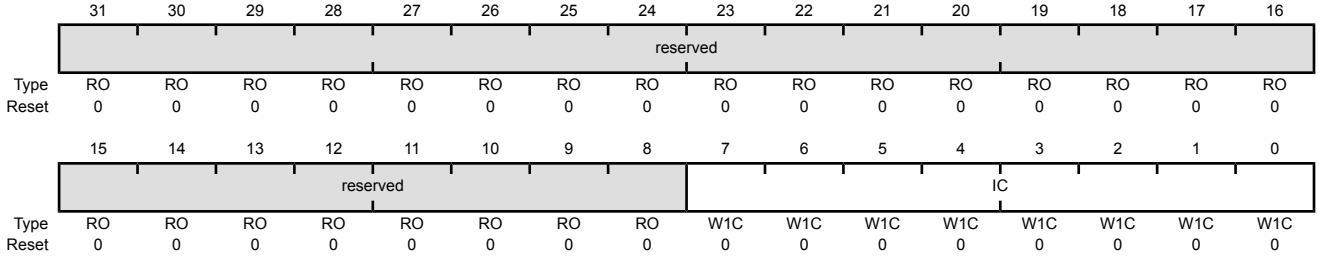

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | MIS | RO | 0x00 | GPIO Masked Interrupt Status<br>Masked value of interrupt due to corresponding pin.<br>The MIS values are defined as follows:<br><br>Value Description<br>0 Corresponding GPIO line interrupt not active.<br>1 Corresponding GPIO line asserting interrupt. |

## Register 9: GPIO Interrupt Clear (GPIOICR), offset 0x41C

The **GPIOICR** register is the interrupt clear register. Writing a 1 to a bit in this register clears the corresponding interrupt edge detection logic register. Writing a 0 has no effect.

GPIO Interrupt Clear (GPIOICR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x41C
Type W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | IC | W1C | 0x00 | GPIO Interrupt Clear<br>The `IC` values are defined as follows:<br><br>Value Description<br>0 Corresponding interrupt is unaffected.<br>1 Corresponding interrupt is cleared. |

## Register 10: GPIO Alternate Function Select (GPIOAFSEL), offset 0x420

The **GPIOAFSEL** register is the mode control select register. Writing a 1 to any bit in this register selects the hardware control for the corresponding GPIO line. All bits are cleared by a reset, therefore no GPIO line is set to hardware control by default.

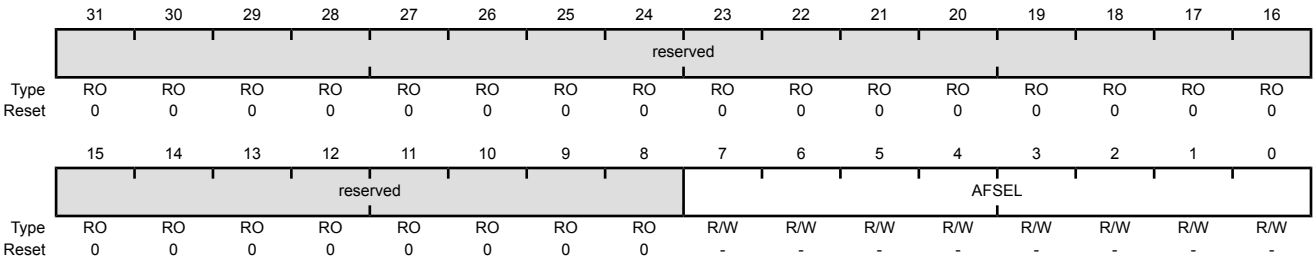
The GPIO commit control registers provide a layer of protection against accidental programming of critical hardware peripherals. Protection is currently provided for the `NMI` pin (`PB7`) and the four JTAG/SWD pins (`PC[3:0]`). Writes to protected bits of the **GPIO Alternate Function Select (GPIOAFSEL)** register (see page 371), **GPIO Pull-Up Select (GPIOPUR)** register (see page 377), and **GPIO Digital Enable (GPIODEN)** register (see page 381) are not committed to storage unless the **GPIO Lock (GPIOLOCK)** register (see page 383) has been unlocked and the appropriate bits of the **GPIO Commit (GPIOCR)** register (see page 384) have been set to 1.

**Important:** All GPIO pins are tri-stated by default (**GPIOAFSEL**=0, **GPIODEN**=0, **GPIOPDR**=0, and **GPIOPUR**=0), with the exception of the four JTAG/SWD pins (`PC[3:0]`). The JTAG/SWD pins default to their JTAG/SWD functionality (**GPIOAFSEL**=1, **GPIODEN=1** and **GPIOPUR**=1). A Power-On-Reset ($\overline{\text{POR}}$) or asserting $\overline{\text{RST}}$ puts both groups of pins back to their default state.

**Caution – It is possible to create a software sequence that prevents the debugger from connecting to the Stellaris microcontroller. If the program code loaded into flash immediately changes the JTAG pins to their GPIO functionality, the debugger may not have enough time to connect and halt the controller before the JTAG pin functionality switches. This may lock the debugger out of the part. This can be avoided with a software routine that restores JTAG functionality based on an external or software trigger.**

GPIO Alternate Function Select (GPIOAFSEL)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x420
Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

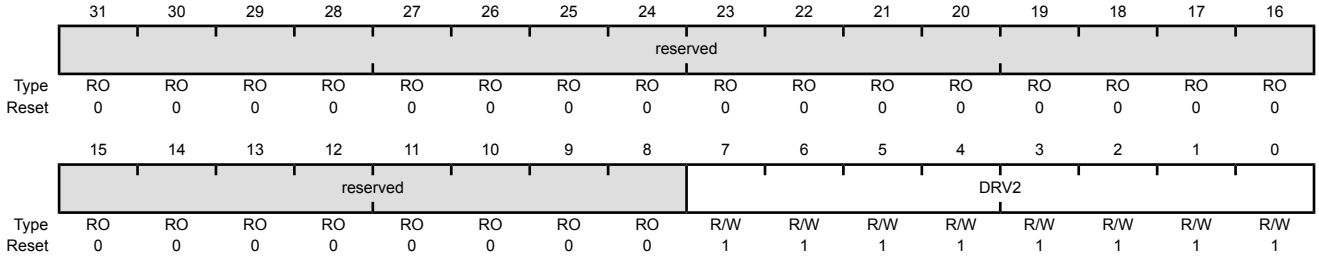
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:0 | AFSEL | R/W | - | GPIO Alternate Function Select<br>The AFSEL values are defined as follows:<br><br>Value Description<br>0 Software control of corresponding GPIO line (GPIO mode).<br>1 Hardware control of corresponding GPIO line (alternate hardware function).<br><br>**Note:** The default reset value for the **GPIOAFSEL**, **GPIOPUR**, and **GPIODEN** registers are 0x0000.0000 for all GPIO pins, with the exception of the four JTAG/SWD pins (PC[3:0]). These four pins default to JTAG/SWD functionality. Because of this, the default reset value of these registers for Port C is 0x0000.000F. |

## Register 11: GPIO 2-mA Drive Select (GPIODR2R), offset 0x500

The **GPIODR2R** register is the 2-mA drive control register. It allows for each GPIO signal in the port to be individually configured without affecting the other pads. When writing a DRV2 bit for a GPIO signal, the corresponding DRV4 bit in the **GPIODR4R** register and the DRV8 bit in the **GPIODR8R** register are automatically cleared by hardware.

GPIO 2-mA Drive Select (GPIODR2R)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x500
Type R/W, reset 0x0000.00FF



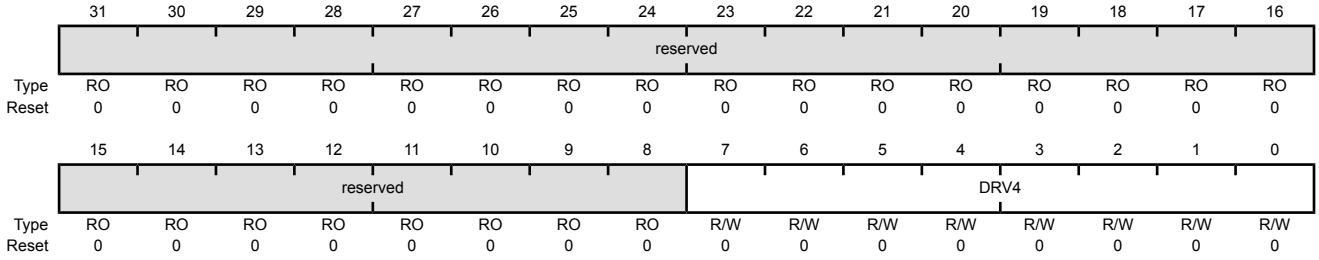

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DRV2 | R/W | 0xFF | Output Pad 2-mA Drive Enable<br>A write of 1 to either **GPIODR4[n]** or **GPIODR8[n]** clears the corresponding 2-mA enable bit. The change is effective on the second clock cycle after the write if accessing GPIO via the APB memory aperture. If using AHB access, the change is effective on the next clock cycle. |

## Register 12: GPIO 4-mA Drive Select (GPIODR4R), offset 0x504

The **GPIODR4R** register is the 4-mA drive control register. It allows for each GPIO signal in the port to be individually configured without affecting the other pads. When writing the DRV4 bit for a GPIO signal, the corresponding DRV2 bit in the **GPIODR2R** register and the DRV8 bit in the **GPIODR8R** register are automatically cleared by hardware.

GPIO 4-mA Drive Select (GPIODR4R)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x504
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DRV4 | R/W | 0x00 | Output Pad 4-mA Drive Enable<br>A write of 1 to either **GPIODR2[n]** or **GPIODR8[n]** clears the corresponding 4-mA enable bit. The change is effective on the second clock cycle after the write if accessing GPIO via the APB memory aperture. If using AHB access, the change is effective on the next clock cycle. |

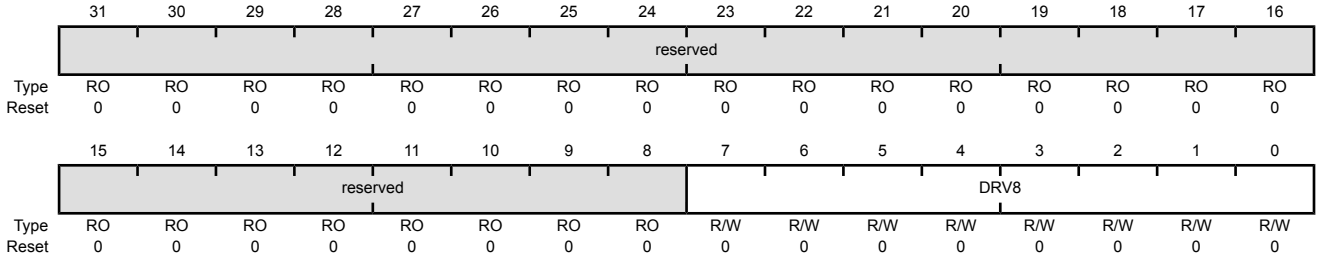
## Register 13: GPIO 8-mA Drive Select (GPIODR8R), offset 0x508

The **GPIODR8R** register is the 8-mA drive control register. It allows for each GPIO signal in the port to be individually configured without affecting the other pads. When writing the DRV8 bit for a GPIO signal, the corresponding DRV2 bit in the **GPIODR2R** register and the DRV4 bit in the **GPIODR4R** register are automatically cleared by hardware. The 8-mA setting is also used for high-current operation.

**Note:** There is no configuration difference between 8-mA and high-current operation. The additional current capacity results from a shift in the $V_{OH}/V_{OL}$ levels. See "Recommended DC Operating Conditions" on page 766 for further information.

GPIO 8-mA Drive Select (GPIODR8R)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x508
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DRV8 | R/W | 0x00 | Output Pad 8-mA Drive Enable<br>A write of 1 to either **GPIODR2[n]** or **GPIODR4[n]** clears the corresponding 8-mA enable bit. The change is effective on the second clock cycle after the write if accessing GPIO via the APB memory aperture. If using AHB access, the change is effective on the next clock cycle. |

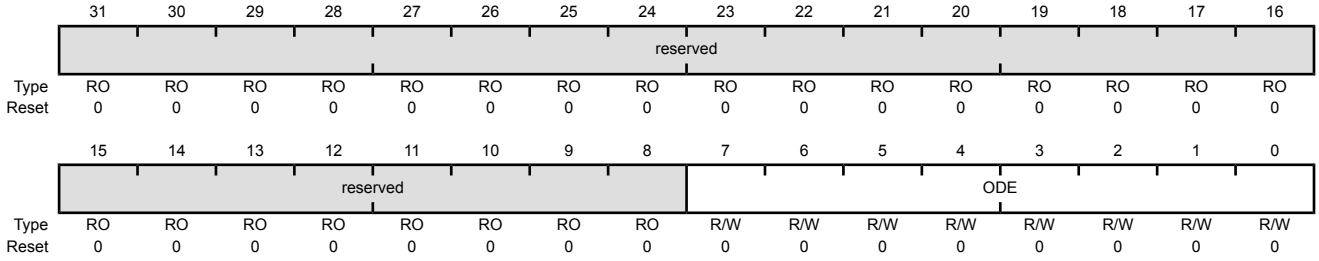
## Register 14: GPIO Open Drain Select (GPIOODR), offset 0x50C

The **GPIOODR** register is the open drain control register. Setting a bit in this register enables the open drain configuration of the corresponding GPIO pad. When open drain mode is enabled, the corresponding bit should also be set in the **GPIO Digital Enable (GPIODEN)** register (see page 381). Corresponding bits in the drive strength registers (**GPIODR2R**, **GPIODR4R**, **GPIODR8R**, and **GPIOSLR** ) can be set to achieve the desired rise and fall times. The GPIO acts as an open-drain input if the corresponding bit in the **GPIODIR** register is cleared. If open drain is selected while the GPIO is configured as an input, the GPIO will remain an input and the open-drain selection has no effect until the GPIO is changed to an output.

When using the $I^2C$ module, in addition to configuring the pin to open drain, the **GPIO Alternate Function Select (GPIOAFSEL)** register bits for the $I^2C$ clock and data pins should be set to 1 (see examples in “Initialization and Configuration” on page 358).

GPIO Open Drain Select (GPIOODR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x50C
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | ODE | R/W | 0x00 | Output Pad Open Drain Enable<br>The ODE values are defined as follows:<br><br>Value Description<br>0 Open drain configuration is disabled.<br>1 Open drain configuration is enabled. |

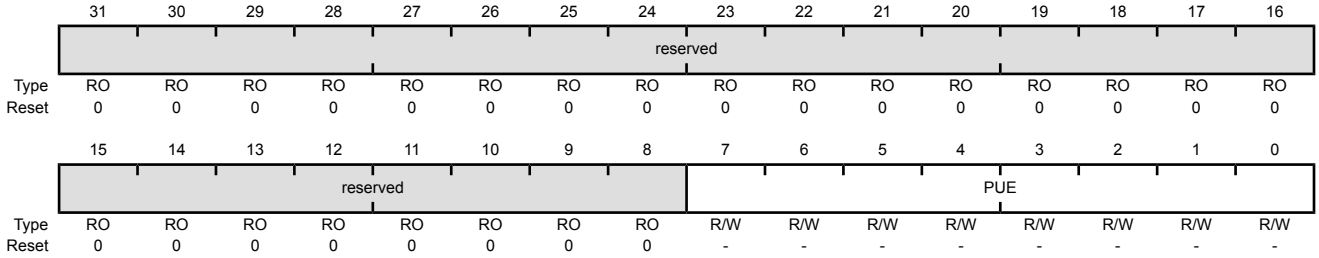
## Register 15: GPIO Pull-Up Select (GPIOPUR), offset 0x510

The **GPIOPUR** register is the pull-up control register. When a bit is set to 1, it enables a weak pull-up resistor on the corresponding GPIO signal. Setting a bit in **GPIOPUR** automatically clears the corresponding bit in the **GPIO Pull-Down Select (GPIOPDR)** register (see page 379). Write access to this register is protected with the **GPIOCR** register. Bits in **GPIOCR** that are set to 0 will prevent writes to the equivalent bit in this register.

**Note:** The GPIO commit control registers provide a layer of protection against accidental programming of critical hardware peripherals. Protection is currently provided for the `NMI` pin (`PB7`) and the four JTAG/SWD pins (`PC[3:0]`). Writes to protected bits of the **GPIO Alternate Function Select (GPIOAFSEL)** register (see page 371), **GPIO Pull-Up Select (GPIOPUR)** register (see page 377), and **GPIO Digital Enable (GPIODEN)** register (see page 381) are not committed to storage unless the **GPIO Lock (GPIOLOCK)** register (see page 383) has been unlocked and the appropriate bits of the **GPIO Commit (GPIOCR)** register (see page 384) have been set to 1.

GPIO Pull-Up Select (GPIOPUR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x510
Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:0 | PUE | R/W | - | Pad Weak Pull-Up Enable |

| Value | Description |
|---|---|
| 0 | The corresponding pin's weak pull-up resistor is disabled. |
| 1 | The corresponding pin's weak pull-up resistor is enabled. |

A write of 1 to **GPIOPDR[n]** clears the corresponding **GPIOPUR[n]** enables. The change is effective on the second clock cycle after the write.

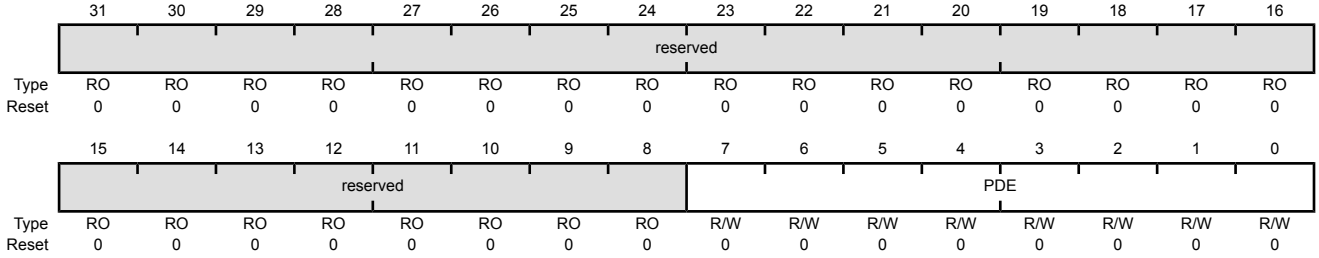
**Note:** The default reset value for the **GPIOAFSEL**, **GPIOPUR**, and **GPIODEN** registers are 0x0000.0000 for all GPIO pins, with the exception of the four JTAG/SWD pins (PC[3:0]). These four pins default to JTAG/SWD functionality. Because of this, the default reset value of these registers for Port C is 0x0000.000F.

## Register 16: GPIO Pull-Down Select (GPIOPDR), offset 0x514

The **GPIOPDR** register is the pull-down control register. When a bit is set to 1, it enables a weak pull-down resistor on the corresponding GPIO signal. Setting a bit in **GPIOPDR** automatically clears the corresponding bit in the **GPIO Pull-Up Select (GPIOPUR)** register (see page 377).

GPIO Pull-Down Select (GPIOPDR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x514
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PDE | R/W | 0x00 | Pad Weak Pull-Down Enable |

| Value | Description |
|---|---|
| 0 | The corresponding pin's weak pull-down resistor is disabled. |
| 1 | The corresponding pin's weak pull-down resistor is enabled. |

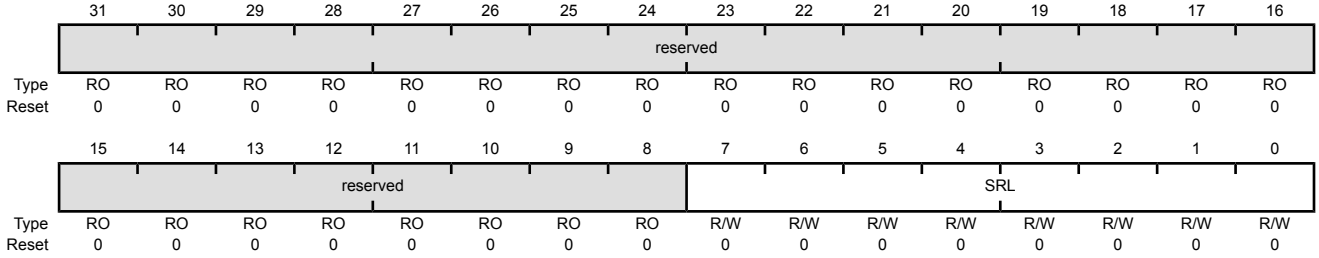
A write of 1 to **GPIOPUR[n]** clears the corresponding **GPIOPDR[n]** enables. The change is effective on the second clock cycle after the write.

## Register 17: GPIO Slew Rate Control Select (GPIOSLR), offset 0x518

The **GPIOSLR** register is the slew rate control register. Slew rate control is only available when using the 8-mA drive strength option via the **GPIO 8-mA Drive Select (GPIODR8R)** register (see page 375).

GPIO Slew Rate Control Select (GPIOSLR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x518
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | SRL | R/W | 0x00 | Slew Rate Limit Enable (8-mA drive only)<br>The SRL values are defined as follows:<br><br>Value Description<br>0 Slew rate control disabled.<br>1 Slew rate control enabled. |

## Register 18: GPIO Digital Enable (GPIODEN), offset 0x51C

**Note:** Pins configured as digital inputs are Schmitt-triggered.

The **GPIODEN** register is the digital enable register. By default, with the exception of the GPIO signals used for JTAG/SWD function, all other GPIO signals are configured out of reset to be undriven (tristate). Their digital function is disabled; they do not drive a logic value on the pin and they do not allow the pin voltage into the GPIO receiver. To use the pin in a digital function (either GPIO or alternate function), the corresponding GPIODEN bit must be set.

**Note:** The GPIO commit control registers provide a layer of protection against accidental programming of critical hardware peripherals. Protection is currently provided for the NMI pin (PB7) and the four JTAG/SWD pins (PC[3:0]). Writes to protected bits of the **GPIO Alternate Function Select (GPIOAFSEL)** register (see page 371), **GPIO Pull-Up Select (GPIOPUR)** register (see page 377), and **GPIO Digital Enable (GPIODEN)** register (see page 381) are not committed to storage unless the **GPIO Lock (GPIOLOCK)** register (see page 383) has been unlocked and the appropriate bits of the **GPIO Commit (GPIOCR)** register (see page 384) have been set to 1.

GPIO Digital Enable (GPIODEN)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x51C
Type R/W, reset -

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | DEN | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - | - | - | - | - | - | - | - |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

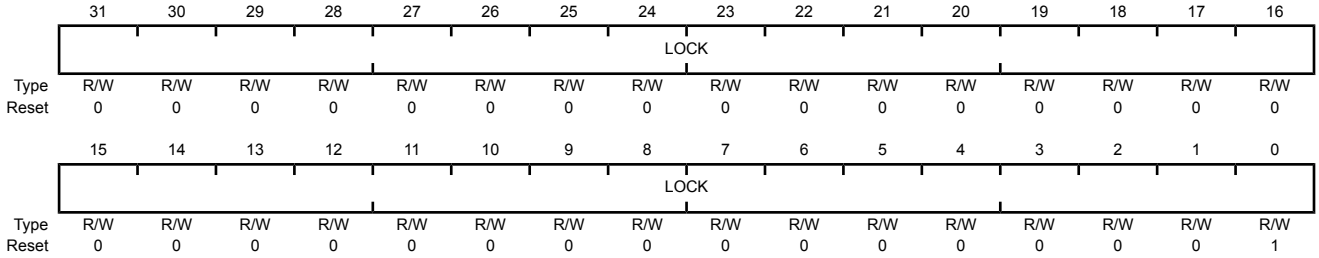
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:0 | DEN | R/W | - | Digital Enable<br>The DEN values are defined as follows:<br><br>Value Description<br>0 Digital functions disabled.<br>1 Digital functions enabled.<br><br>**Note:** The default reset value for the **GPIOAFSEL**, **GPIOPUR**, and **GPIODEN** registers are 0x0000.0000 for all GPIO pins, with the exception of the four JTAG/SWD pins (PC[3:0]). These four pins default to JTAG/SWD functionality. Because of this, the default reset value of these registers for Port C is 0x0000.000F. |

## Register 19: GPIO Lock (GPIOLOCK), offset 0x520

The **GPIOLOCK** register enables write access to the **GPIOCR** register (see page 384). Writing 0x0x4C4F.434B to the **GPIOLOCK** register will unlock the **GPIOCR** register. Writing any other value to the **GPIOLOCK** register re-enables the locked state. Reading the **GPIOLOCK** register returns the lock status rather than the 32-bit value that was previously written. Therefore, when write accesses are disabled, or locked, reading the **GPIOLOCK** register returns 0x00000001. When write accesses are enabled, or unlocked, reading the **GPIOLOCK** register returns 0x00000000.

GPIO Lock (GPIOLOCK)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x520
Type R/W, reset 0x0000.0001




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | LOCK | R/W | 0x0000.0001 | GPIO Lock<br>A write of the value 0x4C4F.434B unlocks the **GPIO Commit (GPIOCR)** register for write access.<br>A write of any other value or a write to the **GPIOCR** register reapplies the lock, preventing any register updates. A read of this register returns the following values:<br>Value Description<br>0x0000.0001 Locked<br>0x0000.0000 Unlocked |

## Register 20: GPIO Commit (GPIOCR), offset 0x524

The **GPIOCR** register is the commit register. The value of the **GPIOCR** register determines which bits of the **GPIOAFSEL**, **GPIOPUR**, and **GPIODEN** registers are committed when a write to these registers is performed. If a bit in the **GPIOCR** register is zero, the data being written to the corresponding bit in the **GPIOAFSEL**, **GPIOPUR**, or **GPIODEN** registers cannot be committed and retains its previous value. If a bit in the **GPIOCR** register is set, the data being written to the corresponding bit of the **GPIOAFSEL**, **GPIOPUR**, or **GPIODEN** registers is committed to the register and reflects the new value.

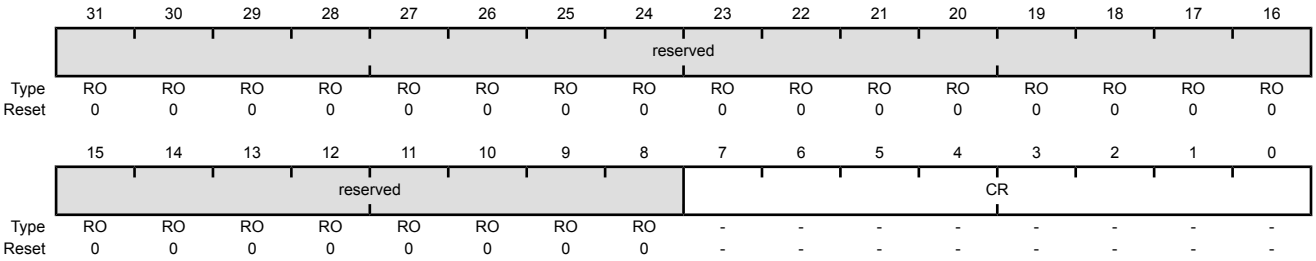
The contents of the **GPIOCR** register can only be modified if the **GPIOLOCK** register is unlocked. Writes to the **GPIOCR** register are ignored if the **GPIOLOCK** register is locked.

**Important:** This register is designed to prevent accidental programming of the registers that control connectivity to the NMI and JTAG/SWD debug hardware. By initializing the bits of the **GPIOCR** register to 0 for PB7 and PC[3:0], the NMI and JTAG/SWD debug port can only be converted to GPIOs through a deliberate set of writes to the **GPIOLOCK**, **GPIOCR**, and the corresponding registers.

Because this protection is currently only implemented on the NMI and JTAG/SWD pins on PB7 and PC[3:0], all of the other bits in the **GPIOCR** registers cannot be written with 0x0. These bits are hardwired to 0x1, ensuring that it is always possible to commit new values to the **GPIOAFSEL**, **GPIOPUR**, or **GPIODEN** register bits of these other pins.

GPIO Commit (GPIOCR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x524
Type -, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:0 | CR | - | - | GPIO Commit<br>On a bit-wise basis, any bit set allows the corresponding **GPIOAFSEL**, **GPIOPUR**, or **GPIODEN** registers to be written.<br>**Note:** The default register type for the **GPIOCR** register is RO for all GPIO pins with the exception of the NMI pin and the four JTAG/SWD pins (PB7 and PC[3:0]). These five pins are currently the only GPIOs that are protected by the **GPIOCR** register. Because of this, the register type for GPIO Port B7 and GPIO Port C[3:0] is R/W.<br>The default reset value for the **GPIOCR** register is 0x0000.00FF for all GPIO pins, with the exception of the NMI pin and the four JTAG/SWD pins (PB7 and PC[3:0]). To ensure that the JTAG port is not accidentally programmed as a GPIO, these four pins default to non-committable. To ensure that the NMI pin is not accidentally programmed as the non-maskable interrupt pin, it defaults to non-committable. Because of this, the default reset value of **GPIOCR** for GPIO Port B is 0x0000.007F while the default reset value of GPIOCR for Port C is 0x0000.00F0. |

## Register 21: GPIO Analog Mode Select (GPIOAMSEL), offset 0x528

**Important:** This register is only valid for ports D and E.

If any pin is to be used as an ADC input, the appropriate bit in **GPIOAMSEL** must be written to 1 to disable the analog isolation circuit.

The **GPIOAMSEL** register controls isolation circuits to the analog side of a unified I/O pad. Because the GPIOs may be driven by a 5V source and affect analog operation, analog circuitry requires isolation from the pins when not used in their analog function.

Each bit of this register controls the isolation circuitry for circuits that share the same pin as the GPIO bit lane.

GPIO Analog Mode Select (GPIOAMSEL)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0x528
Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | GPIOAMSEL | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3:0 | GPIOAMSEL | R/W | 0x00 | GPIO Analog Mode Select |

| Value | Description |
|---|---|
| 0 | Analog function of the pin is disabled, the isolation is enabled, and the pin is capable of digital functions as specified by the other GPIO configuration registers. |
| 1 | Analog function of the pin is enabled, the isolation is disabled, and the pin is capable of analog functions. |

**Note:** This register and bits are required only for GPIO bit lanes that share analog function through a unified I/O pad.

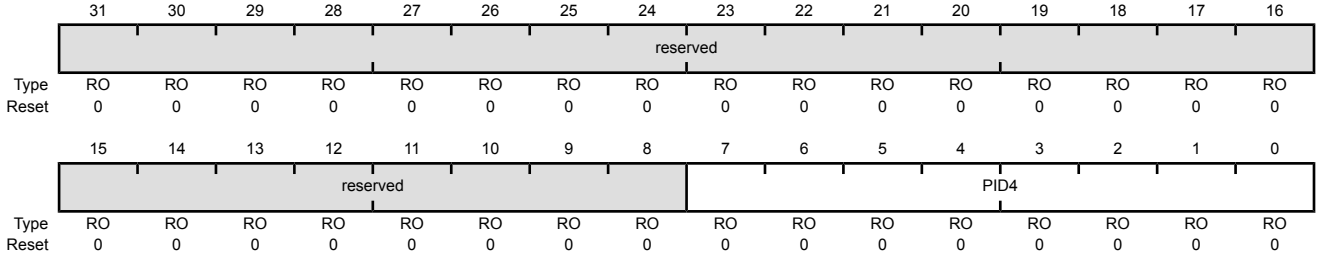
The reset state of this register is 0 for all bit lanes.

## Register 22: GPIO Peripheral Identification 4 (GPIOPeriphID4), offset 0xFD0

The **GPIOPeriphID4**, **GPIOPeriphID5**, **GPIOPeriphID6**, and **GPIOPeriphID7** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 4 (GPIOPeriphID4)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFD0
Type RO, reset 0x0000.0000



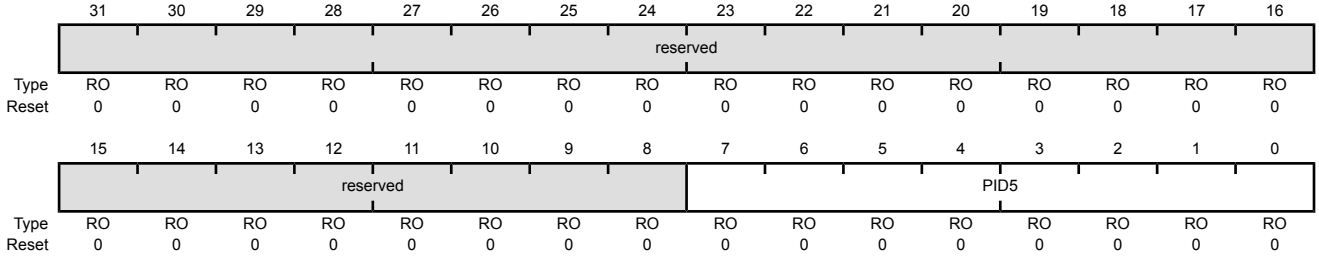

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID4 | RO | 0x00 | GPIO Peripheral ID Register[7:0] |

## Register 23: GPIO Peripheral Identification 5 (GPIOPeriphID5), offset 0xFD4

The **GPIOPeriphID4**, **GPIOPeriphID5**, **GPIOPeriphID6**, and **GPIOPeriphID7** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 5 (GPIOPeriphID5)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFD4
Type RO, reset 0x0000.0000



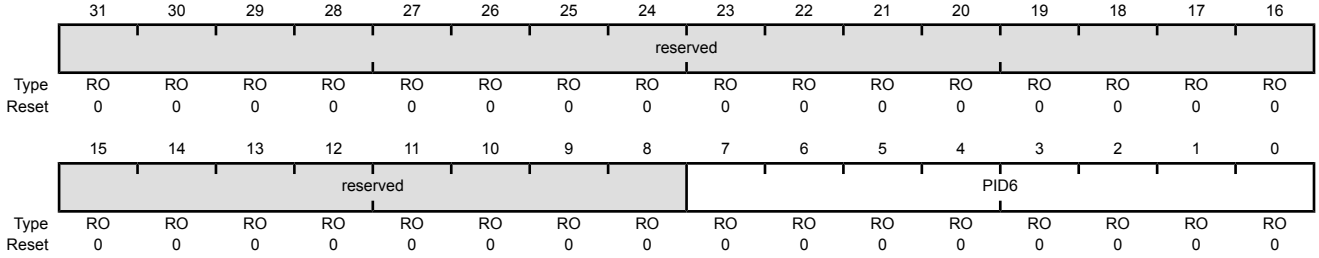

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID5 | RO | 0x00 | GPIO Peripheral ID Register[15:8] |

## Register 24: GPIO Peripheral Identification 6 (GPIOPeriphID6), offset 0xFD8

The **GPIOPeriphID4**, **GPIOPeriphID5**, **GPIOPeriphID6**, and **GPIOPeriphID7** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 6 (GPIOPeriphID6)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFD8
Type RO, reset 0x0000.0000



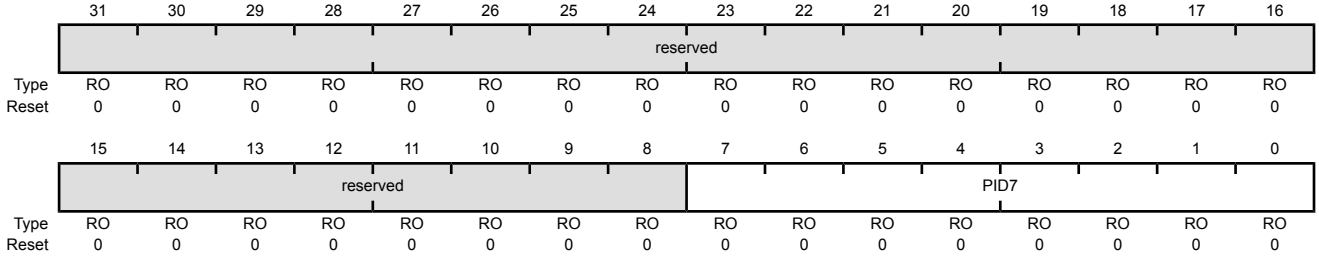

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID6 | RO | 0x00 | GPIO Peripheral ID Register[23:16] |

## Register 25: GPIO Peripheral Identification 7 (GPIOPeriphID7), offset 0xFDC

The **GPIOPeriphID4**, **GPIOPeriphID5**, **GPIOPeriphID6**, and **GPIOPeriphID7** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 7 (GPIOPeriphID7)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFDC
Type RO, reset 0x0000.0000



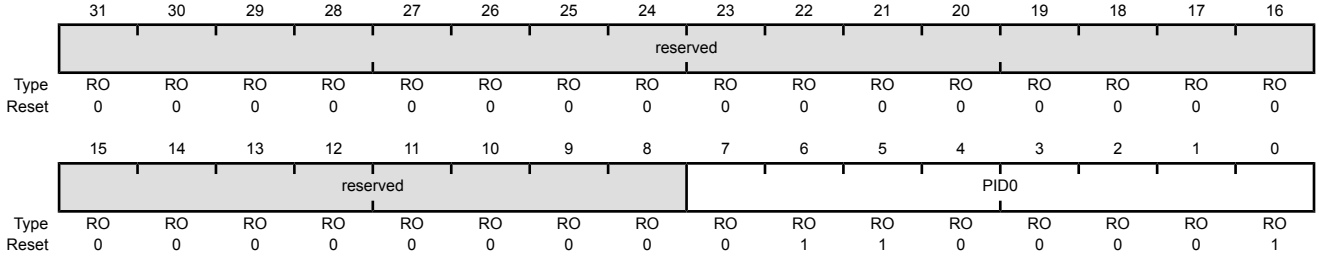

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID7 | RO | 0x00 | GPIO Peripheral ID Register[31:24] |

## Register 26: GPIO Peripheral Identification 0 (GPIOPeriphID0), offset 0xFE0

The **GPIOPeriphID0**, **GPIOPeriphID1**, **GPIOPeriphID2**, and **GPIOPeriphID3** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 0 (GPIOPeriphID0)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFE0
Type RO, reset 0x0000.0061



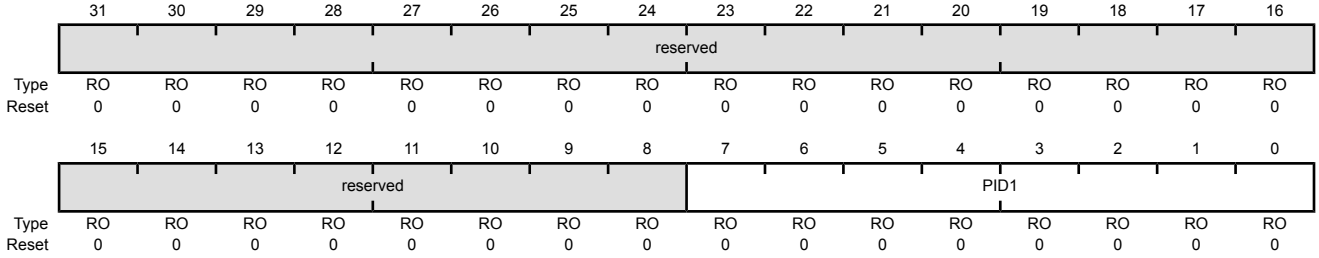

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID0 | RO | 0x61 | GPIO Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

## Register 27: GPIO Peripheral Identification 1 (GPIOPeriphID1), offset 0xFE4

The **GPIOPeriphID0**, **GPIOPeriphID1**, **GPIOPeriphID2**, and **GPIOPeriphID3** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 1 (GPIOPeriphID1)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFE4
Type RO, reset 0x0000.0000



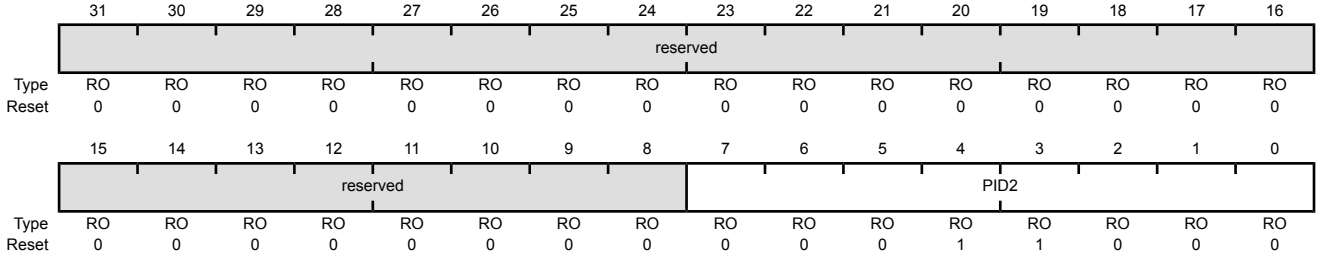

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID1 | RO | 0x00 | GPIO Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

## Register 28: GPIO Peripheral Identification 2 (GPIOPeriphID2), offset 0xFE8

The **GPIOPeriphID0**, **GPIOPeriphID1**, **GPIOPeriphID2**, and **GPIOPeriphID3** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 2 (GPIOPeriphID2)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFE8
Type RO, reset 0x0000.0018



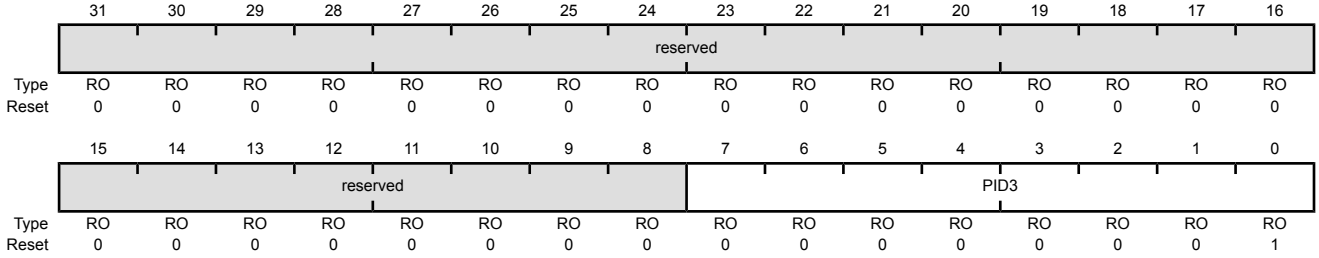

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID2 | RO | 0x18 | GPIO Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 29: GPIO Peripheral Identification 3 (GPIOPeriphID3), offset 0xFEC

The **GPIOPeriphID0**, **GPIOPeriphID1**, **GPIOPeriphID2**, and **GPIOPeriphID3** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 3 (GPIOPeriphID3)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFEC
Type RO, reset 0x0000.0001



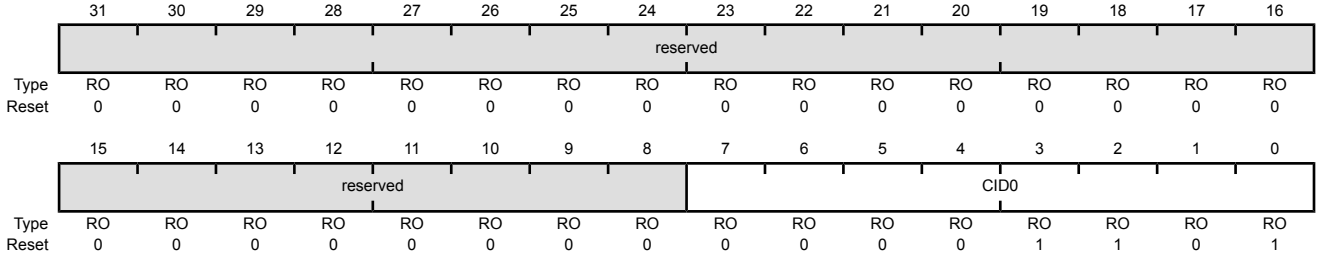

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID3 | RO | 0x01 | GPIO Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

## Register 30: GPIO PrimeCell Identification 0 (GPIOPCellID0), offset 0xFF0

The **GPIOPCellID0**, **GPIOPCellID1**, **GPIOPCellID2**, and **GPIOPCellID3** registers are four 8-bit wide registers, that can conceptually be treated as one 32-bit register. The register is used as a standard cross-peripheral identification system.

GPIO PrimeCell Identification 0 (GPIOPCellID0)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFF0
Type RO, reset 0x0000.000D



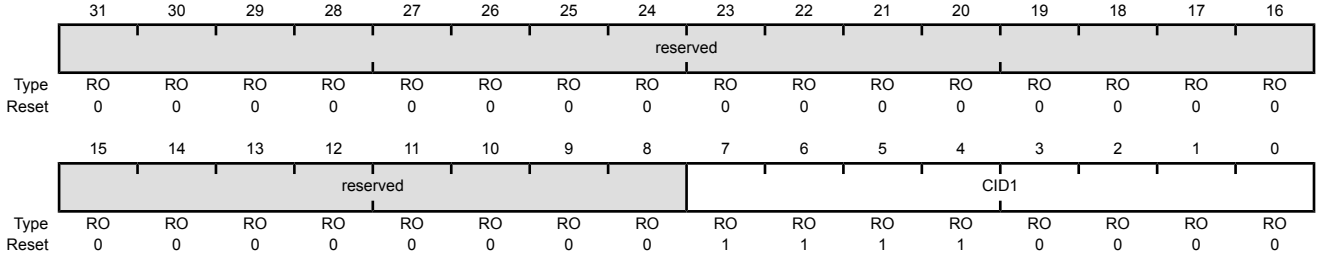

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID0 | RO | 0x0D | GPIO PrimeCell ID Register[7:0]<br>Provides software a standard cross-peripheral identification system. |

## Register 31: GPIO PrimeCell Identification 1 (GPIOPCellID1), offset 0xFF4

The **GPIOPCellID0**, **GPIOPCellID1**, **GPIOPCellID2**, and **GPIOPCellID3** registers are four 8-bit wide registers, that can conceptually be treated as one 32-bit register. The register is used as a standard cross-peripheral identification system.

GPIO PrimeCell Identification 1 (GPIOPCellID1)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFF4
Type RO, reset 0x0000.00F0



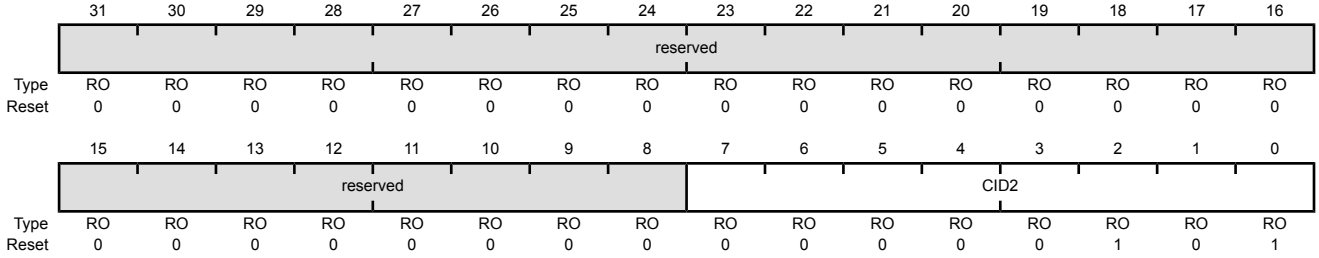

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID1 | RO | 0xF0 | GPIO PrimeCell ID Register[15:8]<br>Provides software a standard cross-peripheral identification system. |

## Register 32: GPIO PrimeCell Identification 2 (GPIOPCellID2), offset 0xFF8

The **GPIOPCellID0**, **GPIOPCellID1**, **GPIOPCellID2**, and **GPIOPCellID3** registers are four 8-bit wide registers, that can conceptually be treated as one 32-bit register. The register is used as a standard cross-peripheral identification system.

GPIO PrimeCell Identification 2 (GPIOPCellID2)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFF8
Type RO, reset 0x0000.0005



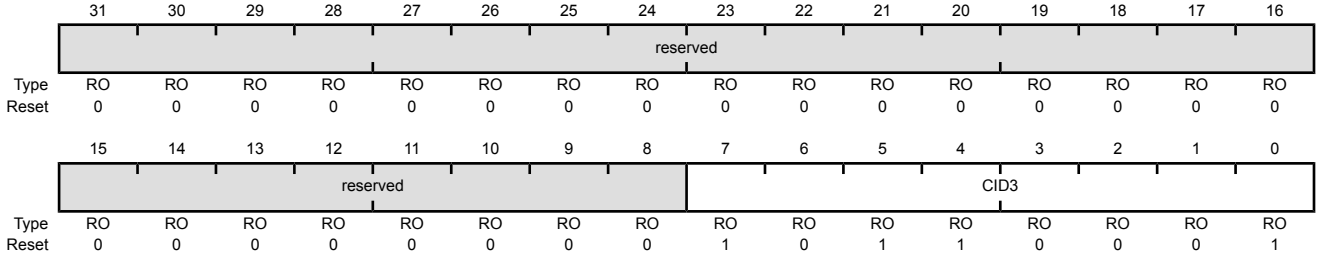

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID2 | RO | 0x05 | GPIO PrimeCell ID Register[23:16]<br>Provides software a standard cross-peripheral identification system. |

## Register 33: GPIO PrimeCell Identification 3 (GPIOPCellID3), offset 0xFFC

The **GPIOPCellID0**, **GPIOPCellID1**, **GPIOPCellID2**, and **GPIOPCellID3** registers are four 8-bit wide registers, that can conceptually be treated as one 32-bit register. The register is used as a standard cross-peripheral identification system.

GPIO PrimeCell Identification 3 (GPIOPCellID3)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
Offset 0xFFC
Type RO, reset 0x0000.00B1




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID3 | RO | 0xB1 | GPIO PrimeCell ID Register[31:24]<br>Provides software a standard cross-peripheral identification system. |

# 10 General-Purpose Timers

Programmable timers can be used to count or time external events that drive the Timer input pins. The Stellaris® General-Purpose Timer Module (GPTM) contains three GPTM blocks (Timer0, Timer1, and Timer 2). Each GPTM block provides two 16-bit timers/counters (referred to as TimerA and TimerB) that can be configured to operate independently as timers or event counters, or configured to operate as one 32-bit timer or one 32-bit Real-Time Clock (RTC).

In addition, timers can be used to trigger analog-to-digital conversions (ADC). The ADC trigger signals from all of the general-purpose timers are ORed together before reaching the ADC module, so only one timer should be used to trigger ADC events.

The GPT Module is one timing resource available on the Stellaris microcontrollers. Other timer resources include the System Timer (SysTick) (see 95).

The General-Purpose Timers provide the following features:

- Three General-Purpose Timer Modules (GPTM), each of which provides two 16-bit timers/counters. Each GPTM can be configured to operate independently:
  - As a single 32-bit timer
  - As one 32-bit Real-Time Clock (RTC) to event capture
  - For Pulse Width Modulation (PWM)
  - To trigger analog-to-digital conversions
- 32-bit Timer modes
  - Programmable one-shot timer
  - Programmable periodic timer
  - Real-Time Clock when using an external 32.768-KHz clock as the input
  - User-enabled stalling when the controller asserts CPU Halt flag during debug
  - ADC event trigger
- 16-bit Timer modes
  - General-purpose timer function with an 8-bit prescaler (for one-shot and periodic modes only)
  - Programmable one-shot timer
  - Programmable periodic timer
  - User-enabled stalling when the controller asserts CPU Halt flag during debug
  - ADC event trigger
- 16-bit Input Capture modes
  - Input edge count capture

  – Input edge time capture

- 16-bit PWM mode
  – Simple PWM mode with software-programmable output inversion of the PWM signal

## 10.1 Block Diagram

**Note:** In Figure 10-1 on page 400, the specific CCP pins available depend on the Stellaris device. See Table 10-1 on page 400 for the available CCPs.

**Figure 10-1. GPTM Module Block Diagram**

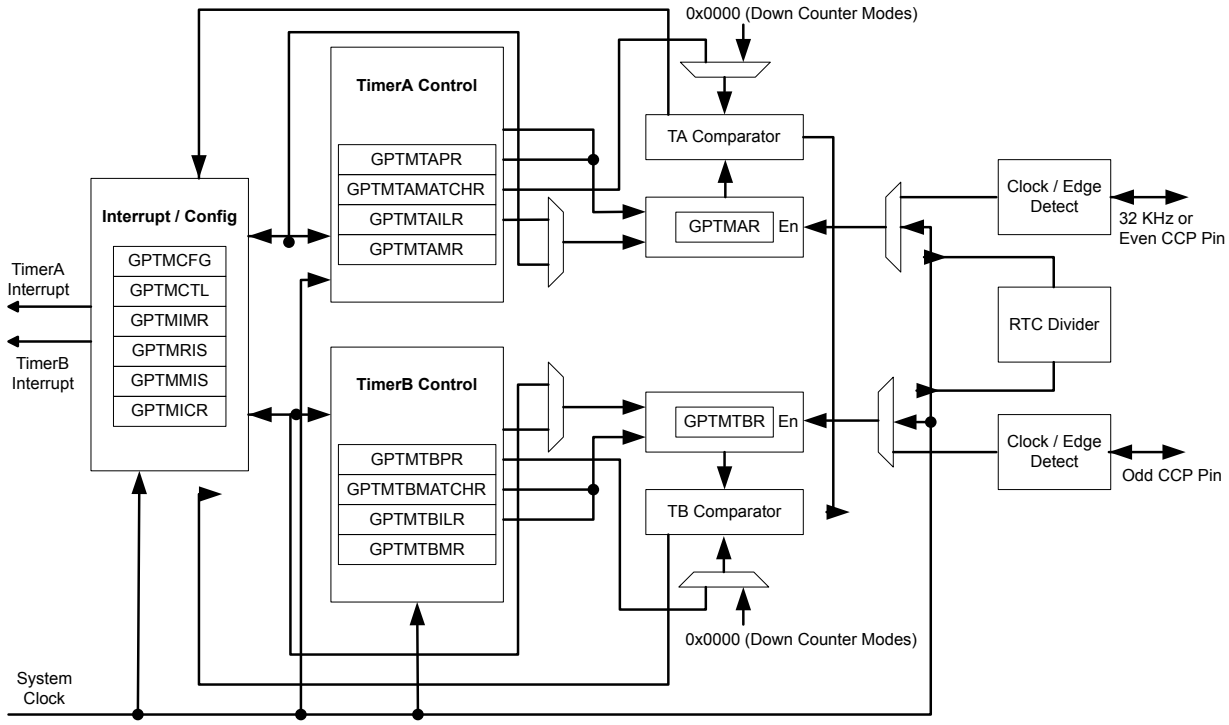



**Table 10-1. Available CCP Pins**

| Timer | 16-Bit Up/Down Counter | Even CCP Pin | Odd CCP Pin |
|---|---|---|---|
| Timer 0 | TimerA | CCP0 | - |
| | TimerB | - | CCP1 |
| Timer 1 | TimerA | CCP2 | - |
| | TimerB | - | CCP3 |
| Timer 2 | TimerA | CCP4 | - |
| | TimerB | - | - |

## 10.2 Signal Description

Table 10-2 on page 401 lists the external signals of the GP Timer module and describes the function of each. The GP Timer signals are alternate functions for some GPIO signals and default to be GPIO signals at reset. The column in the table below titled "Pin Assignment" lists the possible GPIO

pin placements for these GP Timer signals. The AFSEL bit in the **GPIO Alternate Function Select (GPIOAFSEL)** register (page 371) should be set to choose the GP Timer function. For more information on configuring GPIOs, see "General-Purpose Input/Outputs (GPIOs)" on page 351.

**Table 10-2. General-Purpose Timers Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| CCP0 | 57 | I/O | TTL | Capture/Compare/PWM 0. |
| CCP1 | 56 | I/O | TTL | Capture/Compare/PWM 1. |
| CCP2 | 11 | I/O | TTL | Capture/Compare/PWM 2. |
| CCP3 | 8 | I/O | TTL | Capture/Compare/PWM 3. |
| CCP4 | 16 | I/O | TTL | Capture/Compare/PWM 4. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 10.3 Functional Description

The main components of each GPTM block are two free-running 16-bit up/down counters (referred to as TimerA and TimerB), two 16-bit match registers, and two 16-bit load/initialization registers and their associated control functions. The exact functionality of each GPTM is controlled by software and configured through the register interface.

Software configures the GPTM using the **GPTM Configuration (GPTMCFG)** register (see page 411), the **GPTM TimerA Mode (GPTMTAMR)** register (see page 412), and the **GPTM TimerB Mode (GPTMTBMR)** register (see page 414). When in one of the 32-bit modes, the timer can only act as a 32-bit timer. However, when configured in 16-bit mode, the GPTM can have its two 16-bit timers configured in any combination of the 16-bit modes.

### 10.3.1 GPTM Reset Conditions

After reset has been applied to the GPTM module, the module is in an inactive state, and all control registers are cleared and in their default states. Counters TimerA and TimerB are initialized to 0xFFFF, along with their corresponding load registers: the **GPTM TimerA Interval Load (GPTMTAILR)** register (see page 425) and the **GPTM TimerB Interval Load (GPTMTBILR)** register (see page 426). The prescale counters are initialized to 0x00: the **GPTM TimerA Prescale (GPTMTAPR)** register (see page 429) and the **GPTM TimerB Prescale (GPTMTBPR)** register (see page 430).

### 10.3.2 32-Bit Timer Operating Modes

This section describes the three GPTM 32-bit timer modes (One-Shot, Periodic, and RTC) and their configuration.

The GPTM is placed into 32-bit mode by writing a 0 (One-Shot/Periodic 32-bit timer mode) or a 1 (RTC mode) to the **GPTM Configuration (GPTMCFG)** register. In both configurations, certain GPTM registers are concatenated to form pseudo 32-bit registers. These registers include:

- **GPTM TimerA Interval Load (GPTMTAILR)** register [15:0], see page 425
- **GPTM TimerB Interval Load (GPTMTBILR)** register [15:0], see page 426
- **GPTM TimerA (GPTMTAR)** register [15:0], see page 431
- **GPTM TimerB (GPTMTBR)** register [15:0], see page 432

In the 32-bit modes, the GPTM translates a 32-bit write access to **GPTMTAILR** into a write access to both **GPTMTAILR** and **GPTMTBILR**. The resulting word ordering for such a write operation is:

```
GPTMTBILR[15:0]:GPTMTAILR[15:0]
```

Likewise, a read access to **GPTMTAR** returns the value:

```
GPTMTBR[15:0]:GPTMTAR[15:0]
```

#### 10.3.2.1 32-Bit One-Shot/Periodic Timer Mode

In 32-bit one-shot and periodic timer modes, the concatenated versions of the TimerA and TimerB registers are configured as a 32-bit down-counter. The selection of one-shot or periodic mode is determined by the value written to the TAMR field of the **GPTM TimerA Mode (GPTMTAMR)** register (see page 412), and there is no need to write to the **GPTM TimerB Mode (GPTMTBMR)** register.

When software writes the TAEN bit in the **GPTM Control (GPTMCTL)** register (see page 416), the timer begins counting down from its preloaded value. Once the 0x0000.0000 state is reached, the timer reloads its start value from the concatenated **GPTMTAILR** on the next cycle. If configured to be a one-shot timer, the timer stops counting and clears the TAEN bit in the **GPTMCTL** register. If configured as a periodic timer, it continues counting.

In addition to reloading the count value, the GPTM generates interrupts and triggers when it reaches the 0x000.0000 state. The GPTM sets the TATORIS bit in the **GPTM Raw Interrupt Status (GPTMRIS)** register (see page 421), and holds it until it is cleared by writing the **GPTM Interrupt Clear (GPTMICR)** register (see page 423). If the time-out interrupt is enabled in the **GPTM Interrupt Mask (GPTMIMR)** register (see page 419), the GPTM also sets the TATOMIS bit in the **GPTM Masked Interrupt Status (GPTMMIS)** register (see page 422). The ADC trigger is enabled by setting the TAOTE bit in **GPTMCTL**.

If software reloads the **GPTMTAILR** register while the counter is running, the counter loads the new value on the next clock cycle and continues counting from the new value.

If the TASTALL bit in the **GPTMCTL** register is set, the timer freezes counting while the processor is halted by the debugger. The timer resumes counting when the processor resumes execution.

#### 10.3.2.2 32-Bit Real-Time Clock Timer Mode

In Real-Time Clock (RTC) mode, the concatenated versions of the TimerA and TimerB registers are configured as a 32-bit up-counter. When RTC mode is selected for the first time, the counter is loaded with a value of 0x0000.0001. All subsequent load values must be written to the **GPTM TimerA Match (GPTMTAMATCHR)** register (see page 427) by the controller.

The input clock on an even CCP input is required to be 32.768 KHz in RTC mode. The clock signal is then divided down to a 1 Hz rate and is passed along to the input of the 32-bit counter.

When software writes the TAEN bit inthe **GPTMCTL** register, the counter starts counting up from its preloaded value of 0x0000.0001. When the current count value matches the preloaded value in the **GPTMTAMATCHR** register, it rolls over to a value of 0x0000.0000 and continues counting until either a hardware reset, or it is disabled by software (clearing the TAEN bit). When a match occurs, the GPTM asserts the RTCRIS bit in **GPTMRIS**. If the RTC interrupt is enabled in **GPTMIMR**, the GPTM also sets the RTCMIS bit in **GPTMMIS** and generates a controller interrupt. The status flags are cleared by writing the RTCCINT bit in **GPTMICR**.

If the TASTALL and/or TBSTALL bits in the **GPTMCTL** register are set, the timer does not freeze if the RTCEN bit is set in **GPTMCTL**.

### 10.3.3 16-Bit Timer Operating Modes

The GPTM is placed into global 16-bit mode by writing a value of 0x4 to the **GPTM Configuration (GPTMCFG)** register (see page 411). This section describes each of the GPTM 16-bit modes of operation. TimerA and TimerB have identical modes, so a single description is given using an **n** to reference both.

#### 10.3.3.1 16-Bit One-Shot/Periodic Timer Mode

In 16-bit one-shot and periodic timer modes, the timer is configured as a 16-bit down-counter with an optional 8-bit prescaler that effectively extends the counting range of the timer to 24 bits. The selection of one-shot or periodic mode is determined by the value written to the TnMR field of the **GPTMTnMR** register. The optional prescaler is loaded into the **GPTM Timern Prescale (GPTMTnPR)** register.

When software writes the TnEN bit in the **GPTMCTL** register, the timer begins counting down from its preloaded value. Once the 0x0000 state is reached, the timer reloads its start value from **GPTMTnILR** and **GPTMTnPR** on the next cycle. If configured to be a one-shot timer, the timer stops counting and clears the TnEN bit in the **GPTMCTL** register. If configured as a periodic timer, it continues counting.

In addition to reloading the count value, the timer generates interrupts and triggers when it reaches the 0x0000 state. The GPTM sets the TnTORIS bit in the **GPTMRIS** register, and holds it until it is cleared by writing the **GPTMICR** register. If the time-out interrupt is enabled in **GPTMIMR**, the GPTM also sets the TnTOMIS bit in **GPTMISR** and generates a controller interrupt. The ADC trigger is enabled by setting the TnOTE bit in the **GPTMCTL** register.

If software reloads the **GPTMTAILR** register while the counter is running, the counter loads the new value on the next clock cycle and continues counting from the new value.

If the TnSTALL bit in the **GPTMCTL** register is set, the timer freezes counting while the processor is halted by the debugger. The timer resumes counting when the processor resumes execution.

The following example shows a variety of configurations for a 16-bit free running timer while using the prescaler. All values assume a 50-MHz clock with Tc=20 ns (clock period).

**Table 10-3. 16-Bit Timer With Prescaler Configurations**

| Prescale | #Clock (T c)[a] | Max Time | Units |
|---|---|---|---|
| 00000000 | 1 | 1.3107 | mS |
| 00000001 | 2 | 2.6214 | mS |
| 00000010 | 3 | 3.9322 | mS |
| ---------- | -- | -- | -- |
| 11111101 | 254 | 332.9229 | mS |
| 11111110 | 255 | 334.2336 | mS |
| 11111111 | 256 | 335.5443 | mS |

a. Tc is the clock period.

#### 10.3.3.2 16-Bit Input Edge Count Mode

**Note:** For rising-edge detection, the input signal must be High for at least two system clock periods following the rising edge. Similarly, for falling-edge detection, the input signal must be Low for at least two system clock periods following the falling edge. Based on this criteria, the maximum input frequency for edge detection is 1/4 of the system frequency.

**Note:** The prescaler is not available in 16-Bit Input Edge Count mode.

In Edge Count mode, the timer is configured as a down-counter capable of capturing three types of events: rising edge, falling edge, or both. To place the timer in Edge Count mode, the TnCMR bit of the **GPTMTnMR** register must be set to 0. The type of edge that the timer counts is determined by the TnEVENT fields of the **GPTMCTL** register. During initialization, the **GPTM Timern Match (GPTMTnMATCHR)** register is configured so that the difference between the value in the **GPTMTnILR** register and the **GPTMTnMATCHR** register equals the number of edge events that must be counted.

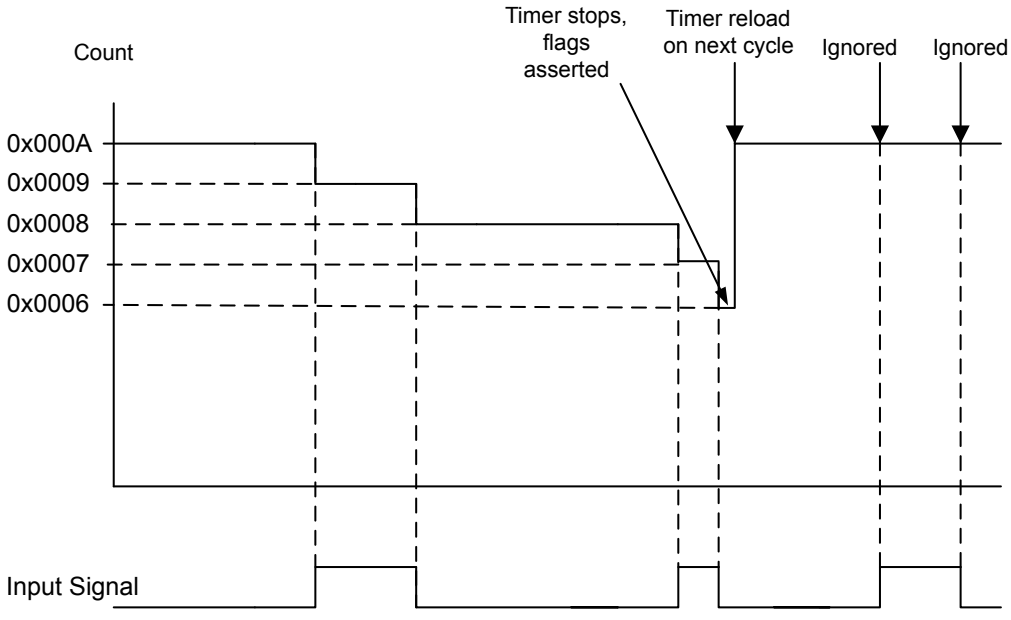
When software writes the TnEN bit in the **GPTM Control (GPTMCTL)** register, the timer is enabled for event capture. Each input event on the CCP pin decrements the counter by 1 until the event count matches **GPTMTnMATCHR**. When the counts match, the GPTM asserts the CnMRIS bit in the **GPTMRIS** register (and the CnMMIS bit, if the interrupt is not masked).

The counter is then reloaded using the value in **GPTMTnILR**, and stopped since the GPTM automatically clears the TnEN bit in the **GPTMCTL** register. Once the event count has been reached, all further events are ignored until TnEN is re-enabled by software.

Figure 10-2 on page 404 shows how input edge count mode works. In this case, the timer start value is set to **GPTMTnILR** =0x000A and the match value is set to **GPTMTnMATCHR** =0x0006 so that four edge events are counted. The counter is configured to detect both edges of the input signal.

Note that the last two edges are not counted since the timer automatically clears the TnEN bit after the current count matches the value in the **GPTMTnMATCHR** register.

**Figure 10-2. 16-Bit Input Edge Count Mode Example**




### 10.3.3.3 16-Bit Input Edge Time Mode

**Note:** For rising-edge detection, the input signal must be High for at least two system clock periods following the rising edge. Similarly, for falling edge detection, the input signal must be Low for at least two system clock periods following the falling edge. Based on this criteria, the maximum input frequency for edge detection is 1/4 of the system frequency.

**Note:** The prescaler is not available in 16-Bit Input Edge Time mode.

In Edge Time mode, the timer is configured as a free-running down-counter initialized to the value loaded in the **GPTMTnILR** register (or 0xFFFF at reset). The timer is capable of capturing three types of events: rising edge, falling edge, or both. The timer is placed into Edge Time mode by setting the TnCMR bit in the **GPTMTnMR** register, and the type of event that the timer captures is determined by the TnEVENT fields of the **GPTMCTL** register.

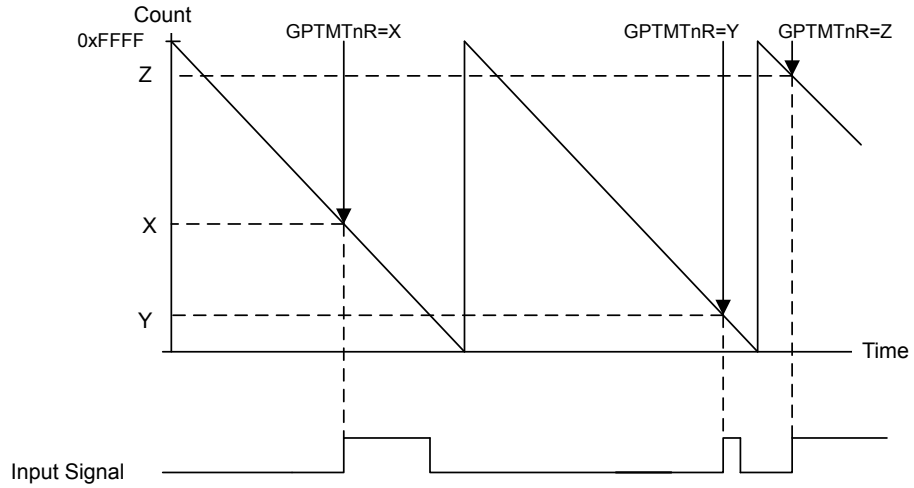
When software writes the TnEN bit in the **GPTMCTL** register, the timer is enabled for event capture. When the selected input event is detected, the current Tn counter value is captured in the **GPTMTnR** register and is available to be read by the controller. The GPTM then asserts the CnERIS bit (and the CnEMIS bit, if the interrupt is not masked).

After an event has been captured, the timer does not stop counting. It continues to count until the TnEN bit is cleared. When the timer reaches the 0x0000 state, it is reloaded with the value from the **GPTMTnILR** register.

Figure 10-3 on page 405 shows how input edge timing mode works. In the diagram, it is assumed that the start value of the timer is the default value of 0xFFFF, and the timer is configured to capture rising edge events.

Each time a rising edge event is detected, the current count value is loaded into the **GPTMTnR** register, and is held there until another rising edge is detected (at which point the new count value is loaded into **GPTMTnR**).

**Figure 10-3. 16-Bit Input Edge Time Mode Example**




### 10.3.3.4 16-Bit PWM Mode

**Note:** The prescaler is not available in 16-Bit PWM mode.

The GPTM supports a simple PWM generation mode. In PWM mode, the timer is configured as a down-counter with a start value (and thus period) defined by **GPTMTnILR**. In this mode, the PWM frequency and period are synchronous events and therefore guaranteed to be glitch free. PWM mode is enabled with the **GPTMTnMR** register by setting the TnAMS bit to 0x1, the TnCMR bit to 0x0, and the TnMR field to 0x2.

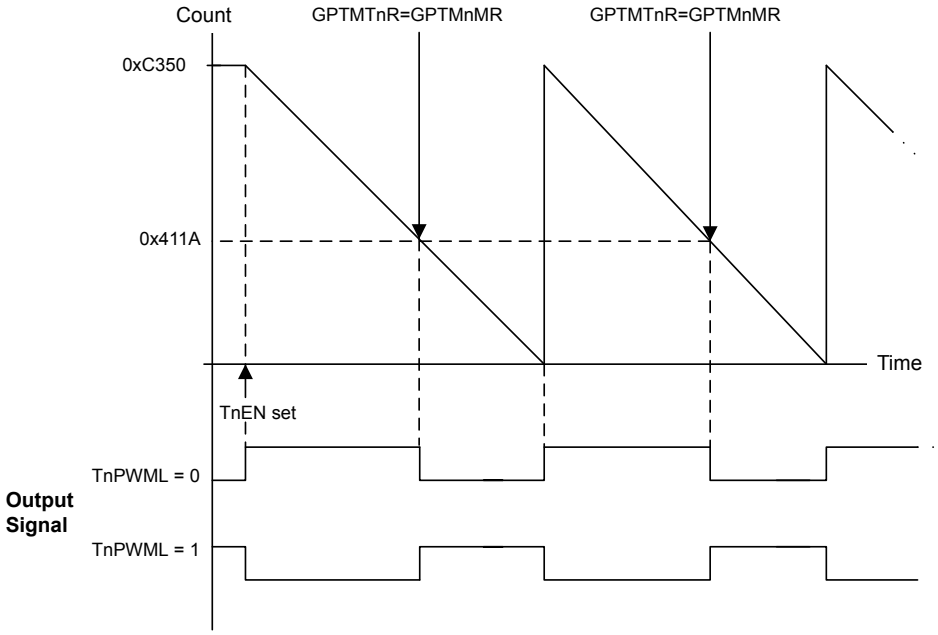
When software writes the TnEN bit in the **GPTMCTL** register, the counter begins counting down until it reaches the 0x0000 state. On the next counter cycle, the counter reloads its start value from

**GPTMTnILR** and continues counting until disabled by software clearing the TnEN bit in the **GPTMCTL** register. No interrupts or status bits are asserted in PWM mode.

The output PWM signal asserts when the counter is at the value of the **GPTMTnILR** register (its start state), and is deasserted when the counter value equals the value in the **GPTM Timern Match Register (GPTMTnMATCHR)**. Software has the capability of inverting the output PWM signal by setting the TnPWML bit in the **GPTMCTL** register.

Figure 10-4 on page 406 shows how to generate an output PWM with a 1-ms period and a 66% duty cycle assuming a 50-MHz input clock and **TnPWML** =0 (duty cycle would be 33% for the **TnPWML** =1 configuration). For this example, the start value is **GPTMTnIRL**=0xC350 and the match value is **GPTMTnMATCHR**=0x411A.

**Figure 10-4. 16-Bit PWM Mode Example**




## 10.4 Initialization and Configuration

To use the general-purpose timers, the peripheral clock must be enabled by setting the TIMER0, TIMER1, and TIMER2 bits in the **RCGC1** register.

This section shows module initialization and configuration examples for each of the supported timer modes.

### 10.4.1 32-Bit One-Shot/Periodic Timer Mode

The GPTM is configured for 32-bit One-Shot and Periodic modes by the following sequence:

1. Ensure the timer is disabled (the TAEN bit in the **GPTMCTL** register is cleared) before making any changes.
2. Write the **GPTM Configuration Register (GPTMCFG)** with a value of 0x0.

3. Set the TAMR field in the **GPTM TimerA Mode Register (GPTMTAMR)**:
   a. Write a value of 0x1 for One-Shot mode.
   b. Write a value of 0x2 for Periodic mode.
4. Load the start value into the **GPTM TimerA Interval Load Register (GPTMTAILR)**.
5. If interrupts are required, set the TATOIM bit in the **GPTM Interrupt Mask Register (GPTMIMR)**.
6. Set the TAEN bit in the **GPTMCTL** register to enable the timer and start counting.
7. Poll the TATORIS bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the TATOCINT bit of the **GPTM Interrupt Clear Register (GPTMICR)**.

In One-Shot mode, the timer stops counting after step 7 on page 407. To re-enable the timer, repeat the sequence. A timer configured in Periodic mode does not stop counting after it times out.

### 10.4.2 32-Bit Real-Time Clock (RTC) Mode

To use the RTC mode, the timer must have a 32.768-KHz input signal on an even CCP input. To enable the RTC feature, follow these steps:

1. Ensure the timer is disabled (the TAEN bit is cleared) before making any changes.
2. Write the **GPTM Configuration Register (GPTMCFG)** with a value of 0x1.
3. Write the desired match value to the **GPTM TimerA Match Register (GPTMTAMATCHR)**.
4. Set/clear the RTCEN bit in the **GPTM Control Register (GPTMCTL)** as desired.
5. If interrupts are required, set the RTCIM bit in the **GPTM Interrupt Mask Register (GPTMIMR)**.
6. Set the TAEN bit in the **GPTMCTL** register to enable the timer and start counting.

When the timer count equals the value in the **GPTMTAMATCHR** register, the GPTM asserts the RTCRIS bit in the **GPTMRIS** register and continues counting until Timer A is disabled or a hardware reset. The interrupt is cleared by writing the RTCCINT bit in the **GPTMICR** register.

### 10.4.3 16-Bit One-Shot/Periodic Timer Mode

A timer is configured for 16-bit One-Shot and Periodic modes by the following sequence:

1. Ensure the timer is disabled (the TnEN bit is cleared) before making any changes.
2. Write the **GPTM Configuration Register (GPTMCFG)** with a value of 0x4.
3. Set the TnMR field in the **GPTM Timer Mode (GPTMTnMR)** register:
   a. Write a value of 0x1 for One-Shot mode.
   b. Write a value of 0x2 for Periodic mode.
4. If a prescaler is to be used, write the prescale value to the **GPTM Timern Prescale Register (GPTMTnPR)**.

5. Load the start value into the **GPTM Timer Interval Load Register (GPTMTnILR)**.
6. If interrupts are required, set the TnTOIM bit in the **GPTM Interrupt Mask Register (GPTMIMR)**.
7. Set the TnEN bit in the **GPTM Control Register (GPTMCTL)** to enable the timer and start counting.
8. Poll the TnTORIS bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the TnTOCINT bit of the **GPTM Interrupt Clear Register (GPTMICR)**.

In One-Shot mode, the timer stops counting after step 8 on page 408. To re-enable the timer, repeat the sequence. A timer configured in Periodic mode does not stop counting after it times out.

### 10.4.4 16-Bit Input Edge Count Mode

A timer is configured to Input Edge Count mode by the following sequence:

1. Ensure the timer is disabled (the TnEN bit is cleared) before making any changes.
2. Write the **GPTM Configuration (GPTMCFG)** register with a value of 0x4.
3. In the **GPTM Timer Mode (GPTMTnMR)** register, write the TnCMR field to 0x0 and the TnMR field to 0x3.
4. Configure the type of event(s) that the timer captures by writing the TnEVENT field of the **GPTM Control (GPTMCTL)** register.
5. Load the timer start value into the **GPTM Timern Interval Load (GPTMTnILR)** register.
6. Load the desired event count into the **GPTM Timern Match (GPTMTnMATCHR)** register.
7. If interrupts are required, set the CnMIM bit in the **GPTM Interrupt Mask (GPTMIMR)** register.
8. Set the TnEN bit in the **GPTMCTL** register to enable the timer and begin waiting for edge events.
9. Poll the CnMRIS bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the CnMCINT bit of the **GPTM Interrupt Clear (GPTMICR)** register.

In Input Edge Count Mode, the timer stops after the desired number of edge events has been detected. To re-enable the timer, ensure that the TnEN bit is cleared and repeat step 4 on page 408 through step 9 on page 408.

### 10.4.5 16-Bit Input Edge Timing Mode

A timer is configured to Input Edge Timing mode by the following sequence:

1. Ensure the timer is disabled (the TnEN bit is cleared) before making any changes.
2. Write the **GPTM Configuration (GPTMCFG)** register with a value of 0x4.
3. In the **GPTM Timer Mode (GPTMTnMR)** register, write the TnCMR field to 0x1 and the TnMR field to 0x3.

4. Configure the type of event that the timer captures by writing the TnEVENT field of the **GPTM Control (GPTMCTL)** register.
5. Load the timer start value into the **GPTM Timern Interval Load (GPTMTnILR)** register.
6. If interrupts are required, set the CnEIM bit in the **GPTM Interrupt Mask (GPTMIMR)** register.
7. Set the TnEN bit in the **GPTM Control (GPTMCTL)** register to enable the timer and start counting.
8. Poll the CnERIS bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the CnECINT bit of the **GPTM Interrupt Clear (GPTMICR)** register. The time at which the event happened can be obtained by reading the **GPTM Timern (GPTMTnR)** register.

In Input Edge Timing mode, the timer continues running after an edge event has been detected, but the timer interval can be changed at any time by writing the **GPTMTnILR** register. The change takes effect at the next cycle after the write.

### 10.4.6 16-Bit PWM Mode

A timer is configured to PWM mode using the following sequence:

1. Ensure the timer is disabled (the TnEN bit is cleared) before making any changes.
2. Write the **GPTM Configuration (GPTMCFG)** register with a value of 0x4.
3. In the **GPTM Timer Mode (GPTMTnMR)** register, set the TnAMS bit to 0x1, the TnCMR bit to 0x0, and the TnMR field to 0x2.
4. Configure the output state of the PWM signal (whether or not it is inverted) in the TnPWML field of the **GPTM Control (GPTMCTL)** register.
5. Load the timer start value into the **GPTM Timern Interval Load (GPTMTnILR)** register.
6. Load the **GPTM Timern Match (GPTMTnMATCHR)** register with the desired value.
7. Set the TnEN bit in the **GPTM Control (GPTMCTL)** register to enable the timer and begin generation of the output PWM signal.

In PWM Timing mode, the timer continues running after the PWM signal has been generated. The PWM period can be adjusted at any time by writing the **GPTMTnILR** register, and the change takes effect at the next cycle after the write.

## 10.5 Register Map

Table 10-4 on page 410 lists the GPTM registers. The offset listed is a hexadecimal increment to the register's address, relative to that timer's base address:

- Timer0: 0x4003.0000
- Timer1: 0x4003.1000
- Timer2: 0x4003.2000

Note that the Timer module clock must be enabled before the registers can be programmed (see page 223). There must be a delay of 3 system clocks after the Timer module clock is enabled before any Timer module registers are accessed.

**Table 10-4. Timers Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | GPTMCFG | R/W | 0x0000.0000 | GPTM Configuration | 411 |
| 0x004 | GPTMTAMR | R/W | 0x0000.0000 | GPTM TimerA Mode | 412 |
| 0x008 | GPTMTBMR | R/W | 0x0000.0000 | GPTM TimerB Mode | 414 |
| 0x00C | GPTMCTL | R/W | 0x0000.0000 | GPTM Control | 416 |
| 0x018 | GPTMIMR | R/W | 0x0000.0000 | GPTM Interrupt Mask | 419 |
| 0x01C | GPTMRIS | RO | 0x0000.0000 | GPTM Raw Interrupt Status | 421 |
| 0x020 | GPTMMIS | RO | 0x0000.0000 | GPTM Masked Interrupt Status | 422 |
| 0x024 | GPTMICR | W1C | 0x0000.0000 | GPTM Interrupt Clear | 423 |
| 0x028 | GPTMTAILR | R/W | 0xFFFF.FFFF | GPTM TimerA Interval Load | 425 |
| 0x02C | GPTMTBILR | R/W | 0x0000.FFFF | GPTM TimerB Interval Load | 426 |
| 0x030 | GPTMTAMATCHR | R/W | 0xFFFF.FFFF | GPTM TimerA Match | 427 |
| 0x034 | GPTMTBMATCHR | R/W | 0x0000.FFFF | GPTM TimerB Match | 428 |
| 0x038 | GPTMTAPR | R/W | 0x0000.0000 | GPTM TimerA Prescale | 429 |
| 0x03C | GPTMTBPR | R/W | 0x0000.0000 | GPTM TimerB Prescale | 430 |
| 0x048 | GPTMTAR | RO | 0xFFFF.FFFF | GPTM TimerA | 431 |
| 0x04C | GPTMTBR | RO | 0x0000.FFFF | GPTM TimerB | 432 |

## 10.6 Register Descriptions

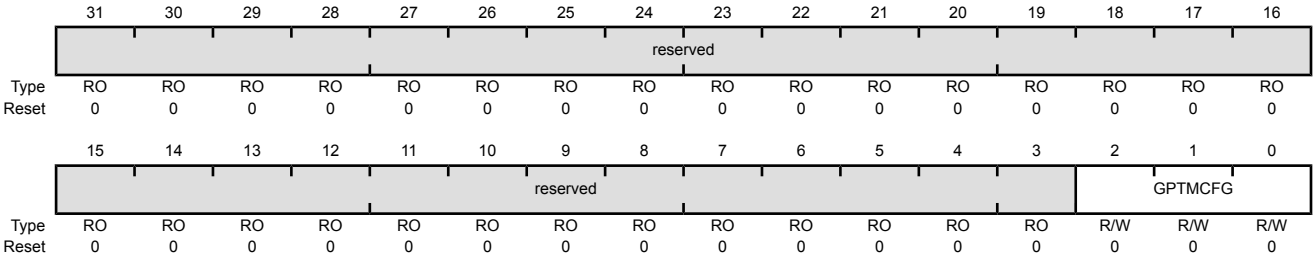
The remainder of this section lists and describes the GPTM registers, in numerical order by address offset.

## Register 1: GPTM Configuration (GPTMCFG), offset 0x000

This register configures the global operation of the GPTM module. The value written to this register determines whether the GPTM is in 32- or 16-bit mode.

GPTM Configuration (GPTMCFG)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x000
Type R/W, reset 0x0000.0000



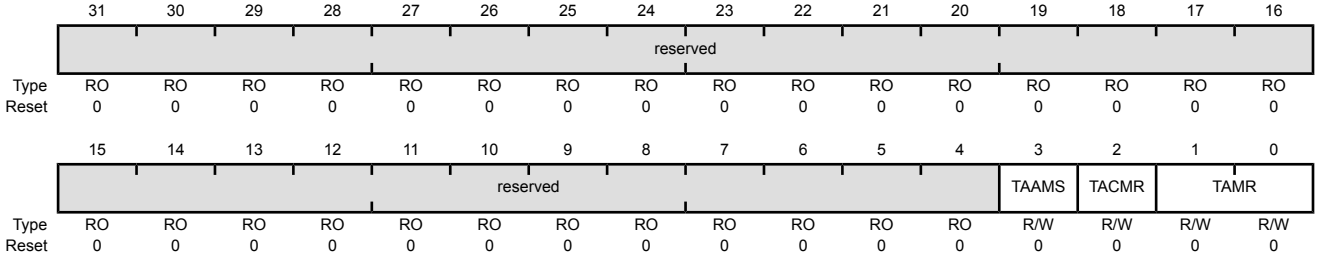

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | GPTMCFG | R/W | 0x0 | GPTM Configuration<br>The GPTMCFG values are defined as follows:<br>Value Description<br>0x0 32-bit timer configuration.<br>0x1 32-bit real-time clock (RTC) counter configuration.<br>0x2 Reserved<br>0x3 Reserved<br>0x4-0x7 16-bit timer configuration, function is controlled by bits 1:0 of **GPTMTAMR** and **GPTMTBMR**. |

## Register 2: GPTM TimerA Mode (GPTMTAMR), offset 0x004

This register configures the GPTM based on the configuration selected in the **GPTMCFG** register. When in 16-bit PWM mode, set the TAAMS bit to 0x1, the TACMR bit to 0x0, and the TAMR field to 0x2.

GPTM TimerA Mode (GPTMTAMR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x004
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TAAMS | R/W | 0 | GPTM TimerA Alternate Mode Select<br>The TAAMS values are defined as follows:<br><br>Value Description<br>0 Capture mode is enabled.<br>1 PWM mode is enabled.<br><br>**Note:** To enable PWM mode, you must also clear the TACMR bit and set the TAMR field to 0x2. |
| 2 | TACMR | R/W | 0 | GPTM TimerA Capture Mode<br>The TACMR values are defined as follows:<br><br>Value Description<br>0 Edge-Count mode<br>1 Edge-Time mode |

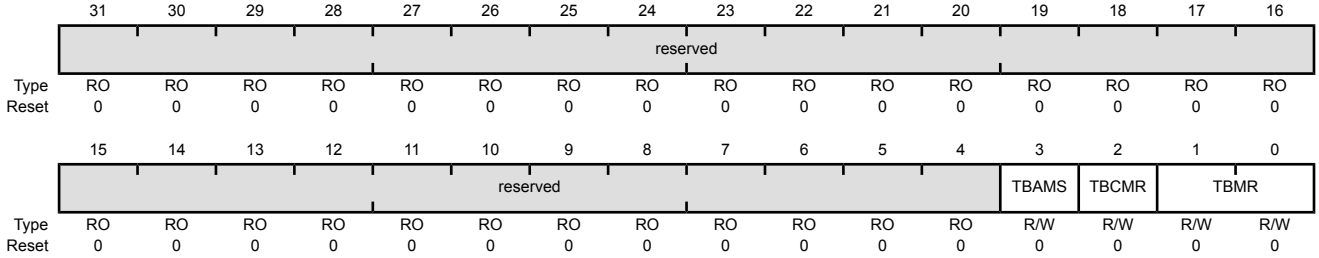
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1:0 | TAMR | R/W | 0x0 | GPTM TimerA Mode<br>The TAMR values are defined as follows:<br><br>Value Description<br>0x0 Reserved<br>0x1 One-Shot Timer mode<br>0x2 Periodic Timer mode<br>0x3 Capture mode<br><br>The Timer mode is based on the timer configuration defined by bits 2:0 in the **GPTMCFG** register (16-or 32-bit).<br>In 16-bit timer configuration, TAMR controls the 16-bit timer modes for TimerA.<br>In 32-bit timer configuration, this register controls the mode and the contents of **GPTMTBMR** are ignored. |

## Register 3: GPTM TimerB Mode (GPTMTBMR), offset 0x008

This register configures the GPTM based on the configuration selected in the **GPTMCFG** register. When in 16-bit PWM mode, set the TBAMS bit to 0x1, the TBCMR bit to 0x0, and the TBMR field to 0x2.

GPTM TimerB Mode (GPTMTBMR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x008
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TBAMS | R/W | 0 | GPTM TimerB Alternate Mode Select<br>The TBAMS values are defined as follows:<br><br>Value Description<br>0 Capture mode is enabled.<br>1 PWM mode is enabled.<br><br>**Note:** To enable PWM mode, you must also clear the TBCMR bit and set the TBMR field to 0x2. |
| 2 | TBCMR | R/W | 0 | GPTM TimerB Capture Mode<br>The TBCMR values are defined as follows:<br><br>Value Description<br>0 Edge-Count mode<br>1 Edge-Time mode |

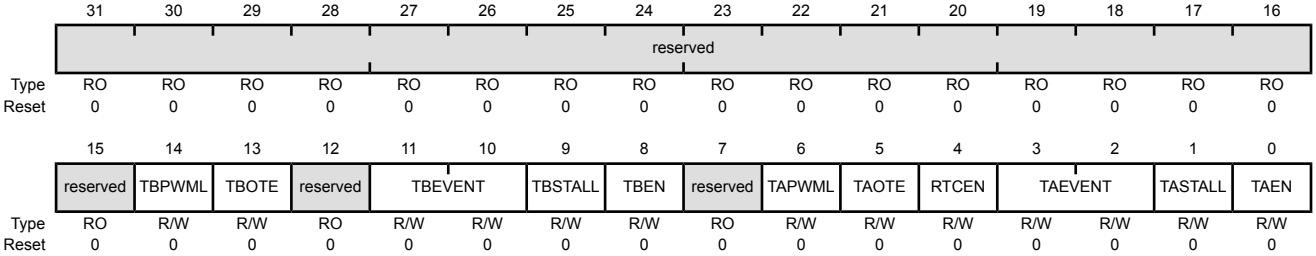
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1:0 | TBMR | R/W | 0x0 | GPTM TimerB Mode<br>The TBMR values are defined as follows:<br><br>Value Description<br>0x0 Reserved<br>0x1 One-Shot Timer mode<br>0x2 Periodic Timer mode<br>0x3 Capture mode<br><br>The timer mode is based on the timer configuration defined by bits 2:0 in the **GPTMCFG** register.<br>In 16-bit timer configuration, these bits control the 16-bit timer modes for TimerB.<br>In 32-bit timer configuration, this register's contents are ignored and **GPTMTAMR** is used. |

## Register 4: GPTM Control (GPTMCTL), offset 0x00C

This register is used alongside the **GPTMCFG** and **GMTMTnMR** registers to fine-tune the timer configuration, and to enable other features such as timer stall and the output trigger. The output trigger can be used to initiate transfers on the ADC module.

GPTM Control (GPTMCTL)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x00C
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:15 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14 | TBPWML | R/W | 0 | GPTM TimerB PWM Output Level<br>The TBPWML values are defined as follows:<br><br>Value Description<br>0 Output is unaffected.<br>1 Output is inverted. |
| 13 | TBOTE | R/W | 0 | GPTM TimerB Output Trigger Enable<br>The TBOTE values are defined as follows:<br><br>Value Description<br>0 The output TimerB ADC trigger is disabled.<br>1 The output TimerB ADC trigger is enabled.<br><br>In addition, the ADC must be enabled and the timer selected as a trigger source with the EMn bit in the **ADCEMUX** register (see page 472). |
| 12 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 11:10 | TBEVENT | R/W | 0x0 | GPTM TimerB Event Mode<br>The TBEVENT values are defined as follows:<br><br>Value Description<br>0x0 Positive edge<br>0x1 Negative edge<br>0x2 Reserved<br>0x3 Both edges |
| 9 | TBSTALL | R/W | 0 | GPTM Timer B Stall Enable<br>The TBSTALL values are defined as follows:<br><br>Value Description<br>0 Timer B continues counting while the processor is halted by the debugger.<br>1 Timer B freezes counting while the processor is halted by the debugger.<br><br>If the processor is executing normally, the TBSTALL bit is ignored. |
| 8 | TBEN | R/W | 0 | GPTM TimerB Enable<br>The TBEN values are defined as follows:<br><br>Value Description<br>0 TimerB is disabled.<br>1 TimerB is enabled and begins counting or the capture logic is enabled based on the **GPTMCFG** register. |
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | TAPWML | R/W | 0 | GPTM TimerA PWM Output Level<br>The TAPWML values are defined as follows:<br><br>Value Description<br>0 Output is unaffected.<br>1 Output is inverted. |
| 5 | TAOTE | R/W | 0 | GPTM TimerA Output Trigger Enable<br>The TAOTE values are defined as follows:<br><br>Value Description<br>0 The output TimerA ADC trigger is disabled.<br>1 The output TimerA ADC trigger is enabled.<br><br>In addition, the ADC must be enabled and the timer selected as a trigger source with the EMn bit in the **ADCEMUX** register (see page 472). |

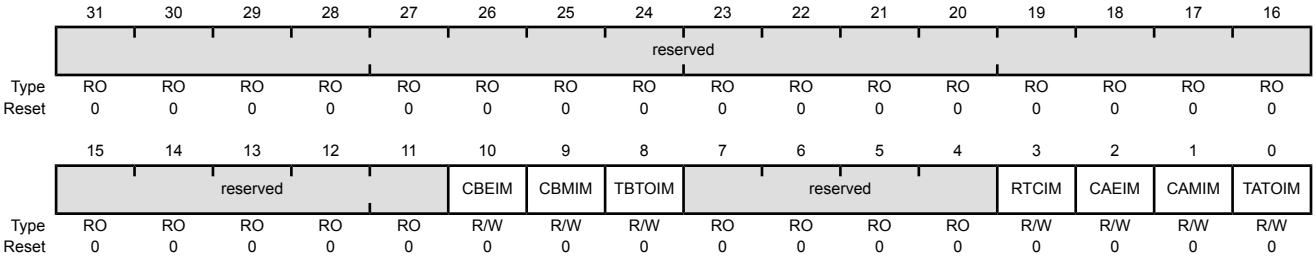
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | RTCEN | R/W | 0 | GPTM RTC Enable<br>The RTCEN values are defined as follows:<br><br>Value Description<br>0 RTC counting is disabled.<br>1 RTC counting is enabled. |
| 3:2 | TAEVENT | R/W | 0x0 | GPTM TimerA Event Mode<br>The TAEVENT values are defined as follows:<br><br>Value Description<br>0x0 Positive edge<br>0x1 Negative edge<br>0x2 Reserved<br>0x3 Both edges |
| 1 | TASTALL | R/W | 0 | GPTM Timer A Stall Enable<br>The TASTALL values are defined as follows:<br><br>Value Description<br>0 Timer A continues counting while the processor is halted by the debugger.<br>1 Timer A freezes counting while the processor is halted by the debugger.<br><br>If the processor is executing normally, the TASTALL bit is ignored. |
| 0 | TAEN | R/W | 0 | GPTM TimerA Enable<br>The TAEN values are defined as follows:<br><br>Value Description<br>0 TimerA is disabled.<br>1 TimerA is enabled and begins counting or the capture logic is enabled based on the **GPTMCFG** register. |

## Register 5: GPTM Interrupt Mask (GPTMIMR), offset 0x018

This register allows software to enable/disable GPTM controller-level interrupts. Writing a 1 enables the interrupt, while writing a 0 disables it.

GPTM Interrupt Mask (GPTMIMR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x018
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | CBEIM | R/W | 0 | GPTM CaptureB Event Interrupt Mask<br>The CBEIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 9 | CBMIM | R/W | 0 | GPTM CaptureB Match Interrupt Mask<br>The CBMIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 8 | TBTOIM | R/W | 0 | GPTM TimerB Time-Out Interrupt Mask<br>The TBTOIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 7:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

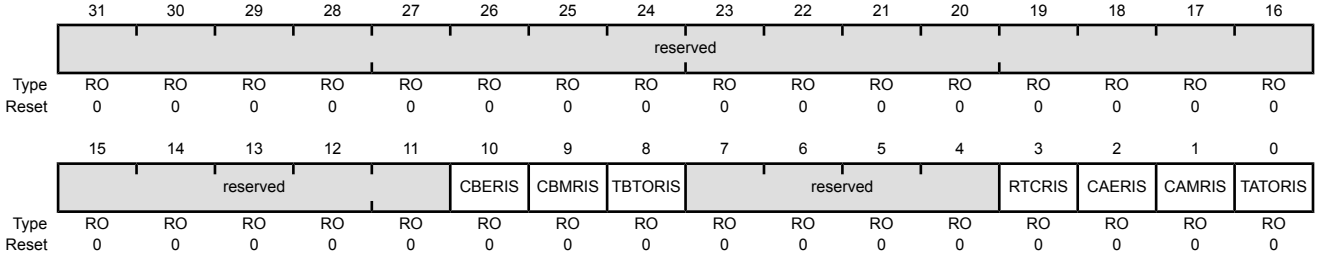
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | RTCIM | R/W | 0 | GPTM RTC Interrupt Mask<br>The RTCIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 2 | CAEIM | R/W | 0 | GPTM CaptureA Event Interrupt Mask<br>The CAEIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 1 | CAMIM | R/W | 0 | GPTM CaptureA Match Interrupt Mask<br>The CAMIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 0 | TATOIM | R/W | 0 | GPTM TimerA Time-Out Interrupt Mask<br>The TATOIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |

## Register 6: GPTM Raw Interrupt Status (GPTMRIS), offset 0x01C

This register shows the state of the GPTM's internal interrupt signal. These bits are set whether or not the interrupt is masked in the **GPTMIMR** register. Each bit can be cleared by writing a 1 to its corresponding bit in **GPTMICR**.

GPTM Raw Interrupt Status (GPTMRIS)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x01C
Type RO, reset 0x0000.0000



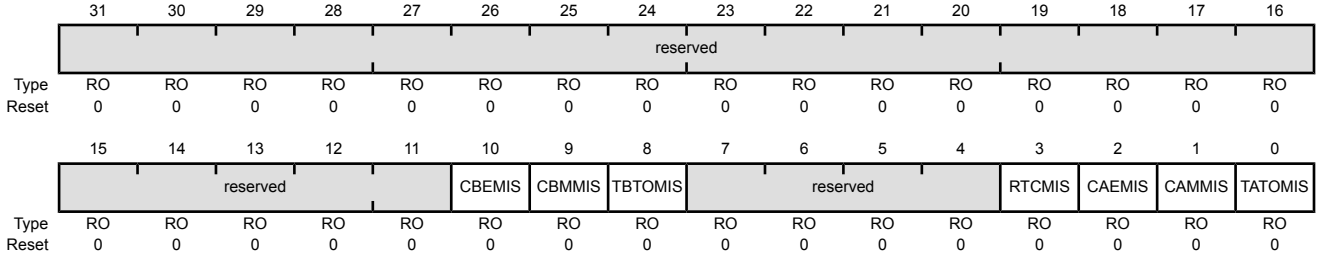

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | CBERIS | RO | 0 | GPTM CaptureB Event Raw Interrupt<br>This is the CaptureB Event interrupt status prior to masking. |
| 9 | CBMRIS | RO | 0 | GPTM CaptureB Match Raw Interrupt<br>This is the CaptureB Match interrupt status prior to masking. |
| 8 | TBTORIS | RO | 0 | GPTM TimerB Time-Out Raw Interrupt<br>This is the TimerB time-out interrupt status prior to masking. |
| 7:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | RTCRIS | RO | 0 | GPTM RTC Raw Interrupt<br>This is the RTC Event interrupt status prior to masking. |
| 2 | CAERIS | RO | 0 | GPTM CaptureA Event Raw Interrupt<br>This is the CaptureA Event interrupt status prior to masking. |
| 1 | CAMRIS | RO | 0 | GPTM CaptureA Match Raw Interrupt<br>This is the CaptureA Match interrupt status prior to masking. |
| 0 | TATORIS | RO | 0 | GPTM TimerA Time-Out Raw Interrupt<br>This the TimerA time-out interrupt status prior to masking. |

## Register 7: GPTM Masked Interrupt Status (GPTMMIS), offset 0x020

This register show the state of the GPTM's controller-level interrupt. If an interrupt is unmasked in **GPTMIMR**, and there is an event that causes the interrupt to be asserted, the corresponding bit is set in this register. All bits are cleared by writing a 1 to the corresponding bit in **GPTMICR**.

GPTM Masked Interrupt Status (GPTMMIS)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x020
Type RO, reset 0x0000.0000



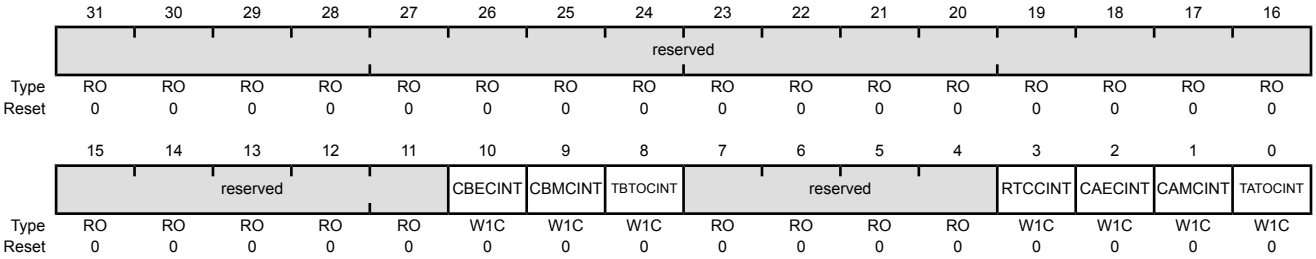

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | CBEMIS | RO | 0 | GPTM CaptureB Event Masked Interrupt<br>This is the CaptureB event interrupt status after masking. |
| 9 | CBMMIS | RO | 0 | GPTM CaptureB Match Masked Interrupt<br>This is the CaptureB match interrupt status after masking. |
| 8 | TBTOMIS | RO | 0 | GPTM TimerB Time-Out Masked Interrupt<br>This is the TimerB time-out interrupt status after masking. |
| 7:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | RTCMIS | RO | 0 | GPTM RTC Masked Interrupt<br>This is the RTC event interrupt status after masking. |
| 2 | CAEMIS | RO | 0 | GPTM CaptureA Event Masked Interrupt<br>This is the CaptureA event interrupt status after masking. |
| 1 | CAMMIS | RO | 0 | GPTM CaptureA Match Masked Interrupt<br>This is the CaptureA match interrupt status after masking. |
| 0 | TATOMIS | RO | 0 | GPTM TimerA Time-Out Masked Interrupt<br>This is the TimerA time-out interrupt status after masking. |

## Register 8: GPTM Interrupt Clear (GPTMICR), offset 0x024

This register is used to clear the status bits in the **GPTMRIS** and **GPTMMIS** registers. Writing a 1 to a bit clears the corresponding bit in the **GPTMRIS** and **GPTMMIS** registers.

GPTM Interrupt Clear (GPTMICR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x024
Type W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | CBECINT | W1C | 0 | GPTM CaptureB Event Interrupt Clear<br>The `CBECINT` values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 9 | CBMCINT | W1C | 0 | GPTM CaptureB Match Interrupt Clear<br>The `CBMCINT` values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 8 | TBTOCINT | W1C | 0 | GPTM TimerB Time-Out Interrupt Clear<br>The `TBTOCINT` values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 7:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

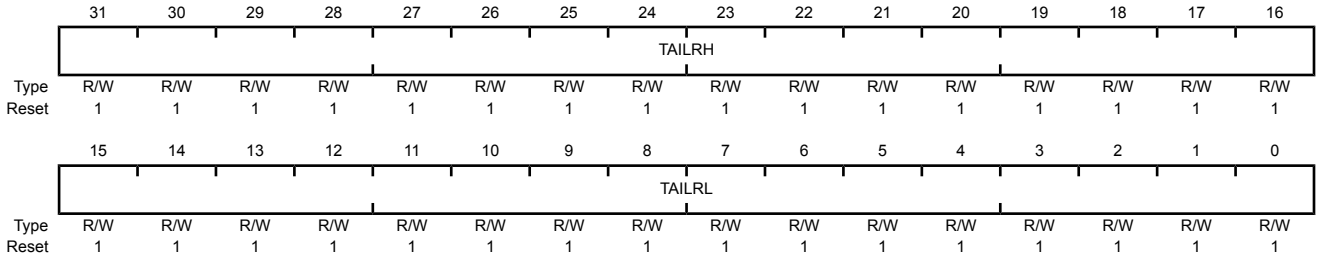
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | RTCCINT | W1C | 0 | GPTM RTC Interrupt Clear<br>The RTCCINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 2 | CAECINT | W1C | 0 | GPTM CaptureA Event Interrupt Clear<br>The CAECINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 1 | CAMCINT | W1C | 0 | GPTM CaptureA Match Interrupt Clear<br>The CAMCINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 0 | TATOCINT | W1C | 0 | GPTM TimerA Time-Out Interrupt Clear<br>The TATOCINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |

## Register 9: GPTM TimerA Interval Load (GPTMTAILR), offset 0x028

This register is used to load the starting count value into the timer. When GPTM is configured to one of the 32-bit modes, **GPTMTAILR** appears as a 32-bit register (the upper 16-bits correspond to the contents of the **GPTM TimerB Interval Load (GPTMTBILR)** register). In 16-bit mode, the upper 16 bits of this register read as 0s and have no effect on the state of **GPTMTBILR**.

GPTM TimerA Interval Load (GPTMTAILR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x028
Type R/W, reset 0xFFFF.FFFF



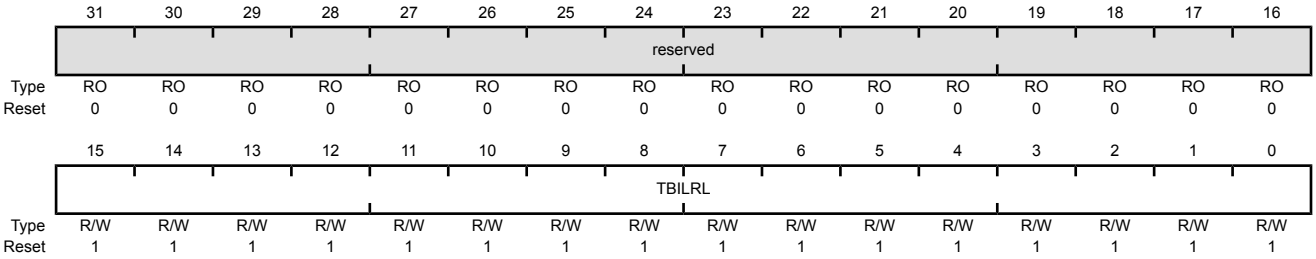

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | TAILRH | R/W | 0xFFFF | GPTM TimerA Interval Load Register High<br>When configured for 32-bit mode via the **GPTMCFG** register, the **GPTM TimerB Interval Load (GPTMTBILR)** register loads this value on a write. A read returns the current value of **GPTMTBILR**.<br>In 16-bit mode, this field reads as 0 and does not have an effect on the state of **GPTMTBILR**. |
| 15:0 | TAILRL | R/W | 0xFFFF | GPTM TimerA Interval Load Register Low<br>For both 16- and 32-bit modes, writing this field loads the counter for TimerA. A read returns the current value of **GPTMTAILR**. |

## Register 10: GPTM TimerB Interval Load (GPTMTBILR), offset 0x02C

This register is used to load the starting count value into TimerB. When the GPTM is configured to a 32-bit mode, **GPTMTBILR** returns the current value of TimerB and ignores writes.

GPTM TimerB Interval Load (GPTMTBILR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x02C
Type R/W, reset 0x0000.FFFF



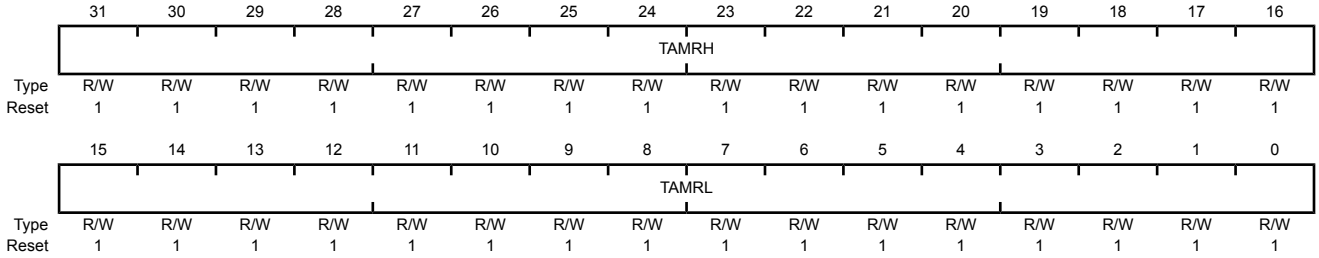

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | TBILRL | R/W | 0xFFFF | GPTM TimerB Interval Load Register<br>When the GPTM is not configured as a 32-bit timer, a write to this field updates **GPTMTBILR**. In 32-bit mode, writes are ignored, and reads return the current value of **GPTMTBILR**. |

## Register 11: GPTM TimerA Match (GPTMTAMATCHR), offset 0x030

This register is used in 32-bit Real-Time Clock mode and 16-bit PWM and Input Edge Count modes.

GPTM TimerA Match (GPTMTAMATCHR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x030
Type R/W, reset 0xFFFF.FFFF



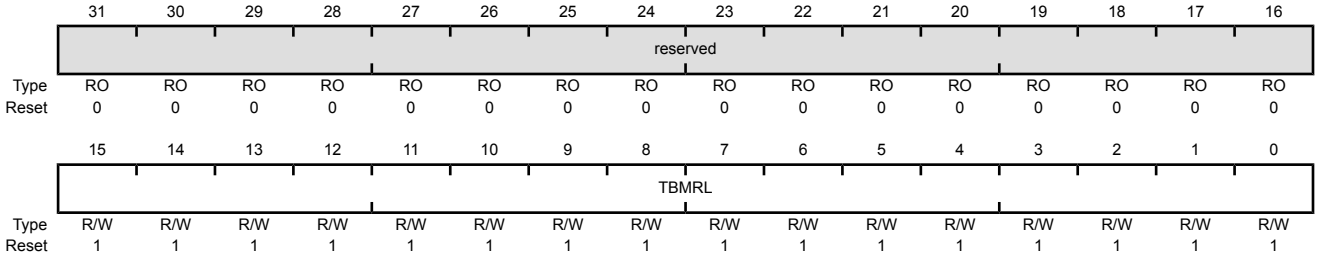

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | TAMRH | R/W | 0xFFFF | GPTM TimerA Match Register High<br>When configured for 32-bit Real-Time Clock (RTC) mode via the **GPTMCFG** register, this value is compared to the upper half of **GPTMTAR**, to determine match events.<br>In 16-bit mode, this field reads as 0 and does not have an effect on the state of **GPTMTBMATCHR**. |
| 15:0 | TAMRL | R/W | 0xFFFF | GPTM TimerA Match Register Low<br>When configured for 32-bit Real-Time Clock (RTC) mode via the **GPTMCFG** register, this value is compared to the lower half of **GPTMTAR**, to determine match events.<br>When configured for PWM mode, this value along with **GPTMTAILR**, determines the duty cycle of the output PWM signal.<br>When configured for Edge Count mode, this value along with **GPTMTAILR**, determines how many edge events are counted. The total number of edge events counted is equal to the value in **GPTMTAILR** minus this value. |

## Register 12: GPTM TimerB Match (GPTMTBMATCHR), offset 0x034

This register is used in 16-bit PWM and Input Edge Count modes.

GPTM TimerB Match (GPTMTBMATCHR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x034
Type R/W, reset 0x0000.FFFF



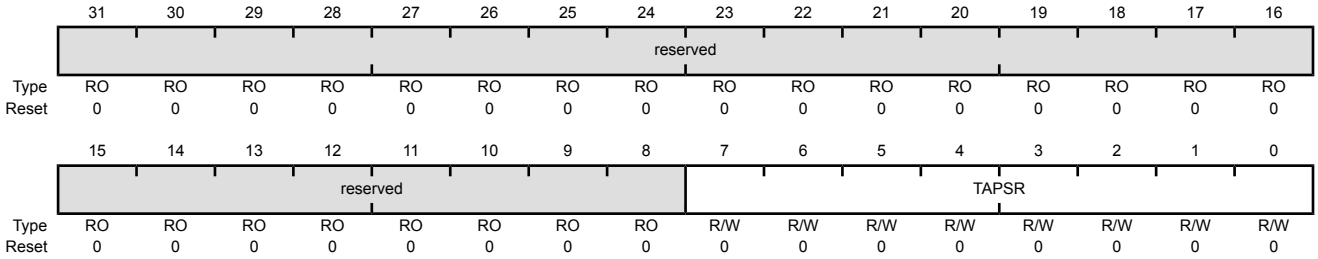

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | TBMRL | R/W | 0xFFFF | GPTM TimerB Match Register Low<br>When configured for PWM mode, this value along with **GPTMTBILR**, determines the duty cycle of the output PWM signal.<br>When configured for Edge Count mode, this value along with **GPTMTBILR**, determines how many edge events are counted. The total number of edge events counted is equal to the value in **GPTMTBILR** minus this value. |

## Register 13: GPTM TimerA Prescale (GPTMTAPR), offset 0x038

This register allows software to extend the range of the 16-bit timers when operating in one-shot or periodic mode.

GPTM TimerA Prescale (GPTMTAPR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x038
Type R/W, reset 0x0000.0000



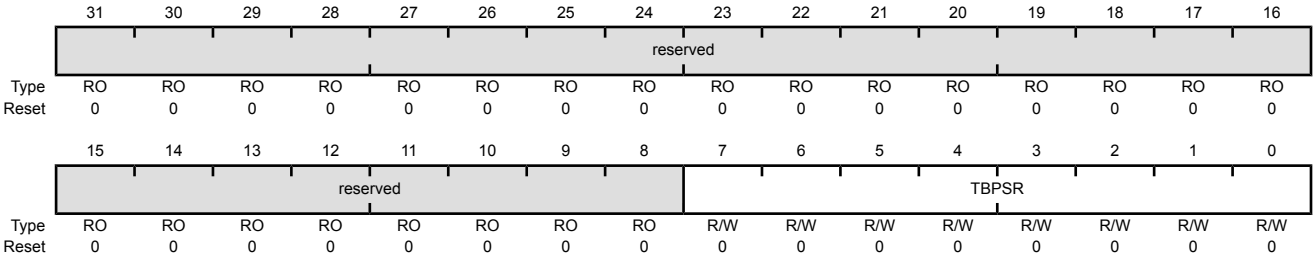

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | TAPSR | R/W | 0x00 | GPTM TimerA Prescale<br>The register loads this value on a write. A read returns the current value of the register.<br>Refer to Table 10-3 on page 403 for more details and an example. |

## Register 14: GPTM TimerB Prescale (GPTMTBPR), offset 0x03C

This register allows software to extend the range of the 16-bit timers when operating in one-shot or periodic mode.

GPTM TimerB Prescale (GPTMTBPR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x03C
Type R/W, reset 0x0000.0000



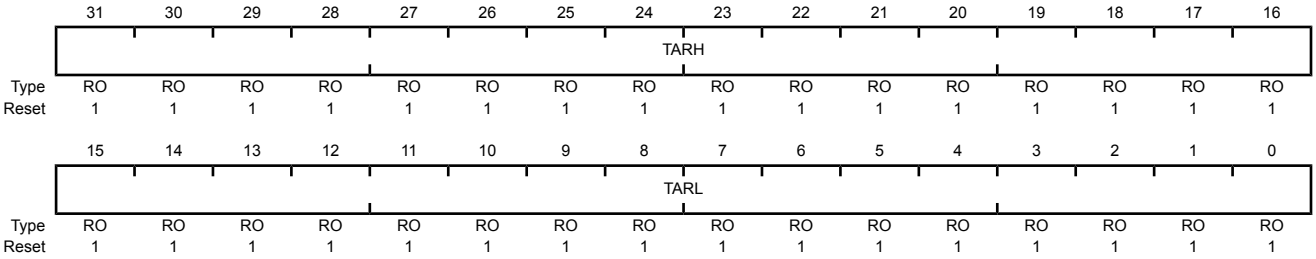

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | TBPSR | R/W | 0x00 | GPTM TimerB Prescale<br>The register loads this value on a write. A read returns the current value of this register.<br>Refer to Table 10-3 on page 403 for more details and an example. |

## Register 15: GPTM TimerA (GPTMTAR), offset 0x048

This register shows the current value of the TimerA counter in all cases except for Input Edge Count mode. When in this mode, this register contains the number of edges that have occurred.

GPTM TimerA (GPTMTAR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x048
Type RO, reset 0xFFFF.FFFF



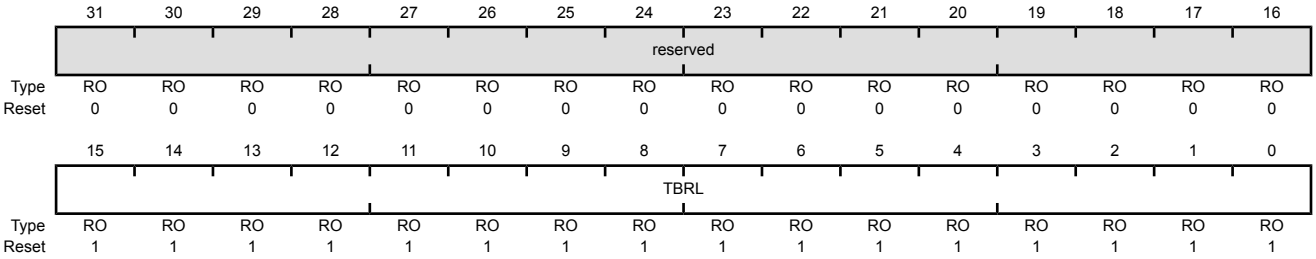

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | TARH | RO | 0xFFFF | GPTM TimerA Register High<br>If the **GPTMCFG** is in a 32-bit mode, TimerB value is read. If the **GPTMCFG** is in a 16-bit mode, this is read as zero. |
| 15:0 | TARL | RO | 0xFFFF | GPTM TimerA Register Low<br>A read returns the current value of the **GPTM TimerA Count Register**, except in Input Edge-Count mode, when it returns the number of edges that have occurred. |

## Register 16: GPTM TimerB (GPTMTBR), offset 0x04C

This register shows the current value of the TimerB counter in all cases except for Input Edge Count mode. When in this mode, this register contains the number of edges that have occurred.

GPTM TimerB (GPTMTBR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x04C
Type RO, reset 0x0000.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | TBRL | RO | 0xFFFF | GPTM TimerB<br>A read returns the current value of the **GPTM TimerB Count Register**, except in Input Edge-Count mode, when it returns the number of edges that have occurred. |

# 11 Watchdog Timer

A watchdog timer can generate nonmaskable interrupts (NMIs) or a reset when a time-out value is reached. The watchdog timer is used to regain control when a system has failed due to a software error or due to the failure of an external device to respond in the expected way.

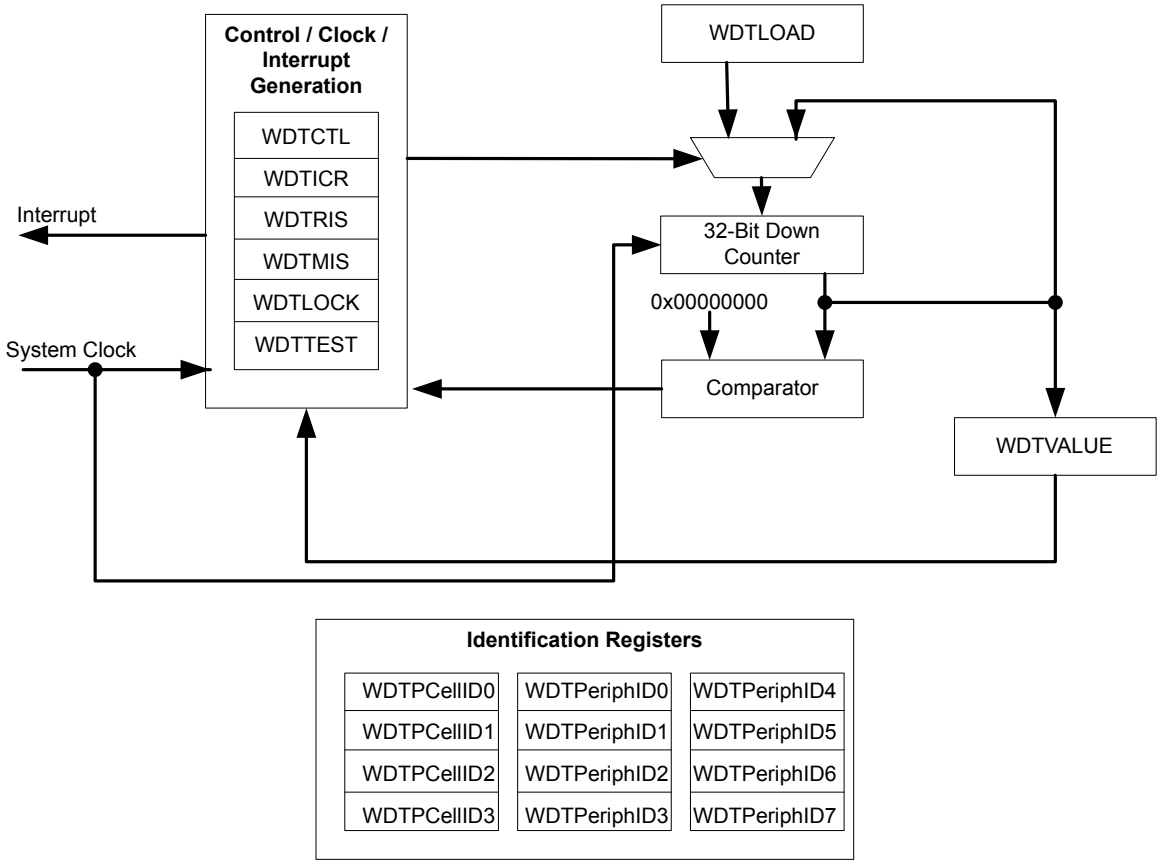
The Stellaris® Watchdog Timer module has the following features:

- 32-bit down counter with a programmable load register
- Separate watchdog clock with an enable
- Programmable interrupt generation logic with interrupt masking
- Lock register protection from runaway software
- Reset generation logic with an enable/disable
- User-enabled stalling when the controller asserts the CPU Halt flag during debug

The Watchdog Timer can be configured to generate an interrupt to the controller on its first time-out, and to generate a reset signal on its second time-out. Once the Watchdog Timer has been configured, the lock register can be written to prevent the timer configuration from being inadvertently altered.

## 11.1 Block Diagram

**Figure 11-1. WDT Module Block Diagram**




## 11.2 Functional Description

The Watchdog Timer module generates the first time-out signal when the 32-bit counter reaches the zero state after being enabled; enabling the counter also enables the watchdog timer interrupt. After the first time-out event, the 32-bit counter is re-loaded with the value of the **Watchdog Timer Load (WDTLOAD)** register, and the timer resumes counting down from that value. Once the Watchdog Timer has been configured, the **Watchdog Timer Lock (WDTLOCK)** register is written, which prevents the timer configuration from being inadvertently altered by software.

If the timer counts down to its zero state again before the first time-out interrupt is cleared, and the reset signal has been enabled (via the `WatchdogResetEnable` function), the Watchdog timer asserts its reset signal to the system. If the interrupt is cleared before the 32-bit counter reaches its second time-out, the 32-bit counter is loaded with the value in the **WDTLOAD** register, and counting resumes from that value.

If **WDTLOAD** is written with a new value while the Watchdog Timer counter is counting, then the counter is loaded with the new value and continues counting.

Writing to **WDTLOAD** does not clear an active interrupt. An interrupt must be specifically cleared by writing to the **Watchdog Interrupt Clear (WDTICR)** register.

The Watchdog module interrupt and reset generation can be enabled or disabled as required. When the interrupt is re-enabled, the 32-bit counter is preloaded with the load register value and not its last state.

## 11.3 Initialization and Configuration

To use the WDT, its peripheral clock must be enabled by setting the WDT bit in the **RCGC0** register. The Watchdog Timer is configured using the following sequence:

1. Load the **WDTLOAD** register with the desired timer load value.
2. If the Watchdog is configured to trigger system resets, set the RESEN bit in the **WDTCTL** register.
3. Set the INTEN bit in the **WDTCTL** register to enable the Watchdog and lock the control register.

If software requires that all of the watchdog registers are locked, the Watchdog Timer module can be fully locked by writing any value to the **WDTLOCK** register. To unlock the Watchdog Timer, write a value of 0x1ACC.E551.

## 11.4 Register Map

Table 11-1 on page 435 lists the Watchdog registers. The offset listed is a hexadecimal increment to the register's address, relative to the Watchdog Timer base address of 0x4000.0000.

**Table 11-1. Watchdog Timer Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | WDTLOAD | R/W | 0xFFFF.FFFF | Watchdog Load | 437 |
| 0x004 | WDTVALUE | RO | 0xFFFF.FFFF | Watchdog Value | 438 |
| 0x008 | WDTCTL | R/W | 0x0000.0000 | Watchdog Control | 439 |
| 0x00C | WDTICR | WO | - | Watchdog Interrupt Clear | 440 |
| 0x010 | WDTRIS | RO | 0x0000.0000 | Watchdog Raw Interrupt Status | 441 |
| 0x014 | WDTMIS | RO | 0x0000.0000 | Watchdog Masked Interrupt Status | 442 |
| 0x418 | WDTTEST | R/W | 0x0000.0000 | Watchdog Test | 443 |
| 0xC00 | WDTLOCK | R/W | 0x0000.0000 | Watchdog Lock | 444 |
| 0xFD0 | WDTPeriphID4 | RO | 0x0000.0000 | Watchdog Peripheral Identification 4 | 445 |
| 0xFD4 | WDTPeriphID5 | RO | 0x0000.0000 | Watchdog Peripheral Identification 5 | 446 |
| 0xFD8 | WDTPeriphID6 | RO | 0x0000.0000 | Watchdog Peripheral Identification 6 | 447 |
| 0xFDC | WDTPeriphID7 | RO | 0x0000.0000 | Watchdog Peripheral Identification 7 | 448 |
| 0xFE0 | WDTPeriphID0 | RO | 0x0000.0005 | Watchdog Peripheral Identification 0 | 449 |
| 0xFE4 | WDTPeriphID1 | RO | 0x0000.0018 | Watchdog Peripheral Identification 1 | 450 |
| 0xFE8 | WDTPeriphID2 | RO | 0x0000.0018 | Watchdog Peripheral Identification 2 | 451 |

**Table 11-1. Watchdog Timer Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0xFEC | WDTPeriphID3 | RO | 0x0000.0001 | Watchdog Peripheral Identification 3 | 452 |
| 0xFF0 | WDTPCellID0 | RO | 0x0000.000D | Watchdog PrimeCell Identification 0 | 453 |
| 0xFF4 | WDTPCellID1 | RO | 0x0000.00F0 | Watchdog PrimeCell Identification 1 | 454 |
| 0xFF8 | WDTPCellID2 | RO | 0x0000.0005 | Watchdog PrimeCell Identification 2 | 455 |
| 0xFFC | WDTPCellID3 | RO | 0x0000.00B1 | Watchdog PrimeCell Identification 3 | 456 |

## 11.5 Register Descriptions

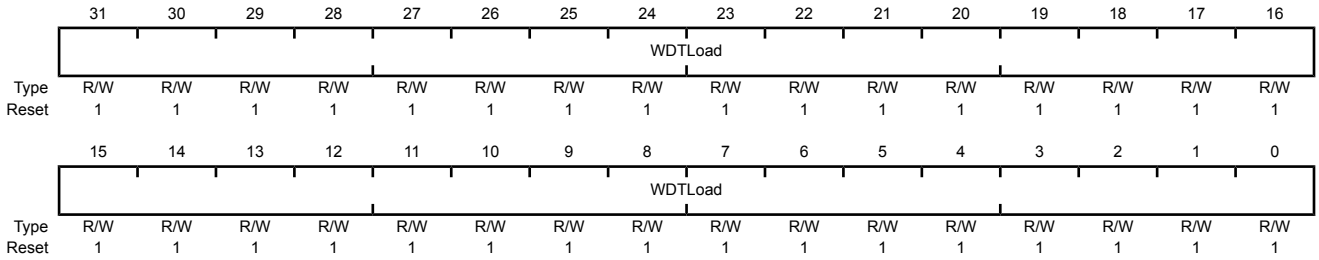
The remainder of this section lists and describes the WDT registers, in numerical order by address offset.

## Register 1: Watchdog Load (WDTLOAD), offset 0x000

This register is the 32-bit interval value used by the 32-bit counter. When this register is written, the value is immediately loaded and the counter restarts counting down from the new value. If the **WDTLOAD** register is loaded with 0x0000.0000, an interrupt is immediately generated.

Watchdog Load (WDTLOAD)

Base 0x4000.0000
Offset 0x000
Type R/W, reset 0xFFFF.FFFF



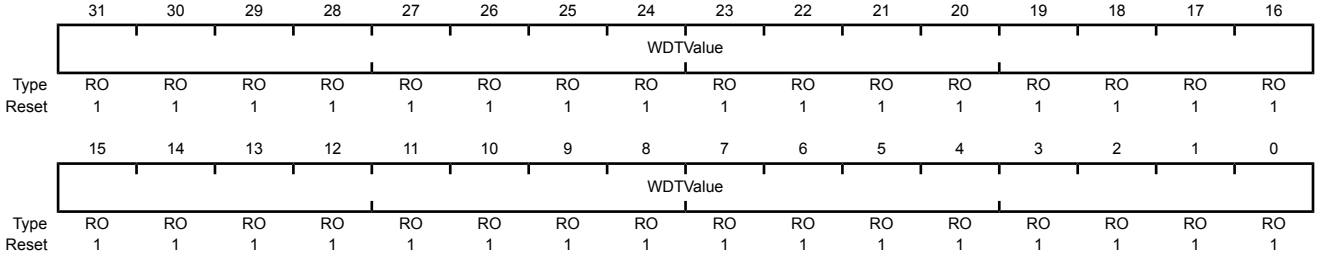

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTLoad | R/W | 0xFFFF.FFFF | Watchdog Load Value |

## Register 2: Watchdog Value (WDTVALUE), offset 0x004

This register contains the current count value of the timer.

Watchdog Value (WDTVALUE)

Base 0x4000.0000
Offset 0x004
Type RO, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTValue | RO | 0xFFFF.FFFF | Watchdog Value<br>Current value of the 32-bit down counter. |

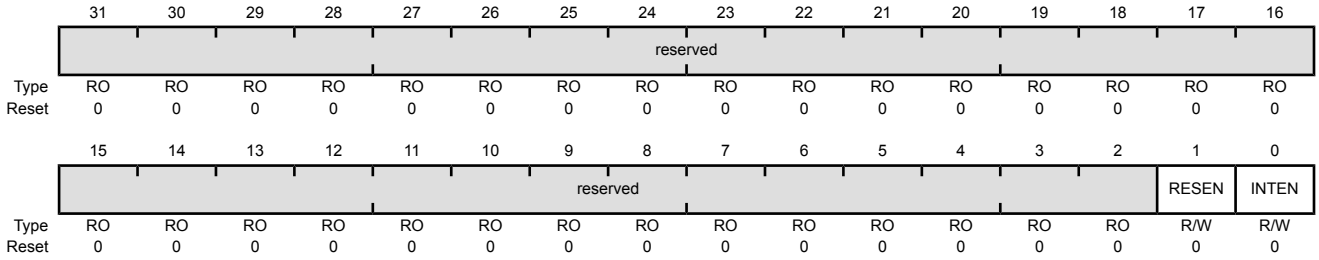
## Register 3: Watchdog Control (WDTCTL), offset 0x008

This register is the watchdog control register. The watchdog timer can be configured to generate a reset signal (on second time-out) or an interrupt on time-out.

When the watchdog interrupt has been enabled, all subsequent writes to the control register are ignored. The only mechanism that can re-enable writes is a hardware reset.

Watchdog Control (WDTCTL)

Base 0x4000.0000
Offset 0x008
Type R/W, reset 0x0000.0000



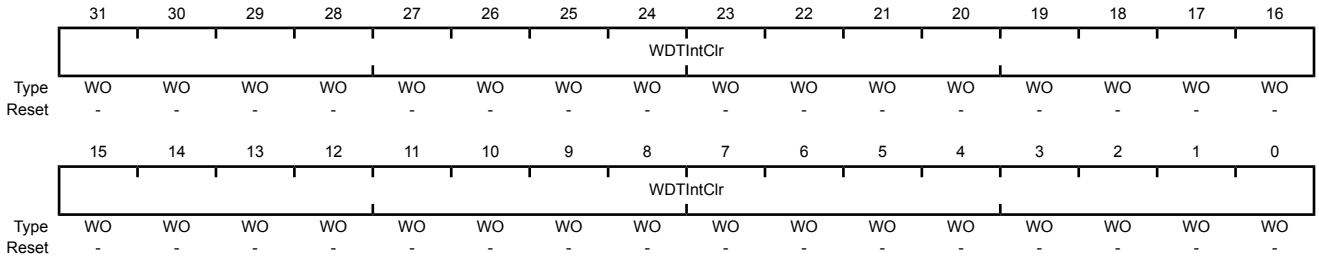

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | RESEN | R/W | 0 | Watchdog Reset Enable<br>The RESEN values are defined as follows:<br><br>Value Description<br>0 Disabled.<br>1 Enable the Watchdog module reset output. |
| 0 | INTEN | R/W | 0 | Watchdog Interrupt Enable<br>The INTEN values are defined as follows:<br><br>Value Description<br>0 Interrupt event disabled (once this bit is set, it can only be cleared by a hardware reset).<br>1 Interrupt event enabled. Once enabled, all writes are ignored. |

## Register 4: Watchdog Interrupt Clear (WDTICR), offset 0x00C

This register is the interrupt clear register. A write of any value to this register clears the Watchdog interrupt and reloads the 32-bit counter from the **WDTLOAD** register. Value for a read or reset is indeterminate.

Watchdog Interrupt Clear (WDTICR)

Base 0x4000.0000
Offset 0x00C
Type WO, reset -



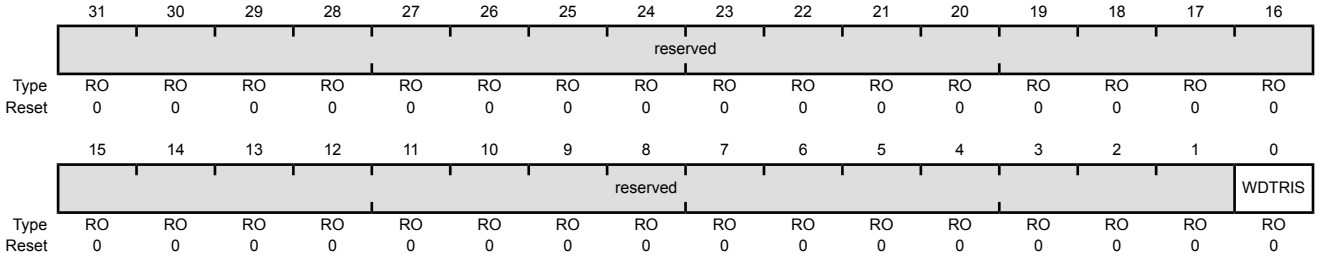

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTIntClr | WO | - | Watchdog Interrupt Clear |

## Register 5: Watchdog Raw Interrupt Status (WDTRIS), offset 0x010

This register is the raw interrupt status register. Watchdog interrupt events can be monitored via this register if the controller interrupt is masked.

Watchdog Raw Interrupt Status (WDTRIS)

Base 0x4000.0000
Offset 0x010
Type RO, reset 0x0000.0000



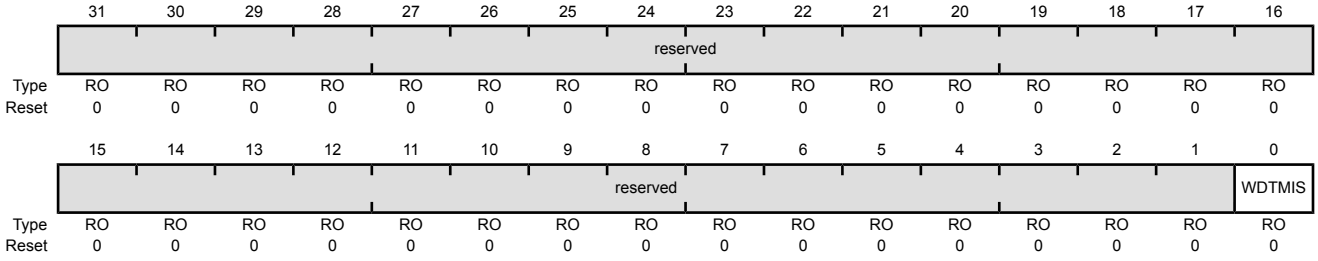

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | WDTRIS | RO | 0 | Watchdog Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of **WDTINTR**. |

## Register 6: Watchdog Masked Interrupt Status (WDTMIS), offset 0x014

This register is the masked interrupt status register. The value of this register is the logical AND of the raw interrupt bit and the Watchdog interrupt enable bit.

Watchdog Masked Interrupt Status (WDTMIS)

Base 0x4000.0000
Offset 0x014
Type RO, reset 0x0000.0000



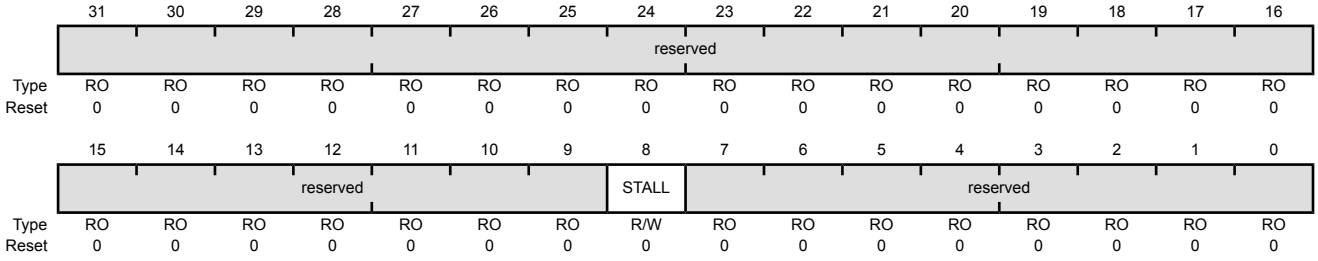

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | WDTMIS | RO | 0 | Watchdog Masked Interrupt Status<br>Gives the masked interrupt state (after masking) of the **WDTINTR** interrupt. |

## Register 7: Watchdog Test (WDTTEST), offset 0x418

This register provides user-enabled stalling when the microcontroller asserts the CPU halt flag during debug.

Watchdog Test (WDTTEST)

Base 0x4000.0000
Offset 0x418
Type R/W, reset 0x0000.0000



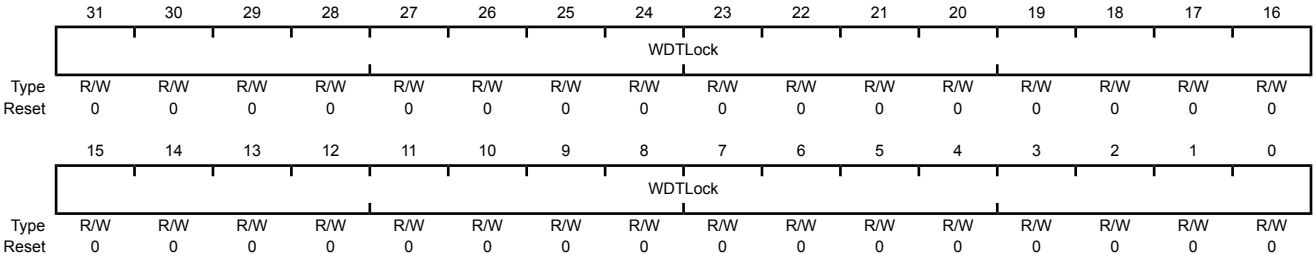

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:9 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 8 | STALL | R/W | 0 | Watchdog Stall Enable<br>When set to 1, if the Stellaris microcontroller is stopped with a debugger, the watchdog timer stops counting. Once the microcontroller is restarted, the watchdog timer resumes counting. |
| 7:0 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 8: Watchdog Lock (WDTLOCK), offset 0xC00

Writing 0x1ACC.E551 to the **WDTLOCK** register enables write access to all other registers. Writing any other value to the **WDTLOCK** register re-enables the locked state for register writes to all the other registers. Reading the **WDTLOCK** register returns the lock status rather than the 32-bit value written. Therefore, when write accesses are disabled, reading the **WDTLOCK** register returns 0x0000.0001 (when locked; otherwise, the returned value is 0x0000.0000 (unlocked)).

Watchdog Lock (WDTLOCK)

Base 0x4000.0000
Offset 0xC00
Type R/W, reset 0x0000.0000



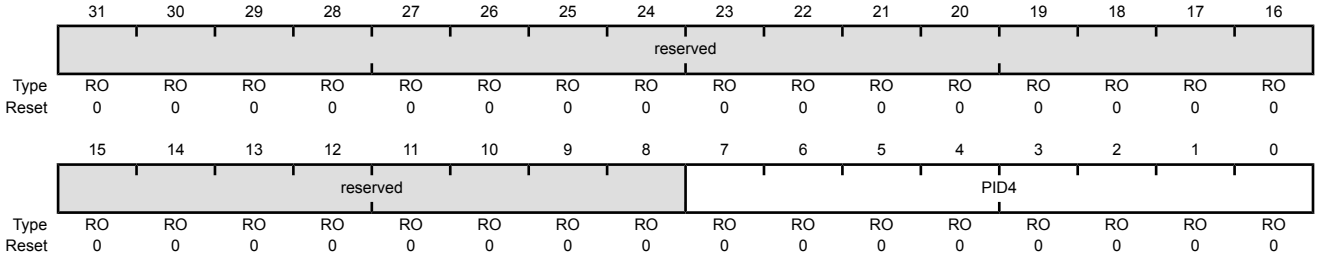

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTLock | R/W | 0x0000 | Watchdog Lock<br>A write of the value 0x1ACC.E551 unlocks the watchdog registers for write access. A write of any other value reapplies the lock, preventing any register updates.<br>A read of this register returns the following values:<br>Value Description<br>0x0000.0001 Locked<br>0x0000.0000 Unlocked |

## Register 9: Watchdog Peripheral Identification 4 (WDTPeriphID4), offset 0xFD0

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 4 (WDTPeriphID4)

Base 0x4000.0000
Offset 0xFD0
Type RO, reset 0x0000.0000



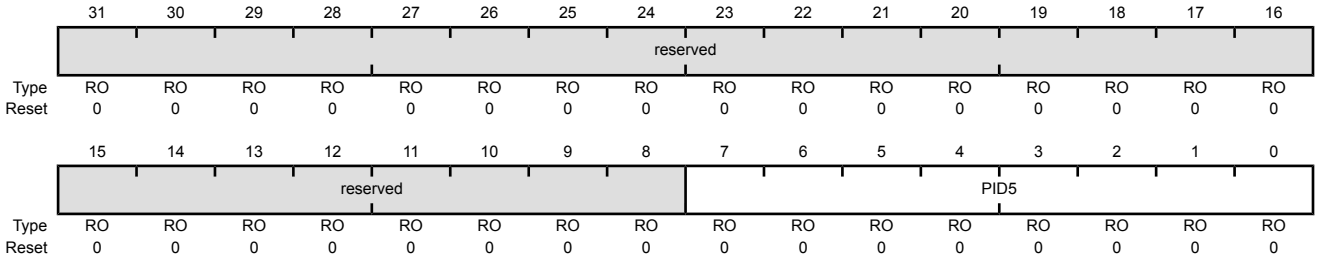

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID4 | RO | 0x00 | WDT Peripheral ID Register[7:0] |

## Register 10: Watchdog Peripheral Identification 5 (WDTPeriphID5), offset 0xFD4

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 5 (WDTPeriphID5)

Base 0x4000.0000
Offset 0xFD4
Type RO, reset 0x0000.0000



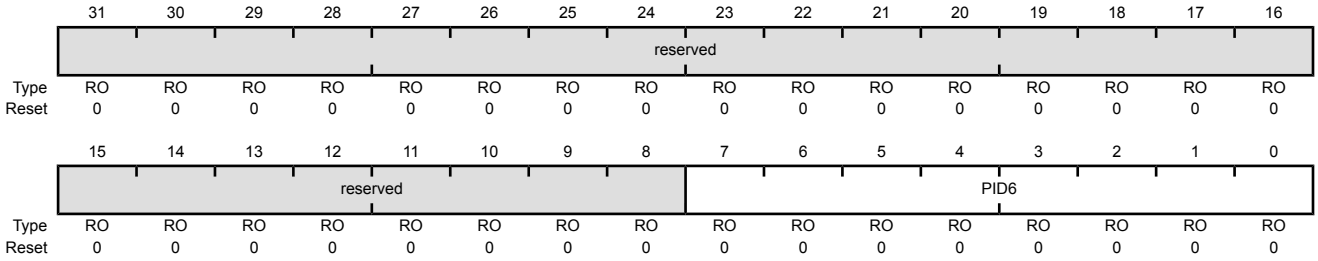

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID5 | RO | 0x00 | WDT Peripheral ID Register[15:8] |

## Register 11: Watchdog Peripheral Identification 6 (WDTPeriphID6), offset 0xFD8

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 6 (WDTPeriphID6)

Base 0x4000.0000
Offset 0xFD8
Type RO, reset 0x0000.0000



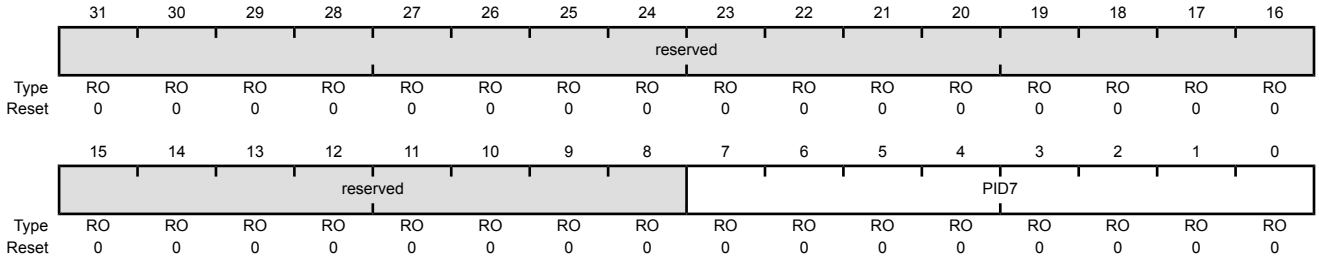

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID6 | RO | 0x00 | WDT Peripheral ID Register[23:16] |

## Register 12: Watchdog Peripheral Identification 7 (WDTPeriphID7), offset 0xFDC

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 7 (WDTPeriphID7)

Base 0x4000.0000
Offset 0xFDC
Type RO, reset 0x0000.0000



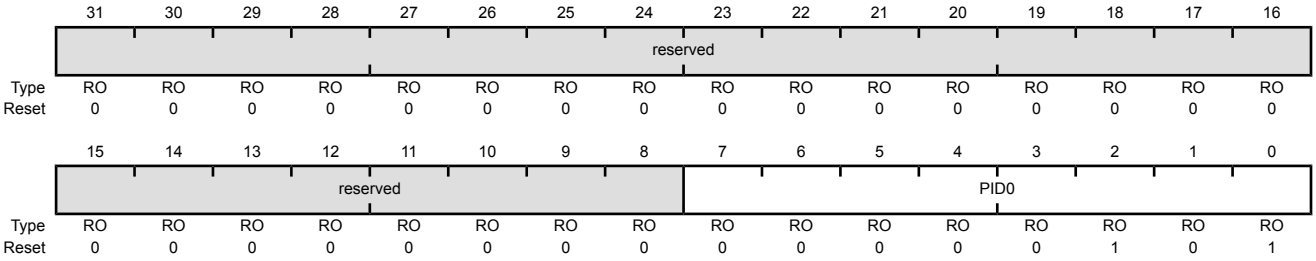

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID7 | RO | 0x00 | WDT Peripheral ID Register[31:24] |

## Register 13: Watchdog Peripheral Identification 0 (WDTPeriphID0), offset 0xFE0

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 0 (WDTPeriphID0)

Base 0x4000.0000
Offset 0xFE0
Type RO, reset 0x0000.0005



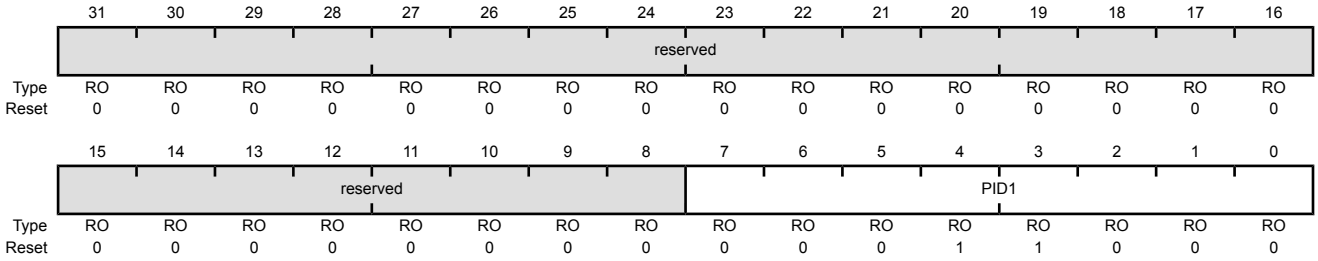

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID0 | RO | 0x05 | Watchdog Peripheral ID Register[7:0] |

## Register 14: Watchdog Peripheral Identification 1 (WDTPeriphID1), offset 0xFE4

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 1 (WDTPeriphID1)

Base 0x4000.0000
Offset 0xFE4
Type RO, reset 0x0000.0018



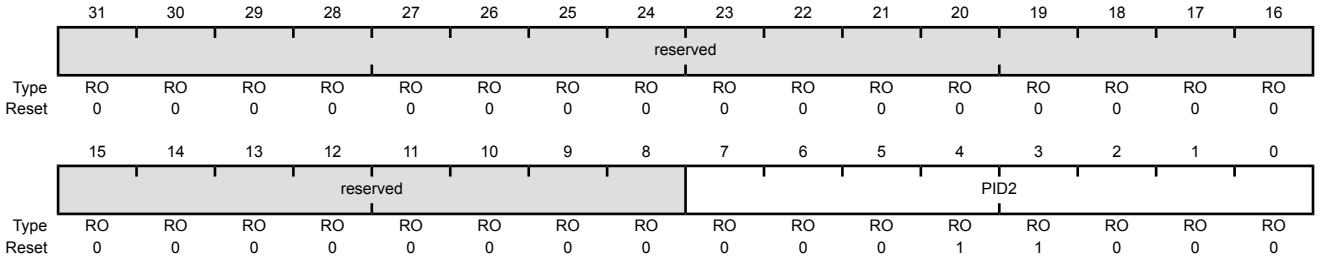

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID1 | RO | 0x18 | Watchdog Peripheral ID Register[15:8] |

## Register 15: Watchdog Peripheral Identification 2 (WDTPeriphID2), offset 0xFE8

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 2 (WDTPeriphID2)

Base 0x4000.0000
Offset 0xFE8
Type RO, reset 0x0000.0018



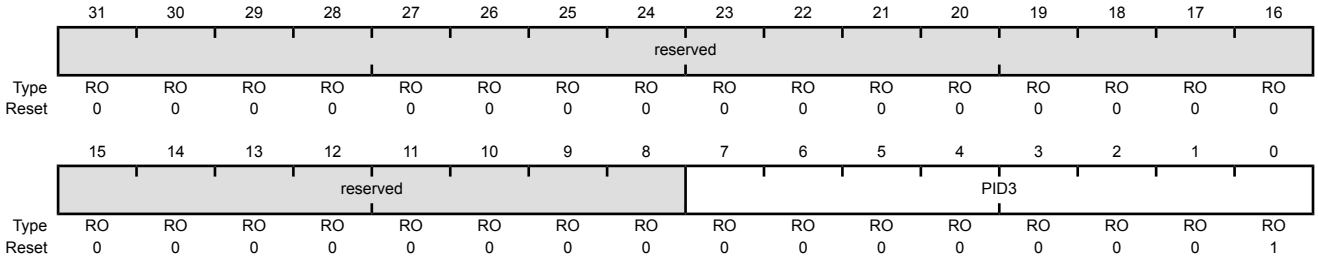

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID2 | RO | 0x18 | Watchdog Peripheral ID Register[23:16] |

## Register 16: Watchdog Peripheral Identification 3 (WDTPeriphID3), offset 0xFEC

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 3 (WDTPeriphID3)

Base 0x4000.0000
Offset 0xFEC
Type RO, reset 0x0000.0001



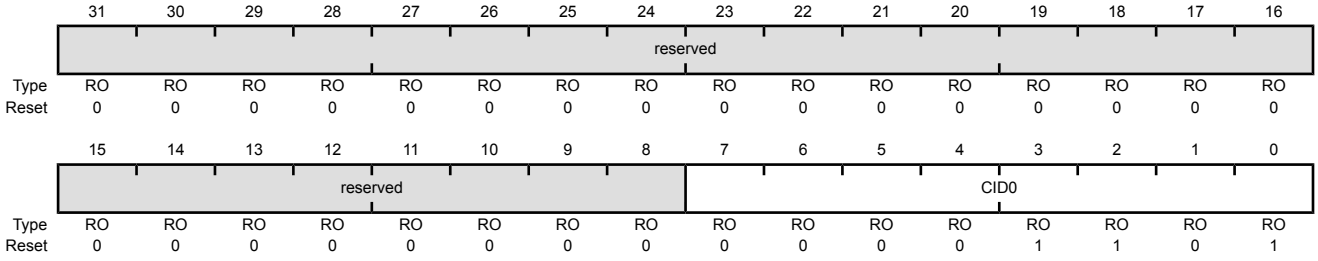

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID3 | RO | 0x01 | Watchdog Peripheral ID Register[31:24] |

## Register 17: Watchdog PrimeCell Identification 0 (WDTPCellID0), offset 0xFF0

The **WDTPCellIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog PrimeCell Identification 0 (WDTPCellID0)

Base 0x4000.0000
Offset 0xFF0
Type RO, reset 0x0000.000D



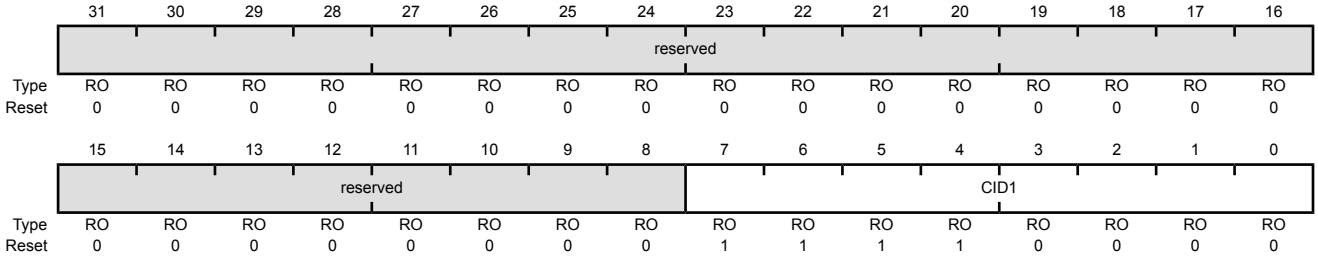

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID0 | RO | 0x0D | Watchdog PrimeCell ID Register[7:0] |

## Register 18: Watchdog PrimeCell Identification 1 (WDTPCellID1), offset 0xFF4

The **WDTPCellIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog PrimeCell Identification 1 (WDTPCellID1)

Base 0x4000.0000
Offset 0xFF4
Type RO, reset 0x0000.00F0



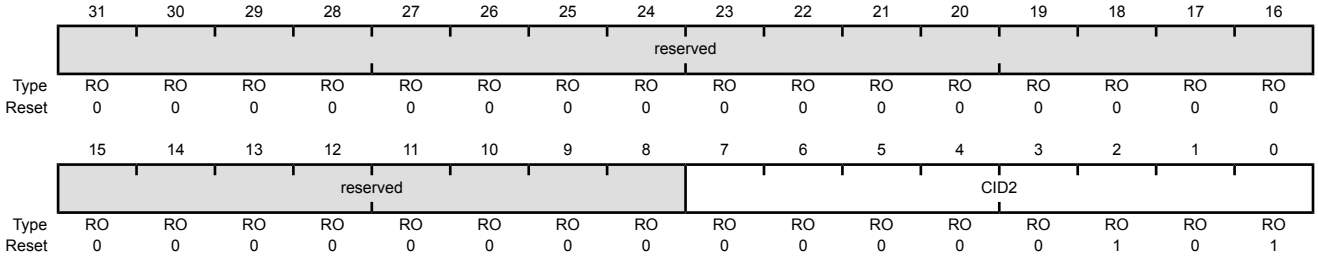

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID1 | RO | 0xF0 | Watchdog PrimeCell ID Register[15:8] |

## Register 19: Watchdog PrimeCell Identification 2 (WDTPCellID2), offset 0xFF8

The **WDTPCellIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog PrimeCell Identification 2 (WDTPCellID2)

Base 0x4000.0000
Offset 0xFF8
Type RO, reset 0x0000.0005



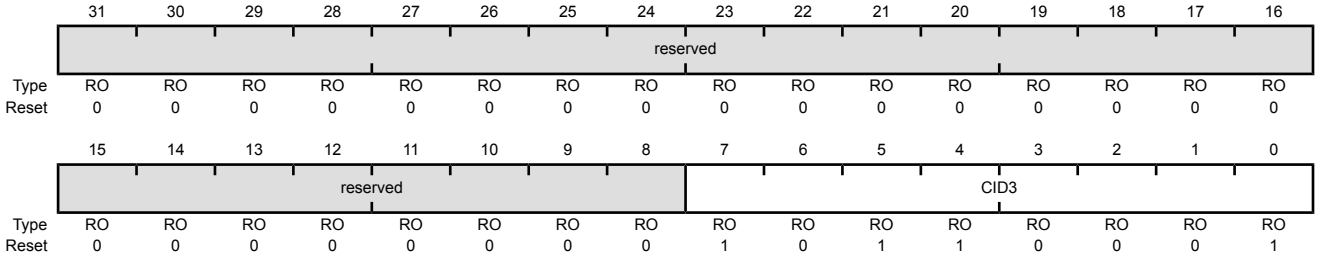

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID2 | RO | 0x05 | Watchdog PrimeCell ID Register[23:16] |

## Register 20: Watchdog PrimeCell Identification 3 (WDTPCellID3 ), offset 0xFFC

The **WDTPCellIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog PrimeCell Identification 3 (WDTPCellID3)

Base 0x4000.0000
Offset 0xFFC
Type RO, reset 0x0000.00B1




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID3 | RO | 0xB1 | Watchdog PrimeCell ID Register[31:24] |

# 12 Analog-to-Digital Converter (ADC)

An analog-to-digital converter (ADC) is a peripheral that converts a continuous analog voltage to a discrete digital number.

The Stellaris® ADC module features 10-bit conversion resolution and supports six input channels, plus an internal temperature sensor. The ADC module contains four programmable sequencer which allows for the sampling of multiple analog input sources without controller intervention. Each sample sequence provides flexible programming with fully configurable input source, trigger events, interrupt generation, and sequence priority.

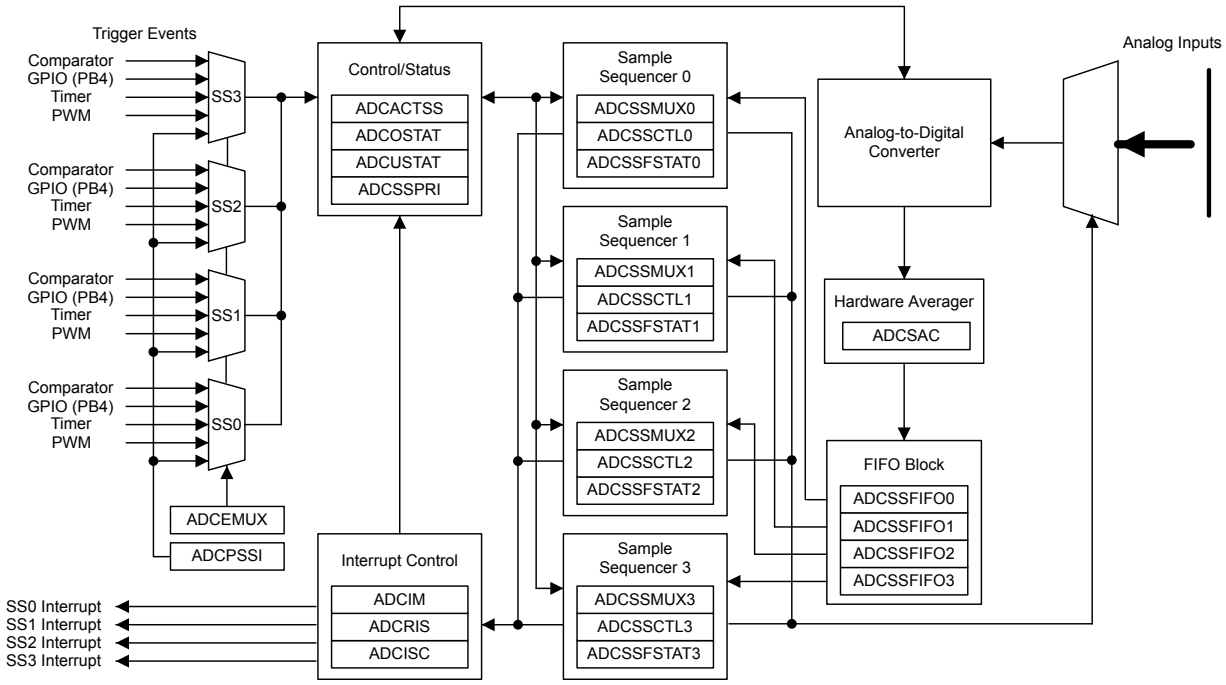
The Stellaris ADC module provides the following features:

- Six analog input channels
- Single-ended and differential-input configurations
- On-chip internal temperature sensor
- Sample rate of 500 thousand samples/second
- Flexible, configurable analog-to-digital conversion
- Four programmable sample conversion sequences from one to eight entries long, with corresponding conversion result FIFOs
- Flexible trigger control
  - Controller (software)
  - Timers
  - GPIO
- Hardware averaging of up to 64 samples for improved accuracy
- Converter uses an internal 3-V reference
- Power and ground for the analog circuitry is separate from the digital power and ground

## 12.1 Block Diagram

Figure 12-1 on page 458 provides details on the internal configuration of the ADC controls and data registers.

**Figure 12-1. ADC Module Block Diagram**




## 12.2 Signal Description

Table 12-1 on page 458 lists the external signals of the ADC module and describes the function of each. The signals are analog functions for some GPIO signals. The column in the table below titled "Pin Assignment" lists the GPIO pin placement for the ADC signals. The AINx analog signals are not 5-V tolerant and go through an isolation circuit before reaching their circuitry. These signals are configured by clearing the corresponding DEN bit in the **GPIO Digital Enable (GPIODEN)** register and setting the corresponding AMSEL bit in the **GPIO Analog Mode Select (GPIOAMSEL)** register. For more information on configuring GPIOs, see “General-Purpose Input/Outputs (GPIOs)” on page 351.

**Table 12-1. ADC Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| ADC0 | 1 | I | Analog | Analog-to-digital converter input 0. |
| ADC1 | 2 | I | Analog | Analog-to-digital converter input 1. |
| ADC2 | 5 | I | Analog | Analog-to-digital converter input 2. |
| ADC3 | 6 | I | Analog | Analog-to-digital converter input 3. |
| ADC4 | 64 | I | Analog | Analog-to-digital converter input 4. |
| ADC5 | 63 | I | Analog | Analog-to-digital converter input 5. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 12.3 Functional Description

The Stellaris ADC collects sample data by using a programmable sequence-based approach instead of the traditional single or double-sampling approaches found on many ADC modules. Each *sample sequence* is a fully programmed series of consecutive (back-to-back) samples, allowing the ADC to collect data from multiple input sources without having to be re-configured or serviced by the

controller. The programming of each sample in the sample sequence includes parameters such as the input source and mode (differential versus single-ended input), interrupt generation on sample completion, and the indicator for the last sample in the sequence.

### 12.3.1 Sample Sequencers

The sampling control and data capture is handled by the sample sequencers. All of the sequencers are identical in implementation except for the number of samples that can be captured and the depth of the FIFO. Table 12-2 on page 459 shows the maximum number of samples that each sequencer can capture and its corresponding FIFO depth. In this implementation, each FIFO entry is a 32-bit word, with the lower 10 bits containing the conversion result.

**Table 12-2. Samples and FIFO Depth of Sequencers**

| Sequencer | Number of Samples | Depth of FIFO |
|---|---|---|
| SS3 | 1 | 1 |
| SS2 | 4 | 4 |
| SS1 | 4 | 4 |
| SS0 | 8 | 8 |

For a given sample sequence, each sample is defined by two 4-bit nibbles in the **ADC Sample Sequence Input Multiplexer Select (ADCSSMUXn)** and **ADC Sample Sequence Control (ADCSSCTLn)** registers, where "n" corresponds to the sequence number. The **ADCSSMUXn** nibbles select the input pin, while the **ADCSSCTLn** nibbles contain the sample control bits corresponding to parameters such as temperature sensor selection, interrupt enable, end of sequence, and differential input mode. Sample sequencers are enabled by setting the respective ASENn bit in the **ADC Active Sample Sequencer (ADCACTSS)** register, and should be configured before being enabled.

When configuring a sample sequence, multiple uses of the same input pin within the same sequence is allowed. In the **ADCSSCTLn** register, the IEn bits can be set for any combination of samples, allowing interrupts to be generated after every sample in the sequence if necessary. Also, the END bit can be set at any point within a sample sequence. For example, if Sequencer 0 is used, the END bit can be set in the nibble associated with the fifth sample, allowing Sequencer 0 to complete execution of the sample sequence after the fifth sample.

After a sample sequence completes execution, the result data can be retrieved from the **ADC Sample Sequence Result FIFO (ADCSSFIFOn)** registers. The FIFOs are simple circular buffers that read a single address to "pop" result data. For software debug purposes, the positions of the FIFO head and tail pointers are visible in the **ADC Sample Sequence FIFO Status (ADCSSFSTATn)** registers along with FULL and EMPTY status flags. Overflow and underflow conditions are monitored using the **ADCOSTAT** and **ADCUSTAT** registers.

### 12.3.2 Module Control

Outside of the sample sequencers, the remainder of the control logic is responsible for tasks such as:

- Interrupt generation
- Sequence prioritization
- Trigger configuration

Most of the ADC control logic runs at the ADC clock rate of 14-18 MHz. The internal ADC divider is configured automatically by hardware when the system XTAL is selected. The automatic clock divider configuration targets 16.667 MHz operation for all Stellaris devices.

#### 12.3.2.1 Interrupts

The register configurations of the sample sequencers dictate which events generate raw interrupts, but do not have control over whether the interrupt is actually sent to the interrupt controller. The ADC module's interrupt signals are controlled by the state of the MASK bits in the **ADC Interrupt Mask (ADCIM)** register. Interrupt status can be viewed at two locations: the **ADC Raw Interrupt Status (ADCRIS)** register, which shows the raw status of the various interrupt signals, and the **ADC Interrupt Status and Clear (ADCISC)** register, which shows active interrupts that are enabled by the **ADCIM** register. Sequencer interrupts are cleared by writing a 1 to the corresponding IN bit in **ADCISC**.

#### 12.3.2.2 Prioritization

When sampling events (triggers) happen concurrently, they are prioritized for processing by the values in the **ADC Sample Sequencer Priority (ADCSSPRI)** register. Valid priority values are in the range of 0-3, with 0 being the highest priority and 3 being the lowest. Multiple active sample sequencer units with the same priority do not provide consistent results, so software must ensure that all active sample sequencer units have a unique priority value.

#### 12.3.2.3 Sampling Events

Sample triggering for each sample sequencer is defined in the **ADC Event Multiplexer Select (ADCEMUX)** register. The external peripheral triggering sources vary by Stellaris family member, but all devices share the "Controller" and "Always" triggers. Software can initiate sampling by setting the SSx bits in the **ADC Processor Sample Sequence Initiate (ADCPSSI)** register.

Care must be taken when using the "Always" trigger. If a sequence's priority is too high, it is possible to starve other lower priority sequences.

### 12.3.3 Hardware Sample Averaging Circuit

Higher precision results can be generated using the hardware averaging circuit, however, the improved results are at the cost of throughput. Up to 64 samples can be accumulated and averaged to form a single data entry in the sequencer FIFO. Throughput is decreased proportionally to the number of samples in the averaging calculation. For example, if the averaging circuit is configured to average 16 samples, the throughput is decreased by a factor of 16.

By default the averaging circuit is off and all data from the converter passes through to the sequencer FIFO. The averaging hardware is controlled by the **ADC Sample Averaging Control (ADCSAC)** register (see page 479). There is a single averaging circuit and all input channels receive the same amount of averaging whether they are single-ended or differential.

### 12.3.4 Analog-to-Digital Converter

The converter itself generates a 10-bit output value for selected analog input. Special analog pads are used to minimize the distortion on the input. An internal 3 V reference is used by the converter resulting in sample values ranging from 0x000 at 0 V input to 0x3FF at 3 V input when in single-ended input mode.

### 12.3.5 Differential Sampling

In addition to traditional single-ended sampling, the ADC module supports differential sampling of two analog input channels. To enable differential sampling, software must set the Dn bit in the **ADCSSCTL0n** register in a step's configuration nibble.

When a sequence step is configured for differential sampling, its corresponding value in the **ADCSSMUXn** register must be set to one of the four differential pairs, numbered 0-3. Differential pair 0 samples analog inputs 0 and 1; differential pair 1 samples analog inputs 2 and 3; and so on (see Table 12-3 on page 461). The ADC does not support other differential pairings such as analog input 0 with analog input 3. The number of differential pairs supported is dependent on the number of analog inputs (see Table 12-3 on page 461).

**Table 12-3. Differential Sampling Pairs**

| Differential Pair | Analog Inputs |
|---|---|
| 0 | 0 and 1 |
| 1 | 2 and 3 |
| 2 | 4 and 5 |

The voltage sampled in differential mode is the difference between the odd and even channels:

$\Delta V$ (differential voltage) = $V_{IN\_EVEN}$ (even channels) – $V_{IN\_ODD}$ (odd channels), therefore:

- If $\Delta V = 0$, then the conversion result = 0x1FF
- If $\Delta V > 0$, then the conversion result > 0x1FF (range is 0x1FF–0x3FF)
- If $\Delta V < 0$, then the conversion result < 0x1FF (range is 0–0x1FF)

The differential pairs assign polarities to the analog inputs: the even-numbered input is always positive, and the odd-numbered input is always negative. In order for a valid conversion result to appear, the negative input must be in the range of ± 1.5 V of the positive input. If an analog input is greater than 3 V or less than 0 V (the valid range for analog inputs), the input voltage is clipped, meaning it appears as either 3 V or 0 V, respectively, to the ADC.

Figure 12-2 on page 462 shows an example of the negative input centered at 1.5 V. In this configuration, the differential range spans from -1.5 V to 1.5 V. Figure 12-3 on page 462 shows an example where the negative input is centered at -0.75 V, meaning inputs on the positive input saturate past a differential voltage of -0.75 V since the input voltage is less than 0 V. Figure 12-4 on page 463 shows an example of the negative input centered at 2.25 V, where inputs on the positive channel saturate past a differential voltage of 0.75 V since the input voltage would be greater than 3 V.

**Figure 12-2. Differential Sampling Range, $V_{IN\_ODD}$ = 1.5 V**

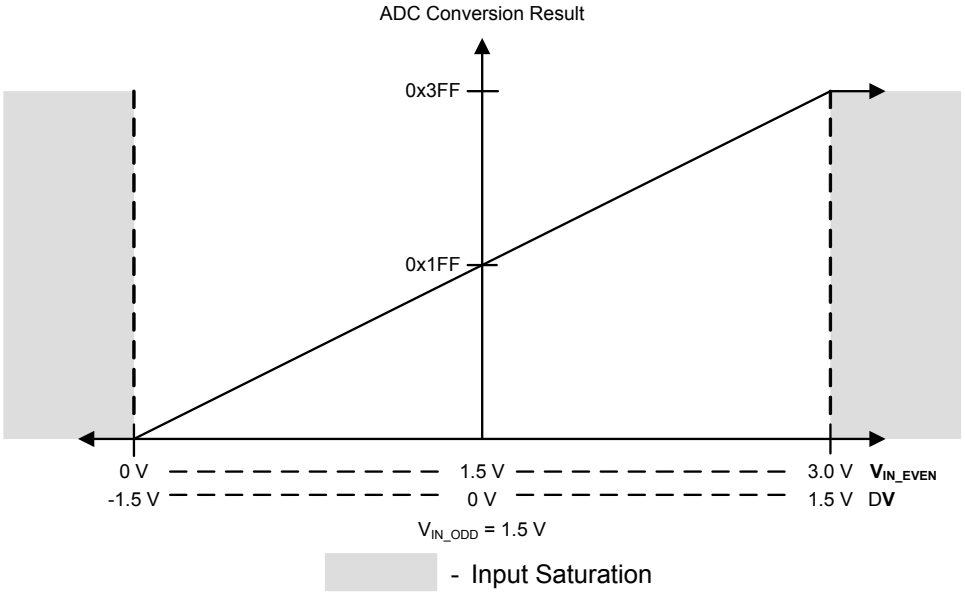



**Figure 12-3. Differential Sampling Range, $V_{IN\_ODD}$ = 0.75 V**

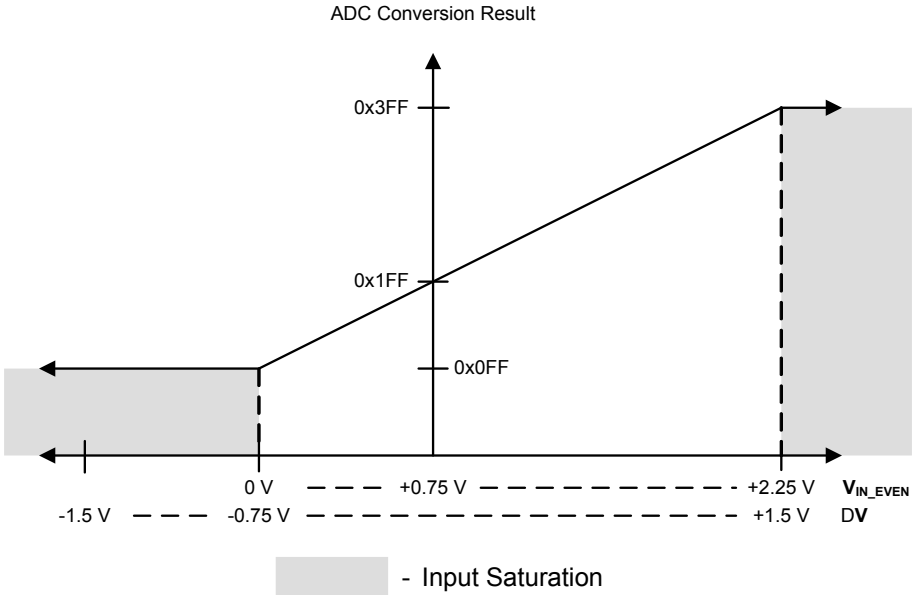
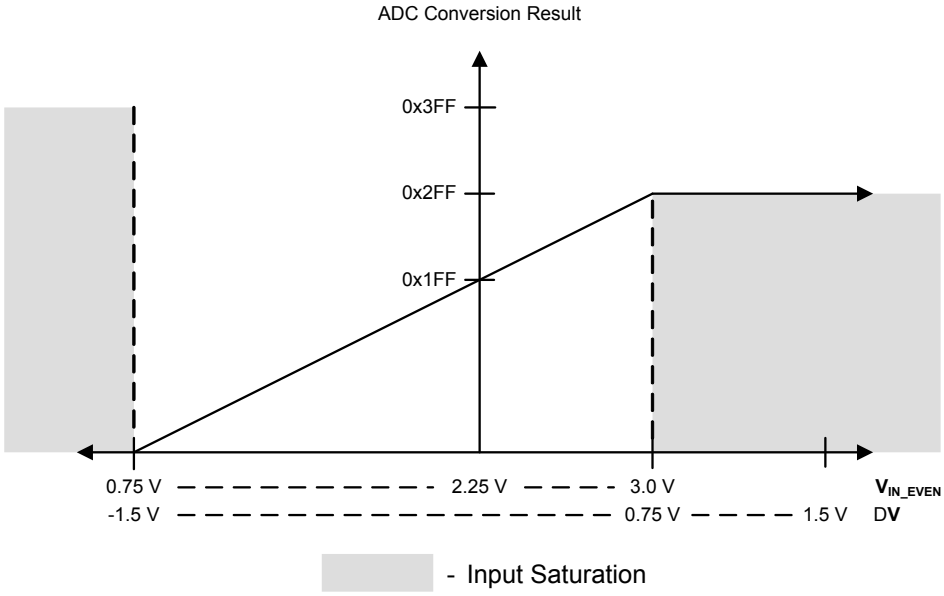


**Figure 12-4. Differential Sampling Range, $V_{IN\_ODD}$ = 2.25 V**




## 12.3.6 Internal Temperature Sensor

The temperature sensor serves two primary purposes: 1) to notify the system that internal temperature is too high or low for reliable operation, and 2) to provide temperature measurements for calibration of the Hibernate module RTC trim value.

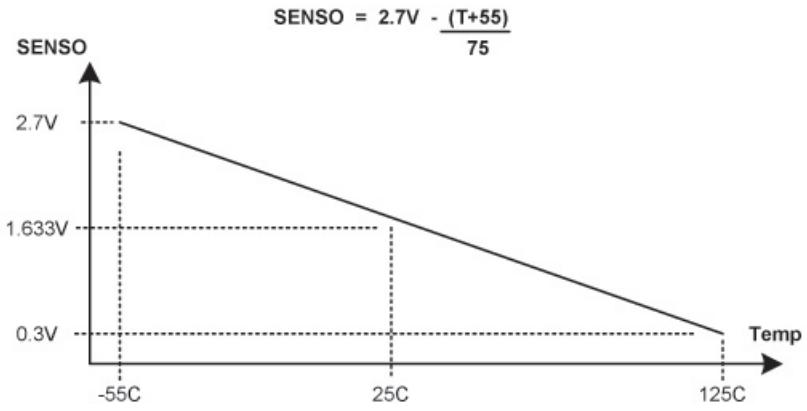
The temperature sensor does not have a separate enable, since it also contains the bandgap reference and must always be enabled. The reference is supplied to other analog modules; not just the ADC.

The internal temperature sensor provides an analog temperature reading as well as a reference voltage. The voltage at the output terminal SENSO is given by the following equation:

*SENSO* = 2.7 - ((T + 55) / 75)

This relation is shown in Figure 12-5 on page 464.

**Figure 12-5. Internal Temperature Sensor Characteristic**




## 12.4 Initialization and Configuration

In order for the ADC module to be used, the PLL must be enabled and using a supported crystal frequency (see the **RCC** register). Using unsupported frequencies can cause faulty operation in the ADC module.

### 12.4.1 Module Initialization

Initialization of the ADC module is a simple process with very few steps. The main steps include enabling the clock to the ADC, disabling the analog isolation circuit associated with all inputs that are to be used, and reconfiguring the sample sequencer priorities (if needed).

The initialization sequence for the ADC is as follows:

1. Enable the ADC clock by writing a value of 0x0001.0000 to the **RCGC0** register (see page 217).
2. Disable the analog isolation circuit for all ADC input pins that are to be used by writing a 1 to the appropriate bits of the **GPIOAMSEL** register (see page 386) in the associated GPIO block.
3. If required by the application, reconfigure the sample sequencer priorities in the **ADCSSPRI** register. The default configuration has Sample Sequencer 0 with the highest priority, and Sample Sequencer 3 as the lowest priority.

### 12.4.2 Sample Sequencer Configuration

Configuration of the sample sequencers is slightly more complex than the module initialization since each sample sequence is completely programmable.

The configuration for each sample sequencer should be as follows:

1. Ensure that the sample sequencer is disabled by writing a 0 to the corresponding ASENn bit in the **ADCACTSS** register. Programming of the sample sequencers is allowed without having them enabled. Disabling the sequencer during programming prevents erroneous execution if a trigger event were to occur during the configuration process.
2. Configure the trigger event for the sample sequencer in the **ADCEMUX** register.

3. For each sample in the sample sequence, configure the corresponding input source in the **ADCSSMUXn** register.

4. For each sample in the sample sequence, configure the sample control bits in the corresponding nibble in the **ADCSSCTLn** register. When programming the last nibble, ensure that the END bit is set. Failure to set the END bit causes unpredictable behavior.

5. If interrupts are to be used, write a 1 to the corresponding MASK bit in the **ADCIM** register.

6. Enable the sample sequencer logic by writing a 1 to the corresponding ASENn bit in the **ADCACTSS** register.

## 12.5 Register Map

Table 12-4 on page 465 lists the ADC registers. The offset listed is a hexadecimal increment to the register's address, relative to the ADC base address of 0x4003.8000.

Note that the ADC module clock must be enabled before the registers can be programmed (see page 217). There must be a delay of 3 system clocks after the ADC module clock is enabled before any ADC module registers are accessed.

**Table 12-4. ADC Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | ADCACTSS | R/W | 0x0000.0000 | ADC Active Sample Sequencer | 467 |
| 0x004 | ADCRIS | RO | 0x0000.0000 | ADC Raw Interrupt Status | 468 |
| 0x008 | ADCIM | R/W | 0x0000.0000 | ADC Interrupt Mask | 469 |
| 0x00C | ADCISC | R/W1C | 0x0000.0000 | ADC Interrupt Status and Clear | 470 |
| 0x010 | ADCOSTAT | R/W1C | 0x0000.0000 | ADC Overflow Status | 471 |
| 0x014 | ADCEMUX | R/W | 0x0000.0000 | ADC Event Multiplexer Select | 472 |
| 0x018 | ADCUSTAT | R/W1C | 0x0000.0000 | ADC Underflow Status | 475 |
| 0x020 | ADCSSPRI | R/W | 0x0000.3210 | ADC Sample Sequencer Priority | 476 |
| 0x028 | ADCPSSI | WO | - | ADC Processor Sample Sequence Initiate | 478 |
| 0x030 | ADCSAC | R/W | 0x0000.0000 | ADC Sample Averaging Control | 479 |
| 0x040 | ADCSSMUX0 | R/W | 0x0000.0000 | ADC Sample Sequence Input Multiplexer Select 0 | 480 |
| 0x044 | ADCSSCTL0 | R/W | 0x0000.0000 | ADC Sample Sequence Control 0 | 482 |
| 0x048 | ADCSSFIFO0 | RO | - | ADC Sample Sequence Result FIFO 0 | 485 |
| 0x04C | ADCSSFSTAT0 | RO | 0x0000.0100 | ADC Sample Sequence FIFO 0 Status | 486 |
| 0x060 | ADCSSMUX1 | R/W | 0x0000.0000 | ADC Sample Sequence Input Multiplexer Select 1 | 487 |
| 0x064 | ADCSSCTL1 | R/W | 0x0000.0000 | ADC Sample Sequence Control 1 | 488 |
| 0x068 | ADCSSFIFO1 | RO | - | ADC Sample Sequence Result FIFO 1 | 485 |
| 0x06C | ADCSSFSTAT1 | RO | 0x0000.0100 | ADC Sample Sequence FIFO 1 Status | 486 |
| 0x080 | ADCSSMUX2 | R/W | 0x0000.0000 | ADC Sample Sequence Input Multiplexer Select 2 | 487 |

**Table 12-4. ADC Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x084 | ADCSSCTL2 | R/W | 0x0000.0000 | ADC Sample Sequence Control 2 | 488 |
| 0x088 | ADCSSFIFO2 | RO | - | ADC Sample Sequence Result FIFO 2 | 485 |
| 0x08C | ADCSSFSTAT2 | RO | 0x0000.0100 | ADC Sample Sequence FIFO 2 Status | 486 |
| 0x0A0 | ADCSSMUX3 | R/W | 0x0000.0000 | ADC Sample Sequence Input Multiplexer Select 3 | 490 |
| 0x0A4 | ADCSSCTL3 | R/W | 0x0000.0002 | ADC Sample Sequence Control 3 | 491 |
| 0x0A8 | ADCSSFIFO3 | RO | - | ADC Sample Sequence Result FIFO 3 | 485 |
| 0x0AC | ADCSSFSTAT3 | RO | 0x0000.0100 | ADC Sample Sequence FIFO 3 Status | 486 |

## 12.6 Register Descriptions

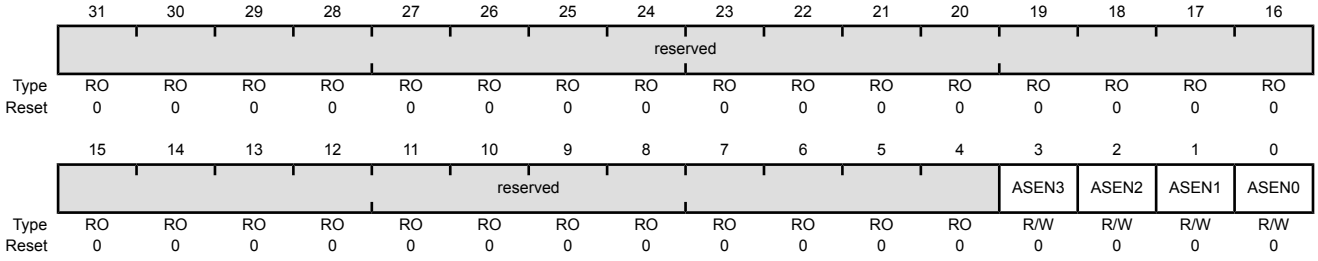
The remainder of this section lists and describes the ADC registers, in numerical order by address offset.

## Register 1: ADC Active Sample Sequencer (ADCACTSS), offset 0x000

This register controls the activation of the sample sequencers. Each sample sequencer can be enabled or disabled independently.

ADC Active Sample Sequencer (ADCACTSS)

Base 0x4003.8000
Offset 0x000
Type R/W, reset 0x0000.0000



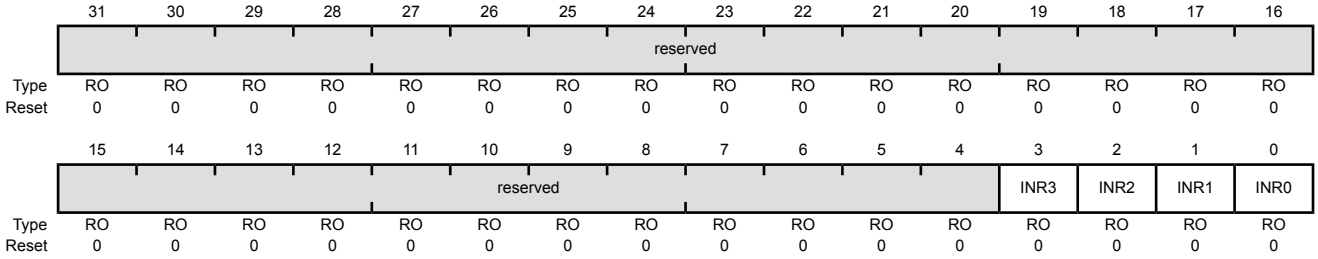

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | ASEN3 | R/W | 0 | ADC SS3 Enable<br>Specifies whether Sample Sequencer 3 is enabled. If set, the sample sequence logic for Sequencer 3 is active. Otherwise, the sequencer is inactive. |
| 2 | ASEN2 | R/W | 0 | ADC SS2 Enable<br>Specifies whether Sample Sequencer 2 is enabled. If set, the sample sequence logic for Sequencer 2 is active. Otherwise, the sequencer is inactive. |
| 1 | ASEN1 | R/W | 0 | ADC SS1 Enable<br>Specifies whether Sample Sequencer 1 is enabled. If set, the sample sequence logic for Sequencer 1 is active. Otherwise, the sequencer is inactive. |
| 0 | ASEN0 | R/W | 0 | ADC SS0 Enable<br>Specifies whether Sample Sequencer 0 is enabled. If set, the sample sequence logic for Sequencer 0 is active. Otherwise, the sequencer is inactive. |

## Register 2: ADC Raw Interrupt Status (ADCRIS), offset 0x004

This register shows the status of the raw interrupt signal of each sample sequencer. These bits may be polled by software to look for interrupt conditions without having to generate controller interrupts.

ADC Raw Interrupt Status (ADCRIS)

Base 0x4003.8000
Offset 0x004
Type RO, reset 0x0000.0000



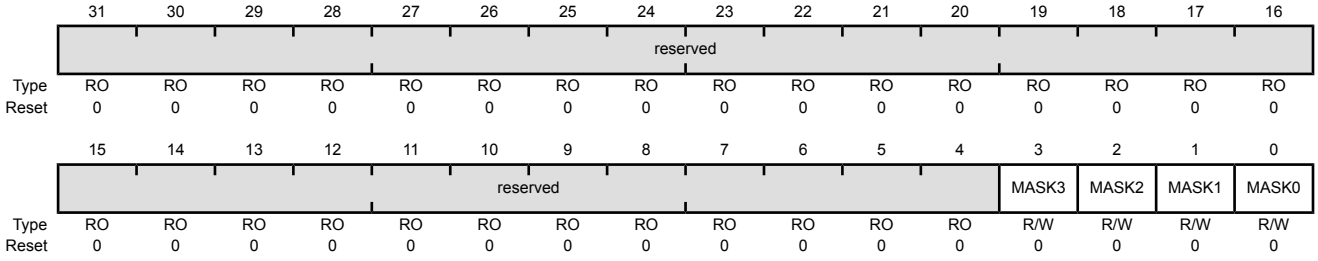

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | INR3 | RO | 0 | SS3 Raw Interrupt Status<br>This bit is set by hardware when a sample with its respective **ADCSSCTL3** `IE` bit has completed conversion. This bit is cleared by setting the `IN3` bit in the **ADCISC** register. |
| 2 | INR2 | RO | 0 | SS2 Raw Interrupt Status<br>This bit is set by hardware when a sample with its respective **ADCSSCTL2** `IE` bit has completed conversion. This bit is cleared by setting the `IN2` bit in the **ADCISC** register. |
| 1 | INR1 | RO | 0 | SS1 Raw Interrupt Status<br>This bit is set by hardware when a sample with its respective **ADCSSCTL1** `IE` bit has completed conversion. This bit is cleared by setting the `IN1` bit in the **ADCISC** register. |
| 0 | INR0 | RO | 0 | SS0 Raw Interrupt Status<br>This bit is set by hardware when a sample with its respective **ADCSSCTL0** `IE` bit has completed conversion. This bit is cleared by setting the `IN30` bit in the **ADCISC** register. |

## Register 3: ADC Interrupt Mask (ADCIM), offset 0x008

This register controls whether the sample sequencer raw interrupt signals are promoted to controller interrupts. Each raw interrupt signal can be masked independently.

ADC Interrupt Mask (ADCIM)

Base 0x4003.8000
Offset 0x008
Type R/W, reset 0x0000.0000



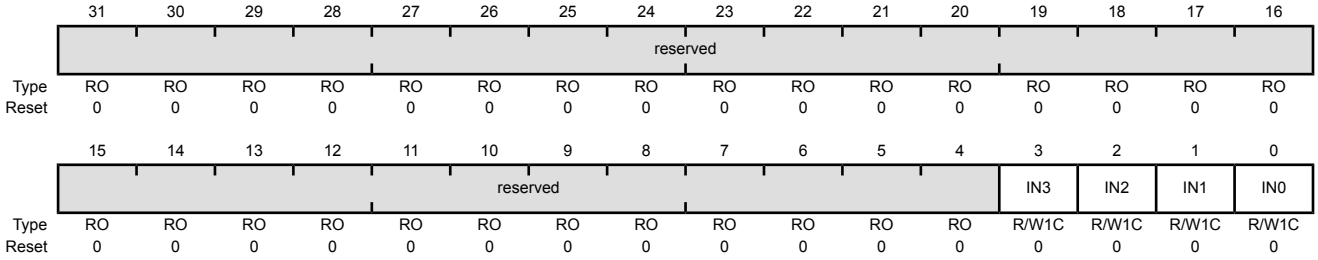

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | MASK3 | R/W | 0 | SS3 Interrupt Mask<br>When set, this bit allows the raw interrupt signal from Sample Sequencer 3 (**ADCRIS** register `INR3` bit) to be promoted to a controller interrupt.<br>When clear, the status of Sample Sequencer 3 does not affect the SS3 interrupt status. |
| 2 | MASK2 | R/W | 0 | SS2 Interrupt Mask<br>When set, this bit allows the raw interrupt signal from Sample Sequencer 2 (**ADCRIS** register `INR2` bit) to be promoted to a controller interrupt.<br>When clear, the status of Sample Sequencer 2 does not affect the SS2 interrupt status. |
| 1 | MASK1 | R/W | 0 | SS1 Interrupt Mask<br>When set, this bit allows the raw interrupt signal from Sample Sequencer 1 (**ADCRIS** register `INR1` bit) to be promoted to a controller interrupt.<br>When clear, the status of Sample Sequencer 1 does not affect the SS1 interrupt status. |
| 0 | MASK0 | R/W | 0 | SS0 Interrupt Mask<br>When set, this bit allows the raw interrupt signal from Sample Sequencer 0 (**ADCRIS** register `INR0` bit) to be promoted to a controller interrupt.<br>When clear, the status of Sample Sequencer 0 does not affect the SS0 interrupt status. |

## Register 4: ADC Interrupt Status and Clear (ADCISC), offset 0x00C

This register provides the mechanism for clearing sample sequence interrupt conditions and shows the status of controller interrupts generated by the sample sequencers. When read, each bit field is the logical AND of the respective INR and MASK bits. Sample sequence nterrupts are cleared by setting the corresponding bit position. If software is polling the **ADCRIS** instead of generating interrupts, the sample sequence INR bits are still cleared via the **ADCISC** register, even if the IN bit is not set.

ADC Interrupt Status and Clear (ADCISC)

Base 0x4003.8000
Offset 0x00C
Type R/W1C, reset 0x0000.0000



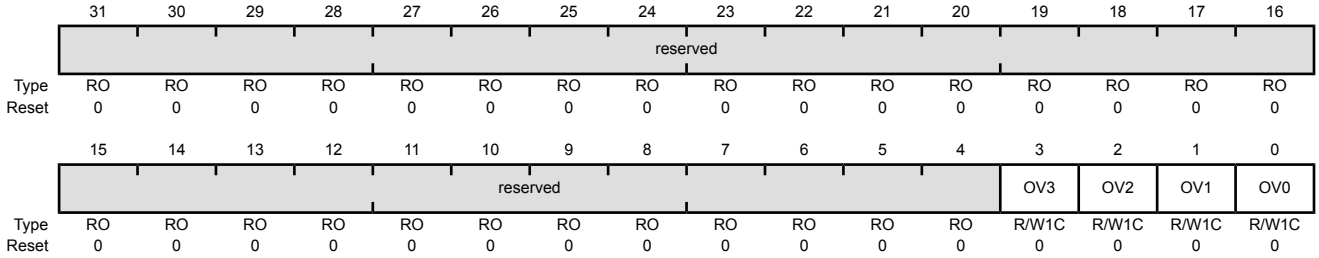

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | IN3 | R/W1C | 0 | SS3 Interrupt Status and Clear<br>This bit is set when both the INR3 bit in the **ADCRIS** register and the MASK3 bit in the **ADCIM** register are set, providing a level-based interrupt to the controller.<br>This bit is cleared by writing a 1. Clearing this bit also clears the INR3 bit. |
| 2 | IN2 | R/W1C | 0 | SS2 Interrupt Status and Clear<br>This bit is set when both the INR2 bit in the **ADCRIS** register and the MASK2 bit in the **ADCIM** register are set, providing a level-based interrupt to the controller.<br>This bit is cleared by writing a 1. Clearing this bit also clears the INR2 bit. |
| 1 | IN1 | R/W1C | 0 | SS1 Interrupt Status and Clear<br>This bit is set when both the INR1 bit in the **ADCRIS** register and the MASK1 bit in the **ADCIM** register are set, providing a level-based interrupt to the controller.<br>This bit is cleared by writing a 1. Clearing this bit also clears the INR1 bit. |
| 0 | IN0 | R/W1C | 0 | SS0 Interrupt Status and Clear<br>This bit is set when both the INR0 bit in the **ADCRIS** register and the MASK0 bit in the **ADCIM** register are set, providing a level-based interrupt to the controller.<br>This bit is cleared by writing a 1. Clearing this bit also clears the INR0 bit. |

## Register 5: ADC Overflow Status (ADCOSTAT), offset 0x010

This register indicates overflow conditions in the sample sequencer FIFOs. Once the overflow condition has been handled by software, the condition can be cleared by writing a 1 to the corresponding bit position.

ADC Overflow Status (ADCOSTAT)

Base 0x4003.8000
Offset 0x010
Type R/W1C, reset 0x0000.0000



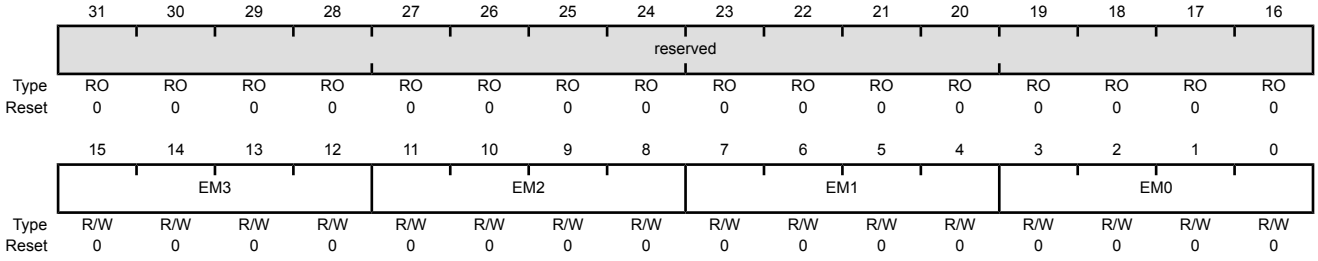

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | OV3 | R/W1C | 0 | SS3 FIFO Overflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 3 has hit an overflow condition where the FIFO is full and a write was requested. When an overflow is detected, the most recent write is dropped.<br>This bit is cleared by writing a 1. |
| 2 | OV2 | R/W1C | 0 | SS2 FIFO Overflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 2 has hit an overflow condition where the FIFO is full and a write was requested. When an overflow is detected, the most recent write is dropped.<br>This bit is cleared by writing a 1. |
| 1 | OV1 | R/W1C | 0 | SS1 FIFO Overflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 1 has hit an overflow condition where the FIFO is full and a write was requested. When an overflow is detected, the most recent write is dropped.<br>This bit is cleared by writing a 1. |
| 0 | OV0 | R/W1C | 0 | SS0 FIFO Overflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 0 has hit an overflow condition where the FIFO is full and a write was requested. When an overflow is detected, the most recent write is dropped.<br>This bit is cleared by writing a 1. |

## Register 6: ADC Event Multiplexer Select (ADCEMUX), offset 0x014

The **ADCEMUX** selects the event (trigger) that initiates sampling for each sample sequencer. Each sample sequencer can be configured with a unique trigger source.

ADC Event Multiplexer Select (ADCEMUX)

Base 0x4003.8000
Offset 0x014
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:12 | EM3 | R/W | 0x0 | SS3 Trigger Select<br>This field selects the trigger source for Sample Sequencer 3.<br>The valid configurations for this field are: |

| Value | Event |
|---|---|
| 0x0 | Controller (default) |
| 0x1 | Reserved |
| 0x2 | Reserved |
| 0x3 | Reserved |
| 0x4 | External (GPIO PB4) |
| 0x5 | Timer<br>In addition, the trigger must be enabled with the TnOTE bit in the **GPTMCTL** register (see page 416). |
| 0x6 | reserved |
| 0x7 | reserved |
| 0x8 | reserved |
| 0x9-0xE | reserved |
| 0xF | Always (continuously sample) |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 11:8 | EM2 | R/W | 0x0 | SS2 Trigger Select<br>This field selects the trigger source for Sample Sequencer 2.<br>The valid configurations for this field are:<br>Value Event<br>0x0 Controller (default)<br>0x1 Reserved<br>0x2 Reserved<br>0x3 Reserved<br>0x4 External (GPIO PB4)<br>0x5 Timer<br>In addition, the trigger must be enabled with the TnOTE bit in the **GPTMCTL** register (see page 416).<br>0x6 reserved<br>0x7 reserved<br>0x8 reserved<br>0x9-0xE reserved<br>0xF Always (continuously sample) |
| 7:4 | EM1 | R/W | 0x0 | SS1 Trigger Select<br>This field selects the trigger source for Sample Sequencer 1.<br>The valid configurations for this field are:<br>Value Event<br>0x0 Controller (default)<br>0x1 Reserved<br>0x2 Reserved<br>0x3 Reserved<br>0x4 External (GPIO PB4)<br>0x5 Timer<br>In addition, the trigger must be enabled with the TnOTE bit in the **GPTMCTL** register (see page 416).<br>0x6 reserved<br>0x7 reserved<br>0x8 reserved<br>0x9-0xE reserved<br>0xF Always (continuously sample) |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3:0 | EM0 | R/W | 0x0 | SS0 Trigger Select<br>This field selects the trigger source for Sample Sequencer 0.<br>The valid configurations for this field are: |

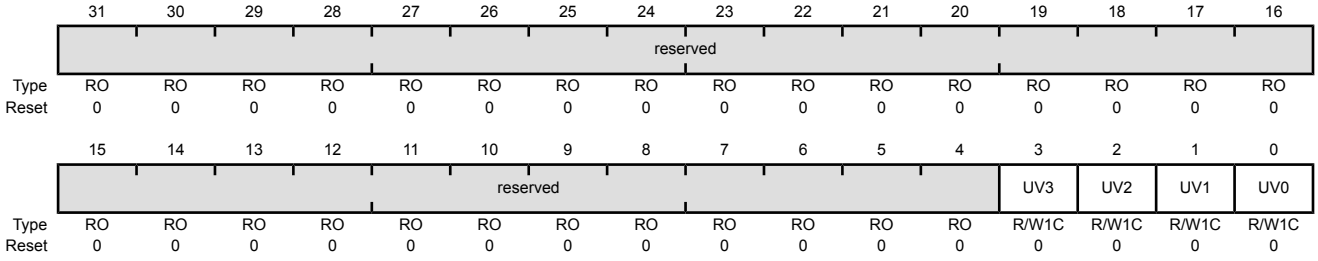
| Value | Event |
|---|---|
| 0x0 | Controller (default) |
| 0x1 | Reserved |
| 0x2 | Reserved |
| 0x3 | Reserved |
| 0x4 | External (GPIO PB4) |
| 0x5 | Timer<br>In addition, the trigger must be enabled with the TnOTE bit in the **GPTMCTL** register (see page 416). |
| 0x6 | reserved |
| 0x7 | reserved |
| 0x8 | reserved |
| 0x9-0xE | reserved |
| 0xF | Always (continuously sample) |

## Register 7: ADC Underflow Status (ADCUSTAT), offset 0x018

This register indicates underflow conditions in the sample sequencer FIFOs. The corresponding underflow condition is cleared by writing a 1 to the relevant bit position.

ADC Underflow Status (ADCUSTAT)

Base 0x4003.8000
Offset 0x018
Type R/W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | UV3 | R/W1C | 0 | SS3 FIFO Underflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 3 has hit an underflow condition where the FIFO is empty and a read was requested. The problematic read does not move the FIFO pointers, and 0s are returned.<br>This bit is cleared by writing a 1. |
| 2 | UV2 | R/W1C | 0 | SS2 FIFO Underflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 2 has hit an underflow condition where the FIFO is empty and a read was requested. The problematic read does not move the FIFO pointers, and 0s are returned.<br>This bit is cleared by writing a 1. |
| 1 | UV1 | R/W1C | 0 | SS1 FIFO Underflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 1 has hit an underflow condition where the FIFO is empty and a read was requested. The problematic read does not move the FIFO pointers, and 0s are returned.<br>This bit is cleared by writing a 1. |
| 0 | UV0 | R/W1C | 0 | SS0 FIFO Underflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 0 has hit an underflow condition where the FIFO is empty and a read was requested. The problematic read does not move the FIFO pointers, and 0s are returned.<br>This bit is cleared by writing a 1. |

## Register 8: ADC Sample Sequencer Priority (ADCSSPRI), offset 0x020

This register sets the priority for each of the sample sequencers. Out of reset, Sequencer 0 has the highest priority, and Sequencer 3 has the lowest priority. When reconfiguring sequence priorities, each sequence must have a unique priority for the ADC to operate properly.

ADC Sample Sequencer Priority (ADCSSPRI)

Base 0x4003.8000
Offset 0x020
Type R/W, reset 0x0000.3210




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:14 | reserved | RO | 0x0000.0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 13:12 | SS3 | R/W | 0x3 | SS3 Priority<br>This field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 3. A priority encoding of 0 is highest and 3 is lowest. The priorities assigned to the sequencers must be uniquely mapped. The ADC may not operate properly if two or more fields are equal. |
| 11:10 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9:8 | SS2 | R/W | 0x2 | SS2 Priority<br>This field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 2. A priority encoding of 0 is highest and 3 is lowest. The priorities assigned to the sequencers must be uniquely mapped. The ADC may not operate properly if two or more fields are equal. |
| 7:6 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:4 | SS1 | R/W | 0x1 | SS1 Priority<br>This field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 1. A priority encoding of 0 is highest and 3 is lowest. The priorities assigned to the sequencers must be uniquely mapped. The ADC may not operate properly if two or more fields are equal. |
| 3:2 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1:0 | SS0 | R/W | 0x0 | SS0 Priority<br>This field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 0. A priority encoding of 0 is highest and 3 is lowest. The priorities assigned to the sequencers must be uniquely mapped. The ADC may not operate properly if two or more fields are equal. |

## Register 9: ADC Processor Sample Sequence Initiate (ADCPSSI), offset 0x028

This register provides a mechanism for application software to initiate sampling in the sample sequencers. Sample sequences can be initiated individually or in any combination. When multiple sequences are triggered simultaneously, the priority encodings in **ADCSSPRI** dictate execution order.

ADC Processor Sample Sequence Initiate (ADCPSSI)

Base 0x4003.8000
Offset 0x028
Type WO, reset -

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | | | | | | | | | | | |
| RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | | | | | | | | SS3 | SS2 | SS1 | SS0 |
| RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | WO | WO | WO | WO |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - | - | - | - |

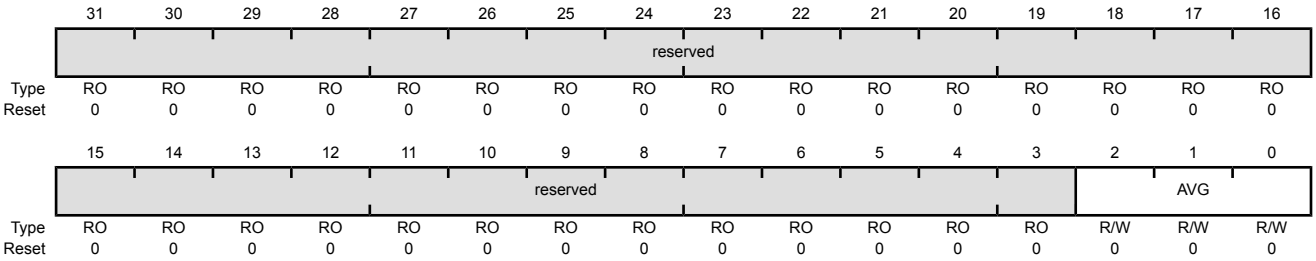
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | SS3 | WO | - | SS3 Initiate<br>When set, this bit triggers sampling on Sample Sequencer 3 if the sequencer is enabled in the **ADCACTSS** register.<br>Only a write by software is valid; a read of this register returns no meaningful data. |
| 2 | SS2 | WO | - | SS2 Initiate<br>When set, this bit triggers sampling on Sample Sequencer 2 if the sequencer is enabled in the **ADCACTSS** register.<br>Only a write by software is valid; a read of this register returns no meaningful data. |
| 1 | SS1 | WO | - | SS1 Initiate<br>When set, this bit triggers sampling on Sample Sequencer 1 if the sequencer is enabled in the **ADCACTSS** register.<br>Only a write by software is valid; a read of this register returns no meaningful data. |
| 0 | SS0 | WO | - | SS0 Initiate<br>When set, this bit triggers sampling on Sample Sequencer 0 if the sequencer is enabled in the **ADCACTSS** register.<br>Only a write by software is valid; a read of this register returns no meaningful data. |

## Register 10: ADC Sample Averaging Control (ADCSAC), offset 0x030

This register controls the amount of hardware averaging applied to conversion results. The final conversion result stored in the FIFO is averaged from $2^{AVG}$ consecutive ADC samples at the specified ADC speed. If AVG is 0, the sample is passed directly through without any averaging. If AVG=6, then 64 consecutive ADC samples are averaged to generate one result in the sequencer FIFO. An AVG = 7 provides unpredictable results.

ADC Sample Averaging Control (ADCSAC)

Base 0x4003.8000
Offset 0x030
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | AVG | R/W | 0x0 | Hardware Averaging Control<br>Specifies the amount of hardware averaging that will be applied to ADC samples. The AVG field can be any value between 0 and 6. Entering a value of 7 creates unpredictable results. |

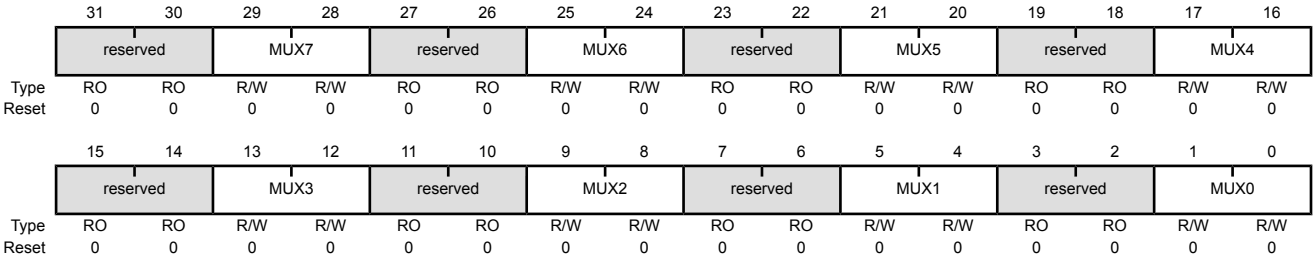
| Value | Description |
|---|---|
| 0x0 | No hardware oversampling |
| 0x1 | 2x hardware oversampling |
| 0x2 | 4x hardware oversampling |
| 0x3 | 8x hardware oversampling |
| 0x4 | 16x hardware oversampling |
| 0x5 | 32x hardware oversampling |
| 0x6 | 64x hardware oversampling |
| 0x7 | Reserved |

## Register 11: ADC Sample Sequence Input Multiplexer Select 0 (ADCSSMUX0), offset 0x040

This register defines the analog input configuration for each sample in a sequence executed with Sample Sequencer 0. This register is 32 bits wide and contains information for eight possible samples.

ADC Sample Sequence Input Multiplexer Select 0 (ADCSSMUX0)

Base 0x4003.8000
Offset 0x040
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 29:28 | MUX7 | R/W | 0x0 | 8th Sample Input Select<br>The MUX7 field is used during the eighth sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. The value set here indicates the corresponding pin, for example, a value of 1 indicates the input is ADC1. |
| 27:26 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 25:24 | MUX6 | R/W | 0x0 | 7th Sample Input Select<br>The MUX6 field is used during the seventh sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 23:22 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 21:20 | MUX5 | R/W | 0x0 | 6th Sample Input Select<br>The MUX5 field is used during the sixth sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 19:18 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 17:16 | MUX4 | R/W | 0x0 | 5th Sample Input Select<br>The MUX4 field is used during the fifth sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 15:14 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 13:12 | MUX3 | R/W | 0x0 | 4th Sample Input Select<br>The MUX3 field is used during the fourth sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 11:10 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9:8 | MUX2 | R/W | 0x0 | 3rd Sample Input Select<br>The MUX72 field is used during the third sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 7:6 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:4 | MUX1 | R/W | 0x0 | 2nd Sample Input Select<br>The MUX1 field is used during the second sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1:0 | MUX0 | R/W | 0x0 | 1st Sample Input Select<br>The MUX0 field is used during the first sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |

## Register 12: ADC Sample Sequence Control 0 (ADCSSCTL0), offset 0x044

This register contains the configuration information for each sample for a sequence executed with a sample sequencer. When configuring a sample sequence, the END bit must be set at some point, whether it be after the first sample, last sample, or any sample in between. This register is 32-bits wide and contains information for eight possible samples.

ADC Sample Sequence Control 0 (ADCSSCTL0)

Base 0x4003.8000
Offset 0x044
Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TS7 | IE7 | END7 | D7 | TS6 | IE6 | END6 | D6 | TS5 | IE5 | END5 | D5 | TS4 | IE4 | END4 | D4 |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TS3 | IE3 | END3 | D3 | TS2 | IE2 | END2 | D2 | TS1 | IE1 | END1 | D1 | TS0 | IE0 | END0 | D0 |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | TS7 | R/W | 0 | 8th Sample Temp Sensor Select<br>This bit is used during the eighth sample of the sample sequence and and specifies the input source of the sample.<br>When set, the temperature sensor is read.<br>When clear, the input pin specified by the **ADCSSMUX** register is read. |
| 30 | IE7 | R/W | 0 | 8th Sample Interrupt Enable<br>This bit is used during the eighth sample of the sample sequence and specifies whether the raw interrupt signal (INR0 bit) is asserted at the end of the sample's conversion. If the MASK0 bit in the **ADCIM** register is set, the interrupt is promoted to a controller-level interrupt.<br>When this bit is set, the raw interrupt is asserted.<br>When this bit is clear, the raw interrupt is not asserted.<br>It is legal to have multiple samples within a sequence generate interrupts. |
| 29 | END7 | R/W | 0 | 8th Sample is End of Sequence<br>The END7 bit indicates that this is the last sample of the sequence. It is possible to end the sequence on any sample position. Samples defined after the sample containing a set END are not requested for conversion even though the fields may be non-zero. It is required that software write the END bit somewhere within the sequence. (Sample Sequencer 3, which only has a single sample in the sequence, is hardwired to have the END0 bit set.)<br>Setting this bit indicates that this sample is the last in the sequence. |
| 28 | D7 | R/W | 0 | 8th Sample Diff Input Select<br>The D7 bit indicates that the analog input is to be differentially sampled. The corresponding **ADCSSMUXx** nibble must be set to the pair number "i", where the paired inputs are "2i and 2i+1". The temperature sensor does not have a differential option. When set, the analog inputs are differentially sampled. |
| 27 | TS6 | R/W | 0 | 7th Sample Temp Sensor Select<br>Same definition as TS7 but used during the seventh sample. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 26 | IE6 | R/W | 0 | 7th Sample Interrupt Enable<br>Same definition as IE7 but used during the seventh sample. |
| 25 | END6 | R/W | 0 | 7th Sample is End of Sequence<br>Same definition as END7 but used during the seventh sample. |
| 24 | D6 | R/W | 0 | 7th Sample Diff Input Select<br>Same definition as D7 but used during the seventh sample. |
| 23 | TS5 | R/W | 0 | 6th Sample Temp Sensor Select<br>Same definition as TS7 but used during the sixth sample. |
| 22 | IE5 | R/W | 0 | 6th Sample Interrupt Enable<br>Same definition as IE7 but used during the sixth sample. |
| 21 | END5 | R/W | 0 | 6th Sample is End of Sequence<br>Same definition as END7 but used during the sixth sample. |
| 20 | D5 | R/W | 0 | 6th Sample Diff Input Select<br>Same definition as D7 but used during the sixth sample. |
| 19 | TS4 | R/W | 0 | 5th Sample Temp Sensor Select<br>Same definition as TS7 but used during the fifth sample. |
| 18 | IE4 | R/W | 0 | 5th Sample Interrupt Enable<br>Same definition as IE7 but used during the fifth sample. |
| 17 | END4 | R/W | 0 | 5th Sample is End of Sequence<br>Same definition as END7 but used during the fifth sample. |
| 16 | D4 | R/W | 0 | 5th Sample Diff Input Select<br>Same definition as D7 but used during the fifth sample. |
| 15 | TS3 | R/W | 0 | 4th Sample Temp Sensor Select<br>Same definition as TS7 but used during the fourth sample. |
| 14 | IE3 | R/W | 0 | 4th Sample Interrupt Enable<br>Same definition as IE7 but used during the fourth sample. |
| 13 | END3 | R/W | 0 | 4th Sample is End of Sequence<br>Same definition as END7 but used during the fourth sample. |
| 12 | D3 | R/W | 0 | 4th Sample Diff Input Select<br>Same definition as D7 but used during the fourth sample. |
| 11 | TS2 | R/W | 0 | 3rd Sample Temp Sensor Select<br>Same definition as TS7 but used during the third sample. |
| 10 | IE2 | R/W | 0 | 3rd Sample Interrupt Enable<br>Same definition as IE7 but used during the third sample. |
| 9 | END2 | R/W | 0 | 3rd Sample is End of Sequence<br>Same definition as END7 but used during the third sample. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 8 | D2 | R/W | 0 | 3rd Sample Diff Input Select<br>Same definition as D7 but used during the third sample. |
| 7 | TS1 | R/W | 0 | 2nd Sample Temp Sensor Select<br>Same definition as TS7 but used during the second sample. |
| 6 | IE1 | R/W | 0 | 2nd Sample Interrupt Enable<br>Same definition as IE7 but used during the second sample. |
| 5 | END1 | R/W | 0 | 2nd Sample is End of Sequence<br>Same definition as END7 but used during the second sample. |
| 4 | D1 | R/W | 0 | 2nd Sample Diff Input Select<br>Same definition as D7 but used during the second sample. |
| 3 | TS0 | R/W | 0 | 1st Sample Temp Sensor Select<br>Same definition as TS7 but used during the first sample. |
| 2 | IE0 | R/W | 0 | 1st Sample Interrupt Enable<br>Same definition as IE7 but used during the first sample. |
| 1 | END0 | R/W | 0 | 1st Sample is End of Sequence<br>Same definition as END7 but used during the first sample. |
| 0 | D0 | R/W | 0 | 1st Sample Diff Input Select<br>Same definition as D7 but used during the first sample. |

## Register 13: ADC Sample Sequence Result FIFO 0 (ADCSSFIFO0), offset 0x048

## Register 14: ADC Sample Sequence Result FIFO 1 (ADCSSFIFO1), offset 0x068

## Register 15: ADC Sample Sequence Result FIFO 2 (ADCSSFIFO2), offset 0x088

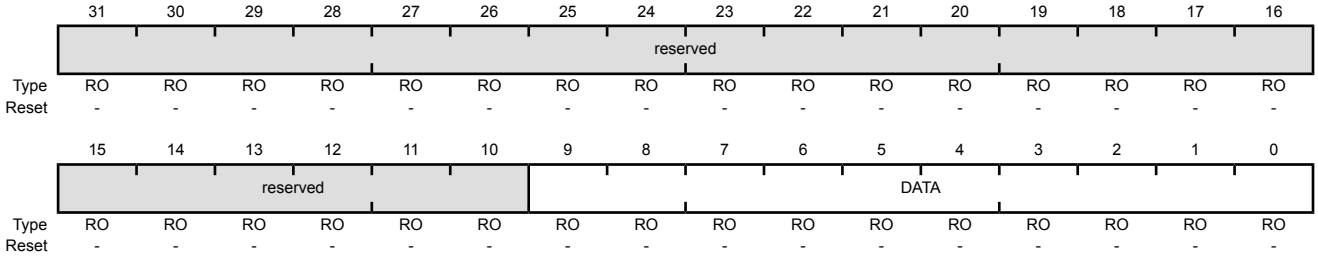
## Register 16: ADC Sample Sequence Result FIFO 3 (ADCSSFIFO3), offset 0x0A8

**Important:** This register is read-sensitive. See the register description for details.

This register contains the conversion results for samples collected with the sample sequencer (the **ADCSSFIFO0** register is used for Sample Sequencer 0, **ADCSSFIFO1** for Sequencer 1, **ADCSSFIFO2** for Sequencer 2, and **ADCSSFIFO3** for Sequencer 3). Reads of this register return conversion result data in the order sample 0, sample 1, and so on, until the FIFO is empty. If the FIFO is not properly handled by software, overflow and underflow conditions are registered in the **ADCOSTAT** and **ADCUSTAT** registers.

ADC Sample Sequence Result FIFO 0 (ADCSSFIFO0)

Base 0x4003.8000
Offset 0x048
Type RO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | reserved | RO | - | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9:0 | DATA | RO | - | Conversion Result Data |

## Register 17: ADC Sample Sequence FIFO 0 Status (ADCSSFSTAT0), offset 0x04C

## Register 18: ADC Sample Sequence FIFO 1 Status (ADCSSFSTAT1), offset 0x06C

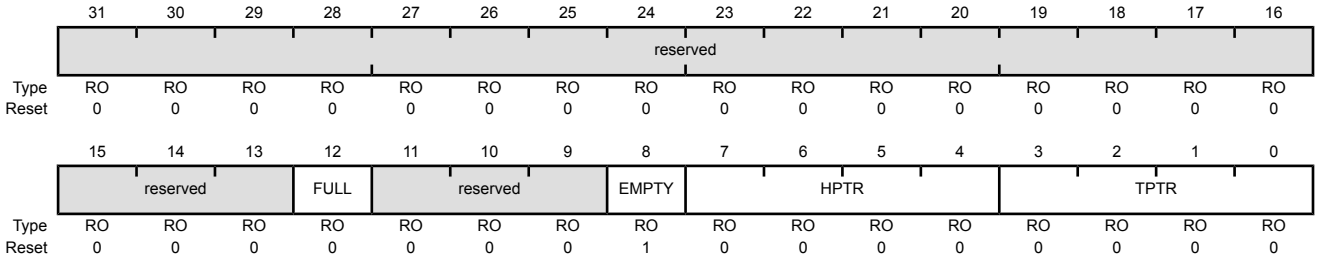
## Register 19: ADC Sample Sequence FIFO 2 Status (ADCSSFSTAT2), offset 0x08C

## Register 20: ADC Sample Sequence FIFO 3 Status (ADCSSFSTAT3), offset 0x0AC

This register provides a window into the sample sequencer, providing full/empty status information as well as the positions of the head and tail pointers. The reset value of 0x100 indicates an empty FIFO. The **ADCSSFSTAT0** register provides status on FIFO0, **ADCSSFSTAT1** on FIFO1, **ADCSSFSTAT2** on FIFO2, and **ADCSSFSTAT3** on FIFO3.

ADC Sample Sequence FIFO 0 Status (ADCSSFSTAT0)

Base 0x4003.8000
Offset 0x04C
Type RO, reset 0x0000.0100




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:13 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12 | FULL | RO | 0 | FIFO Full<br>When set, this bit indicates that the FIFO is currently full. |
| 11:9 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 8 | EMPTY | RO | 1 | FIFO Empty<br>When set, this bit indicates that the FIFO is currently empty. |
| 7:4 | HPTR | RO | 0x0 | FIFO Head Pointer<br>This field contains the current "head" pointer index for the FIFO, that is, the next entry to be written. |
| 3:0 | TPTR | RO | 0x0 | FIFO Tail Pointer<br>This field contains the current "tail" pointer index for the FIFO, that is, the next entry to be read. |

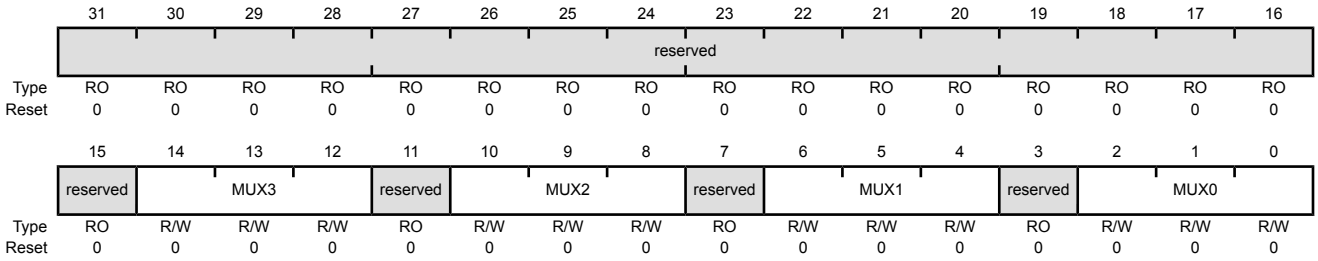
## Register 21: ADC Sample Sequence Input Multiplexer Select 1 (ADCSSMUX1), offset 0x060

## Register 22: ADC Sample Sequence Input Multiplexer Select 2 (ADCSSMUX2), offset 0x080

This register defines the analog input configuration for each sample in a sequence executed with Sample Sequencer 1 or 2. These registers are 16-bits wide and contain information for four possible samples. See the **ADCSSMUX0** register on page 480 for detailed bit descriptions. The **ADCSSMUX1** register affects Sample Sequencer 1 and the **ADCSSMUX2** register affects Sample Sequencer 2.

ADC Sample Sequence Input Multiplexer Select 1 (ADCSSMUX1)

Base 0x4003.8000
Offset 0x060
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:15 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14:12 | MUX3 | R/W | 0x0 | 4th Sample Input Select |
| 11 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10:8 | MUX2 | R/W | 0x0 | 3rd Sample Input Select |
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:4 | MUX1 | R/W | 0x0 | 2nd Sample Input Select |
| 3 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | MUX0 | R/W | 0x0 | 1st Sample Input Select |

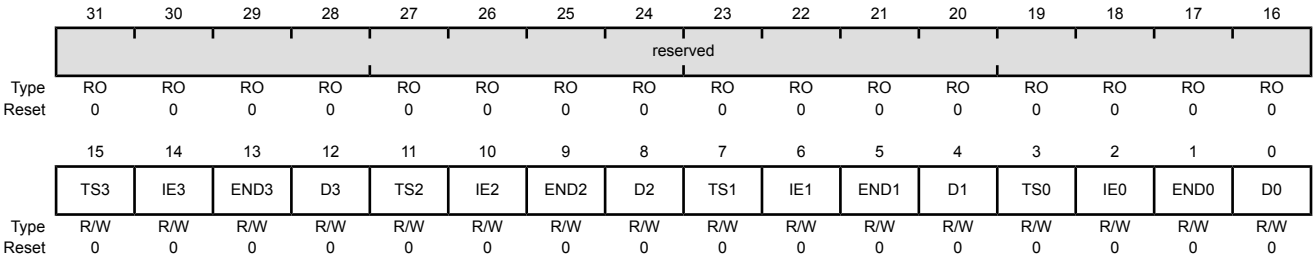
## Register 23: ADC Sample Sequence Control 1 (ADCSSCTL1), offset 0x064

## Register 24: ADC Sample Sequence Control 2 (ADCSSCTL2), offset 0x084

These registers contain the configuration information for each sample for a sequence executed with Sample Sequencer 1 or 2. When configuring a sample sequence, the END bit must be set at some point, whether it be after the first sample, last sample, or any sample in between. These registers are 16-bits wide and contain information for four possible samples. See the **ADCSSCTL0** register on page 482 for detailed bit descriptions. The **ADCSSCTL1** register configures Sample Sequencer 1 and the **ADCSSCTL2** register configures Sample Sequencer 2.

ADC Sample Sequence Control 1 (ADCSSCTL1)

Base 0x4003.8000
Offset 0x064
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15 | TS3 | R/W | 0 | 4th Sample Temp Sensor Select<br>Same definition as TS7 but used during the fourth sample. |
| 14 | IE3 | R/W | 0 | 4th Sample Interrupt Enable<br>Same definition as IE7 but used during the fourth sample. |
| 13 | END3 | R/W | 0 | 4th Sample is End of Sequence<br>Same definition as END7 but used during the fourth sample. |
| 12 | D3 | R/W | 0 | 4th Sample Diff Input Select<br>Same definition as D7 but used during the fourth sample. |
| 11 | TS2 | R/W | 0 | 3rd Sample Temp Sensor Select<br>Same definition as TS7 but used during the third sample. |
| 10 | IE2 | R/W | 0 | 3rd Sample Interrupt Enable<br>Same definition as IE7 but used during the third sample. |
| 9 | END2 | R/W | 0 | 3rd Sample is End of Sequence<br>Same definition as END7 but used during the third sample. |
| 8 | D2 | R/W | 0 | 3rd Sample Diff Input Select<br>Same definition as D7 but used during the third sample. |
| 7 | TS1 | R/W | 0 | 2nd Sample Temp Sensor Select<br>Same definition as TS7 but used during the second sample. |

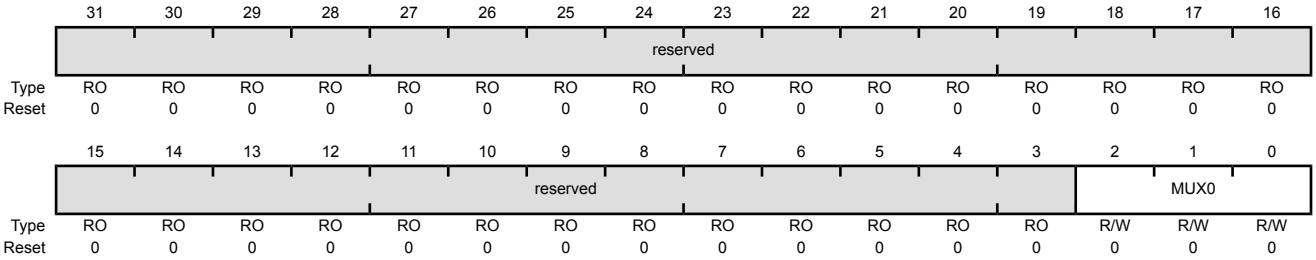
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | IE1 | R/W | 0 | 2nd Sample Interrupt Enable<br>Same definition as IE7 but used during the second sample. |
| 5 | END1 | R/W | 0 | 2nd Sample is End of Sequence<br>Same definition as END7 but used during the second sample. |
| 4 | D1 | R/W | 0 | 2nd Sample Diff Input Select<br>Same definition as D7 but used during the second sample. |
| 3 | TS0 | R/W | 0 | 1st Sample Temp Sensor Select<br>Same definition as TS7 but used during the first sample. |
| 2 | IE0 | R/W | 0 | 1st Sample Interrupt Enable<br>Same definition as IE7 but used during the first sample. |
| 1 | END0 | R/W | 0 | 1st Sample is End of Sequence<br>Same definition as END7 but used during the first sample. |
| 0 | D0 | R/W | 0 | 1st Sample Diff Input Select<br>Same definition as D7 but used during the first sample. |

## Register 25: ADC Sample Sequence Input Multiplexer Select 3 (ADCSSMUX3), offset 0x0A0

This register defines the analog input configuration for a sample executed with Sample Sequencer 3. This register is 4-bits wide and contains information for one possible sample. See the **ADCSSMUX0** register on page 480 for detailed bit descriptions.

ADC Sample Sequence Input Multiplexer Select 3 (ADCSSMUX3)

Base 0x4003.8000
Offset 0x0A0
Type R/W, reset 0x0000.0000



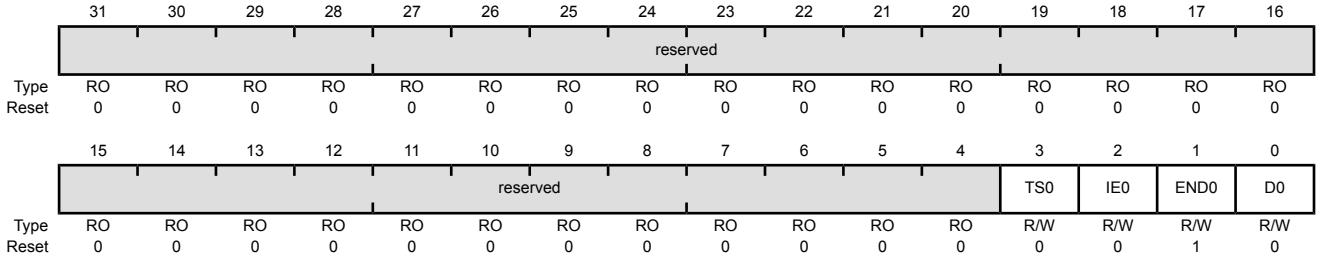

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | MUX0 | R/W | 0 | 1st Sample Input Select |

## Register 26: ADC Sample Sequence Control 3 (ADCSSCTL3), offset 0x0A4

This register contains the configuration information for a sample executed with Sample Sequencer 3. The END bit is always set since there is only one sample in this sequencer. This register is 4-bits wide and contains information for one possible sample. See the **ADCSSCTL0** register on page 482 for detailed bit descriptions.

ADC Sample Sequence Control 3 (ADCSSCTL3)

Base 0x4003.8000
Offset 0x0A4
Type R/W, reset 0x0000.0002




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TS0 | R/W | 0 | 1st Sample Temp Sensor Select<br>Same definition as TS7 but used during the first sample. |
| 2 | IE0 | R/W | 0 | 1st Sample Interrupt Enable<br>Same definition as IE7 but used during the first sample. |
| 1 | END0 | R/W | 1 | 1st Sample is End of Sequence<br>Same definition as END7 but used during the first sample.<br>Since this sequencer has only one entry, this bit must be set. |
| 0 | D0 | R/W | 0 | 1st Sample Diff Input Select<br>Same definition as D7 but used during the first sample. |

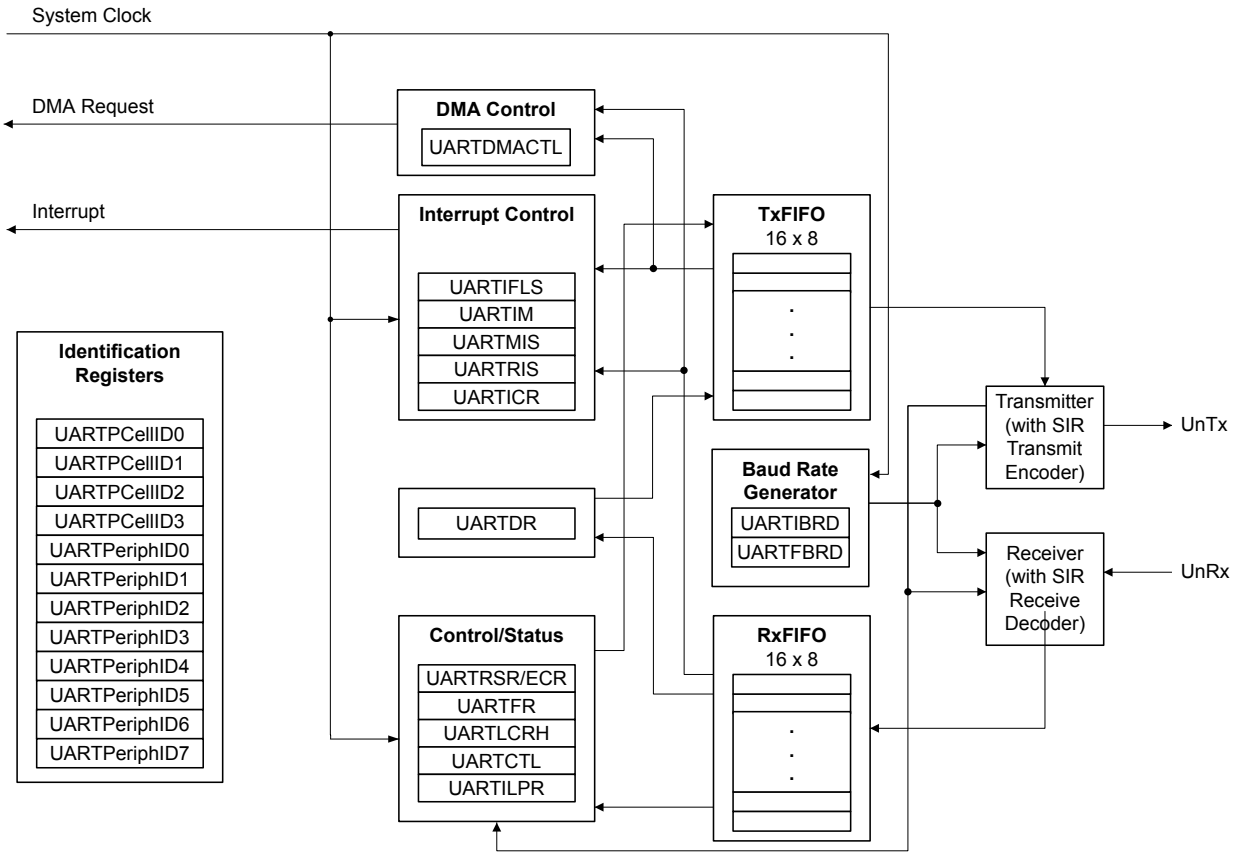
# 13 Universal Asynchronous Receivers/Transmitters (UARTs)

Each Stellaris® Universal Asynchronous Receiver/Transmitter (UART) has the following features:

- Two fully programmable 16C550-type UARTs with IrDA support
- Separate 16x8 transmit (TX) and receive (RX) FIFOs to reduce CPU interrupt service loading
- Programmable baud-rate generator allowing speeds up to 3.125 Mbps
- Programmable FIFO length, including 1-byte deep operation providing conventional double-buffered interface
- FIFO trigger levels of 1/8, 1/4, 1/2, 3/4, and 7/8
- Standard asynchronous communication bits for start, stop, and parity
- Line-break generation and detection
- Fully programmable serial interface characteristics
  - 5, 6, 7, or 8 data bits
  - Even, odd, stick, or no-parity bit generation/detection
  - 1 or 2 stop bit generation
- IrDA serial-IR (SIR) encoder/decoder providing
  - Programmable use of IrDA Serial Infrared (SIR) or UART input/output
  - Support of IrDA SIR encoder/decoder functions for data rates up to 115.2 Kbps half-duplex
  - Support of normal 3/16 and low-power (1.41-2.23 µs) bit durations
  - Programmable internal clock generator enabling division of reference clock by 1 to 256 for low-power mode bit duration
- Dedicated Direct Memory Access (DMA) transmit and receive channels

## 13.1 Block Diagram

**Figure 13-1. UART Module Block Diagram**




## 13.2 Signal Description

Table 13-1 on page 493 lists the external signals of the UART module and describes the function of each. The UART signals are alternate functions for some GPIO signals and default to be GPIO signals at reset, with the exception of the U0Rx and U0Tx pins which default to the UART function. The column in the table below titled "Pin Assignment" lists the possible GPIO pin placements for these UART signals. The AFSEL bit in the **GPIO Alternate Function Select (GPIOAFSEL)** register (page 371) should be set to choose the UART function. For more information on configuring GPIOs, see “General-Purpose Input/Outputs (GPIOs)” on page 351.

**Table 13-1. UART Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| U0Rx | 17 | I | TTL | UART module 0 receive. When in IrDA mode, this signal has IrDA modulation. |
| U0Tx | 18 | O | TTL | UART module 0 transmit. When in IrDA mode, this signal has IrDA modulation. |
| U1Rx | 41 | I | TTL | UART module 1 receive. When in IrDA mode, this signal has IrDA modulation. |

**Table 13-1. UART Signals (64LQFP) *(continued)***

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| U1Tx | 42 | O | TTL | UART module 1 transmit. When in IrDA mode, this signal has IrDA modulation. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 13.3 Functional Description

Each Stellaris UART performs the functions of parallel-to-serial and serial-to-parallel conversions. It is similar in functionality to a 16C550 UART, but is not register compatible.

The UART is configured for transmit and/or receive via the TXE and RXE bits of the **UART Control (UARTCTL)** register (see page 513). Transmit and receive are both enabled out of reset. Before any control registers are programmed, the UART must be disabled by clearing the UARTEN bit in **UARTCTL**. If the UART is disabled during a TX or RX operation, the current transaction is completed prior to the UART stopping.

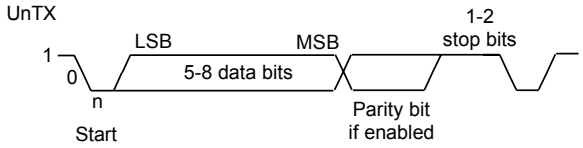
The UART peripheral also includes a serial IR (SIR) encoder/decoder block that can be connected to an infrared transceiver to implement an IrDA SIR physical layer. The SIR function is programmed using the UARTCTL register.

### 13.3.1 Transmit/Receive Logic

The transmit logic performs parallel-to-serial conversion on the data read from the transmit FIFO. The control logic outputs the serial bit stream beginning with a start bit, and followed by the data bits (LSB first), parity bit, and the stop bits according to the programmed configuration in the control registers. See Figure 13-2 on page 494 for details.

The receive logic performs serial-to-parallel conversion on the received bit stream after a valid start pulse has been detected. Overrun, parity, frame error checking, and line-break detection are also performed, and their status accompanies the data that is written to the receive FIFO.

**Figure 13-2. UART Character Frame**




### 13.3.2 Baud-Rate Generation

The baud-rate divisor is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part. The number formed by these two values is used by the baud-rate generator to determine the bit period. Having a fractional baud-rate divider allows the UART to generate all the standard baud rates.

The 16-bit integer is loaded through the **UART Integer Baud-Rate Divisor (UARTIBRD)** register (see page 509) and the 6-bit fractional part is loaded with the **UART Fractional Baud-Rate Divisor (UARTFBRD)** register (see page 510). The baud-rate divisor (BRD) has the following relationship to the system clock (where *BRDI* is the integer part of the BRD and *BRDF* is the fractional part, separated by a decimal place.)

```
BRD = BRDI + BRDF = UARTSysClk / (16 * Baud Rate)
```

where UARTSysClk is the system clock connected to the UART.

The 6-bit fractional number (that is to be loaded into the DIVFRAC bit field in the **UARTFBRD** register) can be calculated by taking the fractional part of the baud-rate divisor, multiplying it by 64, and adding 0.5 to account for rounding errors:

```
UARTFBRD[DIVFRAC] = integer(BRDF * 64 + 0.5)
```

The UART generates an internal baud-rate reference clock at 16x the baud-rate (referred to as Baud16). This reference clock is divided by 16 to generate the transmit clock, and is used for error detection during receive operations.

Along with the **UART Line Control, High Byte (UARTLCRH)** register (see page 511), the **UARTIBRD** and **UARTFBRD** registers form an internal 30-bit register. This internal register is only updated when a write operation to **UARTLCRH** is performed, so any changes to the baud-rate divisor must be followed by a write to the **UARTLCRH** register for the changes to take effect.

To update the baud-rate registers, there are four possible sequences:

- **UARTIBRD** write, **UARTFBRD** write, and **UARTLCRH** write
- **UARTFBRD** write, **UARTIBRD** write, and **UARTLCRH** write
- **UARTIBRD** write and **UARTLCRH** write
- **UARTFBRD** write and **UARTLCRH** write

### 13.3.3 Data Transmission

Data received or transmitted is stored in two 16-byte FIFOs, though the receive FIFO has an extra four bits per character for status information. For transmission, data is written into the transmit FIFO. If the UART is enabled, it causes a data frame to start transmitting with the parameters indicated in the **UARTLCRH** register. Data continues to be transmitted until there is no data left in the transmit FIFO. The BUSY bit in the **UART Flag (UARTFR)** register (see page 506) is asserted as soon as data is written to the transmit FIFO (that is, if the FIFO is non-empty) and remains asserted while data is being transmitted. The BUSY bit is negated only when the transmit FIFO is empty, and the last character has been transmitted from the shift register, including the stop bits. The UART can indicate that it is busy even though the UART may no longer be enabled.

When the receiver is idle (the UnRx is continuously 1) and the data input goes Low (a start bit has been received), the receive counter begins running and data is sampled on the eighth cycle of Baud16 (described in “Transmit/Receive Logic” on page 494).

The start bit is valid and recognized if UnRx is still low on the eighth cycle of Baud16, otherwise it is ignored. After a valid start bit is detected, successive data bits are sampled on every 16th cycle of Baud16 (that is, one bit period later) according to the programmed length of the data characters. The parity bit is then checked if parity mode was enabled. Data length and parity are defined in the **UARTLCRH** register.

Lastly, a valid stop bit is confirmed if UnRx is High, otherwise a framing error has occurred. When a full word is received, the data is stored in the receive FIFO, with any error bits associated with that word.

### 13.3.4 Serial IR (SIR)

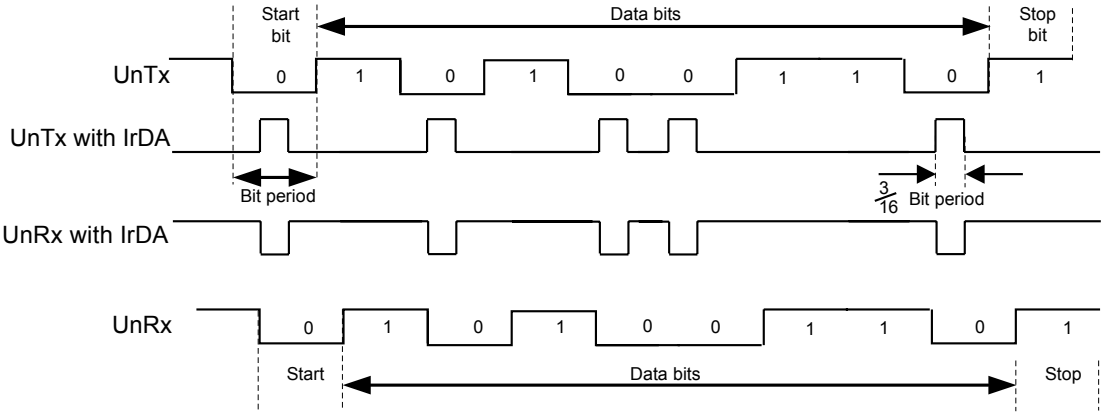
The UART peripheral includes an IrDA serial-IR (SIR) encoder/decoder block. The IrDA SIR block provides functionality that converts between an asynchronous UART data stream, and half-duplex serial SIR interface. No analog processing is performed on-chip. The role of the SIR block is to provide a digital encoded output and decoded input to the UART. The UART signal pins can be

connected to an infrared transceiver to implement an IrDA SIR physical layer link. The SIR block has two modes of operation:

- In normal IrDA mode, a zero logic level is transmitted as high pulse of 3/16th duration of the selected baud rate bit period on the output pin, while logic one levels are transmitted as a static LOW signal. These levels control the driver of an infrared transmitter, sending a pulse of light for each zero. On the reception side, the incoming light pulses energize the photo transistor base of the receiver, pulling its output LOW. This drives the UART input pin LOW.
- In low-power IrDA mode, the width of the transmitted infrared pulse is set to three times the period of the internally generated IrLPBaud16 signal (1.63 µs, assuming a nominal 1.8432 MHz frequency) by changing the appropriate bit in the **UARTCR** register. See page 508 for more information on IrDA low-power pulse-duration configuration.

Figure 13-3 on page 496 shows the UART transmit and receive signals, with and without IrDA modulation.

**Figure 13-3. IrDA Data Modulation**




In both normal and low-power IrDA modes:

- During transmission, the UART data bit is used as the base for encoding
- During reception, the decoded bits are transferred to the UART receive logic

The IrDA SIR physical layer specifies a half-duplex communication link, with a minimum 10 ms delay between transmission and reception. This delay must be generated by software because it is not automatically supported by the UART. The delay is required because the infrared receiver electronics might become biased, or even saturated from the optical power coupled from the adjacent transmitter LED. This delay is known as latency, or receiver setup time.

If the application does not require the use of the UnRx signal, the GPIO pin that has the UnRx signal as an alternate function must be configured as the UnRx signal and pulled High.

### 13.3.5 FIFO Operation

The UART has two 16-entry FIFOs; one for transmit and one for receive. Both FIFOs are accessed via the **UART Data (UARTDR)** register (see page 502). Read operations of the **UARTDR** register return a 12-bit value consisting of 8 data bits and 4 error flags while write operations place 8-bit data in the transmit FIFO.

Out of reset, both FIFOs are disabled and act as 1-byte-deep holding registers. The FIFOs are enabled by setting the FEN bit in **UARTLCRH** (page 511).

FIFO status can be monitored via the **UART Flag (UARTFR)** register (see page 506) and the **UART Receive Status (UARTRSR)** register. Hardware monitors empty, full and overrun conditions. The **UARTFR** register contains empty and full flags (TXFE, TXFF, RXFE, and RXFF bits) and the **UARTRSR** register shows overrun status via the OE bit.

The trigger points at which the FIFOs generate interrupts is controlled via the **UART Interrupt FIFO Level Select (UARTIFLS)** register (see page 515). Both FIFOs can be individually configured to trigger interrupts at different levels. Available configurations include 1/8, ¼, ½, ¾, and 7/8. For example, if the ¼ option is selected for the receive FIFO, the UART generates a receive interrupt after 4 data bytes are received. Out of reset, both FIFOs are configured to trigger an interrupt at the ½ mark.

### 13.3.6 Interrupts

The UART can generate interrupts when the following conditions are observed:

- Overrun Error
- Break Error
- Parity Error
- Framing Error
- Receive Timeout
- Transmit (when condition defined in the TXIFLSEL bit in the **UARTIFLS** register is met)
- Receive (when condition defined in the RXIFLSEL bit in the **UARTIFLS** register is met)

All of the interrupt events are ORed together before being sent to the interrupt controller, so the UART can only generate a single interrupt request to the controller at any given time. Software can service multiple interrupt events in a single interrupt service routine by reading the **UART Masked Interrupt Status (UARTMIS)** register (see page 520).

The interrupt events that can trigger a controller-level interrupt are defined in the **UART Interrupt Mask (UARTIM** ) register (see page 517) by setting the corresponding IM bit to 1. If interrupts are not used, the raw interrupt status is always visible via the **UART Raw Interrupt Status (UARTRIS)** register (see page 519).

Interrupts are always cleared (for both the **UARTMIS** and **UARTRIS** registers) by setting the corresponding bit in the **UART Interrupt Clear (UARTICR)** register (see page 521).

The receive interrupt changes state when one of the following events occurs:

- If the FIFOs are enabled and the receive FIFO reaches the programmed trigger level, the RXRIS bit is set. The receive interrupt is cleared by reading data from the receive FIFO until it becomes less than the trigger level, or by clearing the interrupt by writing a 1 to the RXIC bit.
- If the FIFOs are disabled (have a depth of one location) and data is received thereby filling the location, the RXRIS bit is set. The receive interrupt is cleared by performing a single read of the receive FIFO, or by clearing the interrupt by writing a 1 to the RXIC bit.

The transmit interrupt changes state when one of the following events occurs:

- If the FIFOs are enabled and the transmit FIFO progresses through the programmed trigger level, the TXRIS bit is set. The transmit interrupt is based on a transition through level, therefore the FIFO must be written past the programmed trigger level otherwise no further transmit interrupts will be generated. The transmit interrupt is cleared by writing data to the transmit FIFO until it becomes greater than the trigger level, or by clearing the interrupt by writing a 1 to the TXIC bit.
- If the FIFOs are disabled (have a depth of one location) and there is no data present in the transmitters single location, the TXRIS bit is set. It is cleared by performing a single write to the transmit FIFO, or by clearing the interrupt by writing a 1 to the TXIC bit.

### 13.3.7 Loopback Operation

The UART can be placed into an internal loopback mode for diagnostic or debug work. This is accomplished by setting the LBE bit in the **UARTCTL** register (see page 513). In loopback mode, data transmitted on UnTx is received on the UnRx input.

### 13.3.8 DMA Operation

The UART provides an interface connected to the µDMA controller. The DMA operation of the UART is enabled through the **UART DMA Control (UARTDMACTL)** register. When DMA operation is enabled, the UART will assert a DMA request on the receive or transmit channel when the associated FIFO can transfer data. For the receive channel, a single transfer request is asserted whenever there is any data in the receive FIFO. A burst transfer request is asserted whenever the amount of data in the receive FIFO is at or above the FIFO trigger level. For the transmit channel, a single transfer request is asserted whenever there is at least one empty location in the transmit FIFO. The burst request is asserted whenever the transmit FIFO contains fewer characters than the FIFO trigger level. The single and burst DMA transfer requests are handled automatically by the µDMA controller depending how the DMA channel is configured.

To enable DMA operation for the receive channel, the RXDMAE bit of the **DMA Control (UARTDMACTL)** register should be set. To enable DMA operation for the transmit channel, the TXDMAE bit of **UARTDMACTL** should be set. The UART can also be configured to stop using DMA for the receive channel if a receive error occurs. If the DMAERR bit of **UARTDMACR** is set, then when a receive error occurs, the DMA receive requests will be automatically disabled. This error condition can be cleared by clearing the UART error interrupt.

If DMA is enabled, then the µDMA controller will trigger an interrupt when a transfer is complete. The interrupt will occur on the UART interrupt vector. Therefore, if interrupts are used for UART operation and DMA is enabled, the UART interrupt handler must be designed to handle the µDMA completion interrupt.

See “Micro Direct Memory Access (µDMA)” on page 290 for more details about programming the µDMA controller.

### 13.3.9 IrDA SIR block

The IrDA SIR block contains an IrDA serial IR (SIR) protocol encoder/decoder. When enabled, the SIR block uses the UnTx and UnRx pins for the SIR protocol, which should be connected to an IR transceiver.

The SIR block can receive and transmit, but it is only half-duplex so it cannot do both at the same time. Transmission must be stopped before data can be received. The IrDA SIR physical layer specifies a minimum 10-ms delay between transmission and reception.

## 13.4 Initialization and Configuration

To use the UARTs, the peripheral clock must be enabled by setting the UART0 or UART1 bits in the **RCGC1** register.

This section discusses the steps that are required to use a UART module. For this example, the UART clock is assumed to be 20 MHz and the desired UART configuration is:

- 115200 baud rate
- Data length of 8 bits
- One stop bit
- No parity
- FIFOs disabled
- No interrupts

The first thing to consider when programming the UART is the baud-rate divisor (BRD), since the **UARTIBRD** and **UARTFBRD** registers must be written before the **UARTLCRH** register. Using the equation described in "Baud-Rate Generation" on page 494, the BRD can be calculated:

```
BRD = 20,000,000 / (16 * 115,200) = 10.8507
```

which means that the DIVINT field of the **UARTIBRD** register (see page 509) should be set to 10. The value to be loaded into the **UARTFBRD** register (see page 510) is calculated by the equation:

```
UARTFBRD[DIVFRAC] = integer(0.8507 * 64 + 0.5) = 54
```

With the BRD values in hand, the UART configuration is written to the module in the following order:

1. Disable the UART by clearing the UARTEN bit in the **UARTCTL** register.
2. Write the integer portion of the BRD to the **UARTIBRD** register.
3. Write the fractional portion of the BRD to the **UARTFBRD** register.
4. Write the desired serial parameters to the **UARTLCRH** register (in this case, a value of 0x0000.0060).
5. Optionally, configure the uDMA channel (see "Micro Direct Memory Access (µDMA)" on page 290) and enable the DMA option(s) in the **UARTDMACTL** register.
6. Enable the UART by setting the UARTEN bit in the **UARTCTL** register.

## 13.5 Register Map

Table 13-2 on page 500 lists the UART registers. The offset listed is a hexadecimal increment to the register's address, relative to that UART's base address:

- UART0: 0x4000.C000
- UART1: 0x4000.D000

Note that the UART module clock must be enabled before the registers can be programmed (see page 223). There must be a delay of 3 system clocks after the UART module clock is enabled before any UART module registers are accessed.

**Note:** The UART must be disabled (see the UARTEN bit in the **UARTCTL** register on page 513) before any of the control registers are reprogrammed. When the UART is disabled during a TX or RX operation, the current transaction is completed prior to the UART stopping.

**Table 13-2. UART Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | UARTDR | R/W | 0x0000.0000 | UART Data | 502 |
| 0x004 | UARTRSR/UARTECR | R/W | 0x0000.0000 | UART Receive Status/Error Clear | 504 |
| 0x018 | UARTFR | RO | 0x0000.0090 | UART Flag | 506 |
| 0x020 | UARTILPR | R/W | 0x0000.0000 | UART IrDA Low-Power Register | 508 |
| 0x024 | UARTIBRD | R/W | 0x0000.0000 | UART Integer Baud-Rate Divisor | 509 |
| 0x028 | UARTFBRD | R/W | 0x0000.0000 | UART Fractional Baud-Rate Divisor | 510 |
| 0x02C | UARTLCRH | R/W | 0x0000.0000 | UART Line Control | 511 |
| 0x030 | UARTCTL | R/W | 0x0000.0300 | UART Control | 513 |
| 0x034 | UARTIFLS | R/W | 0x0000.0012 | UART Interrupt FIFO Level Select | 515 |
| 0x038 | UARTIM | R/W | 0x0000.0000 | UART Interrupt Mask | 517 |
| 0x03C | UARTRIS | RO | 0x0000.0000 | UART Raw Interrupt Status | 519 |
| 0x040 | UARTMIS | RO | 0x0000.0000 | UART Masked Interrupt Status | 520 |
| 0x044 | UARTICR | W1C | 0x0000.0000 | UART Interrupt Clear | 521 |
| 0x048 | UARTDMACTL | R/W | 0x0000.0000 | UART DMA Control | 523 |
| 0xFD0 | UARTPeriphID4 | RO | 0x0000.0000 | UART Peripheral Identification 4 | 524 |
| 0xFD4 | UARTPeriphID5 | RO | 0x0000.0000 | UART Peripheral Identification 5 | 525 |
| 0xFD8 | UARTPeriphID6 | RO | 0x0000.0000 | UART Peripheral Identification 6 | 526 |
| 0xFDC | UARTPeriphID7 | RO | 0x0000.0000 | UART Peripheral Identification 7 | 527 |
| 0xFE0 | UARTPeriphID0 | RO | 0x0000.0011 | UART Peripheral Identification 0 | 528 |
| 0xFE4 | UARTPeriphID1 | RO | 0x0000.0000 | UART Peripheral Identification 1 | 529 |
| 0xFE8 | UARTPeriphID2 | RO | 0x0000.0018 | UART Peripheral Identification 2 | 530 |
| 0xFEC | UARTPeriphID3 | RO | 0x0000.0001 | UART Peripheral Identification 3 | 531 |
| 0xFF0 | UARTPCellID0 | RO | 0x0000.000D | UART PrimeCell Identification 0 | 532 |
| 0xFF4 | UARTPCellID1 | RO | 0x0000.00F0 | UART PrimeCell Identification 1 | 533 |
| 0xFF8 | UARTPCellID2 | RO | 0x0000.0005 | UART PrimeCell Identification 2 | 534 |
| 0xFFC | UARTPCellID3 | RO | 0x0000.00B1 | UART PrimeCell Identification 3 | 535 |

## 13.6 Register Descriptions

The remainder of this section lists and describes the UART registers, in numerical order by address offset.

## Register 1: UART Data (UARTDR), offset 0x000

**Important:** This register is read-sensitive. See the register description for details.

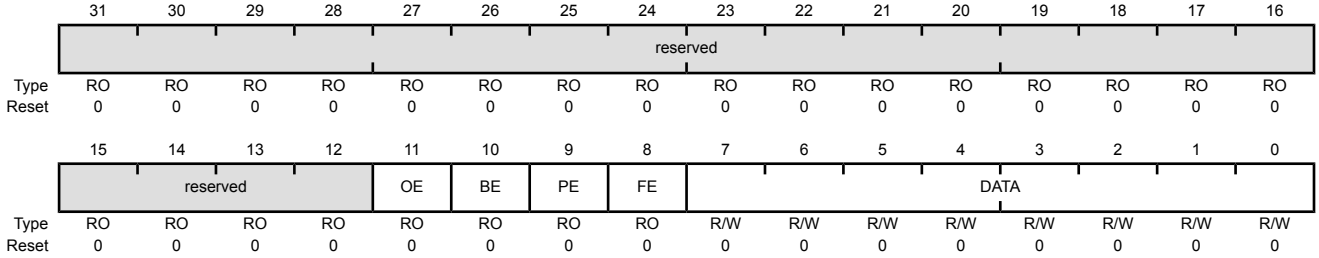
This register is the data register (the interface to the FIFOs).

When FIFOs are enabled, data written to this location is pushed onto the transmit FIFO. If FIFOs are disabled, data is stored in the transmitter holding register (the bottom word of the transmit FIFO). A write to this register initiates a transmission from the UART.

For received data, if the FIFO is enabled, the data byte and the 4-bit status (break, frame, parity, and overrun) is pushed onto the 12-bit wide receive FIFO. If FIFOs are disabled, the data byte and status are stored in the receiving holding register (the bottom word of the receive FIFO). The received data can be retrieved by reading this register.

UART Data (UARTDR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x000
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:12 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 11 | OE | RO | 0 | UART Overrun Error<br>The OE values are defined as follows:<br>Value Description<br>0 There has been no data loss due to a FIFO overrun.<br>1 New data was received when the FIFO was full, resulting in data loss. |
| 10 | BE | RO | 0 | UART Break Error<br>This bit is set to 1 when a break condition is detected, indicating that the receive data input was held Low for longer than a full-word transmission time (defined as start, data, parity, and stop bits).<br>In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the received data input goes to a 1 (marking state) and the next valid start bit is received. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 9 | PE | RO | 0 | UART Parity Error<br>This bit is set to 1 when the parity of the received data character does not match the parity defined by bits 2 and 7 of the **UARTLCRH** register.<br>In FIFO mode, this error is associated with the character at the top of the FIFO. |
| 8 | FE | RO | 0 | UART Framing Error<br>This bit is set to 1 when the received character does not have a valid stop bit (a valid stop bit is 1). |
| 7:0 | DATA | R/W | 0 | Data Transmitted or Received<br>When written, the data that is to be transmitted via the UART. When read, the data that was received by the UART. |

## Register 2: UART Receive Status/Error Clear (UARTRSR/UARTECR), offset 0x004

The **UARTRSR/UARTECR** register is the receive status register/error clear register.

In addition to the **UARTDR** register, receive status can also be read from the **UARTRSR** register. If the status is read from this register, then the status information corresponds to the entry read from **UARTDR** prior to reading **UARTRSR**. The status information for overrun is set immediately when an overrun condition occurs.

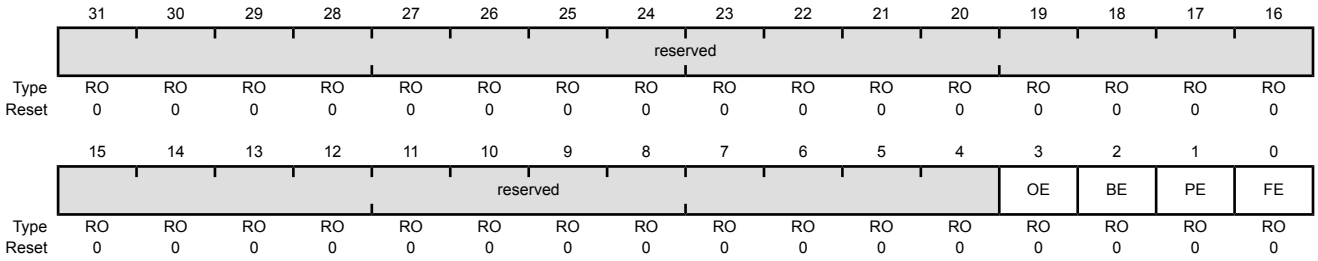
The **UARTRSR** register cannot be written.

A write of any value to the **UARTECR** register clears the framing, parity, break, and overrun errors. All the bits are cleared to 0 on reset.

### Reads

UART Receive Status/Error Clear (UARTRSR/UARTECR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x004
Type RO, reset 0x0000.0000



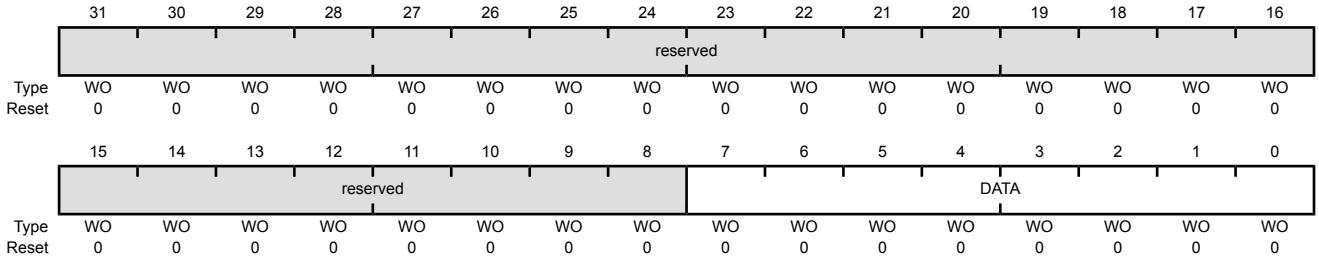

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | OE | RO | 0 | UART Overrun Error<br>When this bit is set to 1, data is received and the FIFO is already full. This bit is cleared to 0 by a write to **UARTECR**.<br>The FIFO contents remain valid since no further data is written when the FIFO is full, only the contents of the shift register are overwritten. The CPU must now read the data in order to empty the FIFO. |
| 2 | BE | RO | 0 | UART Break Error<br>This bit is set to 1 when a break condition is detected, indicating that the received data input was held Low for longer than a full-word transmission time (defined as start, data, parity, and stop bits).<br>This bit is cleared to 0 by a write to **UARTECR**.<br>In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state) and the next valid start bit is received. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | PE | RO | 0 | UART Parity Error<br>This bit is set to 1 when the parity of the received data character does not match the parity defined by bits 2 and 7 of the **UARTLCRH** register.<br>This bit is cleared to 0 by a write to **UARTECR**. |
| 0 | FE | RO | 0 | UART Framing Error<br>This bit is set to 1 when the received character does not have a valid stop bit (a valid stop bit is 1).<br>This bit is cleared to 0 by a write to **UARTECR**.<br>In FIFO mode, this error is associated with the character at the top of the FIFO. |

## Writes

UART Receive Status/Error Clear (UARTRSR/UARTECR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x004
Type WO, reset 0x0000.0000

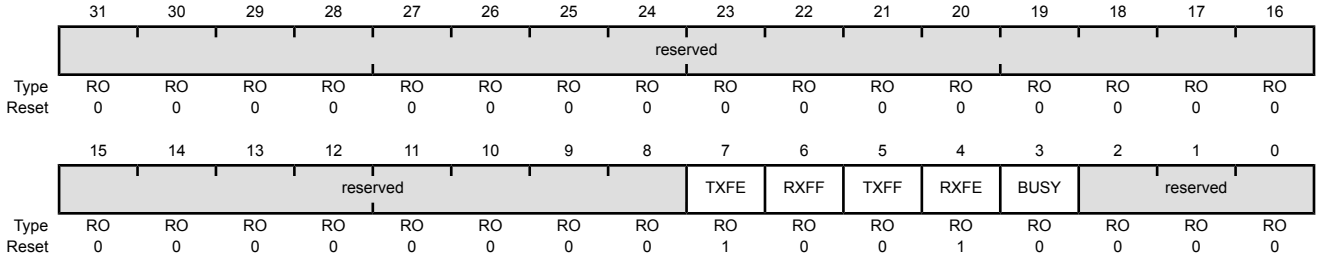



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | WO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DATA | WO | 0 | Error Clear<br>A write to this register of any data clears the framing, parity, break, and overrun flags. |

## Register 3: UART Flag (UARTFR), offset 0x018

The **UARTFR** register is the flag register. After reset, the TXFF, RXFF, and BUSY bits are 0, and TXFE and RXFE bits are 1.

UART Flag (UARTFR)
UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x018
Type RO, reset 0x0000.0090




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7 | TXFE | RO | 1 | UART Transmit FIFO Empty<br>The meaning of this bit depends on the state of the FEN bit in the **UARTLCRH** register.<br>If the FIFO is disabled (FEN is 0), this bit is set when the transmit holding register is empty.<br>If the FIFO is enabled (FEN is 1), this bit is set when the transmit FIFO is empty. |
| 6 | RXFF | RO | 0 | UART Receive FIFO Full<br>The meaning of this bit depends on the state of the FEN bit in the **UARTLCRH** register.<br>If the FIFO is disabled, this bit is set when the receive holding register is full.<br>If the FIFO is enabled, this bit is set when the receive FIFO is full. |
| 5 | TXFF | RO | 0 | UART Transmit FIFO Full<br>The meaning of this bit depends on the state of the FEN bit in the **UARTLCRH** register.<br>If the FIFO is disabled, this bit is set when the transmit holding register is full.<br>If the FIFO is enabled, this bit is set when the transmit FIFO is full. |
| 4 | RXFE | RO | 1 | UART Receive FIFO Empty<br>The meaning of this bit depends on the state of the FEN bit in the **UARTLCRH** register.<br>If the FIFO is disabled, this bit is set when the receive holding register is empty.<br>If the FIFO is enabled, this bit is set when the receive FIFO is empty. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | BUSY | RO | 0 | UART Busy<br>When this bit is 1, the UART is busy transmitting data. This bit remains set until the complete byte, including all stop bits, has been sent from the shift register.<br>This bit is set as soon as the transmit FIFO becomes non-empty (regardless of whether UART is enabled). |
| 2:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 4: UART IrDA Low-Power Register (UARTILPR), offset 0x020

The **UARTILPR** register is an 8-bit read/write register that stores the low-power counter divisor value used to derive the low-power SIR pulse width clock by dividing down the system clock (SysClk). All the bits are cleared to 0 when reset.

The internal `IrLPBaud16` clock is generated by dividing down SysClk according to the low-power divisor value written to **UARTILPR**. The duration of SIR pulses generated when low-power mode is enabled is three times the period of the `IrLPBaud16` clock. The low-power divisor value is calculated as follows:

$$ILPDVSR = SysClk / F_{IrLPBaud16}$$

where $F_{IrLPBaud16}$ is nominally 1.8432 MHz.

You must choose the divisor so that 1.42 MHz < $F_{IrLPBaud16}$ < 2.12 MHz, which results in a low-power pulse duration of 1.41–2.11 µs (three times the period of `IrLPBaud16`). The minimum frequency of `IrLPBaud16` ensures that pulses less than one period of `IrLPBaud16` are rejected, but that pulses greater than 1.4 µs are accepted as valid pulses.

**Note:** Zero is an illegal value. Programming a zero value results in no `IrLPBaud16` pulses being generated.

UART IrDA Low-Power Register (UARTILPR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x020
Type R/W, reset 0x0000.0000

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | ILPDVSR | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

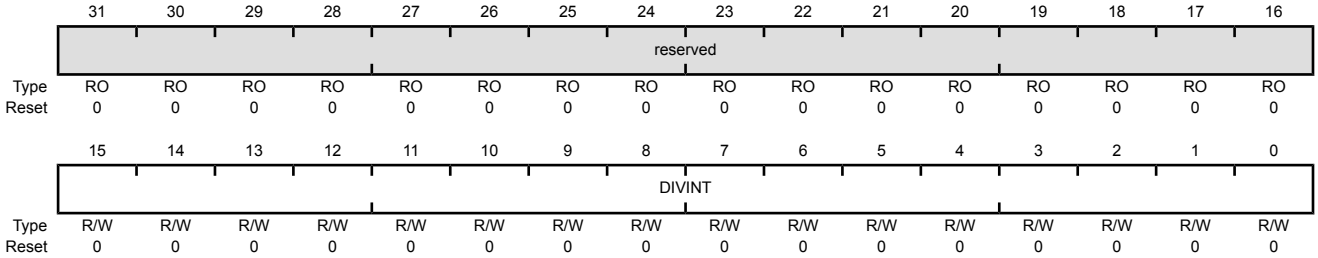
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | ILPDVSR | R/W | 0x00 | IrDA Low-Power Divisor<br>This is an 8-bit low-power divisor value. |

## Register 5: UART Integer Baud-Rate Divisor (UARTIBRD), offset 0x024

The **UARTIBRD** register is the integer part of the baud-rate divisor value. All the bits are cleared on reset. The minimum possible divide ratio is 1 (when **UARTIBRD**=0), in which case the **UARTFBRD** register is ignored. When changing the **UARTIBRD** register, the new value does not take effect until transmission/reception of the current character is complete. Any changes to the baud-rate divisor must be followed by a write to the **UARTLCRH** register. See “Baud-Rate Generation” on page 494 for configuration details.

UART Integer Baud-Rate Divisor (UARTIBRD)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x024
Type R/W, reset 0x0000.0000



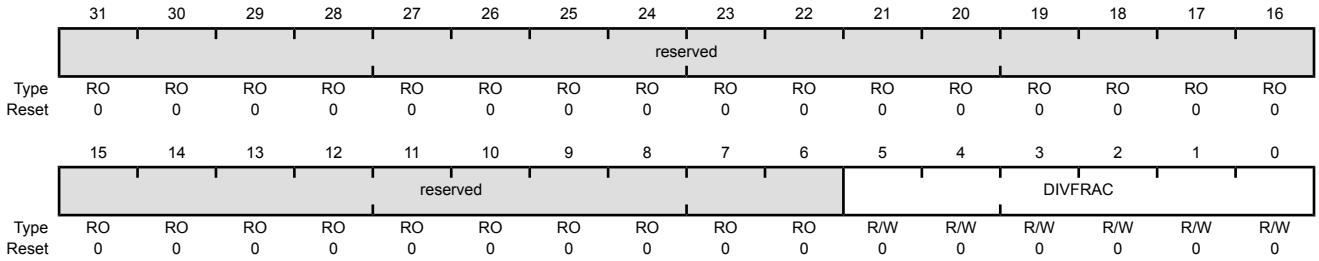

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | DIVINT | R/W | 0x0000 | Integer Baud-Rate Divisor |

## Register 6: UART Fractional Baud-Rate Divisor (UARTFBRD), offset 0x028

The **UARTFBRD** register is the fractional part of the baud-rate divisor value. All the bits are cleared on reset. When changing the **UARTFBRD** register, the new value does not take effect until transmission/reception of the current character is complete. Any changes to the baud-rate divisor must be followed by a write to the **UARTLCRH** register. See “Baud-Rate Generation” on page 494 for configuration details.

UART Fractional Baud-Rate Divisor (UARTFBRD)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x028
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:0 | DIVFRAC | R/W | 0x000 | Fractional Baud-Rate Divisor |

## Register 7: UART Line Control (UARTLCRH), offset 0x02C

The **UARTLCRH** register is the line control register. Serial parameters such as data length, parity, and stop bit selection are implemented in this register.

When updating the baud-rate divisor (**UARTIBRD** and/or **UARTIFRD**), the **UARTLCRH** register must also be written. The write strobe for the baud-rate divisor registers is tied to the **UARTLCRH** register.

UART Line Control (UARTLCRH)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x02C
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7 | SPS | R/W | 0 | UART Stick Parity Select<br>When bits 1, 2, and 7 of **UARTLCRH** are set, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set and 2 is cleared, the parity bit is transmitted and checked as a 1.<br>When this bit is cleared, stick parity is disabled. |
| 6:5 | WLEN | R/W | 0 | UART Word Length<br>The bits indicate the number of data bits transmitted or received in a frame as follows:<br>Value Description<br>0x3 8 bits<br>0x2 7 bits<br>0x1 6 bits<br>0x0 5 bits (default) |
| 4 | FEN | R/W | 0 | UART Enable FIFOs<br>If this bit is set to 1, transmit and receive FIFO buffers are enabled (FIFO mode).<br>When cleared to 0, FIFOs are disabled (Character mode). The FIFOs become 1-byte-deep holding registers. |
| 3 | STP2 | R/W | 0 | UART Two Stop Bits Select<br>If this bit is set to 1, two stop bits are transmitted at the end of a frame. The receive logic does not check for two stop bits being received. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | EPS | R/W | 0 | UART Even Parity Select<br>If this bit is set to 1, even parity generation and checking is performed during transmission and reception, which checks for an even number of 1s in data and parity bits.<br>When cleared to 0, then odd parity is performed, which checks for an odd number of 1s.<br>This bit has no effect when parity is disabled by the PEN bit. |
| 1 | PEN | R/W | 0 | UART Parity Enable<br>If this bit is set to 1, parity checking and generation is enabled; otherwise, parity is disabled and no parity bit is added to the data frame. |
| 0 | BRK | R/W | 0 | UART Send Break<br>If this bit is set to 1, a Low level is continually output on the UnTX output, after completing transmission of the current character. For the proper execution of the break command, the software must set this bit for at least two frames (character periods). For normal use, this bit must be cleared to 0. |

## Register 8: UART Control (UARTCTL), offset 0x030

The **UARTCTL** register is the control register. All the bits are cleared on reset except for the Transmit Enable (TXE) and Receive Enable (RXE) bits, which are set to 1.

To enable the UART module, the UARTEN bit must be set to 1. If software requires a configuration change in the module, the UARTEN bit must be cleared before the configuration changes are written. If the UART is disabled during a transmit or receive operation, the current transaction is completed prior to the UART stopping.

**Note:** The **UARTCTL** register should not be changed while the UART is enabled or else the results are unpredictable. The following sequence is recommended for making changes to the **UARTCTL** register.

1. Disable the UART.
2. Wait for the end of transmission or reception of the current character.
3. Flush the transmit FIFO by disabling bit 4 (FEN) in the line control register (**UARTLCRH**).
4. Reprogram the control register.
5. Enable the UART.

UART Control (UARTCTL)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x030
Type R/W, reset 0x0000.0300

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | RXE | TXE | LBE | reserved | | | | SIRLP | SIREN | UARTEN |
| Type | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | RO | RO | RO | RO | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9 | RXE | R/W | 1 | UART Receive Enable<br>If this bit is set to 1, the receive section of the UART is enabled. When the UART is disabled in the middle of a receive, it completes the current character before stopping.<br>**Note:** To enable reception, the UARTEN bit must also be set. |
| 8 | TXE | R/W | 1 | UART Transmit Enable<br>If this bit is set to 1, the transmit section of the UART is enabled. When the UART is disabled in the middle of a transmission, it completes the current character before stopping.<br>**Note:** To enable transmission, the UARTEN bit must also be set. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | LBE | R/W | 0 | UART Loop Back Enable<br>If this bit is set to 1, the UnTX path is fed through the UnRX path. |
| 6:3 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | SIRLP | R/W | 0 | UART SIR Low Power Mode<br>This bit selects the IrDA encoding mode. If this bit is cleared to 0, low-level bits are transmitted as an active High pulse with a width of 3/16th of the bit period. If this bit is set to 1, low-level bits are transmitted with a pulse width which is 3 times the period of the IrLPBaud16 input signal, regardless of the selected bit rate. Setting this bit uses less power, but might reduce transmission distances. See page 508 for more information. |
| 1 | SIREN | R/W | 0 | UART SIR Enable<br>If this bit is set to 1, the IrDA SIR block is enabled, and the UART will transmit and receive data using SIR protocol. |
| 0 | UARTEN | R/W | 0 | UART Enable<br>If this bit is set to 1, the UART is enabled. When the UART is disabled in the middle of transmission or reception, it completes the current character before stopping. |

## Register 9: UART Interrupt FIFO Level Select (UARTIFLS), offset 0x034

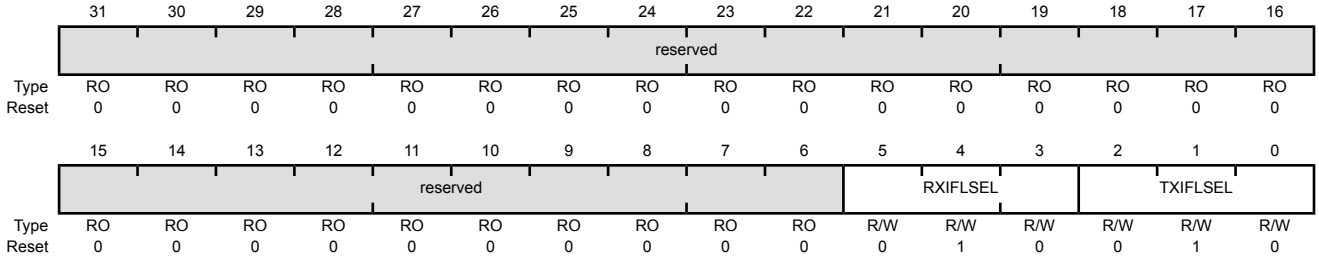
The **UARTIFLS** register is the interrupt FIFO level select register. You can use this register to define the FIFO level at which the TXRIS and RXRIS bits in the **UARTRIS** register are triggered.

The interrupts are generated based on a transition through a level rather than being based on the level. That is, the interrupts are generated when the fill level progresses through the trigger level. For example, if the receive trigger level is set to the half-way mark, the interrupt is triggered as the module is receiving the 9th character.

Out of reset, the TXIFLSEL and RXIFLSEL bits are configured so that the FIFOs trigger an interrupt at the half-way mark.

UART Interrupt FIFO Level Select (UARTIFLS)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x034
Type R/W, reset 0x0000.0012




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:3 | RXIFLSEL | R/W | 0x2 | UART Receive Interrupt FIFO Level Select<br>The trigger points for the receive interrupt are as follows:<br>Value Description<br>0x0 RX FIFO ≥ ⅛ full<br>0x1 RX FIFO ≥ ¼ full<br>0x2 RX FIFO ≥ ½ full (default)<br>0x3 RX FIFO ≥ ¾ full<br>0x4 RX FIFO ≥ ⅞ full<br>0x5-0x7 Reserved |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2:0 | TXIFLSEL | R/W | 0x2 | UART Transmit Interrupt FIFO Level Select<br>The trigger points for the transmit interrupt are as follows:<br><br>Value Description<br>0x0 TX FIFO ≤ ⅞ empty<br>0x1 TX FIFO ≤ ¾ empty<br>0x2 TX FIFO ≤ ½ empty (default)<br>0x3 TX FIFO ≤ ¼ empty<br>0x4 TX FIFO ≤ ⅛ empty<br>0x5-0x7 Reserved |

## Register 10: UART Interrupt Mask (UARTIM), offset 0x038

The **UARTIM** register is the interrupt mask set/clear register.

On a read, this register gives the current value of the mask on the relevant interrupt. Writing a 1 to a bit allows the corresponding raw interrupt signal to be routed to the interrupt controller. Writing a 0 prevents the raw interrupt signal from being sent to the interrupt controller.

UART Interrupt Mask (UARTIM)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x038
Type R/W, reset 0x0000.0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | | | | | | | | | | | |
| RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | OEIM | BEIM | PEIM | FEIM | RTIM | TXIM | RXIM | reserved | | | |
| RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W | R/W | RO | RO | RO | RO |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | OEIM | R/W | 0 | UART Overrun Error Interrupt Mask<br>On a read, the current mask for the `OEIM` interrupt is returned.<br>Setting this bit to 1 promotes the `OEIM` interrupt to the interrupt controller. |
| 9 | BEIM | R/W | 0 | UART Break Error Interrupt Mask<br>On a read, the current mask for the `BEIM` interrupt is returned.<br>Setting this bit to 1 promotes the `BEIM` interrupt to the interrupt controller. |
| 8 | PEIM | R/W | 0 | UART Parity Error Interrupt Mask<br>On a read, the current mask for the `PEIM` interrupt is returned.<br>Setting this bit to 1 promotes the `PEIM` interrupt to the interrupt controller. |
| 7 | FEIM | R/W | 0 | UART Framing Error Interrupt Mask<br>On a read, the current mask for the `FEIM` interrupt is returned.<br>Setting this bit to 1 promotes the `FEIM` interrupt to the interrupt controller. |
| 6 | RTIM | R/W | 0 | UART Receive Time-Out Interrupt Mask<br>On a read, the current mask for the `RTIM` interrupt is returned.<br>Setting this bit to 1 promotes the `RTIM` interrupt to the interrupt controller. |
| 5 | TXIM | R/W | 0 | UART Transmit Interrupt Mask<br>On a read, the current mask for the `TXIM` interrupt is returned.<br>Setting this bit to 1 promotes the `TXIM` interrupt to the interrupt controller. |
| 4 | RXIM | R/W | 0 | UART Receive Interrupt Mask<br>On a read, the current mask for the `RXIM` interrupt is returned.<br>Setting this bit to 1 promotes the `RXIM` interrupt to the interrupt controller. |

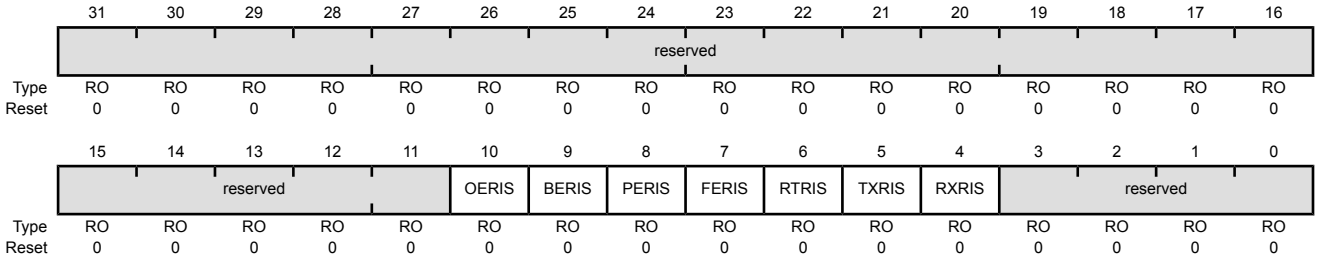
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3:0 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 11: UART Raw Interrupt Status (UARTRIS), offset 0x03C

The **UARTRIS** register is the raw interrupt status register. On a read, this register gives the current raw status value of the corresponding interrupt. A write has no effect.

UART Raw Interrupt Status (UARTRIS)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x03C
Type RO, reset 0x0000.0000



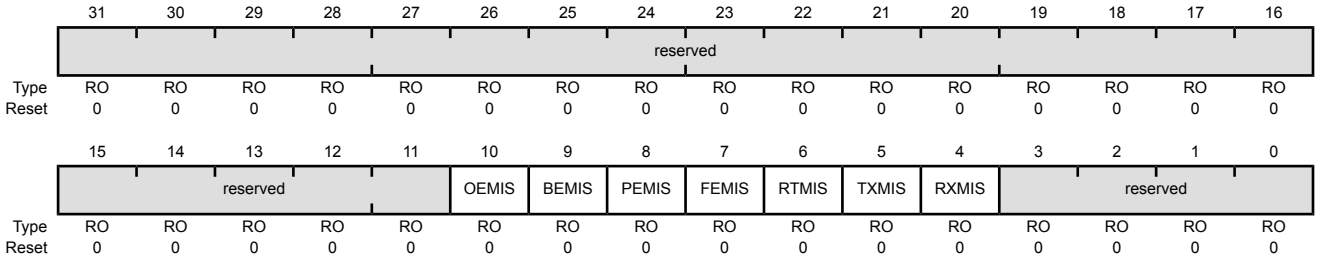

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | OERIS | RO | 0 | UART Overrun Error Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 9 | BERIS | RO | 0 | UART Break Error Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 8 | PERIS | RO | 0 | UART Parity Error Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 7 | FERIS | RO | 0 | UART Framing Error Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 6 | RTRIS | RO | 0 | UART Receive Time-Out Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 5 | TXRIS | RO | 0 | UART Transmit Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 4 | RXRIS | RO | 0 | UART Receive Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 3:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 12: UART Masked Interrupt Status (UARTMIS), offset 0x040

The **UARTMIS** register is the masked interrupt status register. On a read, this register gives the current masked status value of the corresponding interrupt. A write has no effect.

UART Masked Interrupt Status (UARTMIS)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x040
Type RO, reset 0x0000.0000



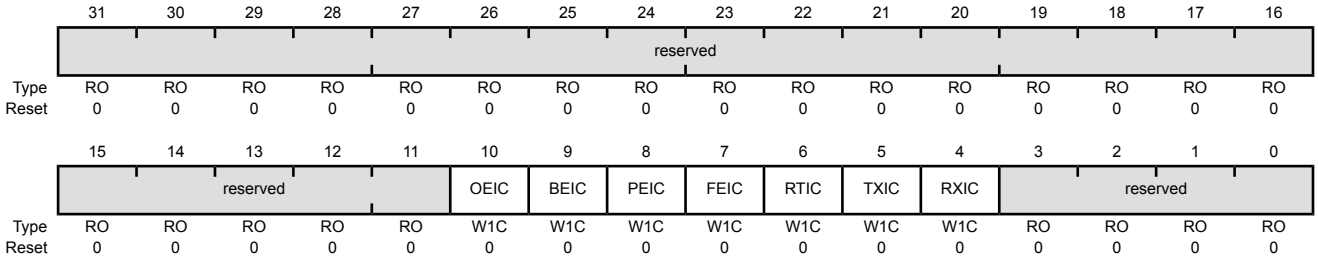

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | OEMIS | RO | 0 | UART Overrun Error Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 9 | BEMIS | RO | 0 | UART Break Error Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 8 | PEMIS | RO | 0 | UART Parity Error Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 7 | FEMIS | RO | 0 | UART Framing Error Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 6 | RTMIS | RO | 0 | UART Receive Time-Out Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 5 | TXMIS | RO | 0 | UART Transmit Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 4 | RXMIS | RO | 0 | UART Receive Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 3:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 13: UART Interrupt Clear (UARTICR), offset 0x044

The **UARTICR** register is the interrupt clear register. On a write of 1, the corresponding interrupt (both raw interrupt and masked interrupt, if enabled) is cleared. A write of 0 has no effect.

UART Interrupt Clear (UARTICR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x044
Type W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | OEIC | W1C | 0 | Overrun Error Interrupt Clear<br>The `OEIC` values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 9 | BEIC | W1C | 0 | Break Error Interrupt Clear<br>The `BEIC` values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 8 | PEIC | W1C | 0 | Parity Error Interrupt Clear<br>The `PEIC` values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 7 | FEIC | W1C | 0 | Framing Error Interrupt Clear<br>The `FEIC` values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |

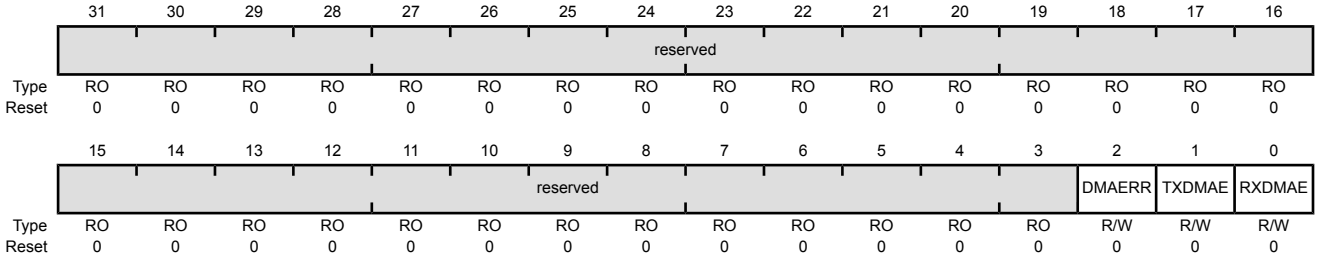
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | RTIC | W1C | 0 | Receive Time-Out Interrupt Clear<br>The RTIC values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 5 | TXIC | W1C | 0 | Transmit Interrupt Clear<br>The TXIC values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 4 | RXIC | W1C | 0 | Receive Interrupt Clear<br>The RXIC values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 3:0 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 14: UART DMA Control (UARTDMACTL), offset 0x048

The **UARTDMACTL** register is the DMA control register.

UART DMA Control (UARTDMACTL)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x048
Type R/W, reset 0x0000.0000



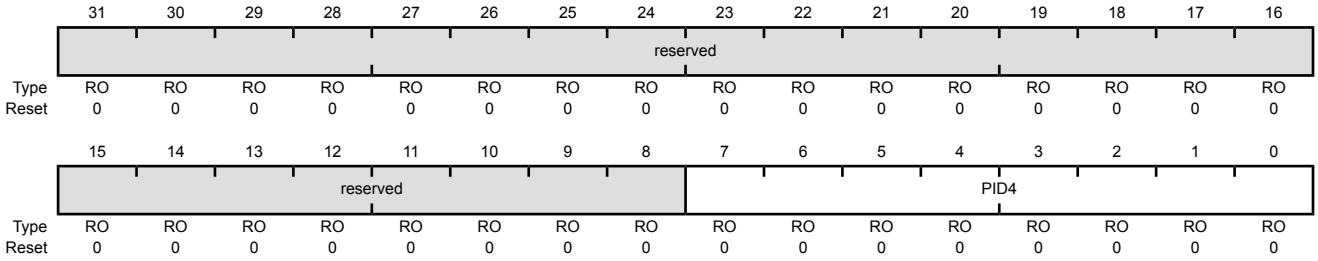

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | DMAERR | R/W | 0 | DMA on Error<br>If this bit is set to 1, DMA receive requests are automatically disabled when a receive error occurs. |
| 1 | TXDMAE | R/W | 0 | Transmit DMA Enable<br>If this bit is set to 1, DMA for the transmit FIFO is enabled. |
| 0 | RXDMAE | R/W | 0 | Receive DMA Enable<br>If this bit is set to 1, DMA for the receive FIFO is enabled. |

## Register 15: UART Peripheral Identification 4 (UARTPeriphID4), offset 0xFD0

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 4 (UARTPeriphID4)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFD0
Type RO, reset 0x0000.0000



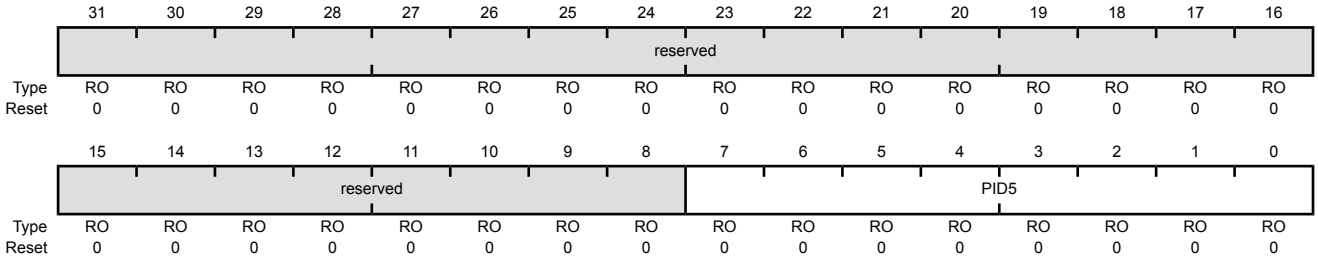

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID4 | RO | 0x0000 | UART Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

## Register 16: UART Peripheral Identification 5 (UARTPeriphID5), offset 0xFD4

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 5 (UARTPeriphID5)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFD4
Type RO, reset 0x0000.0000



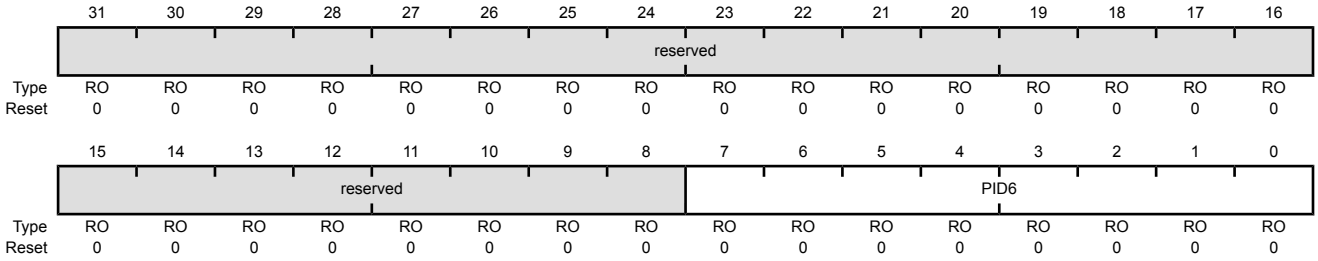

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID5 | RO | 0x0000 | UART Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

## Register 17: UART Peripheral Identification 6 (UARTPeriphID6), offset 0xFD8

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 6 (UARTPeriphID6)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFD8
Type RO, reset 0x0000.0000



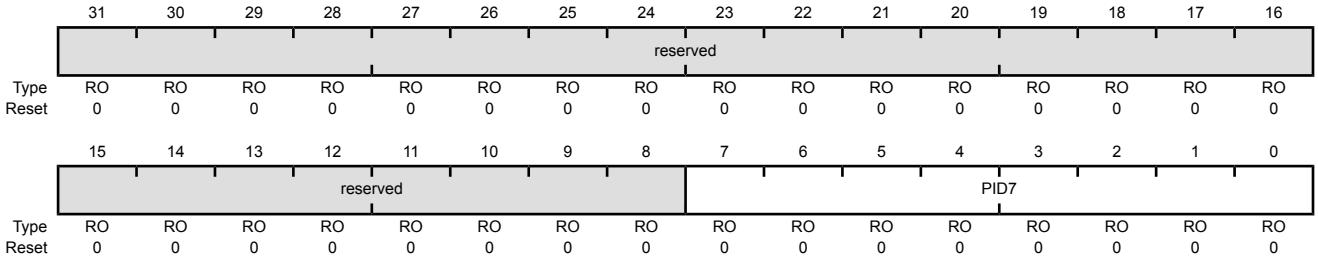

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID6 | RO | 0x0000 | UART Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 18: UART Peripheral Identification 7 (UARTPeriphID7), offset 0xFDC

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 7 (UARTPeriphID7)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFDC
Type RO, reset 0x0000.0000



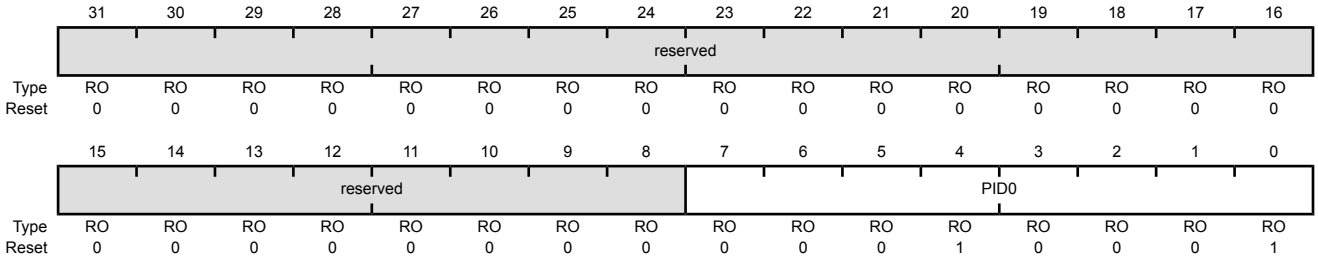

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID7 | RO | 0x0000 | UART Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

## Register 19: UART Peripheral Identification 0 (UARTPeriphID0), offset 0xFE0

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 0 (UARTPeriphID0)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFE0
Type RO, reset 0x0000.0011



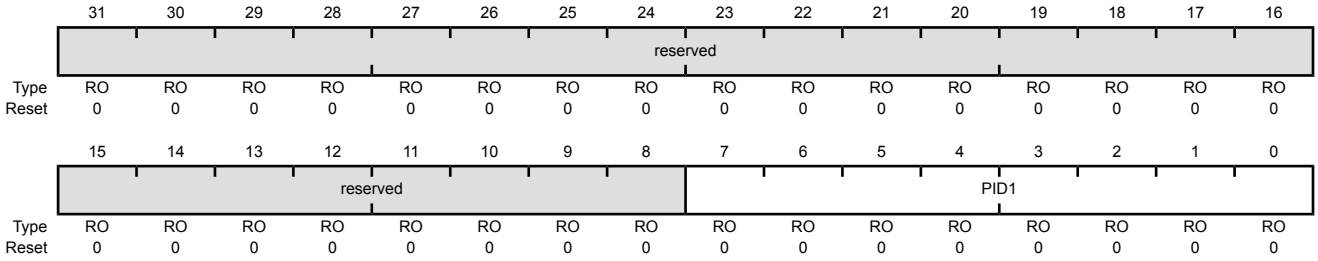

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID0 | RO | 0x11 | UART Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

## Register 20: UART Peripheral Identification 1 (UARTPeriphID1), offset 0xFE4

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 1 (UARTPeriphID1)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFE4
Type RO, reset 0x0000.0000



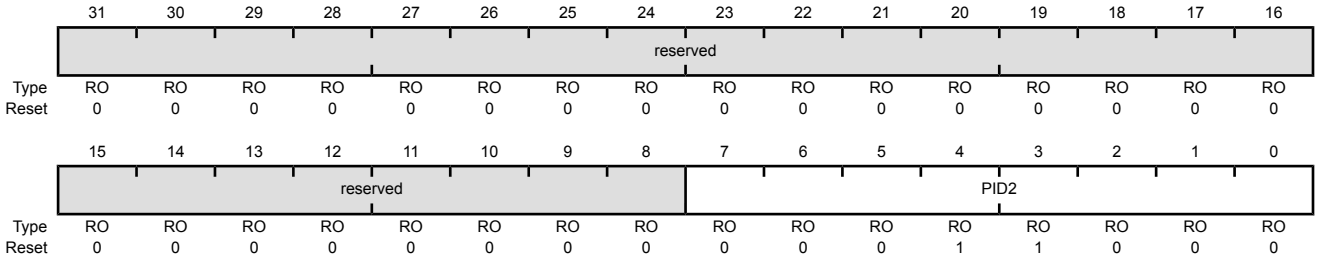

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID1 | RO | 0x00 | UART Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

## Register 21: UART Peripheral Identification 2 (UARTPeriphID2), offset 0xFE8

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 2 (UARTPeriphID2)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFE8
Type RO, reset 0x0000.0018



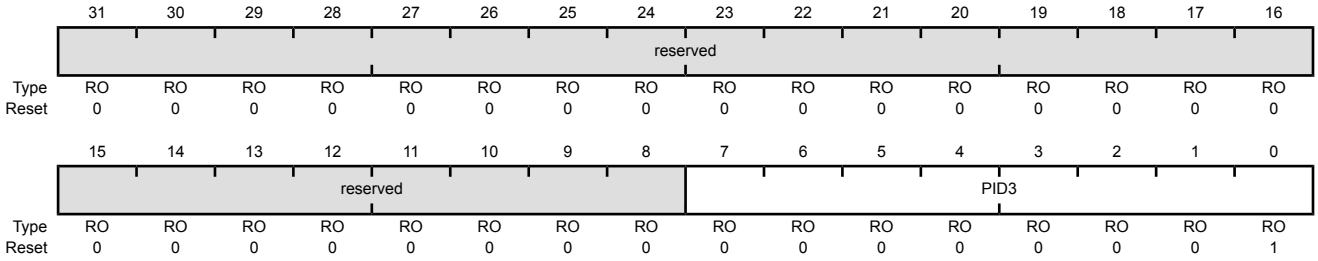

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID2 | RO | 0x18 | UART Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 22: UART Peripheral Identification 3 (UARTPeriphID3), offset 0xFEC

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 3 (UARTPeriphID3)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFEC
Type RO, reset 0x0000.0001



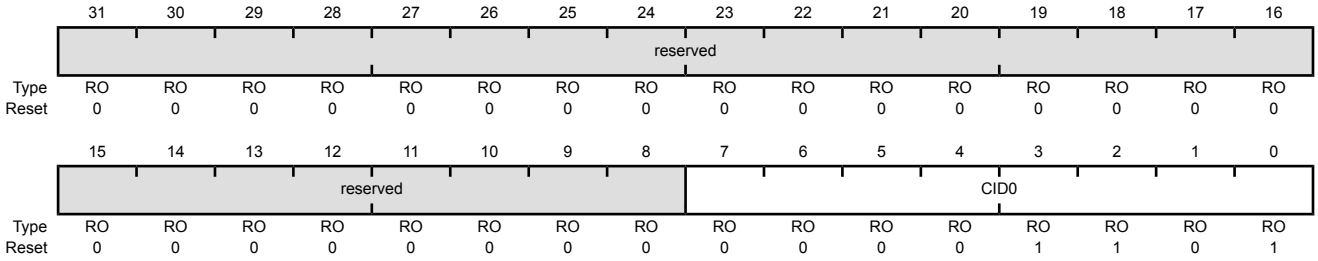

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID3 | RO | 0x01 | UART Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

## Register 23: UART PrimeCell Identification 0 (UARTPCellID0), offset 0xFF0

The **UARTPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART PrimeCell Identification 0 (UARTPCellID0)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFF0
Type RO, reset 0x0000.000D



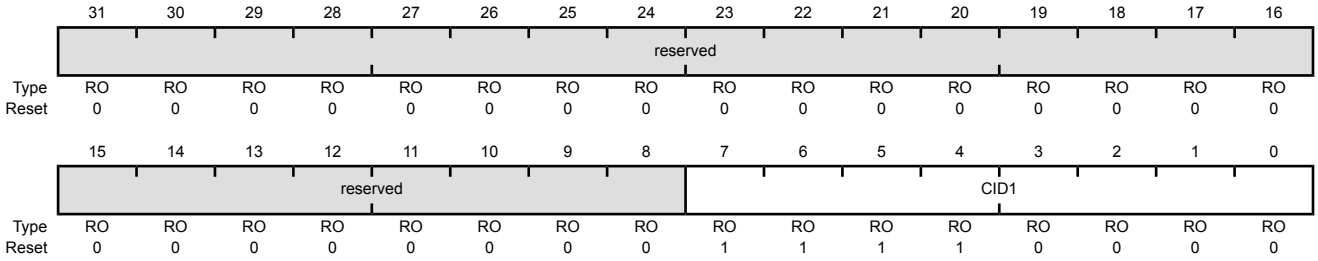

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID0 | RO | 0x0D | UART PrimeCell ID Register[7:0]<br>Provides software a standard cross-peripheral identification system. |

## Register 24: UART PrimeCell Identification 1 (UARTPCellID1), offset 0xFF4

The **UARTPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART PrimeCell Identification 1 (UARTPCellID1)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFF4
Type RO, reset 0x0000.00F0



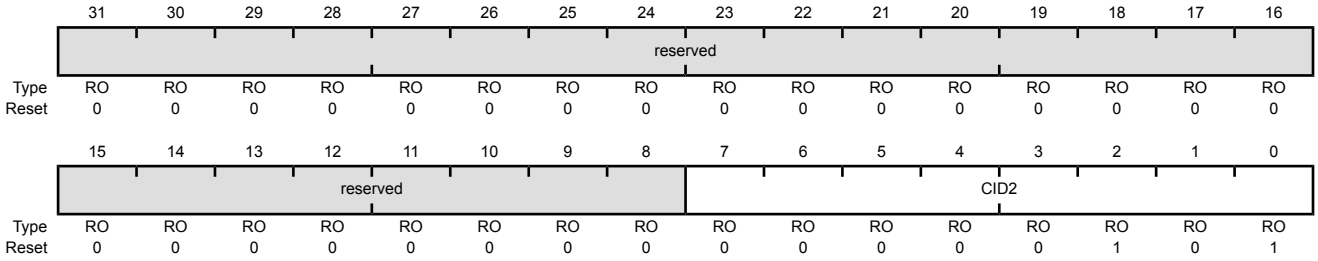

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID1 | RO | 0xF0 | UART PrimeCell ID Register[15:8]<br>Provides software a standard cross-peripheral identification system. |

## Register 25: UART PrimeCell Identification 2 (UARTPCellID2), offset 0xFF8

The **UARTPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART PrimeCell Identification 2 (UARTPCellID2)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFF8
Type RO, reset 0x0000.0005



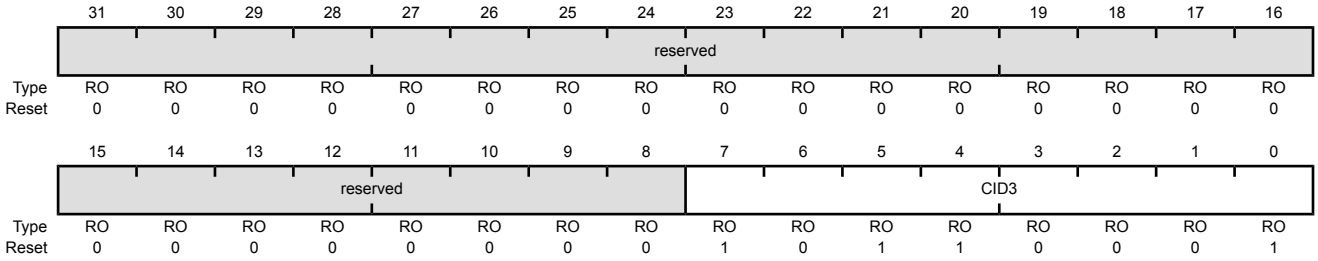

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID2 | RO | 0x05 | UART PrimeCell ID Register[23:16]<br>Provides software a standard cross-peripheral identification system. |

## Register 26: UART PrimeCell Identification 3 (UARTPCellID3), offset 0xFFC

The **UARTPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART PrimeCell Identification 3 (UARTPCellID3)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFFC
Type RO, reset 0x0000.00B1




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID3 | RO | 0xB1 | UART PrimeCell ID Register[31:24]<br>Provides software a standard cross-peripheral identification system. |

# 14 Synchronous Serial Interface (SSI)

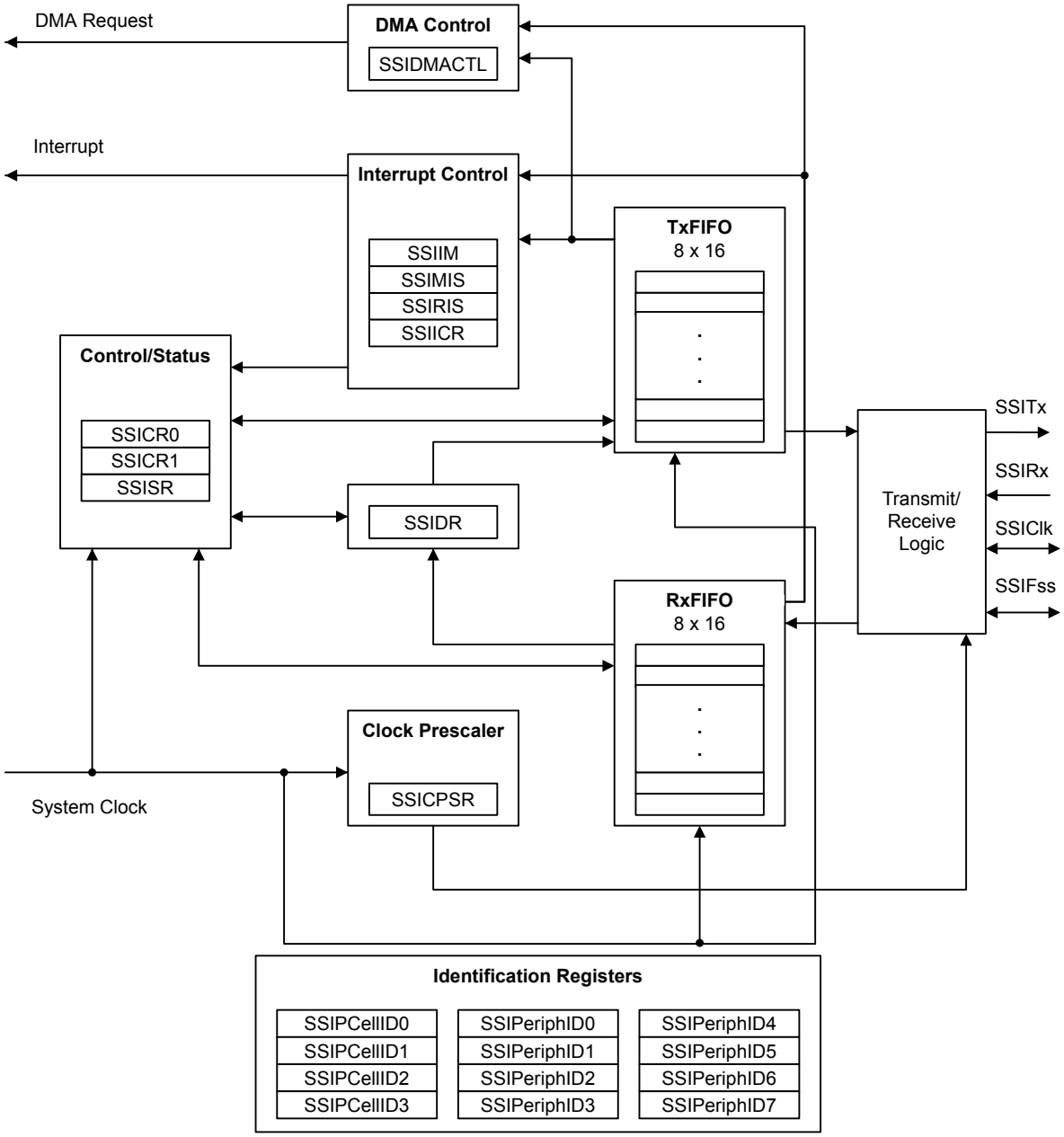
The Stellaris® Synchronous Serial Interface (SSI) is a master or slave interface for synchronous serial communication with peripheral devices that have either Freescale SPI, MICROWIRE, or Texas Instruments synchronous serial interfaces.

The Stellaris SSI module has the following features:

- Master or slave operation
- Support for Direct Memory Access (DMA)
- Programmable clock bit rate and prescale
- Separate transmit and receive FIFOs, 16 bits wide, 8 locations deep
- Programmable interface operation for Freescale SPI, MICROWIRE, or Texas Instruments synchronous serial interfaces
- Programmable data frame size from 4 to 16 bits
- Internal loopback test mode for diagnostic/debug testing

## 14.1 Block Diagram

**Figure 14-1. SSI Module Block Diagram**




## 14.2 Signal Description

Table 14-1 on page 538 lists the external signals of the SSI module and describes the function of each. The SSI signals are alternate functions for some GPIO signals and default to be GPIO signals at reset., with the exception of the SSI0Clk, SSI0Fss, SSI0Rx, and SSI0Tx pins which default to the SSI function. The column in the table below titled "Pin Assignment" lists the possible GPIO pin placements for the SSI signals. The AFSEL bit in the **GPIO Alternate Function Select**

**(GPIOAFSEL)** register (page 371) should be set to choose the SSI function. For more information on configuring GPIOs, see “General-Purpose Input/Outputs (GPIOs)” on page 351.

**Table 14-1. SSI Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| SSI0Clk | 19 | I/O | TTL | SSI module 0 clock. |
| SSI0Fss | 20 | I/O | TTL | SSI module 0 frame signal. |
| SSI0Rx | 21 | I | TTL | SSI module 0 receive. |
| SSI0Tx | 22 | O | TTL | SSI module 0 transmit. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 14.3 Functional Description

The SSI performs serial-to-parallel conversion on data received from a peripheral device. The CPU accesses data, control, and status information. The transmit and receive paths are buffered with internal FIFO memories allowing up to eight 16-bit values to be stored independently in both transmit and receive modes. The SSI also supports the DMA interface. The transmit and receive FIFOs can be programmed as destination/source addresses in the DMA module. DMA operation is enabled by setting the appropriate bit(s) in the **SSIDMACTL** register (see page 563).

### 14.3.1 Bit Rate Generation

The SSI includes a programmable bit rate clock divider and prescaler to generate the serial output clock. Bit rates are supported to 2 MHz and higher, although maximum bit rate is determined by peripheral devices.

The serial bit rate is derived by dividing down the input clock (FSysClk). The clock is first divided by an even prescale value CPSDVSR from 2 to 254, which is programmed in the **SSI Clock Prescale (SSICPSR)** register (see page 557). The clock is further divided by a value from 1 to 256, which is 1 + *SCR*, where *SCR* is the value programmed in the **SSI Control0 (SSICR0)** register (see page 550).

The frequency of the output clock SSIClk is defined by:

```
SSIClk = FSysClk / (CPSDVSR * (1 + SCR))
```

**Note:** For master mode, the system clock must be at least two times faster than the SSIClk. For slave mode, the system clock must be at least 12 times faster than the SSIClk.

See “Synchronous Serial Interface (SSI)” on page 777 to view SSI timing parameters.

### 14.3.2 FIFO Operation

#### 14.3.2.1 Transmit FIFO

The common transmit FIFO is a 16-bit wide, 8-locations deep, first-in, first-out memory buffer. The CPU writes data to the FIFO by writing the **SSI Data (SSIDR)** register (see page 554), and data is stored in the FIFO until it is read out by the transmission logic.

When configured as a master or a slave, parallel data is written into the transmit FIFO prior to serial conversion and transmission to the attached slave or master, respectively, through the SSITx pin.

In slave mode, the SSI transmits data each time the master initiates a transaction. If the transmit FIFO is empty and the master initiates, the slave transmits the 8th most recent value in the transmit FIFO. If less than 8 values have been written to the transmit FIFO since the SSI module clock was enabled using the SSI bit in the **RGCG1** register, then 0 is transmitted. Care should be taken to

ensure that valid data is in the FIFO as needed. The SSI can be configured to generate an interrupt or a µDMA request when the FIFO is empty.

#### 14.3.2.2 Receive FIFO

The common receive FIFO is a 16-bit wide, 8-locations deep, first-in, first-out memory buffer. Received data from the serial interface is stored in the buffer until read out by the CPU, which accesses the read FIFO by reading the **SSIDR** register.

When configured as a master or slave, serial data received through the SSIRx pin is registered prior to parallel loading into the attached slave or master receive FIFO, respectively.

### 14.3.3 Interrupts

The SSI can generate interrupts when the following conditions are observed:

- Transmit FIFO service
- Receive FIFO service
- Receive FIFO time-out
- Receive FIFO overrun

All of the interrupt events are ORed together before being sent to the interrupt controller, so the SSI can only generate a single interrupt request to the controller at any given time. You can mask each of the four individual maskable interrupts by setting the appropriate bits in the **SSI Interrupt Mask (SSIIM)** register (see page 558). Setting the appropriate mask bit to 1 enables the interrupt.

Provision of the individual outputs, as well as a combined interrupt output, allows use of either a global interrupt service routine, or modular device drivers to handle interrupts. The transmit and receive dynamic dataflow interrupts have been separated from the status interrupts so that data can be read or written in response to the FIFO trigger levels. The status of the individual interrupt sources can be read from the **SSI Raw Interrupt Status (SSIRIS)** and **SSI Masked Interrupt Status (SSIMIS)** registers (see page 560 and page 561, respectively).

### 14.3.4 Frame Formats

Each data frame is between 4 and 16 bits long, depending on the size of data programmed, and is transmitted starting with the MSB. There are three basic frame types that can be selected:

- Texas Instruments synchronous serial
- Freescale SPI
- MICROWIRE

For all three formats, the serial clock (SSIClk) is held inactive while the SSI is idle, and SSIClk transitions at the programmed frequency only during active transmission or reception of data. The idle state of SSIClk is utilized to provide a receive timeout indication that occurs when the receive FIFO still contains data after a timeout period.

For Freescale SPI and MICROWIRE frame formats, the serial frame (SSIFss ) pin is active Low, and is asserted (pulled down) during the entire transmission of the frame.

For Texas Instruments synchronous serial frame format, the SSIFss pin is pulsed for one serial clock period starting at its rising edge, prior to the transmission of each frame. For this frame format,

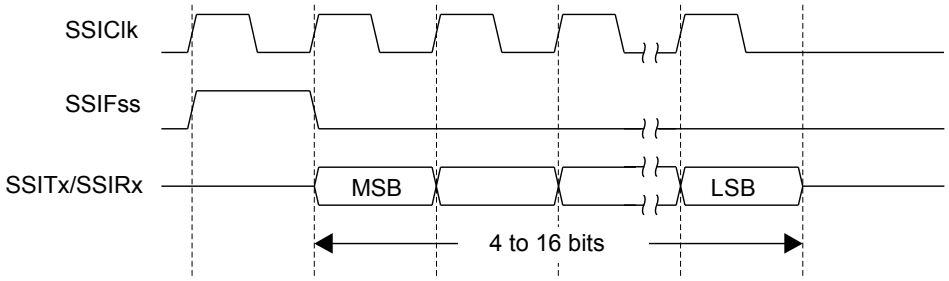
both the SSI and the off-chip slave device drive their output data on the rising edge of SSIClk, and latch data from the other device on the falling edge.

Unlike the full-duplex transmission of the other two frame formats, the MICROWIRE format uses a special master-slave messaging technique, which operates at half-duplex. In this mode, when a frame begins, an 8-bit control message is transmitted to the off-chip slave. During this transmit, no incoming data is received by the SSI. After the message has been sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message has been sent, responds with the requested data. The returned data can be 4 to 16 bits in length, making the total frame length anywhere from 13 to 25 bits.

#### 14.3.4.1 Texas Instruments Synchronous Serial Frame Format

Figure 14-2 on page 540 shows the Texas Instruments synchronous serial frame format for a single transmitted frame.

**Figure 14-2. TI Synchronous Serial Frame Format (Single Transfer)**

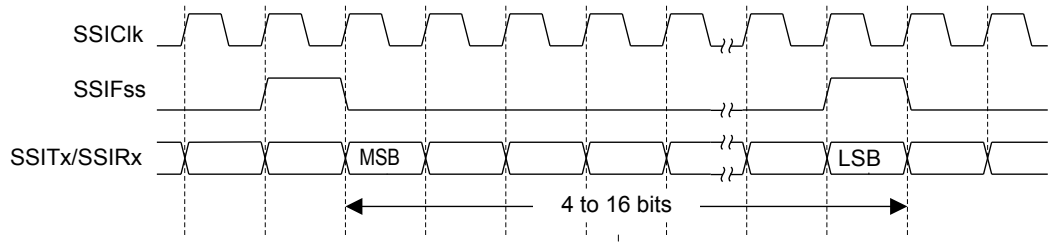



In this mode, SSIClk and SSIFss are forced Low, and the transmit data line SSITx is tristated whenever the SSI is idle. Once the bottom entry of the transmit FIFO contains data, SSIFss is pulsed High for one SSIClk period. The value to be transmitted is also transferred from the transmit FIFO to the serial shift register of the transmit logic. On the next rising edge of SSIClk, the MSB of the 4 to 16-bit data frame is shifted out on the SSITx pin. Likewise, the MSB of the received data is shifted onto the SSIRx pin by the off-chip serial slave device.

Both the SSI and the off-chip serial slave device then clock each data bit into their serial shifter on the falling edge of each SSIClk. The received data is transferred from the serial shifter to the receive FIFO on the first rising edge of SSIClk after the LSB has been latched.

Figure 14-3 on page 541 shows the Texas Instruments synchronous serial frame format when back-to-back frames are transmitted.

**Figure 14-3. TI Synchronous Serial Frame Format (Continuous Transfer)**




### 14.3.4.2 Freescale SPI Frame Format

The Freescale SPI interface is a four-wire interface where the SSIFss signal behaves as a slave select. The main feature of the Freescale SPI format is that the inactive state and phase of the SSIClk signal are programmable through the SPO and SPH bits within the **SSISCR0** control register.

#### *SPO Clock Polarity Bit*

When the SPO clock polarity control bit is Low, it produces a steady state Low value on the SSIClk pin. If the SPO bit is High, a steady state High value is placed on the SSIClk pin when data is not being transferred.

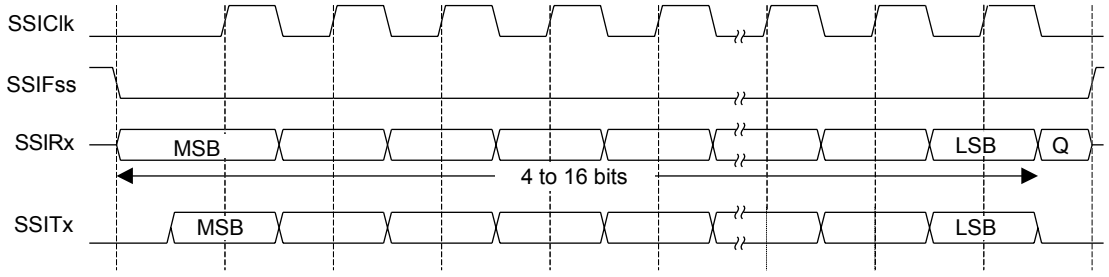
#### *SPH Phase Control Bit*

The SPH phase control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge. When the SPH phase control bit is Low, data is captured on the first clock edge transition. If the SPH bit is High, data is captured on the second clock edge transition.

### 14.3.4.3 Freescale SPI Frame Format with SPO=0 and SPH=0

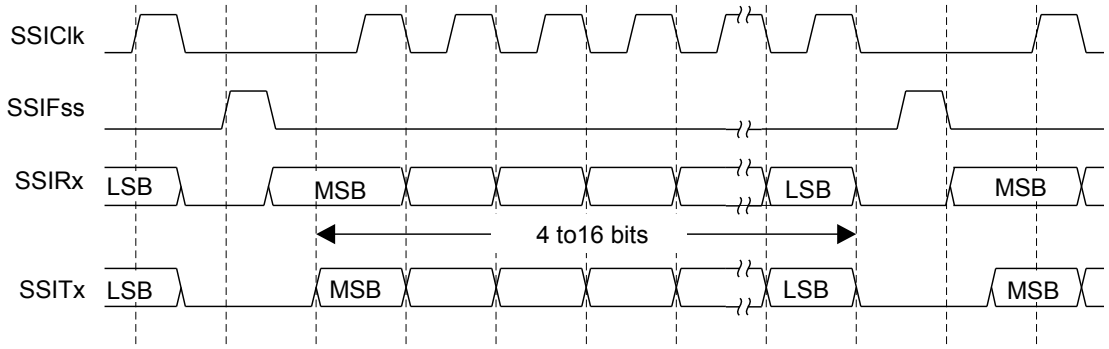
Single and continuous transmission signal sequences for Freescale SPI format with SPO=0 and SPH=0 are shown in Figure 14-4 on page 541 and Figure 14-5 on page 542.

**Figure 14-4. Freescale SPI Format (Single Transfer) with SPO=0 and SPH=0**




**Note:** Q is undefined.

**Figure 14-5. Freescale SPI Format (Continuous Transfer) with SPO=0 and SPH=0**




In this configuration, during idle periods:

- SSIClk is forced Low
- SSIFss is forced High
- The transmit data line SSITx is arbitrarily forced Low
- When the SSI is configured as a master, it enables the SSIClk pad
- When the SSI is configured as a slave, it disables the SSIClk pad

If the SSI is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSIFss master signal being driven Low. This causes slave data to be enabled onto the SSIRx input line of the master. The master SSITx output pad is enabled.

One half SSIClk period later, valid master data is transferred to the SSITx pin. Now that both the master and slave data have been set, the SSIClk master clock pin goes High after one further half SSIClk period.

The data is now captured on the rising and propagated on the falling edges of the SSIClk signal.

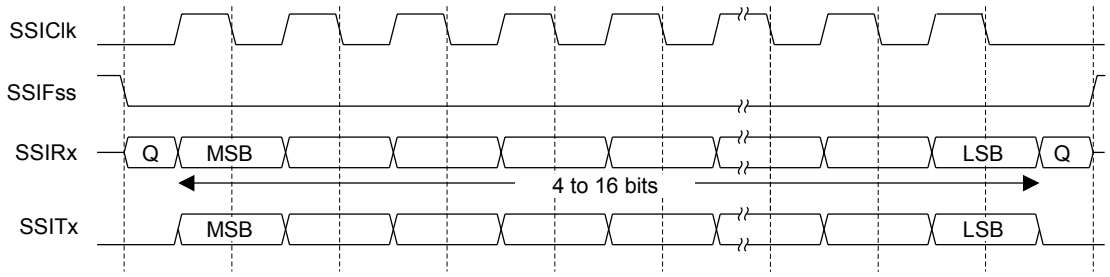
In the case of a single word transmission, after all bits of the data word have been transferred, the SSIFss line is returned to its idle High state one SSIClk period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSIFss signal must be pulsed High between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the SPH bit is logic zero. Therefore, the master device must raise the SSIFss pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSIFss pin is returned to its idle state one SSIClk period after the last bit has been captured.

### 14.3.4.4 Freescale SPI Frame Format with SPO=0 and SPH=1

The transfer signal sequence for Freescale SPI format with SPO=0 and SPH=1 is shown in Figure 14-6 on page 543, which covers both single and continuous transfers.

**Figure 14-6. Freescale SPI Frame Format with SPO=0 and SPH=1**




**Note:** Q is undefined.

In this configuration, during idle periods:

- SSIClk is forced Low
- SSIFss is forced High
- The transmit data line SSITx is arbitrarily forced Low
- When the SSI is configured as a master, it enables the SSIClk pad
- When the SSI is configured as a slave, it disables the SSIClk pad

If the SSI is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSIFss master signal being driven Low. The master SSITx output is enabled. After a further one half SSIClk period, both master and slave valid data is enabled onto their respective transmission lines. At the same time, the SSIClk is enabled with a rising edge transition.

Data is then captured on the falling edges and propagated on the rising edges of the SSIClk signal.

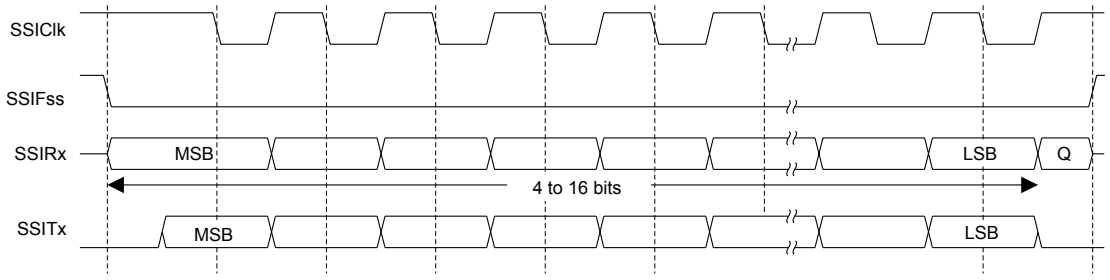
In the case of a single word transfer, after all bits have been transferred, the SSIFss line is returned to its idle High state one SSIClk period after the last bit has been captured.

For continuous back-to-back transfers, the SSIFss pin is held Low between successive data words and termination is the same as that of the single word transfer.

### 14.3.4.5 Freescale SPI Frame Format with SPO=1 and SPH=0

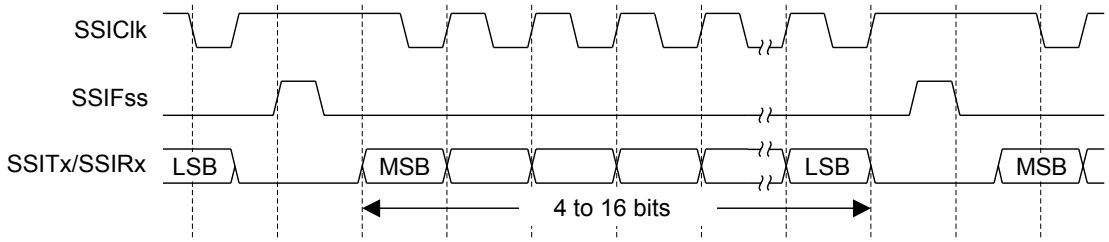
Single and continuous transmission signal sequences for Freescale SPI format with SPO=1 and SPH=0 are shown in Figure 14-7 on page 543 and Figure 14-8 on page 544.

**Figure 14-7. Freescale SPI Frame Format (Single Transfer) with SPO=1 and SPH=0**




**Note:** Q is undefined.

**Figure 14-8. Freescale SPI Frame Format (Continuous Transfer) with SPO=1 and SPH=0**




In this configuration, during idle periods:

- SSIClk is forced High
- SSIFss is forced High
- The transmit data line SSITx is arbitrarily forced Low
- When the SSI is configured as a master, it enables the SSIClk pad
- When the SSI is configured as a slave, it disables the SSIClk pad

If the SSI is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSIFss master signal being driven Low, which causes slave data to be immediately transferred onto the SSIRx line of the master. The master SSITx output pad is enabled.

One half period later, valid master data is transferred to the SSITx line. Now that both the master and slave data have been set, the SSIClk master clock pin becomes Low after one further half SSIClk period. This means that data is captured on the falling edges and propagated on the rising edges of the SSIClk signal.

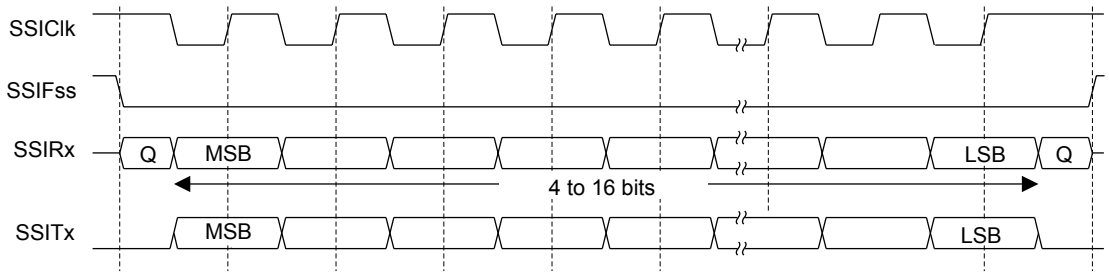
In the case of a single word transmission, after all bits of the data word are transferred, the SSIFss line is returned to its idle High state one SSIClk period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSIFss signal must be pulsed High between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the SPH bit is logic zero. Therefore, the master device must raise the SSIFss pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSIFss pin is returned to its idle state one SSIClk period after the last bit has been captured.

### 14.3.4.6 Freescale SPI Frame Format with SPO=1 and SPH=1

The transfer signal sequence for Freescale SPI format with SPO=1 and SPH=1 is shown in Figure 14-9 on page 545, which covers both single and continuous transfers.

**Figure 14-9. Freescale SPI Frame Format with SPO=1 and SPH=1**




**Note:** Q is undefined.

In this configuration, during idle periods:

- SSIClk is forced High
- SSIFss is forced High
- The transmit data line SSITx is arbitrarily forced Low
- When the SSI is configured as a master, it enables the SSIClk pad
- When the SSI is configured as a slave, it disables the SSIClk pad

If the SSI is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSIFss master signal being driven Low. The master SSITx output pad is enabled. After a further one-half SSIClk period, both master and slave data are enabled onto their respective transmission lines. At the same time, SSIClk is enabled with a falling edge transition. Data is then captured on the rising edges and propagated on the falling edges of the SSIClk signal.

After all bits have been transferred, in the case of a single word transmission, the SSIFss line is returned to its idle high state one SSIClk period after the last bit has been captured.

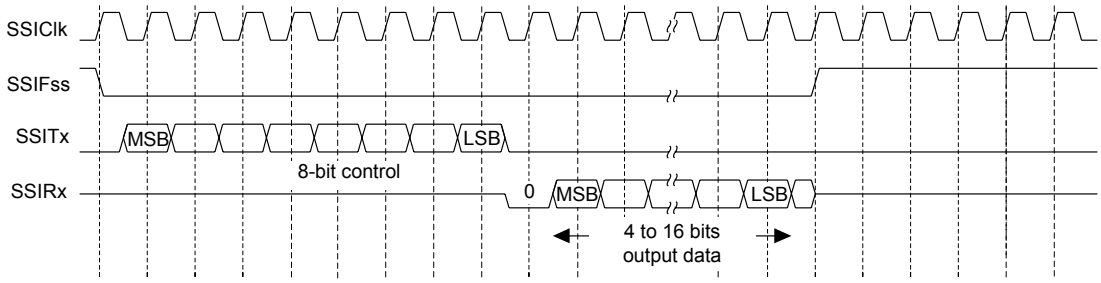
For continuous back-to-back transmissions, the SSIFss pin remains in its active Low state, until the final bit of the last word has been captured, and then returns to its idle state as described above.

For continuous back-to-back transfers, the SSIFss pin is held Low between successive data words and termination is the same as that of the single word transfer.

### 14.3.4.7 MICROWIRE Frame Format

Figure 14-10 on page 545 shows the MICROWIRE frame format, again for a single frame. Figure 14-11 on page 546 shows the same format when back-to-back frames are transmitted.

**Figure 14-10. MICROWIRE Frame Format (Single Frame)**



MICROWIRE format is very similar to SPI format, except that transmission is half-duplex instead of full-duplex, using a master-slave message passing technique. Each serial transmission begins with an 8-bit control word that is transmitted from the SSI to the off-chip slave device. During this transmission, no incoming data is received by the SSI. After the message has been sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message has been sent, responds with the required data. The returned data is 4 to 16 bits in length, making the total frame length anywhere from 13 to 25 bits.

In this configuration, during idle periods:

- SSIClk is forced Low
- SSIFss is forced High
- The transmit data line SSITx is arbitrarily forced Low

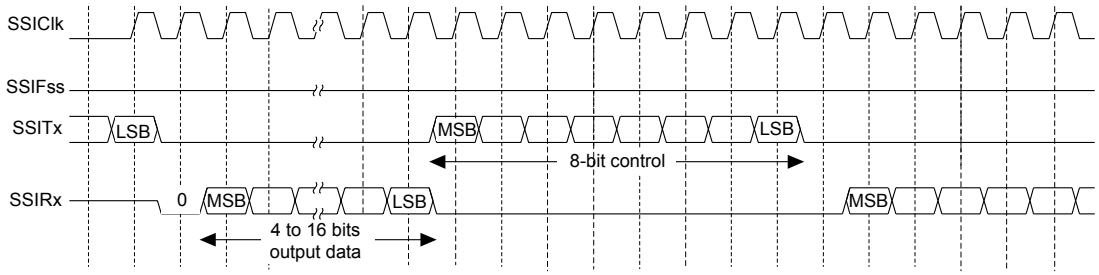
A transmission is triggered by writing a control byte to the transmit FIFO. The falling edge of SSIFss causes the value contained in the bottom entry of the transmit FIFO to be transferred to the serial shift register of the transmit logic, and the MSB of the 8-bit control frame to be shifted out onto the SSITx pin. SSIFss remains Low for the duration of the frame transmission. The SSIRx pin remains tristated during this transmission.

The off-chip serial slave device latches each control bit into its serial shifter on the rising edge of each SSIClk. After the last bit is latched by the slave device, the control byte is decoded during a one clock wait-state, and the slave responds by transmitting data back to the SSI. Each bit is driven onto the SSIRx line on the falling edge of SSIClk. The SSI in turn latches each bit on the rising edge of SSIClk. At the end of the frame, for single transfers, the SSIFss signal is pulled High one clock period after the last bit has been latched in the receive serial shifter, which causes the data to be transferred to the receive FIFO.

**Note:** The off-chip slave device can tristate the receive line either on the falling edge of SSIClk after the LSB has been latched by the receive shifter, or when the SSIFss pin goes High.

For continuous transfers, data transmission begins and ends in the same manner as a single transfer. However, the SSIFss line is continuously asserted (held Low) and transmission of data occurs back-to-back. The control byte of the next frame follows directly after the LSB of the received data from the current frame. Each of the received values is transferred from the receive shifter on the falling edge of SSIClk, after the LSB of the frame has been latched into the SSI.

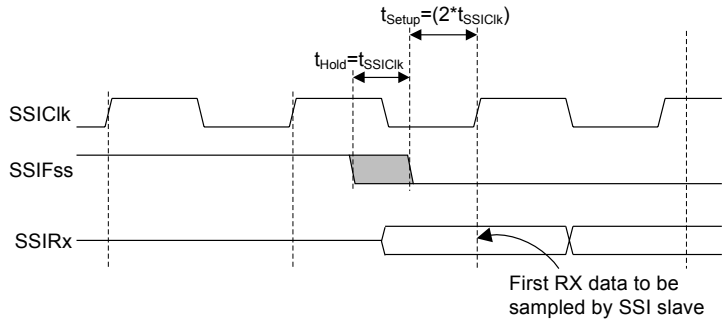
**Figure 14-11. MICROWIRE Frame Format (Continuous Transfer)**




In the MICROWIRE mode, the SSI slave samples the first bit of receive data on the rising edge of SSIClk after SSIFss has gone Low. Masters that drive a free-running SSIClk must ensure that the SSIFss signal has sufficient setup and hold margins with respect to the rising edge of SSIClk.

Figure 14-12 on page 547 illustrates these setup and hold time requirements. With respect to the SSIClk rising edge on which the first bit of receive data is to be sampled by the SSI slave, SSIFss must have a setup of at least two times the period of SSIClk on which the SSI operates. With respect to the SSIClk rising edge previous to this edge, SSIFss must have a hold of at least one SSIClk period.

**Figure 14-12. MICROWIRE Frame Format, SSIFss Input Setup and Hold Requirements**




### 14.3.5 DMA Operation

The SSI peripheral provides an interface connected to the µDMA controller. The DMA operation of the SSI is enabled through the **SSI DMA Control (SSIDMACTL)** register. When DMA operation is enabled, the SSI will assert a DMA request on the receive or transmit channel when the associated FIFO can transfer data. For the receive channel, a single transfer request is asserted whenever there is any data in the receive FIFO. A burst transfer request is asserted whenever the amount of data in the receive FIFO is 4 or more items. For the transmit channel, a single transfer request is asserted whenever there is at least one empty location in the transmit FIFO. The burst request is asserted whenever the transmit FIFO has 4 or more empty slots. The single and burst DMA transfer requests are handled automatically by the µDMA controller depending how the DMA channel is configured. To enable DMA operation for the receive channel, the RXDMAE bit of the **DMA Control (SSIDMACTL)** register should be set. To enable DMA operation for the transmit channel, the TXDMAE bit of **SSIDMACTL** should be set. If DMA is enabled, then the µDMA controller will trigger an interrupt when a transfer is complete. The interrupt will occur on the SSI interrupt vector. Therefore, if interrupts are used for SSI operation and DMA is enabled, the SSI interrupt handler must be designed to handle the µDMA completion interrupt.

See “Micro Direct Memory Access (µDMA)” on page 290 for more details about programming the µDMA controller.

## 14.4 Initialization and Configuration

To use the SSI, its peripheral clock must be enabled by setting the SSI bit in the **RCGC1** register.

For each of the frame formats, the SSI is configured using the following steps:

1. Ensure that the SSE bit in the **SSICR1** register is disabled before making any configuration changes.
2. Select whether the SSI is a master or slave:
   a. For master operations, set the **SSICR1** register to 0x0000.0000.
   b. For slave mode (output enabled), set the **SSICR1** register to 0x0000.0004.

   c. For slave mode (output disabled), set the **SSICR1** register to 0x0000.000C.

3. Configure the clock prescale divisor by writing the **SSICPSR** register.
4. Write the **SSICR0** register with the following configuration:
   - Serial clock rate (SCR)
   - Desired clock phase/polarity, if using Freescale SPI mode (SPH and SPO)
   - The protocol mode: Freescale SPI, TI SSF, MICROWIRE (FRF)
   - The data size (DSS)
5. Optionally, configure the µDMA channel (see "Micro Direct Memory Access (µDMA)" on page 290) and enable the DMA option(s) in the **SSIDMACTL** register.
6. Enable the SSI by setting the SSE bit in the **SSICR1** register.

As an example, assume the SSI must be configured to operate with the following parameters:

- Master operation
- Freescale SPI mode (SPO=1, SPH=1)
- 1 Mbps bit rate
- 8 data bits

Assuming the system clock is 20 MHz, the bit rate calculation would be:

```
FSSIClk = FSysClk / (CPSDVSR * (1 + SCR))
1x10^6 = 20x10^6 / (CPSDVSR * (1 + SCR))
```

In this case, if CPSDVSR=2, SCR must be 9.

The configuration sequence would be as follows:

1. Ensure that the SSE bit in the **SSICR1** register is disabled.
2. Write the **SSICR1** register with a value of 0x0000.0000.
3. Write the **SSICPSR** register with a value of 0x0000.0002.
4. Write the **SSICR0** register with a value of 0x0000.09C7.
5. The SSI is then enabled by setting the SSE bit in the **SSICR1** register to 1.

## 14.5 Register Map

Table 14-2 on page 549 lists the SSI registers. The offset listed is a hexadecimal increment to the register's address, relative to that SSI module's base address:

- SSI0: 0x4000.8000

Note that the SSI module clock must be enabled before the registers can be programmed (see page 223). There must be a delay of 3 system clocks after the SSI module clock is enabled before any SSI module registers are accessed.

**Note:** The SSI must be disabled (see the SSE bit in the **SSICR1** register) before any of the control registers are reprogrammed.

**Table 14-2. SSI Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | SSICR0 | R/W | 0x0000.0000 | SSI Control 0 | 550 |
| 0x004 | SSICR1 | R/W | 0x0000.0000 | SSI Control 1 | 552 |
| 0x008 | SSIDR | R/W | 0x0000.0000 | SSI Data | 554 |
| 0x00C | SSISR | RO | 0x0000.0003 | SSI Status | 555 |
| 0x010 | SSICPSR | R/W | 0x0000.0000 | SSI Clock Prescale | 557 |
| 0x014 | SSIIM | R/W | 0x0000.0000 | SSI Interrupt Mask | 558 |
| 0x018 | SSIRIS | RO | 0x0000.0008 | SSI Raw Interrupt Status | 560 |
| 0x01C | SSIMIS | RO | 0x0000.0000 | SSI Masked Interrupt Status | 561 |
| 0x020 | SSIICR | W1C | 0x0000.0000 | SSI Interrupt Clear | 562 |
| 0x024 | SSIDMACTL | R/W | 0x0000.0000 | SSI DMA Control | 563 |
| 0xFD0 | SSIPeriphID4 | RO | 0x0000.0000 | SSI Peripheral Identification 4 | 564 |
| 0xFD4 | SSIPeriphID5 | RO | 0x0000.0000 | SSI Peripheral Identification 5 | 565 |
| 0xFD8 | SSIPeriphID6 | RO | 0x0000.0000 | SSI Peripheral Identification 6 | 566 |
| 0xFDC | SSIPeriphID7 | RO | 0x0000.0000 | SSI Peripheral Identification 7 | 567 |
| 0xFE0 | SSIPeriphID0 | RO | 0x0000.0022 | SSI Peripheral Identification 0 | 568 |
| 0xFE4 | SSIPeriphID1 | RO | 0x0000.0000 | SSI Peripheral Identification 1 | 569 |
| 0xFE8 | SSIPeriphID2 | RO | 0x0000.0018 | SSI Peripheral Identification 2 | 570 |
| 0xFEC | SSIPeriphID3 | RO | 0x0000.0001 | SSI Peripheral Identification 3 | 571 |
| 0xFF0 | SSIPCellID0 | RO | 0x0000.000D | SSI PrimeCell Identification 0 | 572 |
| 0xFF4 | SSIPCellID1 | RO | 0x0000.00F0 | SSI PrimeCell Identification 1 | 573 |
| 0xFF8 | SSIPCellID2 | RO | 0x0000.0005 | SSI PrimeCell Identification 2 | 574 |
| 0xFFC | SSIPCellID3 | RO | 0x0000.00B1 | SSI PrimeCell Identification 3 | 575 |

## 14.6 Register Descriptions

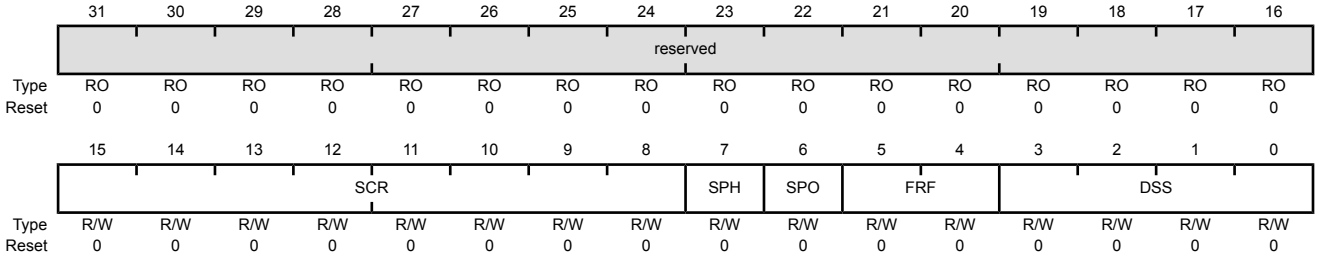
The remainder of this section lists and describes the SSI registers, in numerical order by address offset.

## Register 1: SSI Control 0 (SSICR0), offset 0x000

**SSICR0** is control register 0 and contains bit fields that control various functions within the SSI module. Functionality such as protocol mode, clock rate, and data size are configured in this register.

SSI Control 0 (SSICR0)

SSI0 base: 0x4000.8000
Offset 0x000
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:8 | SCR | R/W | 0x0000 | SSI Serial Clock Rate<br>The value SCR is used to generate the transmit and receive bit rate of the SSI. The bit rate is:<br>`BR=FSSIClk/(CPSDVSR * (1 + SCR))`<br>where CPSDVSR is an even value from 2-254 programmed in the **SSICPSR** register, and SCR is a value from 0-255. |
| 7 | SPH | R/W | 0 | SSI Serial Clock Phase<br>This bit is only applicable to the Freescale SPI Format.<br>The SPH control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge.<br>When the SPH bit is 0, data is captured on the first clock edge transition. If SPH is 1, data is captured on the second clock edge transition. |
| 6 | SPO | R/W | 0 | SSI Serial Clock Polarity<br>This bit is only applicable to the Freescale SPI Format.<br>When the SPO bit is 0, it produces a steady state Low value on the SSIClk pin. If SPO is 1, a steady state High value is placed on the SSIClk pin when data is not being transferred. |
| 5:4 | FRF | R/W | 0x0 | SSI Frame Format Select<br>The FRF values are defined as follows:<br>Value Frame Format<br>0x0 Freescale SPI Frame Format<br>0x1 Texas Instruments Synchronous Serial Frame Format<br>0x2 MICROWIRE Frame Format<br>0x3 Reserved |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3:0 | DSS | R/W | 0x00 | SSI Data Size Select<br>The DSS values are defined as follows: |

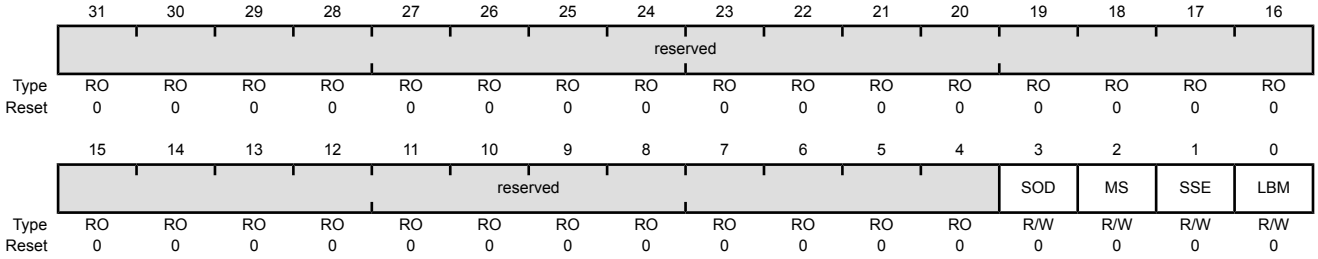
| Value | Data Size |
|---|---|
| 0x0-0x2 | Reserved |
| 0x3 | 4-bit data |
| 0x4 | 5-bit data |
| 0x5 | 6-bit data |
| 0x6 | 7-bit data |
| 0x7 | 8-bit data |
| 0x8 | 9-bit data |
| 0x9 | 10-bit data |
| 0xA | 11-bit data |
| 0xB | 12-bit data |
| 0xC | 13-bit data |
| 0xD | 14-bit data |
| 0xE | 15-bit data |
| 0xF | 16-bit data |

## Register 2: SSI Control 1 (SSICR1), offset 0x004

**SSICR1** is control register 1 and contains bit fields that control various functions within the SSI module. Master and slave mode functionality is controlled by this register.

SSI Control 1 (SSICR1)

SSI0 base: 0x4000.8000
Offset 0x004
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | SOD | R/W | 0 | SSI Slave Mode Output Disable<br>This bit is relevant only in the Slave mode (MS=1). In multiple-slave systems, it is possible for the SSI master to broadcast a message to all slaves in the system while ensuring that only one slave drives data onto the serial output line. In such systems, the TXD lines from multiple slaves could be tied together. To operate in such a system, the SOD bit can be configured so that the SSI slave does not drive the SSITx pin.<br>The SOD values are defined as follows:<br>Value Description<br>0 SSI can drive SSITx output in Slave Output mode.<br>1 SSI must not drive the SSITx output in Slave mode. |
| 2 | MS | R/W | 0 | SSI Master/Slave Select<br>This bit selects Master or Slave mode and can be modified only when SSI is disabled (SSE=0).<br>The MS values are defined as follows:<br>Value Description<br>0 Device configured as a master.<br>1 Device configured as a slave. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | SSE | R/W | 0 | SSI Synchronous Serial Port Enable<br>Setting this bit enables SSI operation.<br>The SSE values are defined as follows:<br><br>Value Description<br>0 SSI operation disabled.<br>1 SSI operation enabled.<br><br>**Note:** This bit must be set to 0 before any control registers are reprogrammed. |
| 0 | LBM | R/W | 0 | SSI Loopback Mode<br>Setting this bit enables Loopback Test mode.<br>The LBM values are defined as follows:<br><br>Value Description<br>0 Normal serial port operation enabled.<br>1 Output of the transmit serial shift register is connected internally to the input of the receive serial shift register. |

## Register 3: SSI Data (SSIDR), offset 0x008

**Important:** This register is read-sensitive. See the register description for details.

**SSIDR** is the data register and is 16-bits wide. When **SSIDR** is read, the entry in the receive FIFO (pointed to by the current FIFO read pointer) is accessed. As data values are removed by the SSI receive logic from the incoming data frame, they are placed into the entry in the receive FIFO (pointed to by the current FIFO write pointer).

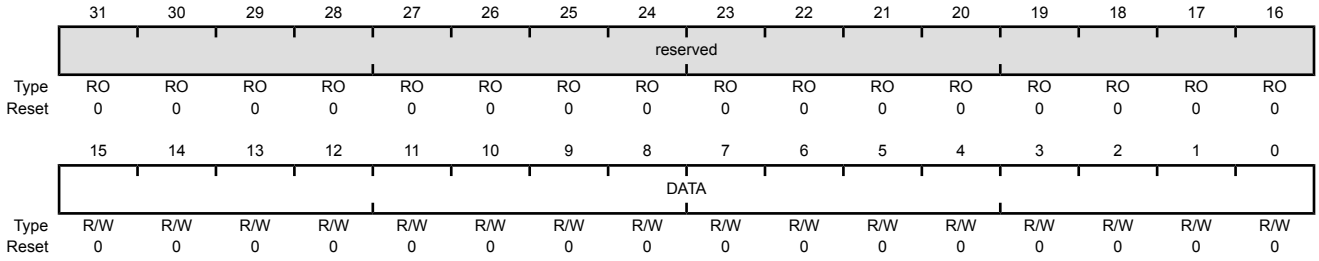
When **SSIDR** is written to, the entry in the transmit FIFO (pointed to by the write pointer) is written to. Data values are removed from the transmit FIFO one value at a time by the transmit logic. It is loaded into the transmit serial shifter, then serially shifted out onto the SSITx pin at the programmed bit rate.

When a data size of less than 16 bits is selected, the user must right-justify data written to the transmit FIFO. The transmit logic ignores the unused bits. Received data less than 16 bits is automatically right-justified in the receive buffer.

When the SSI is programmed for MICROWIRE frame format, the default size for transmit data is eight bits (the most significant byte is ignored). The receive data size is controlled by the programmer. The transmit FIFO and the receive FIFO are not cleared even when the SSE bit in the **SSICR1** register is set to zero. This allows the software to fill the transmit FIFO before enabling the SSI.

SSI Data (SSIDR)

SSI0 base: 0x4000.8000
Offset 0x008
Type R/W, reset 0x0000.0000



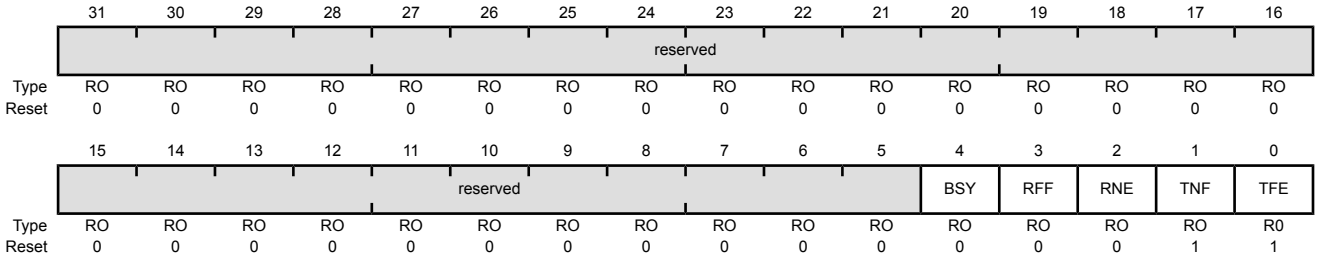

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | DATA | R/W | 0x0000 | SSI Receive/Transmit Data<br>A read operation reads the receive FIFO. A write operation writes the transmit FIFO.<br>Software must right-justify data when the SSI is programmed for a data size that is less than 16 bits. Unused bits at the top are ignored by the transmit logic. The receive logic automatically right-justifies the data. |

## Register 4: SSI Status (SSISR), offset 0x00C

**SSISR** is a status register that contains bits that indicate the FIFO fill status and the SSI busy status.

SSI Status (SSISR)

SSI0 base: 0x4000.8000
Offset 0x00C
Type RO, reset 0x0000.0003




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | BSY | RO | 0 | SSI Busy Bit<br>The BSY values are defined as follows:<br><br>Value Description<br>0 SSI is idle.<br>1 SSI is currently transmitting and/or receiving a frame, or the transmit FIFO is not empty. |
| 3 | RFF | RO | 0 | SSI Receive FIFO Full<br>The RFF values are defined as follows:<br><br>Value Description<br>0 Receive FIFO is not full.<br>1 Receive FIFO is full. |
| 2 | RNE | RO | 0 | SSI Receive FIFO Not Empty<br>The RNE values are defined as follows:<br><br>Value Description<br>0 Receive FIFO is empty.<br>1 Receive FIFO is not empty. |
| 1 | TNF | RO | 1 | SSI Transmit FIFO Not Full<br>The TNF values are defined as follows:<br><br>Value Description<br>0 Transmit FIFO is full.<br>1 Transmit FIFO is not full. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | TFE | R0 | 1 | SSI Transmit FIFO Empty<br>The TFE values are defined as follows:<br><br>Value Description<br>0 Transmit FIFO is not empty.<br>1 Transmit FIFO is empty. |

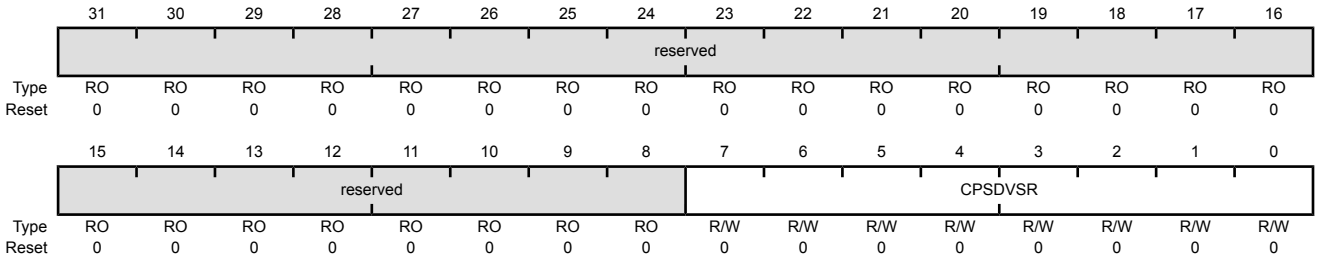
## Register 5: SSI Clock Prescale (SSICPSR), offset 0x010

**SSICPSR** is the clock prescale register and specifies the division factor by which the system clock must be internally divided before further use.

The value programmed into this register must be an even number between 2 and 254. The least-significant bit of the programmed number is hard-coded to zero. If an odd number is written to this register, data read back from this register has the least-significant bit as zero.

SSI Clock Prescale (SSICPSR)

SSI0 base: 0x4000.8000
Offset 0x010
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CPSDVSR | R/W | 0x00 | SSI Clock Prescale Divisor<br>This value must be an even number from 2 to 254, depending on the frequency of SSIClk. The LSB always returns 0 on reads. |

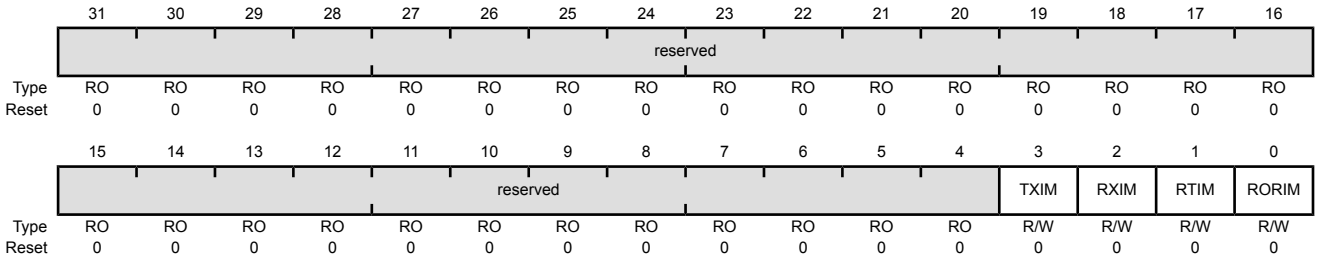
## Register 6: SSI Interrupt Mask (SSIIM), offset 0x014

The **SSIIM** register is the interrupt mask set or clear register. It is a read/write register and all bits are cleared to 0 on reset.

On a read, this register gives the current value of the mask on the relevant interrupt. A write of 1 to the particular bit sets the mask, enabling the interrupt to be read. A write of 0 clears the corresponding mask.

SSI Interrupt Mask (SSIIM)

SSI0 base: 0x4000.8000
Offset 0x014
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TXIM | R/W | 0 | SSI Transmit FIFO Interrupt Mask<br>The `TXIM` values are defined as follows:<br><br>Value Description<br>0 TX FIFO half-empty or less condition interrupt is masked.<br>1 TX FIFO half-empty or less condition interrupt is not masked. |
| 2 | RXIM | R/W | 0 | SSI Receive FIFO Interrupt Mask<br>The `RXIM` values are defined as follows:<br><br>Value Description<br>0 RX FIFO half-full or more condition interrupt is masked.<br>1 RX FIFO half-full or more condition interrupt is not masked. |
| 1 | RTIM | R/W | 0 | SSI Receive Time-Out Interrupt Mask<br>The `RTIM` values are defined as follows:<br><br>Value Description<br>0 RX FIFO time-out interrupt is masked.<br>1 RX FIFO time-out interrupt is not masked. |

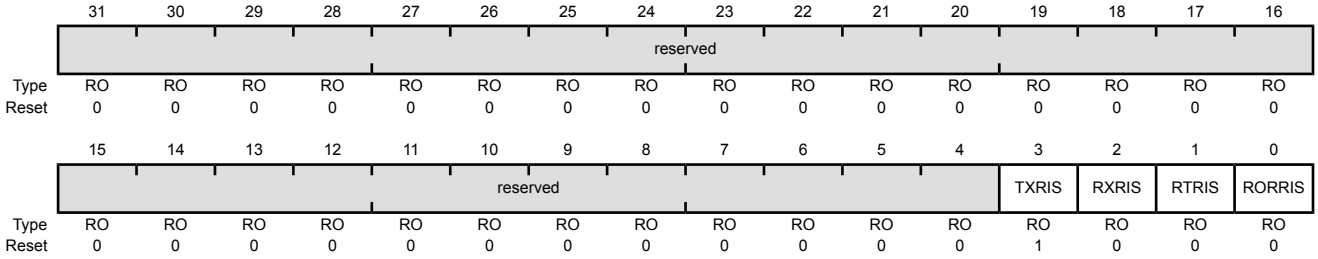
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | RORIM | R/W | 0 | SSI Receive Overrun Interrupt Mask<br>The RORIM values are defined as follows:<br><br>Value Description<br>0 RX FIFO overrun interrupt is masked.<br>1 RX FIFO overrun interrupt is not masked. |

## Register 7: SSI Raw Interrupt Status (SSIRIS), offset 0x018

The **SSIRIS** register is the raw interrupt status register. On a read, this register gives the current raw status value of the corresponding interrupt prior to masking. A write has no effect.

SSI Raw Interrupt Status (SSIRIS)

SSI0 base: 0x4000.8000
Offset 0x018
Type RO, reset 0x0000.0008



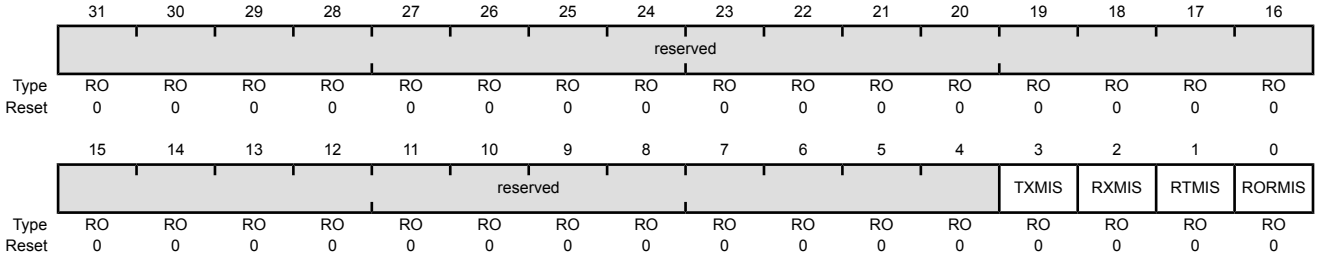

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TXRIS | RO | 1 | SSI Transmit FIFO Raw Interrupt Status<br>Indicates that the transmit FIFO is half empty or less, when set. |
| 2 | RXRIS | RO | 0 | SSI Receive FIFO Raw Interrupt Status<br>Indicates that the receive FIFO is half full or more, when set. |
| 1 | RTRIS | RO | 0 | SSI Receive Time-Out Raw Interrupt Status<br>Indicates that the receive time-out has occurred, when set. |
| 0 | RORRIS | RO | 0 | SSI Receive Overrun Raw Interrupt Status<br>Indicates that the receive FIFO has overflowed, when set. |

## Register 8: SSI Masked Interrupt Status (SSIMIS), offset 0x01C

The **SSIMIS** register is the masked interrupt status register. On a read, this register gives the current masked status value of the corresponding interrupt. A write has no effect.

SSI Masked Interrupt Status (SSIMIS)

SSI0 base: 0x4000.8000
Offset 0x01C
Type RO, reset 0x0000.0000



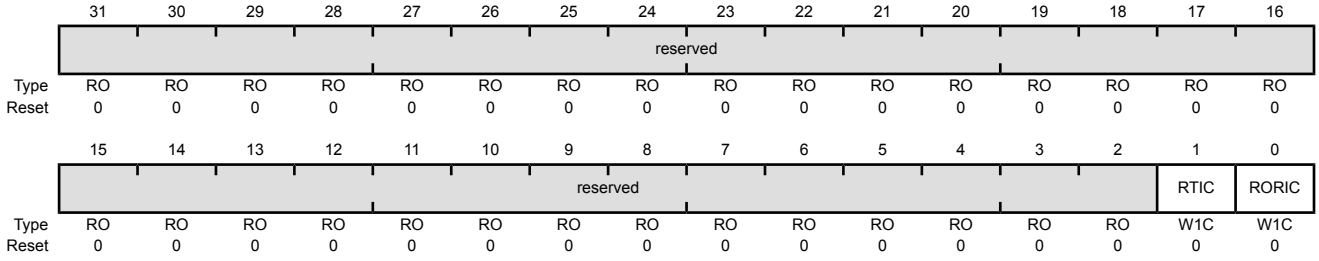

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TXMIS | RO | 0 | SSI Transmit FIFO Masked Interrupt Status<br>Indicates that the transmit FIFO is half empty or less, when set. |
| 2 | RXMIS | RO | 0 | SSI Receive FIFO Masked Interrupt Status<br>Indicates that the receive FIFO is half full or more, when set. |
| 1 | RTMIS | RO | 0 | SSI Receive Time-Out Masked Interrupt Status<br>Indicates that the receive time-out has occurred, when set. |
| 0 | RORMIS | RO | 0 | SSI Receive Overrun Masked Interrupt Status<br>Indicates that the receive FIFO has overflowed, when set. |

## Register 9: SSI Interrupt Clear (SSIICR), offset 0x020

The **SSIICR** register is the interrupt clear register. On a write of 1, the corresponding interrupt is cleared. A write of 0 has no effect.

SSI Interrupt Clear (SSIICR)

SSI0 base: 0x4000.8000
Offset 0x020
Type W1C, reset 0x0000.0000



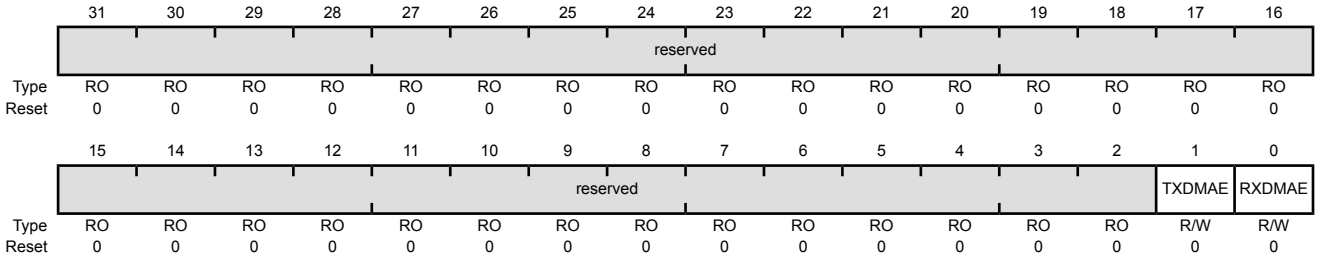

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | RTIC | W1C | 0 | SSI Receive Time-Out Interrupt Clear<br>The `RTIC` values are defined as follows:<br><br>Value Description<br>0 No effect on interrupt.<br>1 Clears interrupt. |
| 0 | RORIC | W1C | 0 | SSI Receive Overrun Interrupt Clear<br>The `RORIC` values are defined as follows:<br><br>Value Description<br>0 No effect on interrupt.<br>1 Clears interrupt. |

## Register 10: SSI DMA Control (SSIDMACTL), offset 0x024

The **SSIDMACTL** register is the DMA control register.

SSI DMA Control (SSIDMACTL)

SSI0 base: 0x4000.8000
Offset 0x024
Type R/W, reset 0x0000.0000



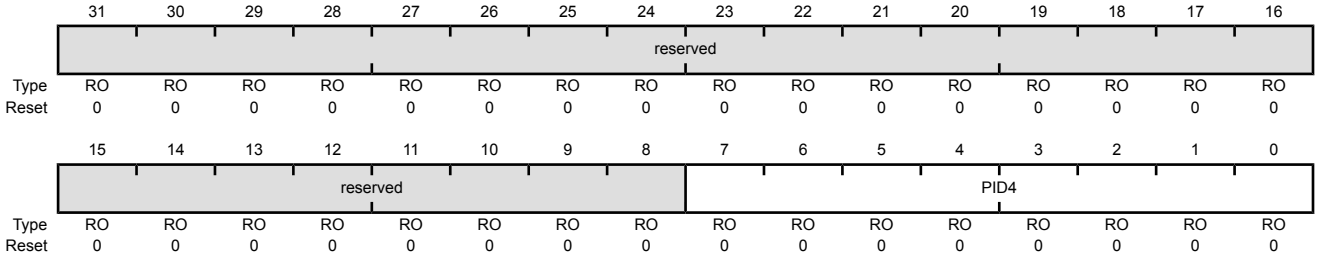

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | TXDMAE | R/W | 0 | Transmit DMA Enable<br>If this bit is set to 1, DMA for the transmit FIFO is enabled. |
| 0 | RXDMAE | R/W | 0 | Receive DMA Enable<br>If this bit is set to 1, DMA for the receive FIFO is enabled. |

## Register 11: SSI Peripheral Identification 4 (SSIPeriphID4), offset 0xFD0

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 4 (SSIPeriphID4)

SSI0 base: 0x4000.8000
Offset 0xFD0
Type RO, reset 0x0000.0000



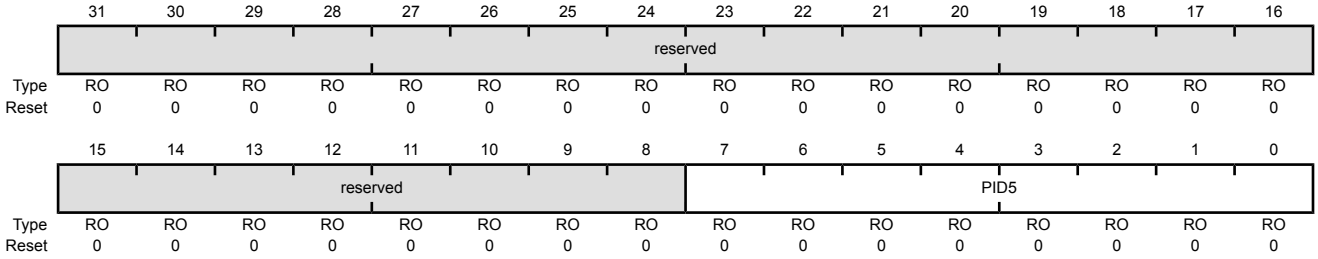

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID4 | RO | 0x00 | SSI Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

## Register 12: SSI Peripheral Identification 5 (SSIPeriphID5), offset 0xFD4

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 5 (SSIPeriphID5)

SSI0 base: 0x4000.8000
Offset 0xFD4
Type RO, reset 0x0000.0000



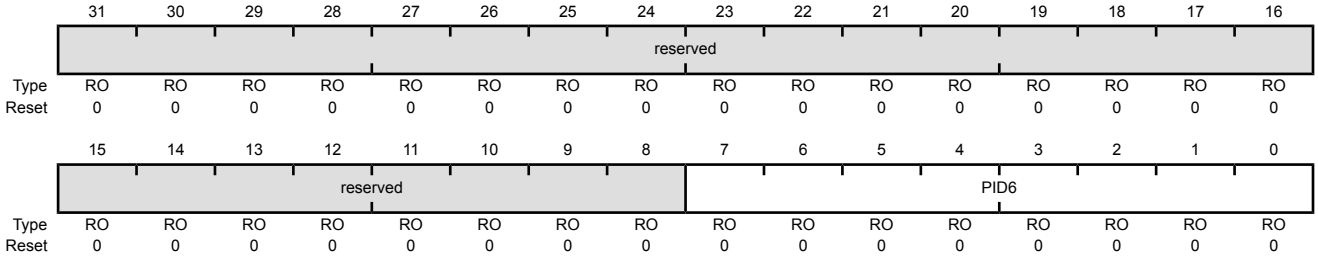

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID5 | RO | 0x00 | SSI Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

## Register 13: SSI Peripheral Identification 6 (SSIPeriphID6), offset 0xFD8

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 6 (SSIPeriphID6)

SSI0 base: 0x4000.8000
Offset 0xFD8
Type RO, reset 0x0000.0000



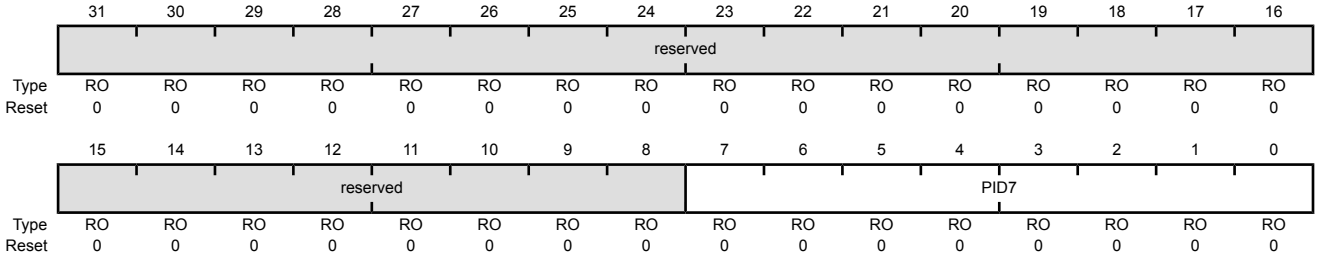

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID6 | RO | 0x00 | SSI Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 14: SSI Peripheral Identification 7 (SSIPeriphID7), offset 0xFDC

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 7 (SSIPeriphID7)

SSI0 base: 0x4000.8000
Offset 0xFDC
Type RO, reset 0x0000.0000



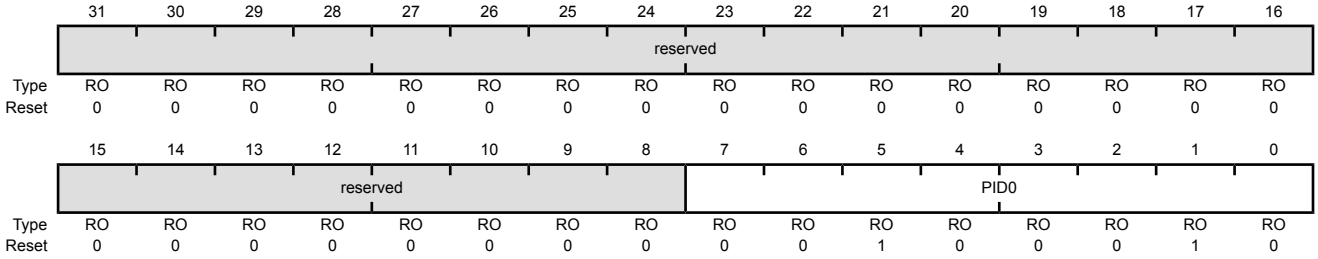

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID7 | RO | 0x00 | SSI Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

## Register 15: SSI Peripheral Identification 0 (SSIPeriphID0), offset 0xFE0

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 0 (SSIPeriphID0)

SSI0 base: 0x4000.8000
Offset 0xFE0
Type RO, reset 0x0000.0022



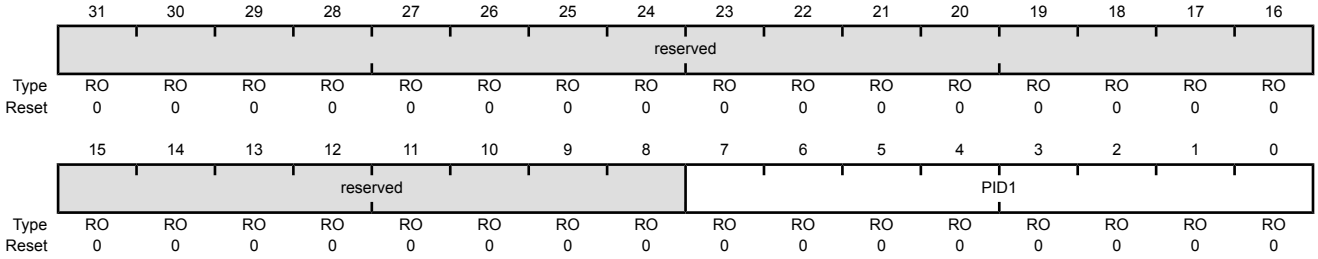

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID0 | RO | 0x22 | SSI Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

## Register 16: SSI Peripheral Identification 1 (SSIPeriphID1), offset 0xFE4

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 1 (SSIPeriphID1)

SSI0 base: 0x4000.8000
Offset 0xFE4
Type RO, reset 0x0000.0000



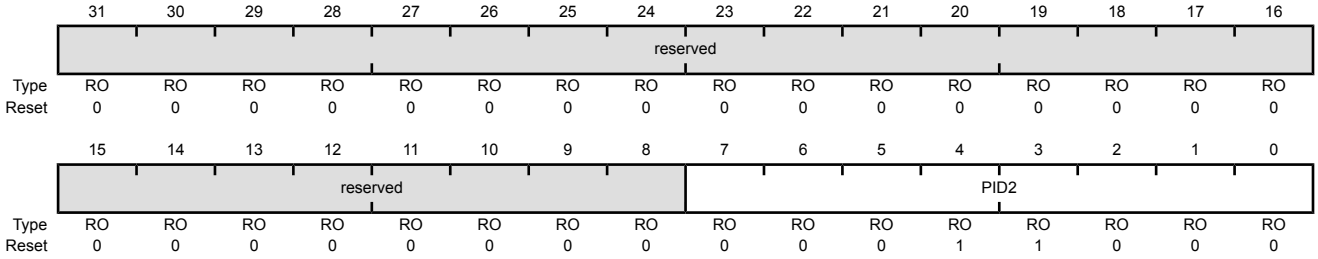

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID1 | RO | 0x00 | SSI Peripheral ID Register [15:8]<br>Can be used by software to identify the presence of this peripheral. |

## Register 17: SSI Peripheral Identification 2 (SSIPeriphID2), offset 0xFE8

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 2 (SSIPeriphID2)

SSI0 base: 0x4000.8000
Offset 0xFE8
Type RO, reset 0x0000.0018



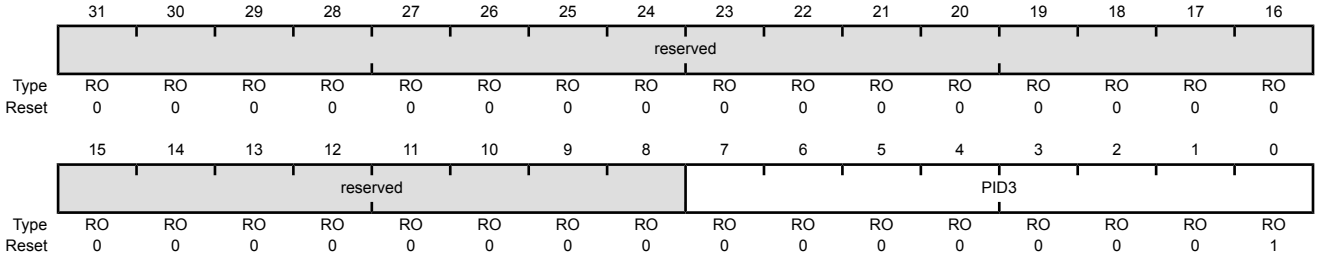

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID2 | RO | 0x18 | SSI Peripheral ID Register [23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 18: SSI Peripheral Identification 3 (SSIPeriphID3), offset 0xFEC

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 3 (SSIPeriphID3)

SSI0 base: 0x4000.8000
Offset 0xFEC
Type RO, reset 0x0000.0001



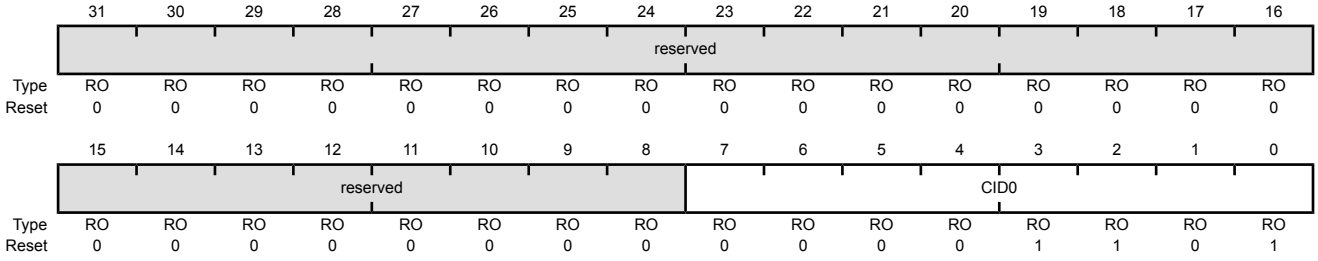

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID3 | RO | 0x01 | SSI Peripheral ID Register [31:24]<br>Can be used by software to identify the presence of this peripheral. |

## Register 19: SSI PrimeCell Identification 0 (SSIPCellID0), offset 0xFF0

The **SSIPCellIDn** registers are hard-coded, and the fields within the register determine the reset value.

SSI PrimeCell Identification 0 (SSIPCellID0)

SSI0 base: 0x4000.8000
Offset 0xFF0
Type RO, reset 0x0000.000D



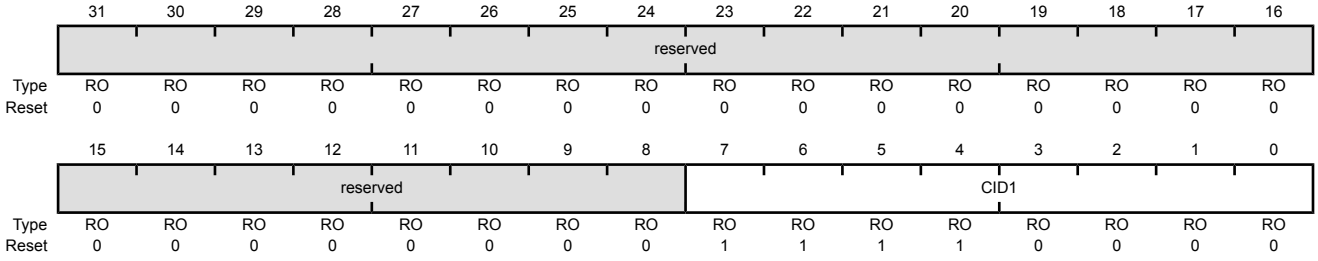

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID0 | RO | 0x0D | SSI PrimeCell ID Register [7:0]<br>Provides software a standard cross-peripheral identification system. |

## Register 20: SSI PrimeCell Identification 1 (SSIPCellID1), offset 0xFF4

The **SSIPCellIDn** registers are hard-coded, and the fields within the register determine the reset value.

SSI PrimeCell Identification 1 (SSIPCellID1)

SSI0 base: 0x4000.8000
Offset 0xFF4
Type RO, reset 0x0000.00F0



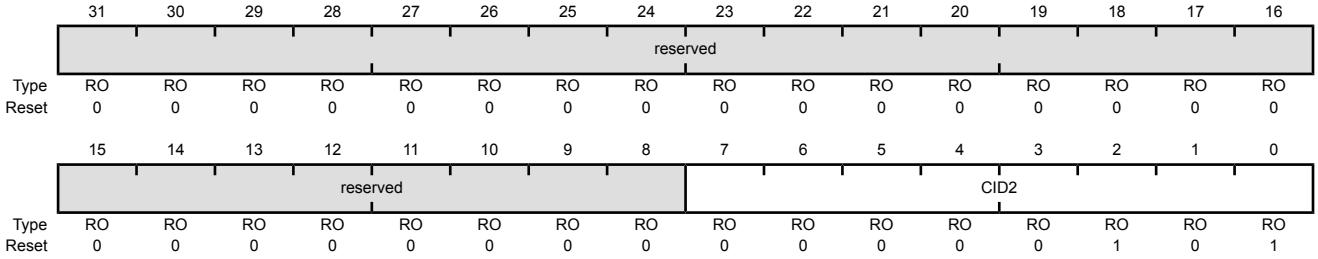

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID1 | RO | 0xF0 | SSI PrimeCell ID Register [15:8]<br>Provides software a standard cross-peripheral identification system. |

## Register 21: SSI PrimeCell Identification 2 (SSIPCellID2), offset 0xFF8

The **SSIPCellIDn** registers are hard-coded, and the fields within the register determine the reset value.

SSI PrimeCell Identification 2 (SSIPCellID2)

SSI0 base: 0x4000.8000
Offset 0xFF8
Type RO, reset 0x0000.0005



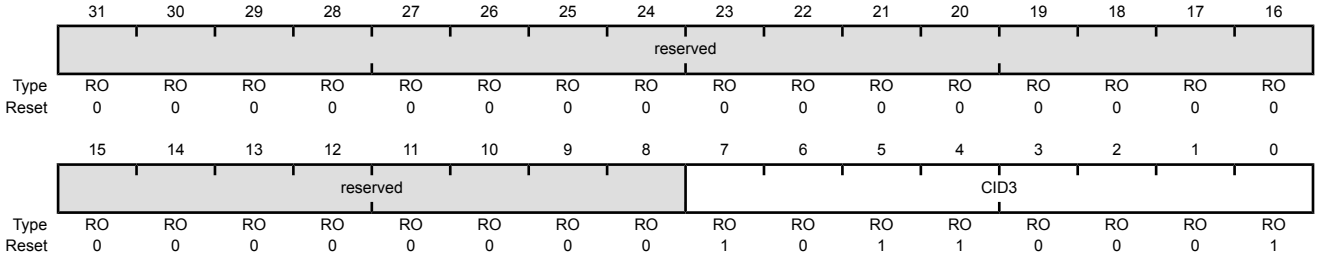

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID2 | RO | 0x05 | SSI PrimeCell ID Register [23:16]<br>Provides software a standard cross-peripheral identification system. |

## Register 22: SSI PrimeCell Identification 3 (SSIPCellID3), offset 0xFFC

The **SSIPCellIDn** registers are hard-coded, and the fields within the register determine the reset value.

SSI PrimeCell Identification 3 (SSIPCellID3)

SSI0 base: 0x4000.8000
Offset 0xFFC
Type RO, reset 0x0000.00B1




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID3 | RO | 0xB1 | SSI PrimeCell ID Register [31:24]<br>Provides software a standard cross-peripheral identification system. |

# 15 Inter-Integrated Circuit ($I^2C$) Interface

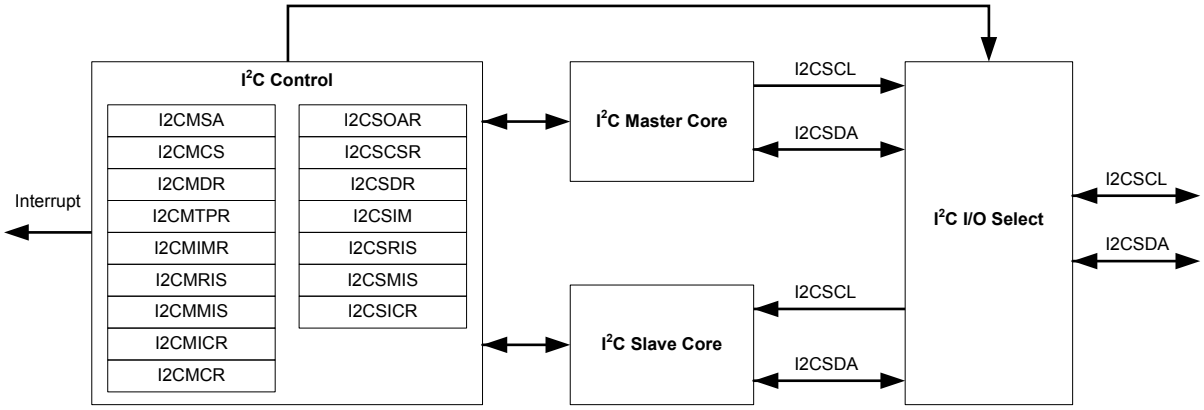
The Inter-Integrated Circuit ($I^2C$) bus provides bi-directional data transfer through a two-wire design (a serial data line SDA and a serial clock line SCL), and interfaces to external $I^2C$ devices such as serial memory (RAMs and ROMs), networking devices, LCDs, tone generators, and so on. The $I^2C$ bus may also be used for system testing and diagnostic purposes in product development and manufacture. The LM3S5632 microcontroller includes two $I^2C$ modules, providing the ability to interact (both send and receive) with other $I^2C$ devices on the bus.

The Stellaris® $I^2C$ interface has the following features:

- Two $I^2C$ modules, each with the following features:
- Devices on the $I^2C$ bus can be designated as either a master or a slave
  - Supports both sending and receiving data as either a master or a slave
  - Supports simultaneous master and slave operation
- Four $I^2C$ modes
  - Master transmit
  - Master receive
  - Slave transmit
  - Slave receive
- Two transmission speeds: Standard (100 Kbps) and Fast (400 Kbps)
- Master and slave interrupt generation
  - Master generates interrupts when a transmit or receive operation completes (or aborts due to an error)
  - Slave generates interrupts when data has been sent or requested by a master
- Master with arbitration and clock synchronization, multimaster support, and 7-bit addressing mode

## 15.1 Block Diagram

**Figure 15-1. I²C Block Diagram**




## 15.2 Signal Description

Table 15-1 on page 577 lists the external signals of the I²C interface and describes the function of each. The I²C interface signals are alternate functions for some GPIO signals and default to be GPIO signals at reset., with the exception of the I2C0SCL and I2CSDA pins which default to the I²C function. The column in the table below titled "Pin Assignment" lists the possible GPIO pin placements for the I²C signals. The AFSEL bit in the **GPIO Alternate Function Select (GPIOAFSEL)** register (page 371) should be set to choose the I²C function. Note that the I²C pins should be set to open drain using the **GPIO Open Drain Select (GPIOODR)** register. For more information on configuring GPIOs, see “General-Purpose Input/Outputs (GPIOs)” on page 351.

**Table 15-1. I2C Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| I2C0SCL | 47 | I/O | OD | I²C module 0 clock. |
| I2C0SDA | 27 | I/O | OD | I²C module 0 data. |
| I2C1SCL | 25 | I/O | OD | I²C module 1 clock. |
| I2C1SDA | 26 | I/O | OD | I²C module 1 data. |

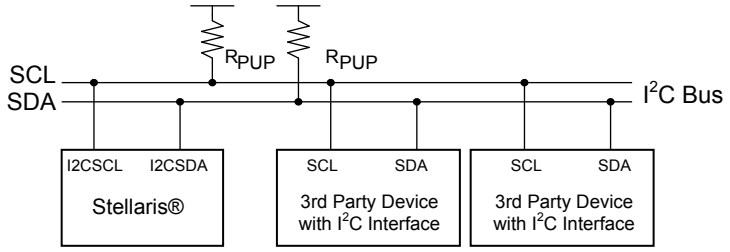
a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 15.3 Functional Description

Each I²C module is comprised of both master and slave functions which are implemented as separate peripherals. For proper operation, the SDA and SCL pins must be connected to bi-directional open-drain pads. A typical I²C bus configuration is shown in Figure 15-2 on page 578.

See “Inter-Integrated Circuit (I²C) Interface” on page 779 for I²C timing diagrams.

**Figure 15-2. I2C Bus Configuration**




## 15.3.1 I2C Bus Functional Overview

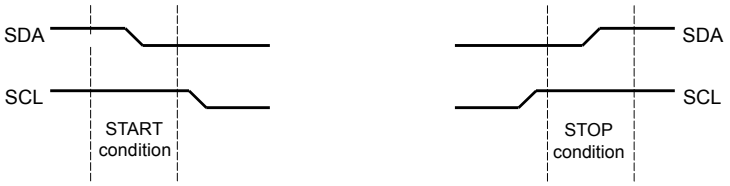
The I2C bus uses only two signals: SDA and SCL, named I2CSDA and I2CSCL on Stellaris microcontrollers. SDA is the bi-directional serial data line and SCL is the bi-directional serial clock line. The bus is considered idle when both lines are High.

Every transaction on the I2C bus is nine bits long, consisting of eight data bits and a single acknowledge bit. The number of bytes per transfer (defined as the time between a valid START and STOP condition, described in "START and STOP Conditions" on page 578) is unrestricted, but each byte has to be followed by an acknowledge bit, and data must be transferred MSB first. When a receiver cannot receive another complete byte, it can hold the clock line SCL Low and force the transmitter into a wait state. The data transfer continues when the receiver releases the clock SCL.

### 15.3.1.1 START and STOP Conditions

The protocol of the I2C bus defines two states to begin and end a transaction: START and STOP. A High-to-Low transition on the SDA line while the SCL is High is defined as a START condition, and a Low-to-High transition on the SDA line while SCL is High is defined as a STOP condition. The bus is considered busy after a START condition and free after a STOP condition. See Figure 15-3 on page 578.

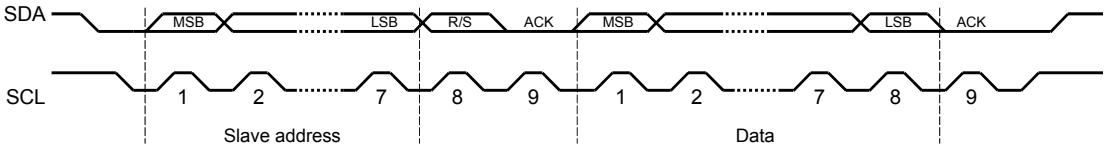
**Figure 15-3. START and STOP Conditions**




### 15.3.1.2 Data Format with 7-Bit Address

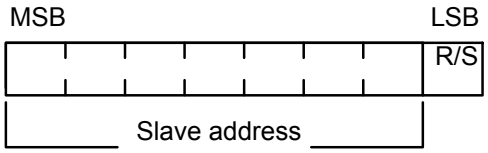
Data transfers follow the format shown in Figure 15-4 on page 579. After the START condition, a slave address is sent. This address is 7-bits long followed by an eighth bit, which is a data direction bit (R/S bit in the **I2CMSA** register). A zero indicates a transmit operation (send), and a one indicates a request for data (receive). A data transfer is always terminated by a STOP condition generated by the master, however, a master can initiate communications with another device on the bus by generating a repeated START condition and addressing another slave without first generating a STOP condition. Various combinations of receive/send formats are then possible within a single transfer.

**Figure 15-4. Complete Data Transfer with a 7-Bit Address**




The first seven bits of the first byte make up the slave address (see Figure 15-5 on page 579). The eighth bit determines the direction of the message. A zero in the R/S position of the first byte means that the master will write (send) data to the selected slave, and a one in this position means that the master will receive data from the slave.

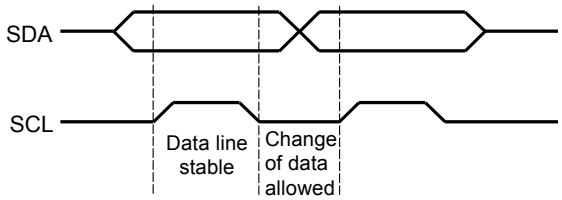
**Figure 15-5. R/S Bit in First Byte**




### 15.3.1.3 Data Validity

The data on the SDA line must be stable during the high period of the clock, and the data line can only change when SCL is Low (see Figure 15-6 on page 579).

**Figure 15-6. Data Validity During Bit Transfer on the I$^2$C Bus**




### 15.3.1.4 Acknowledge

All bus transactions have a required acknowledge clock cycle that is generated by the master. During the acknowledge cycle, the transmitter (which can be the master or slave) releases the SDA line. To acknowledge the transaction, the receiver must pull down SDA during the acknowledge clock cycle. The data sent out by the receiver during the acknowledge cycle must comply with the data validity requirements described in “Data Validity” on page 579.

When a slave receiver does not acknowledge the slave address, SDA must be left High by the slave so that the master can generate a STOP condition and abort the current transfer. If the master device is acting as a receiver during a transfer, it is responsible for acknowledging each transfer made by the slave. Since the master controls the number of bytes in the transfer, it signals the end of data to the slave transmitter by not generating an acknowledge on the last data byte. The slave transmitter must then release SDA to allow the master to generate the STOP or a repeated START condition.

### 15.3.1.5 Arbitration

A master may start a transfer only if the bus is idle. It's possible for two or more masters to generate a START condition within minimum hold time of the START condition. In these situations, an

arbitration scheme takes place on the SDA line, while SCL is High. During arbitration, the first of the competing master devices to place a '1' (High) on SDA while another master transmits a '0' (Low) will switch off its data output stage and retire until the bus is idle again.

Arbitration can take place over several bits. Its first stage is a comparison of address bits, and if both masters are trying to address the same device, arbitration continues on to the comparison of data bits.

### 15.3.2 Available Speed Modes

The I2C clock rate is determined by the parameters: CLK_PRD, TIMER_PRD, SCL_LP, and SCL_HP.

where:

CLK_PRD is the system clock period

SCL_LP is the low phase of SCL (fixed at 6)

SCL_HP is the high phase of SCL (fixed at 4)

TIMER_PRD is the programmed value in the **I2C Master Timer Period (I2CMTPR)** register (see page 598).

The I2C clock period is calculated as follows:

```
SCL_PERIOD = 2*(1 + TIMER_PRD)*(SCL_LP + SCL_HP)*CLK_PRD
```

For example:

```
CLK_PRD = 50 ns
TIMER_PRD = 2
SCL_LP=6
SCL_HP=4
```

yields a SCL frequency of:

```
1/T = 333 Khz
```

Table 15-2 on page 580 gives examples of timer period, system clock, and speed mode (Standard or Fast).

**Table 15-2. Examples of I2C Master Timer Period versus Speed Mode**

| System Clock | Timer Period | Standard Mode | Timer Period | Fast Mode |
|---|---|---|---|---|
| 4 MHz | 0x01 | 100 Kbps | - | - |
| 6 MHz | 0x02 | 100 Kbps | - | - |
| 12.5 MHz | 0x06 | 89 Kbps | 0x01 | 312 Kbps |
| 16.7 MHz | 0x08 | 93 Kbps | 0x02 | 278 Kbps |
| 20 MHz | 0x09 | 100 Kbps | 0x02 | 333 Kbps |
| 25 MHz | 0x0C | 96.2 Kbps | 0x03 | 312 Kbps |
| 33 MHz | 0x10 | 97.1 Kbps | 0x04 | 330 Kbps |
| 40 MHz | 0x13 | 100 Kbps | 0x04 | 400 Kbps |
| 50 MHz | 0x18 | 100 Kbps | 0x06 | 357 Kbps |

### 15.3.3 Interrupts

The I²C can generate interrupts when the following conditions are observed:

- Master transaction completed
- Master arbitration lost
- Master transaction error
- Slave transaction received
- Slave transaction requested

There is a separate interrupt signal for the I²C master and I²C slave modules. While both modules can generate interrupts for multiple conditions, only a single interrupt signal is sent to the interrupt controller.

#### 15.3.3.1 I²C Master Interrupts

The I²C master module generates an interrupt when a transaction completes (either transmit or receive), when arbitration is lost, or when an error occurs during a transaction. To enable the I²C master interrupt, software must set the IM bit in the **I²C Master Interrupt Mask (I2CMIMR)** register. When an interrupt condition is met, software must check the ERROR and ARBLST bits in the **I²C Master Control/Status (I2CMCS)** register to verify that an error didn't occur during the last transaction and to ensure that arbitration has not been lost. An error condition is asserted if the last transaction wasn't acknowledged by the slave. If an error is not detected and the master has not lost arbitration, the application can proceed with the transfer. The interrupt is cleared by writing a 1 to the IC bit in the **I²C Master Interrupt Clear (I2CMICR)** register.

If the application doesn't require the use of interrupts, the raw interrupt status is always visible via the **I²C Master Raw Interrupt Status (I2CMRIS)** register.

#### 15.3.3.2 I²C Slave Interrupts

The slave module can generate an interrupt when data has been received or requested. This interrupt is enabled by writing a 1 to the DATAIM bit in the **I²C Slave Interrupt Mask (I2CSIMR)** register. Software determines whether the module should write (transmit) or read (receive) data from the **I²C Slave Data (I2CSDR)** register, by checking the RREQ and TREQ bits of the **I²C Slave Control/Status (I2CSCSR)** register. If the slave module is in receive mode and the first byte of a transfer is received, the FBR bit is set along with the RREQ bit. The interrupt is cleared by writing a 1 to the DATAIC bit in the **I²C Slave Interrupt Clear (I2CSICR)** register.

In addition, the slave module can generate an interrupt when a start and stop condition is detected. These interrupts are enabled by writing a 1 to the STARTIM and STOPIM bits of the **I²C Slave Interrupt Mask (I2CSIMR)** register and cleared by writing a 1 to the STOPIC and STARTIC bits of the **I²C Slave Interrupt Clear (I2CSICR)** register.

If the application doesn't require the use of interrupts, the raw interrupt status is always visible via the **I²C Slave Raw Interrupt Status (I2CSRIS)** register.

### 15.3.4 Loopback Operation

The I²C modules can be placed into an internal loopback mode for diagnostic or debug work. This is accomplished by setting the LPBK bit in the **I²C Master Configuration (I2CMCR)** register. In loopback mode, the SDA and SCL signals from the master and slave modules are tied together.

### 15.3.5 Command Sequence Flow Charts

This section details the steps required to perform the various $I^2C$ transfer types in both master and slave mode.

#### 15.3.5.1 $I^2C$ Master Command Sequences

The figures that follow show the command sequences available for the $I^2C$ master.

**Figure 15-7. Master Single SEND**

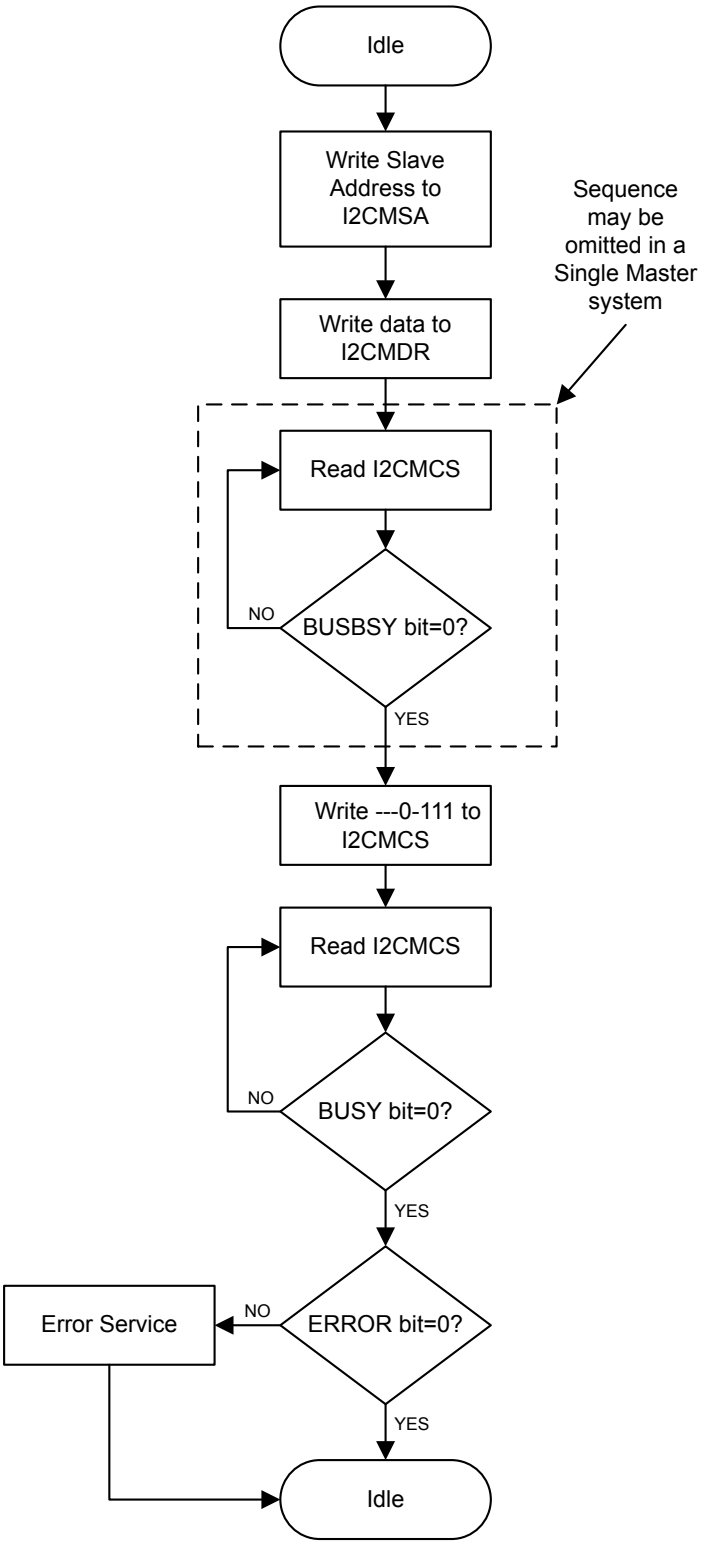


**Figure 15-8. Master Single RECEIVE**

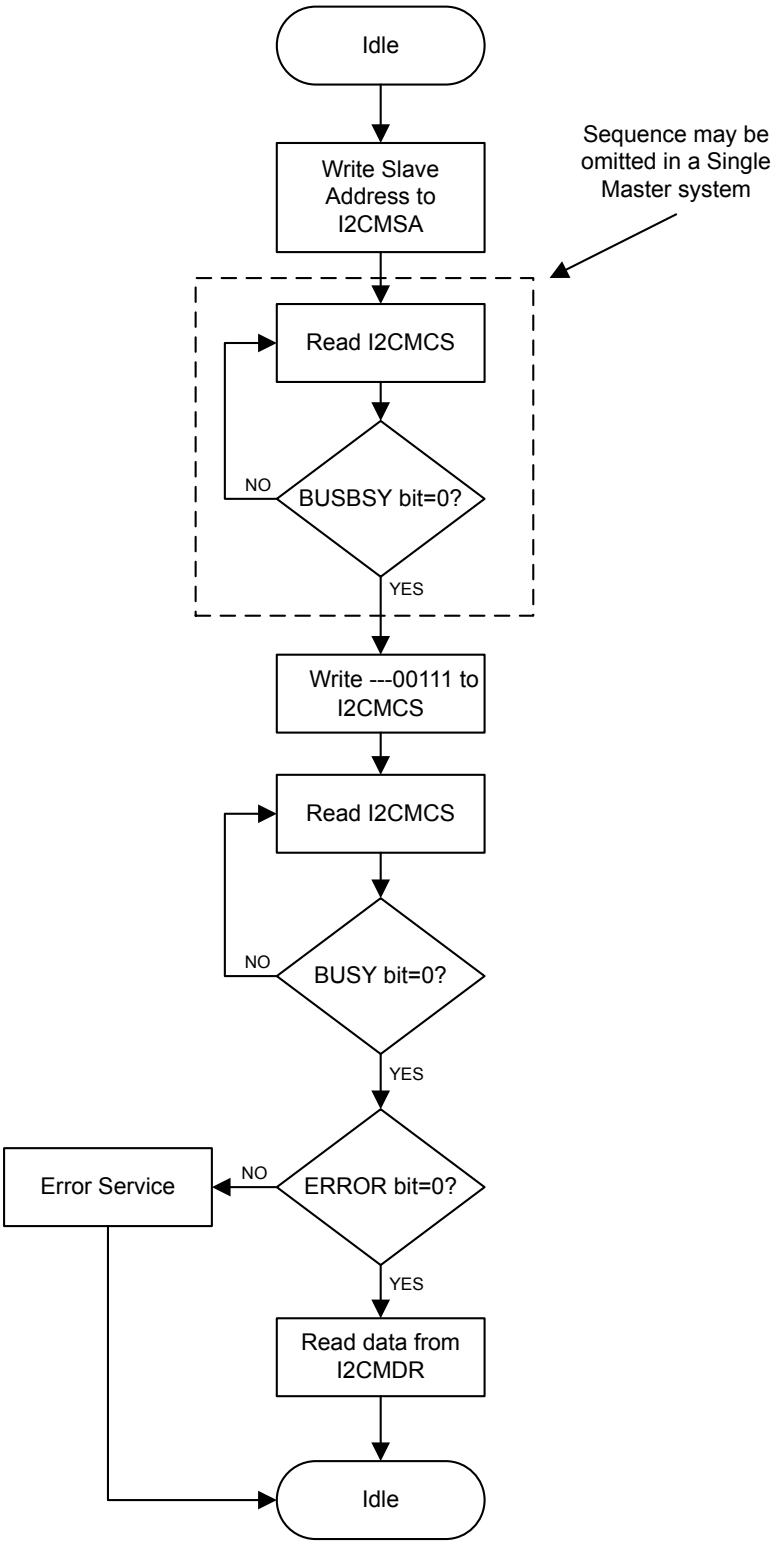


**Figure 15-9. Master Burst SEND**

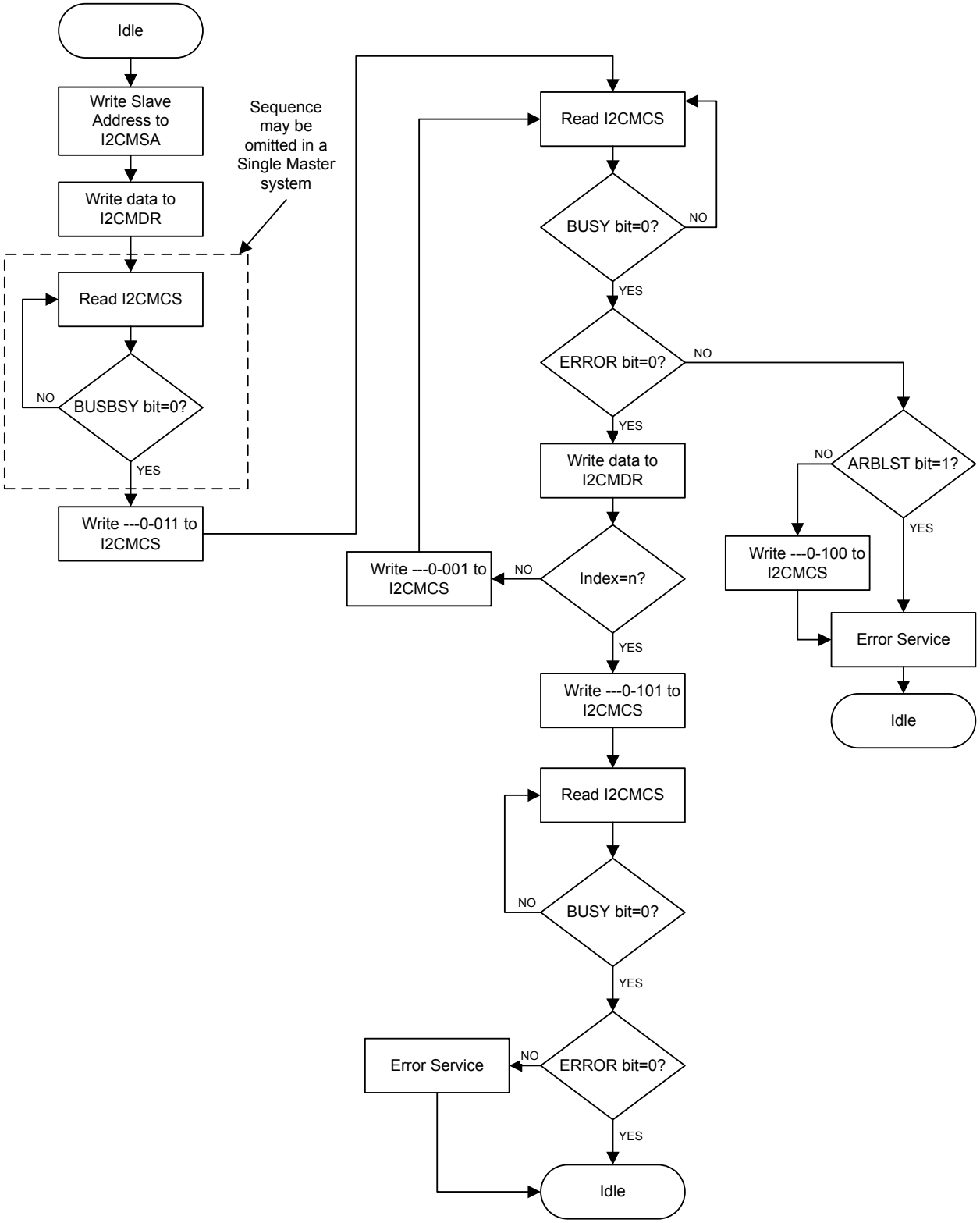


**Figure 15-10. Master Burst RECEIVE**

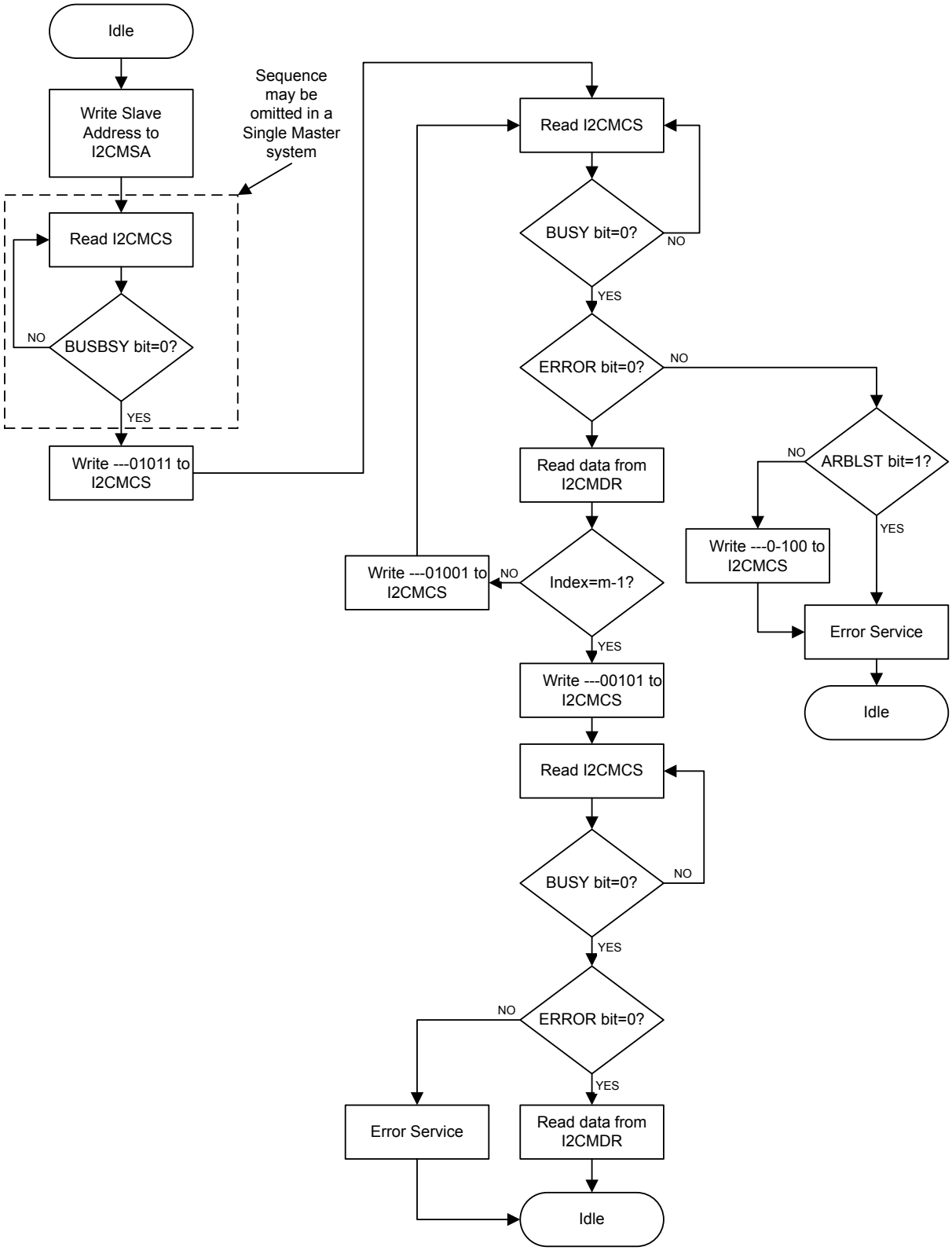


**Figure 15-11. Master Burst RECEIVE after Burst SEND**

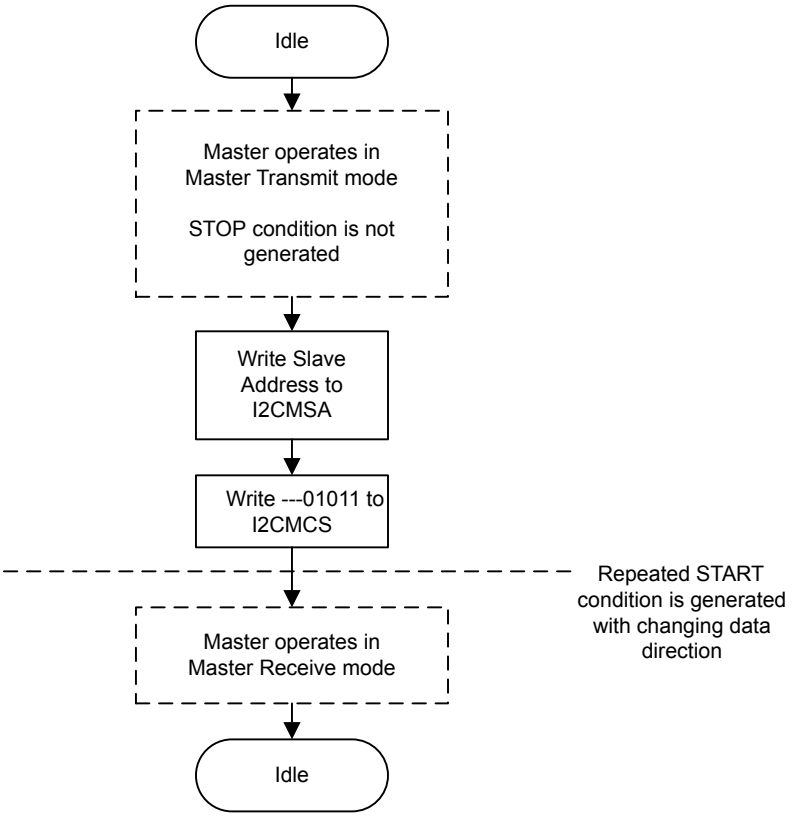
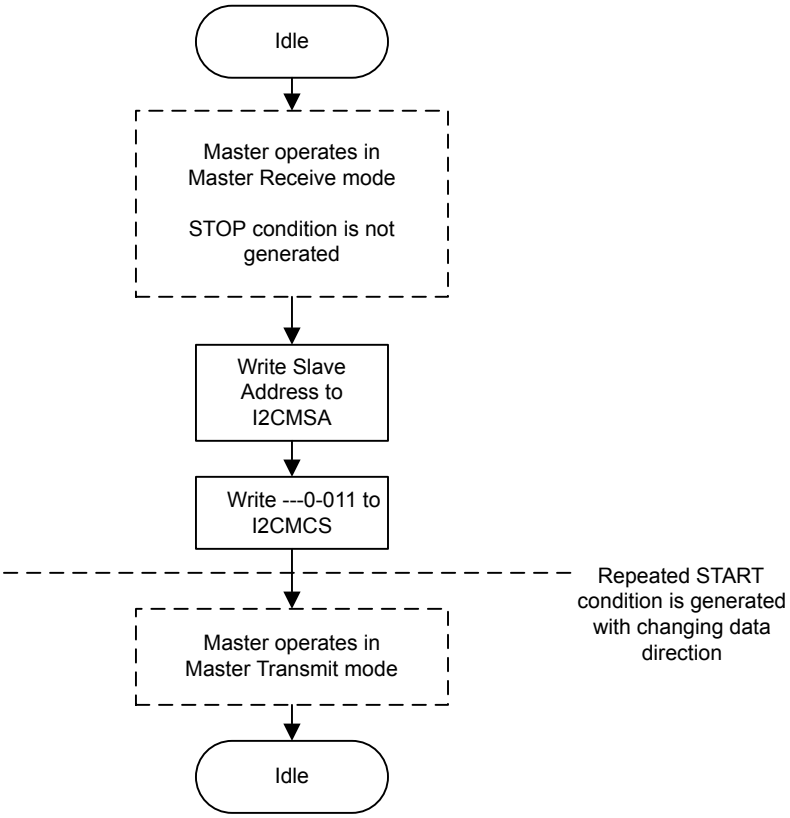


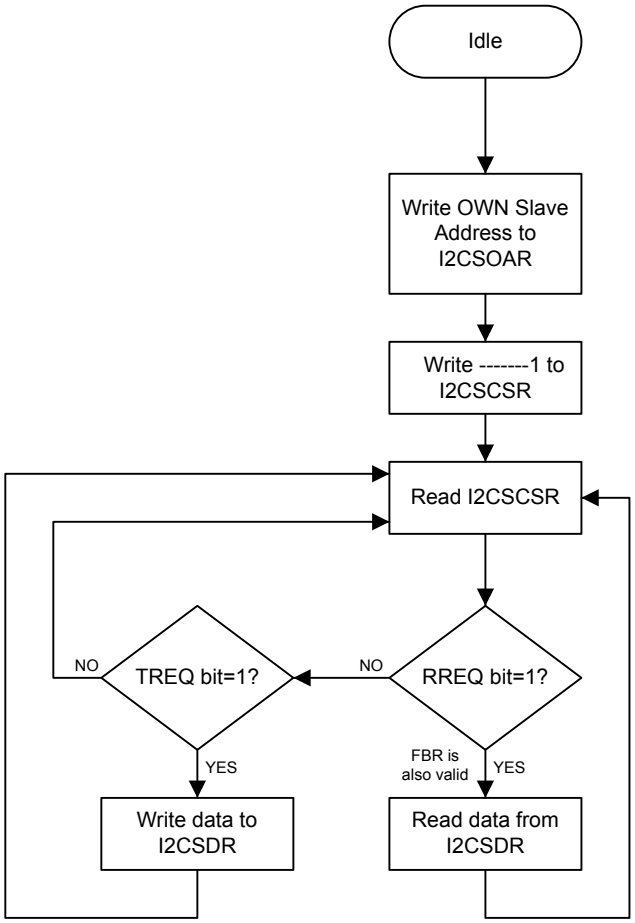
**Figure 15-12. Master Burst SEND after Burst RECEIVE**




### 15.3.5.2 I2C Slave Command Sequences

Figure 15-13 on page 589 presents the command sequence available for the I2C slave.

**Figure 15-13. Slave Command Sequence**




## 15.4 Initialization and Configuration

The following example shows how to configure the I$^2$C module to send a single byte as a master. This assumes the system clock is 20 MHz.

1. Enable the I$^2$C clock by writing a value of 0x0000.1000 to the **RCGC1** register in the System Control module.
2. Enable the clock to the appropriate GPIO module via the **RCGC2** register in the System Control module.
3. In the GPIO module, enable the appropriate pins for their alternate function using the **GPIOAFSEL** register. Also, be sure to enable the same pins for Open Drain operation.
4. Initialize the I$^2$C Master by writing the **I2CMCR** register with a value of 0x0000.0020.
5. Set the desired SCL clock speed of 100 Kbps by writing the **I2CMTPR** register with the correct value. The value written to the **I2CMTPR** register represents the number of system clock periods in one SCL clock period. The TPR value is determined by the following equation:

```
TPR = (System Clock / (2 * (SCL_LP + SCL_HP) * SCL_CLK)) - 1;
TPR = (20MHz / (2 * (6 + 4) * 100000)) - 1;
TPR = 9
```

Write the **I2CMTPR** register with the value of 0x0000.0009.

6. Specify the slave address of the master and that the next operation will be a Send by writing the **I2CMSA** register with a value of 0x0000.0076. This sets the slave address to 0x3B.
7. Place data (byte) to be sent in the data register by writing the **I2CMDR** register with the desired data.
8. Initiate a single byte send of the data from Master to Slave by writing the **I2CMCS** register with a value of 0x0000.0007 (STOP, START, RUN).
9. Wait until the transmission completes by polling the **I2CMCS** register's BUSBSY bit until it has been cleared.

## 15.5 Register Map

Table 15-3 on page 590 lists the I2C registers. All addresses given are relative to the I2C base addresses for the master and slave:

- I2C 0: 0x4002.0000
- I2C 1: 0x4002.1000

Note that the I2C module clock must be enabled before the registers can be programmed (see page 223). There must be a delay of 3 system clocks after the I2C module clock is enabled before any I2C module registers are accessed.

The hw_i2c.h file in the StellarisWare® Driver Library uses a base address of 0x800 for the I2C slave registers. Be aware when using registers with offsets between 0x800 and 0x818 that StellarisWare uses an offset between 0x000 and 0x018 with the slave base address.

**Table 15-3. Inter-Integrated Circuit (I2C) Interface Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| **I2C Master** | | | | | |
| 0x000 | I2CMSA | R/W | 0x0000.0000 | I2C Master Slave Address | 592 |
| 0x004 | I2CMCS | R/W | 0x0000.0000 | I2C Master Control/Status | 593 |
| 0x008 | I2CMDR | R/W | 0x0000.0000 | I2C Master Data | 597 |
| 0x00C | I2CMTPR | R/W | 0x0000.0001 | I2C Master Timer Period | 598 |
| 0x010 | I2CMIMR | R/W | 0x0000.0000 | I2C Master Interrupt Mask | 599 |
| 0x014 | I2CMRIS | RO | 0x0000.0000 | I2C Master Raw Interrupt Status | 600 |
| 0x018 | I2CMMIS | RO | 0x0000.0000 | I2C Master Masked Interrupt Status | 601 |
| 0x01C | I2CMICR | WO | 0x0000.0000 | I2C Master Interrupt Clear | 602 |
| 0x020 | I2CMCR | R/W | 0x0000.0000 | I2C Master Configuration | 603 |

**Table 15-3. Inter-Integrated Circuit ($I^2C$) Interface Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| **$I^2C$ Slave** | | | | | |
| 0x800 | I2CSOAR | R/W | 0x0000.0000 | I2C Slave Own Address | 605 |
| 0x804 | I2CSCSR | RO | 0x0000.0000 | I2C Slave Control/Status | 606 |
| 0x808 | I2CSDR | R/W | 0x0000.0000 | I2C Slave Data | 608 |
| 0x80C | I2CSIMR | R/W | 0x0000.0000 | I2C Slave Interrupt Mask | 609 |
| 0x810 | I2CSRIS | RO | 0x0000.0000 | I2C Slave Raw Interrupt Status | 610 |
| 0x814 | I2CSMIS | RO | 0x0000.0000 | I2C Slave Masked Interrupt Status | 611 |
| 0x818 | I2CSICR | WO | 0x0000.0000 | I2C Slave Interrupt Clear | 612 |

## 15.6 Register Descriptions ($I^2C$ Master)

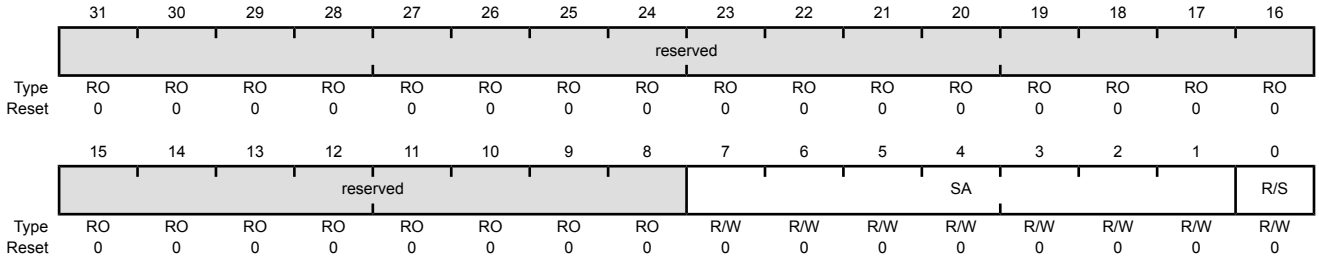
The remainder of this section lists and describes the $I^2C$ master registers, in numerical order by address offset. See also “Register Descriptions ($I^2C$ Slave)” on page 604.

## Register 1: I2C Master Slave Address (I2CMSA), offset 0x000

This register consists of eight bits: seven address bits (A6-A0), and a Receive/Send bit, which determines if the next operation is a Receive (High), or Send (Low).

I2C Master Slave Address (I2CMSA)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x000
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:1 | SA | R/W | 0 | I2C Slave Address<br>This field specifies bits A6 through A0 of the slave address. |
| 0 | R/S | R/W | 0 | Receive/Send<br>The R/S bit specifies if the next operation is a Receive (High) or Send (Low).<br><br>Value Description<br>0 Send.<br>1 Receive. |

## Register 2: I²C Master Control/Status (I2CMCS), offset 0x004

This register accesses four control bits when written, and accesses seven status bits when read.

The status register consists of seven bits, which when read determine the state of the I²C bus controller.

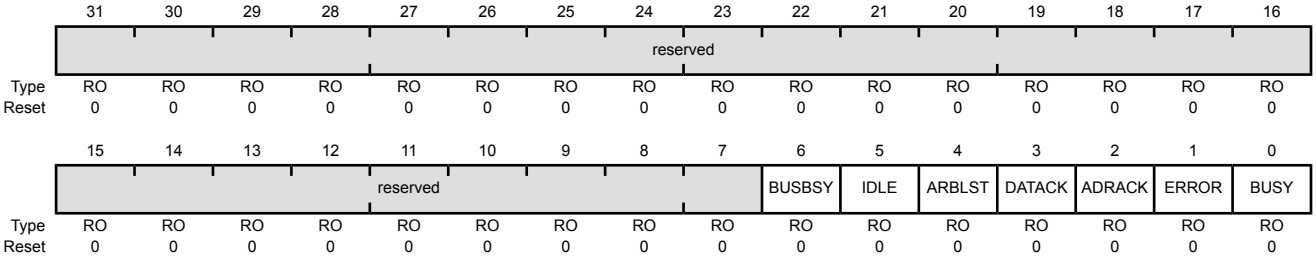
The control register consists of four bits: the RUN, START, STOP, and ACK bits. The START bit causes the generation of the START, or REPEATED START condition.

The STOP bit determines if the cycle stops at the end of the data cycle, or continues on to a burst. To generate a single send cycle, the **I²C Master Slave Address (I2CMSA)** register is written with the desired address, the R/S bit is set to 0, and the Control register is written with ACK=X (0 or 1), STOP=1, START=1, and RUN=1 to perform the operation and stop. When the operation is completed (or aborted due an error), the interrupt pin becomes active and the data may be read from the **I2CMDR** register. When the I²C module operates in Master receiver mode, the ACK bit must be set normally to logic 1. This causes the I²C bus controller to send an acknowledge automatically after each byte. This bit must be reset when the I²C bus controller requires no further data to be sent from the slave transmitter.

### Reads

I2C Master Control/Status (I2CMCS)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x004
Type RO, reset 0x0000.0000



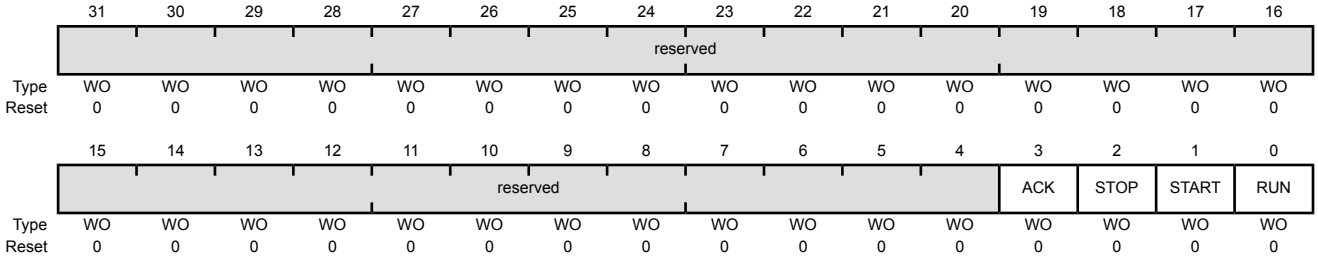

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | BUSBSY | RO | 0 | Bus Busy<br>This bit specifies the state of the I²C bus. If set, the bus is busy; otherwise, the bus is idle. The bit changes based on the START and STOP conditions. |
| 5 | IDLE | RO | 0 | I²C Idle<br>This bit specifies the I²C controller state. If set, the controller is idle; otherwise the controller is not idle. |
| 4 | ARBLST | RO | 0 | Arbitration Lost<br>This bit specifies the result of bus arbitration. If set, the controller lost arbitration; otherwise, the controller won arbitration. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | DATACK | RO | 0 | Acknowledge Data<br>This bit specifies the result of the last data operation. If set, the transmitted data was not acknowledged; otherwise, the data was acknowledged. |
| 2 | ADRACK | RO | 0 | Acknowledge Address<br>This bit specifies the result of the last address operation. If set, the transmitted address was not acknowledged; otherwise, the address was acknowledged. |
| 1 | ERROR | RO | 0 | Error<br>This bit specifies the result of the last bus operation. If set, an error occurred on the last operation; otherwise, no error was detected. The error can be from the slave address not being acknowledged or the transmit data not being acknowledged. |
| 0 | BUSY | RO | 0 | I2C Busy<br>This bit specifies the state of the controller. If set, the controller is busy; otherwise, the controller is idle. When the BUSY bit is set, the other status bits are not valid. |

## Writes

I2C Master Control/Status (I2CMCS)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x004
Type WO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | WO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | ACK | WO | 0 | Data Acknowledge Enable<br>When set, causes received data byte to be acknowledged automatically by the master. See field decoding in Table 15-4 on page 595. |
| 2 | STOP | WO | 0 | Generate STOP<br>When set, causes the generation of the STOP condition. See field decoding in Table 15-4 on page 595. |
| 1 | START | WO | 0 | Generate START<br>When set, causes the generation of a START or repeated START condition. See field decoding in Table 15-4 on page 595. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | RUN | WO | 0 | I²C Master Enable<br>When set, allows the master to send or receive data. See field decoding in Table 15-4 on page 595. |

**Table 15-4. Write Field Decoding for I2CMCS[3:0] Field (Sheet 1 of 3)**

| Current State | I2CMSA[0] | I2CMCS[3:0] | | | | Description |
|---|---|---|---|---|---|---|
| | R/S | ACK | STOP | START | RUN | |
| Idle | 0 | X[a] | 0 | 1 | 1 | START condition followed by SEND (master goes to the Master Transmit state). |
| | 0 | X | 1 | 1 | 1 | START condition followed by a SEND and STOP condition (master remains in Idle state). |
| | 1 | 0 | 0 | 1 | 1 | START condition followed by RECEIVE operation with negative ACK (master goes to the Master Receive state). |
| | 1 | 0 | 1 | 1 | 1 | START condition followed by RECEIVE and STOP condition (master remains in Idle state). |
| | 1 | 1 | 0 | 1 | 1 | START condition followed by RECEIVE (master goes to the Master Receive state). |
| | 1 | 1 | 1 | 1 | 1 | Illegal. |
| | All other combinations not listed are non-operations. | | | | | NOP. |
| Master Transmit | X | X | 0 | 0 | 1 | SEND operation (master remains in Master Transmit state). |
| | X | X | 1 | 0 | 0 | STOP condition (master goes to Idle state). |
| | X | X | 1 | 0 | 1 | SEND followed by STOP condition (master goes to Idle state). |
| | 0 | X | 0 | 1 | 1 | Repeated START condition followed by a SEND (master remains in Master Transmit state). |
| | 0 | X | 1 | 1 | 1 | Repeated START condition followed by SEND and STOP condition (master goes to Idle state). |
| | 1 | 0 | 0 | 1 | 1 | Repeated START condition followed by a RECEIVE operation with a negative ACK (master goes to Master Receive state). |
| | 1 | 0 | 1 | 1 | 1 | Repeated START condition followed by a SEND and STOP condition (master goes to Idle state). |
| | 1 | 1 | 0 | 1 | 1 | Repeated START condition followed by RECEIVE (master goes to Master Receive state). |
| | 1 | 1 | 1 | 1 | 1 | Illegal. |
| | All other combinations not listed are non-operations. | | | | | NOP. |

**Table 15-4. Write Field Decoding for I2CMCS[3:0] Field (Sheet 1 of 3) *(continued)***

| Current State | I2CMSA[0] | I2CMCS[3:0] | | | | Description |
|---|---|---|---|---|---|---|
| | R/S | ACK | STOP | START | RUN | |
| Master Receive | X | 0 | 0 | 0 | 1 | RECEIVE operation with negative ACK (master remains in Master Receive state). |
| | X | X | 1 | 0 | 0 | STOP condition (master goes to Idle state).[b] |
| | X | 0 | 1 | 0 | 1 | RECEIVE followed by STOP condition (master goes to Idle state). |
| | X | 1 | 0 | 0 | 1 | RECEIVE operation (master remains in Master Receive state). |
| | X | 1 | 1 | 0 | 1 | Illegal. |
| | 1 | 0 | 0 | 1 | 1 | Repeated START condition followed by RECEIVE operation with a negative ACK (master remains in Master Receive state). |
| | 1 | 0 | 1 | 1 | 1 | Repeated START condition followed by RECEIVE and STOP condition (master goes to Idle state). |
| | 1 | 1 | 0 | 1 | 1 | Repeated START condition followed by RECEIVE (master remains in Master Receive state). |
| | 0 | X | 0 | 1 | 1 | Repeated START condition followed by SEND (master goes to Master Transmit state). |
| | 0 | X | 1 | 1 | 1 | Repeated START condition followed by SEND and STOP condition (master goes to Idle state). |
| | All other combinations not listed are non-operations. | | | | | NOP. |

a. An X in a table cell indicates the bit can be 0 or 1.

b. In Master Receive mode, a STOP condition should be generated only after a Data Negative Acknowledge executed by the master or an Address Negative Acknowledge executed by the slave.

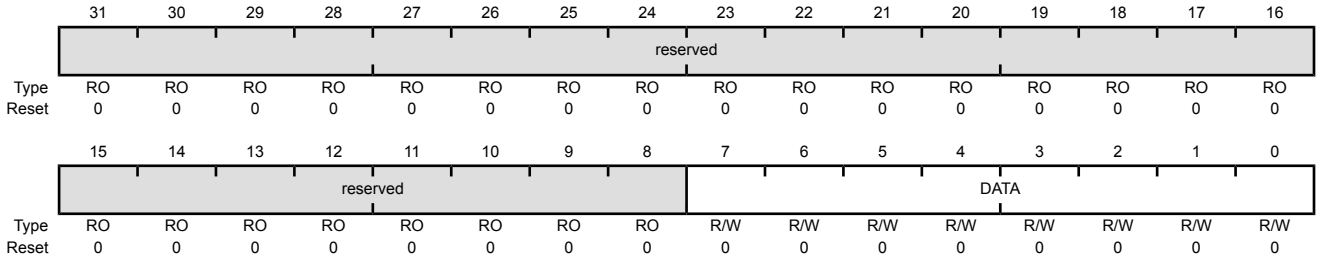
## Register 3: I²C Master Data (I2CMDR), offset 0x008

**Important:** This register is read-sensitive. See the register description for details.

This register contains the data to be transmitted when in the Master Transmit state, and the data received when in the Master Receive state.

I2C Master Data (I2CMDR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x008
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DATA | R/W | 0x00 | Data Transferred<br>Data transferred during transaction. |

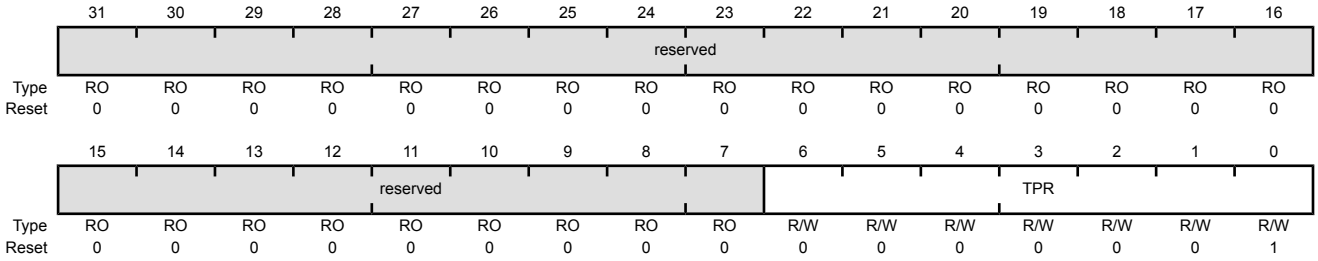
## Register 4: I2C Master Timer Period (I2CMTPR), offset 0x00C

This register specifies the period of the SCL clock.

**Caution – Take care not to set bit 7 when accessing this register as unpredictable behavior can occur.**

I2C Master Timer Period (I2CMTPR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x00C
Type R/W, reset 0x0000.0001



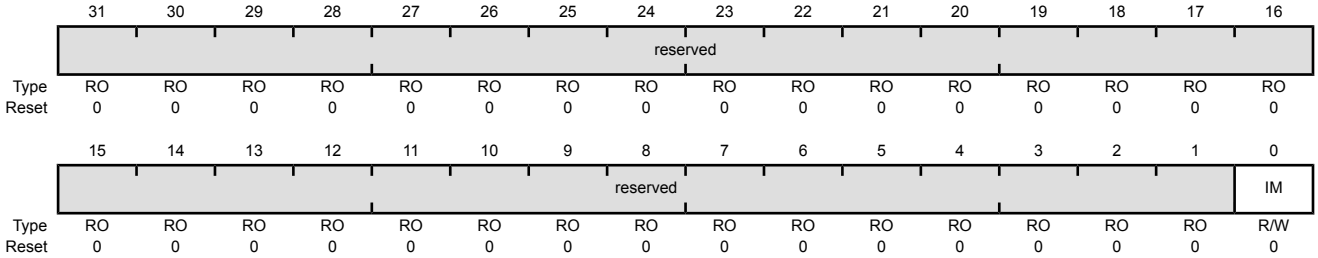

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | TPR | R/W | 0x1 | SCL Clock Period<br>This field specifies the period of the SCL clock.<br>`SCL_PRD = 2*(1 + TPR)*(SCL_LP + SCL_HP)*CLK_PRD`<br>where:<br>SCL_PRD is the SCL line period (I2C clock).<br>TPR is the Timer Period register value (range of 1 to 127).<br>SCL_LP is the SCL Low period (fixed at 6).<br>SCL_HP is the SCL High period (fixed at 4). |

## Register 5: I²C Master Interrupt Mask (I2CMIMR), offset 0x010

This register controls whether a raw interrupt is promoted to a controller interrupt.

I2C Master Interrupt Mask (I2CMIMR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x010
Type R/W, reset 0x0000.0000



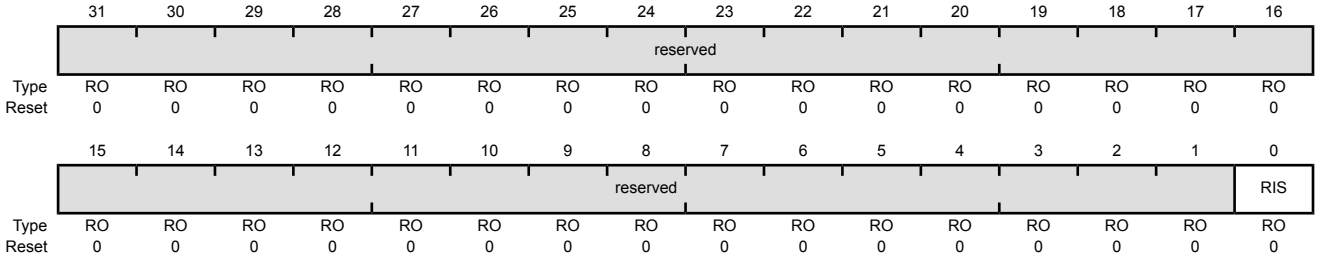

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | IM | R/W | 0 | Interrupt Mask<br>This bit controls whether a raw interrupt is promoted to a controller interrupt. If set, the interrupt is not masked and the interrupt is promoted; otherwise, the interrupt is masked. |

## Register 6: I²C Master Raw Interrupt Status (I2CMRIS), offset 0x014

This register specifies whether an interrupt is pending.

I2C Master Raw Interrupt Status (I2CMRIS)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x014
Type RO, reset 0x0000.0000



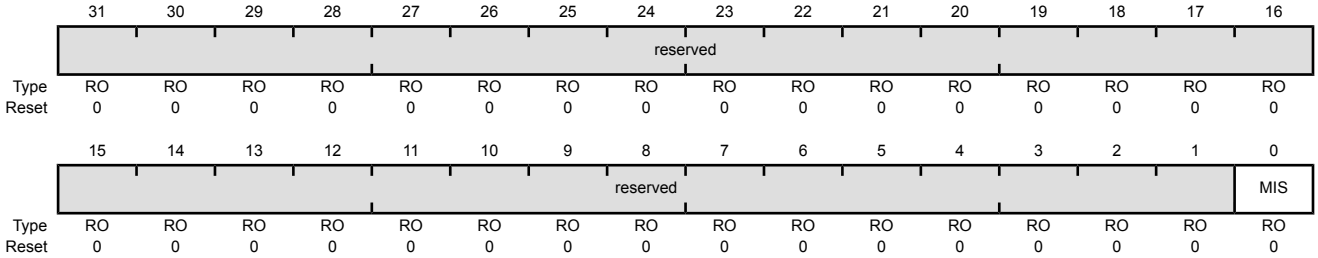

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | RIS | RO | 0 | Raw Interrupt Status<br>This bit specifies the raw interrupt state (prior to masking) of the I²C master block. If set, an interrupt is pending; otherwise, an interrupt is not pending. |

## Register 7: I²C Master Masked Interrupt Status (I2CMMIS), offset 0x018

This register specifies whether an interrupt was signaled.

I2C Master Masked Interrupt Status (I2CMMIS)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x018
Type RO, reset 0x0000.0000



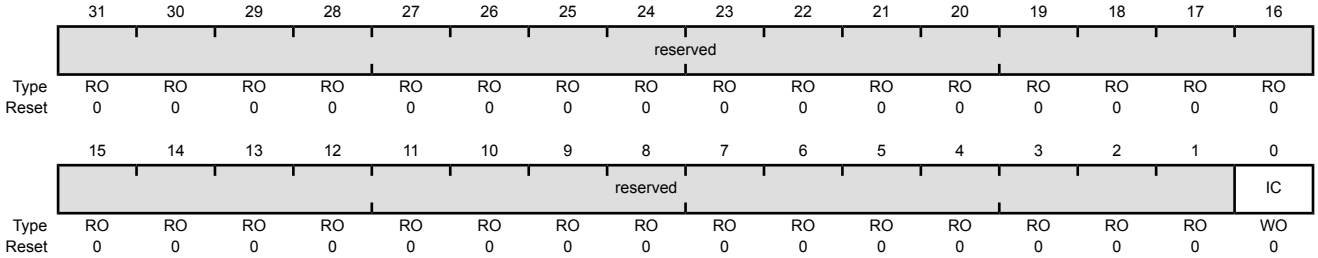

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | MIS | RO | 0 | Masked Interrupt Status<br>This bit specifies the raw interrupt state (after masking) of the I²C master block. If set, an interrupt was signaled; otherwise, an interrupt has not been generated since the bit was last cleared. |

## Register 8: I2C Master Interrupt Clear (I2CMICR), offset 0x01C

This register clears the raw interrupt.

I2C Master Interrupt Clear (I2CMICR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x01C
Type WO, reset 0x0000.0000



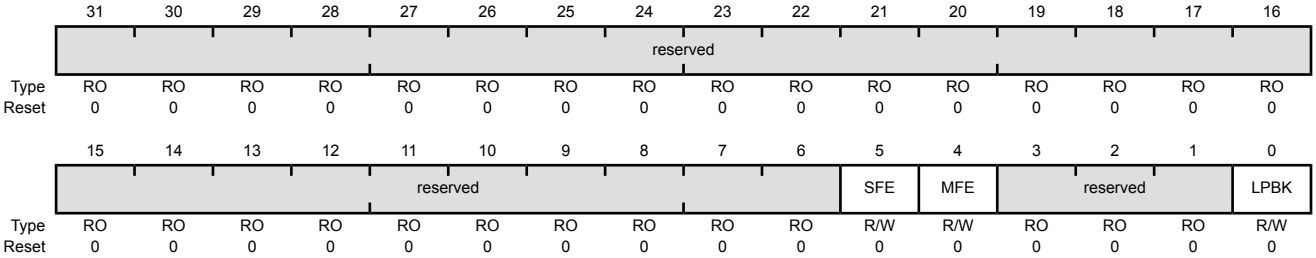

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | IC | WO | 0 | Interrupt Clear<br>This bit controls the clearing of the raw interrupt. A write of 1 clears the interrupt; otherwise, a write of 0 has no affect on the interrupt state. A read of this register returns no meaningful data. |

## Register 9: I²C Master Configuration (I2CMCR), offset 0x020

This register configures the mode (Master or Slave) and sets the interface for test mode loopback.

I2C Master Configuration (I2CMCR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x020
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5 | SFE | R/W | 0 | I²C Slave Function Enable<br>This bit specifies whether the interface may operate in Slave mode. If set, Slave mode is enabled; otherwise, Slave mode is disabled. |
| 4 | MFE | R/W | 0 | I²C Master Function Enable<br>This bit specifies whether the interface may operate in Master mode. If set, Master mode is enabled; otherwise, Master mode is disabled and the interface clock is disabled. |
| 3:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | LPBK | R/W | 0 | I²C Loopback<br>This bit specifies whether the interface is operating normally or in Loopback mode. If set, the device is put in a test mode loopback configuration; otherwise, the device operates normally. |

## 15.7 Register Descriptions ($I^2C$ Slave)

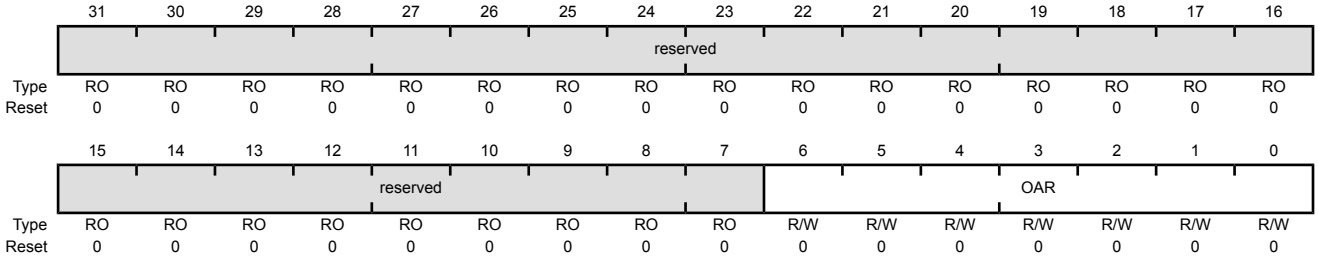
The remainder of this section lists and describes the $I^2C$ slave registers, in numerical order by address offset. See also “Register Descriptions ($I^2C$ Master)” on page 591.

## Register 10: I²C Slave Own Address (I2CSOAR), offset 0x800

This register consists of seven address bits that identify the Stellaris I²C device on the I²C bus.

I2C Slave Own Address (I2CSOAR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x800
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | OAR | R/W | 0x00 | I²C Slave Own Address<br>This field specifies bits A6 through A0 of the slave address. |

## Register 11: I2C Slave Control/Status (I2CSCSR), offset 0x804

This register accesses one control bit when written, and three status bits when read.

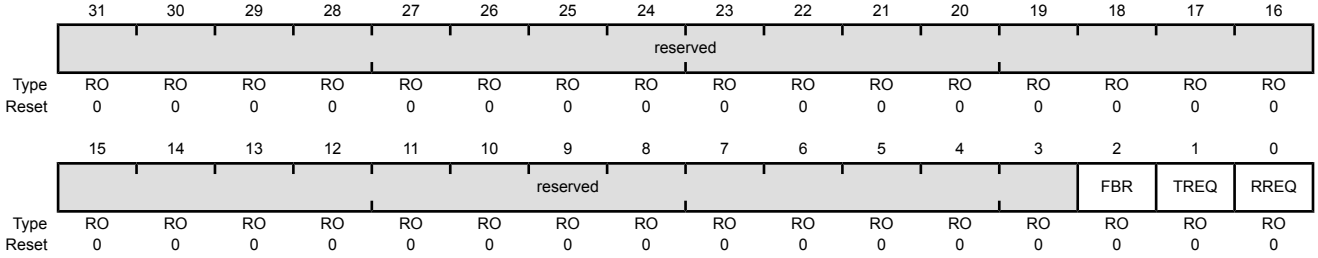
The read-only Status register consists of three bits: the FBR, RREQ, and TREQ bits. The First Byte Received (FBR) bit is set only after the Stellaris device detects its own slave address and receives the first data byte from the I2C master. The Receive Request (RREQ) bit indicates that the Stellaris I2C device has received a data byte from an I2C master. Read one data byte from the **I2C Slave Data (I2CSDR)** register to clear the RREQ bit. The Transmit Request (TREQ) bit indicates that the Stellaris I2C device is addressed as a Slave Transmitter. Write one data byte into the **I2C Slave Data (I2CSDR)** register to clear the TREQ bit.

The write-only Control register consists of one bit: the DA bit. The DA bit enables and disables the Stellaris I2C slave operation.

### Reads

I2C Slave Control/Status (I2CSCSR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x804
Type RO, reset 0x0000.0000

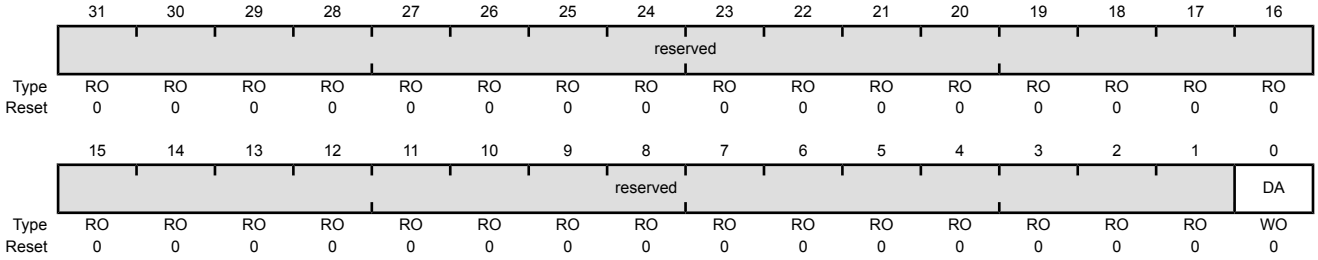



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | FBR | RO | 0 | First Byte Received<br>Indicates that the first byte following the slave's own address is received. This bit is only valid when the RREQ bit is set, and is automatically cleared when data has been read from the **I2CSDR** register.<br>**Note:** This bit is not used for slave transmit operations. |
| 1 | TREQ | RO | 0 | Transmit Request<br>This bit specifies the state of the I2C slave with regards to outstanding transmit requests. If set, the I2C unit has been addressed as a slave transmitter and uses clock stretching to delay the master until data has been written to the **I2CSDR** register. Otherwise, there is no outstanding transmit request. |
| 0 | RREQ | RO | 0 | Receive Request<br>This bit specifies the status of the I2C slave with regards to outstanding receive requests. If set, the I2C unit has outstanding receive data from the I2C master and uses clock stretching to delay the master until the data has been read from the **I2CSDR** register. Otherwise, no receive data is outstanding. |

### Writes

I2C Slave Control/Status (I2CSCSR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x804
Type WO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DA | WO | 0 | Device Active<br><br>Value Description<br>0 Disables the $I^2C$ slave operation.<br>1 Enables the $I^2C$ slave operation.<br><br>Once this bit has been set, it should not be set again unless it has been cleared by writing a 0 or by a reset, otherwise transfer failures may occur. |

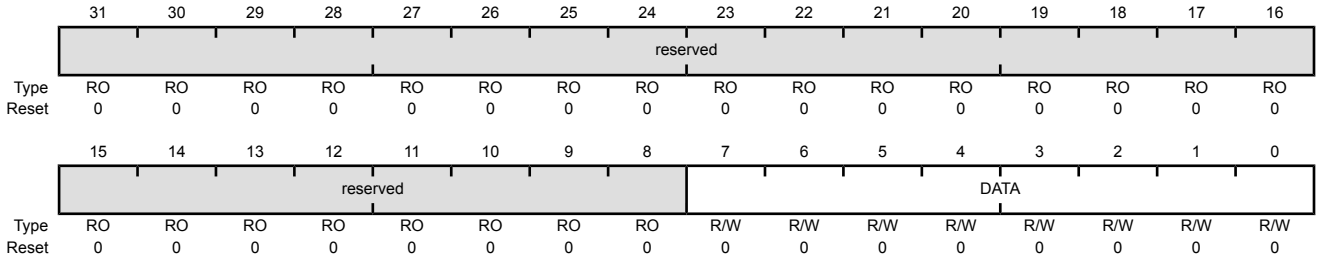
## Register 12: I2C Slave Data (I2CSDR), offset 0x808

**Important:** This register is read-sensitive. See the register description for details.

This register contains the data to be transmitted when in the Slave Transmit state, and the data received when in the Slave Receive state.

I2C Slave Data (I2CSDR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x808
Type R/W, reset 0x0000.0000



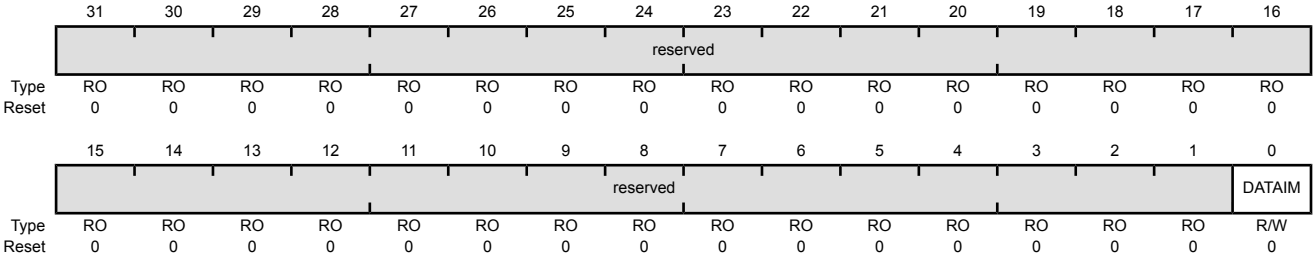

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DATA | R/W | 0x0 | Data for Transfer<br>This field contains the data for transfer during a slave receive or transmit operation. |

## Register 13: I²C Slave Interrupt Mask (I2CSIMR), offset 0x80C

This register controls whether a raw interrupt is promoted to a controller interrupt.

I2C Slave Interrupt Mask (I2CSIMR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x80C
Type R/W, reset 0x0000.0000



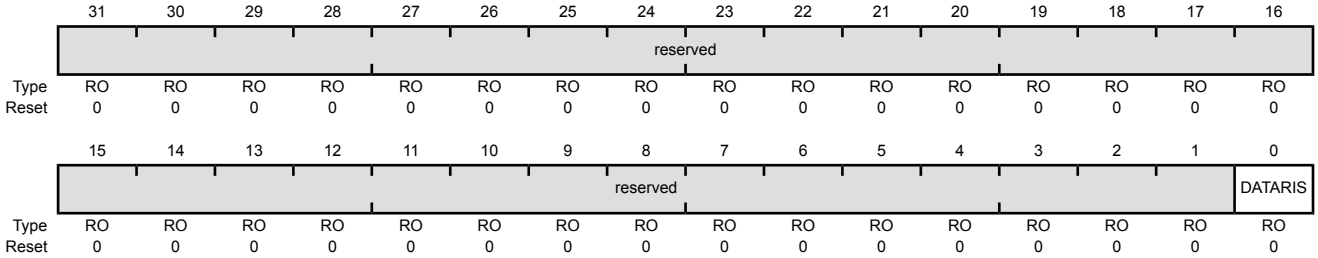

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DATAIM | R/W | 0 | Data Interrupt Mask<br>This bit controls whether the raw interrupt for data received and data requested is promoted to a controller interrupt. If set, the interrupt is not masked and the interrupt is promoted; otherwise, the interrupt is masked. |

## Register 14: I2C Slave Raw Interrupt Status (I2CSRIS), offset 0x810

This register specifies whether an interrupt is pending.

I2C Slave Raw Interrupt Status (I2CSRIS)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x810
Type RO, reset 0x0000.0000



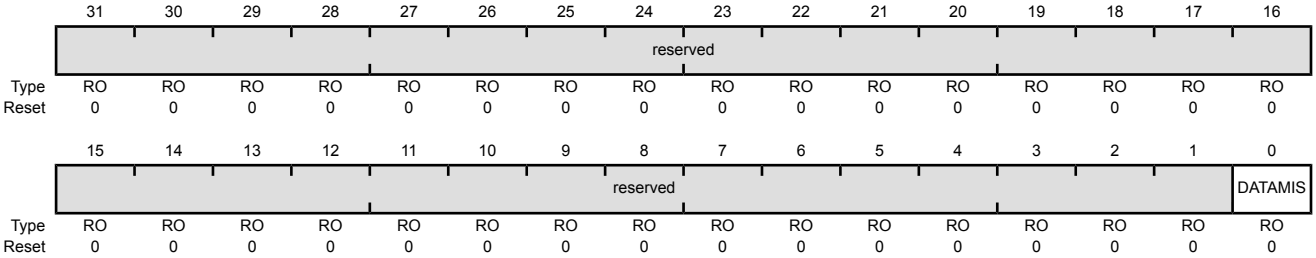

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DATARIS | RO | 0 | Data Raw Interrupt Status<br>This bit specifies the raw interrupt state for data received and data requested (prior to masking) of the I2C slave block. If set, an interrupt is pending; otherwise, an interrupt is not pending. |

## Register 15: I²C Slave Masked Interrupt Status (I2CSMIS), offset 0x814

This register specifies whether an interrupt was signaled.

I2C Slave Masked Interrupt Status (I2CSMIS)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x814
Type RO, reset 0x0000.0000



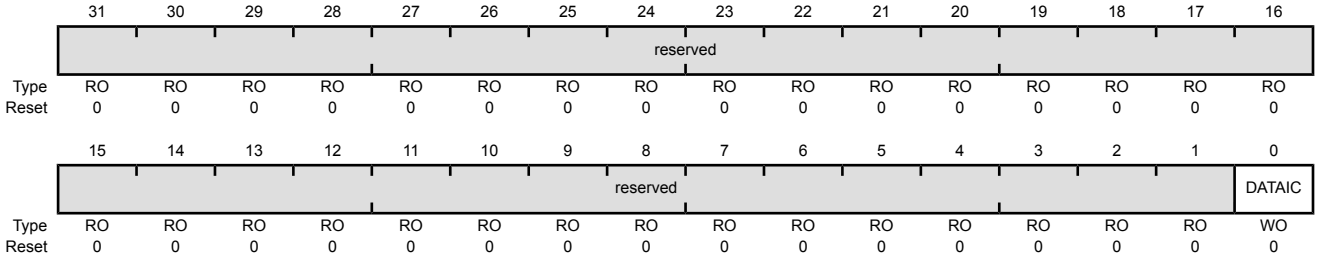

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DATAMIS | RO | 0 | Data Masked Interrupt Status<br>This bit specifies the interrupt state for data received and data requested (after masking) of the I²C slave block. If set, an interrupt was signaled; otherwise, an interrupt has not been generated since the bit was last cleared. |

## Register 16: I2C Slave Interrupt Clear (I2CSICR), offset 0x818

This register clears the raw interrupt. A read of this register returns no meaningful data.

I2C Slave Interrupt Clear (I2CSICR)

I2C 0 base: 0x4002.0000
I2C 1 base: 0x4002.1000
Offset 0x818
Type WO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DATAIC | WO | 0 | Data Interrupt Clear<br>This bit controls the clearing of the raw interrupt for data received and data requested. When set, it clears the DATARIS interrupt bit; otherwise, it has no effect on the DATARIS bit value. |

# 16 Controller Area Network (CAN) Module

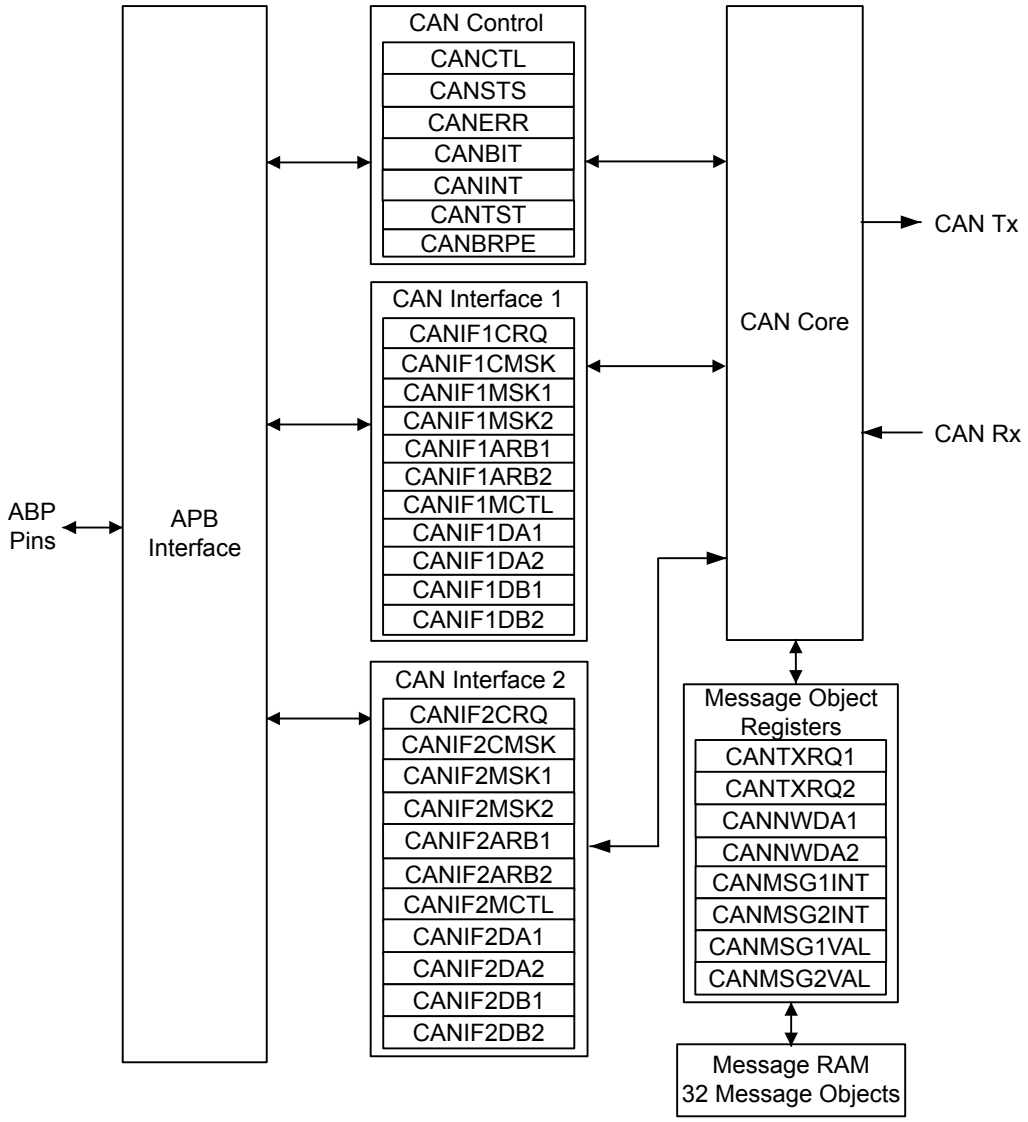
Controller Area Network (CAN) is a multicast, shared serial bus standard for connecting electronic control units (ECUs). CAN was specifically designed to be robust in electromagnetically-noisy environments and can utilize a differential balanced line like RS-485 or a more robust twisted-pair wire. Originally created for automotive purposes, it is also used in many embedded control applications (such as industrial and medical). Bit rates up to 1Mbps are possible at network lengths less than 40 meters. Decreased bit rates allow longer network distances (for example, 125 Kbps at 500 meters).

The Stellaris® CAN controller supports the following features:

- CAN protocol version 2.0 part A/B
- Bit rates up to 1 Mbps
- 32 message objects with individual identifier masks
- Maskable interrupt
- Disable Automatic Retransmission mode for Time-Triggered CAN (TTCAN) applications
- Programmable Loopback mode for self-test operation
- Programmable FIFO mode enables storage of multiple message objects
- Gluelessly attaches to an external CAN interface through the CANnTX and CANnRX signals

## 16.1 Block Diagram

**Figure 16-1. CAN Controller Block Diagram**




## 16.2 Signal Description

Table 16-1 on page 614 lists the external signals of the CAN controller and describes the function of each. The CAN controller signals are alternate functions for some GPIO signals and default to be GPIO signals at reset. The column in the table below titled "Pin Assignment" lists the possible GPIO pin placements for the CAN signals. The AFSEL bit in the **GPIO Alternate Function Select (GPIOAFSEL)** register (page 371) should be set to choose the CAN controller function. For more information on configuring GPIOs, see “General-Purpose Input/Outputs (GPIOs)” on page 351.

**Table 16-1. Controller Area Network Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| CAN0Rx | 61 | I | Analog | CAN module 0 receive. |

**Table 16-1. Controller Area Network Signals (64LQFP) *(continued)***

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| CAN0Tx | 62 | O | Analog | CAN module 0 transmit. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 16.3 Functional Description

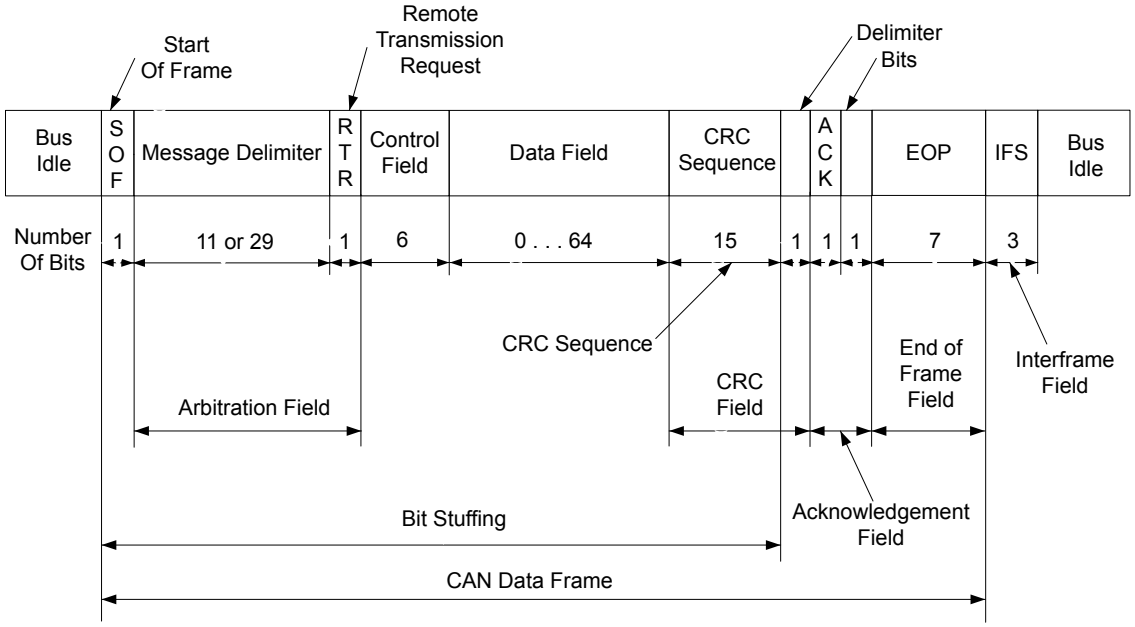
The Stellaris CAN controller conforms to the CAN protocol version 2.0 (parts A and B). Message transfers that include data, remote, error, and overload frames with an 11-bit identifier (standard) or a 29-bit identifier (extended) are supported. Transfer rates can be programmed up to 1 Mbps.

The CAN module consists of three major parts:

- CAN protocol controller and message handler
- Message memory
- CAN register interface

A data frame contains data for transmission, whereas a remote frame contains no data and is used to request the transmission of a specific message object. The CAN data/remote frame is constructed as shown in Figure 16-2 on page 615.

**Figure 16-2. CAN Data/Remote Frame**




The protocol controller transfers and receives the serial data from the CAN bus and passes the data on to the message handler. The message handler then loads this information into the appropriate message object based on the current filtering and identifiers in the message object memory. The message handler is also responsible for generating interrupts based on events on the CAN bus.

The message object memory is a set of 32 identical memory blocks that hold the current configuration, status, and actual data for each message object. These are accessed via either of the CAN message object register interfaces.

The message memory is not directly accessible in the Stellaris memory map, so the Stellaris CAN controller provides an interface to communicate with the message memory via two CAN interface register sets for communicating with the message objects. As there is no direct access to the message object memory, these two interfaces must be used to read or write to each message object. The two message object interfaces allow parallel access to the CAN controller message objects when multiple objects may have new information that must be processed. In general, one interface is used for transmit data and one for receive data.

### 16.3.1 Initialization

Software initialization is started by setting the INIT bit in the **CAN Control (CANCTL)** register (with software or by a hardware reset) or by going bus-off, which occurs when the transmitter's error counter exceeds a count of 255. While INIT is set, all message transfers to and from the CAN bus are stopped and the CANnTX signal is held High. Entering the initialization state does not change the configuration of the CAN controller, the message objects, or the error counters. However, some configuration registers are only accessible while in the initialization state.

To initialize the CAN controller, set the **CAN Bit Timing (CANBIT)** register and configure each message object. If a message object is not needed, label it as not valid by clearing the MSGVAL bit in the **CAN IFn Arbitration 2 (CANIFnARB2)** register. Otherwise, the whole message object must be initialized, as the fields of the message object may not have valid information, causing unexpected results. Both the INIT and CCE bits in the **CANCTL** register must be set in order to access the **CANBIT** register and the **CAN Baud Rate Prescaler Extension (CANBRPE)** register to configure the bit timing. To leave the initialization state, the INIT bit must be cleared. Afterwards, the internal Bit Stream Processor (BSP) synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (indicating a bus idle condition) before it takes part in bus activities and starts message transfers. Message object initialization does not require the CAN to be in the initialization state and can be done on the fly. However, message objects should all be configured to particular identifiers or set to not valid before message transfer starts. To change the configuration of a message object during normal operation, clear the MSGVAL bit in the **CANIFnARB2** register to indicate that the message object is not valid during the change. When the configuration is completed, set the MSGVAL bit again to indicate that the message object is once again valid.

### 16.3.2 Operation

There are two sets of CAN Interface Registers (**CANIF1x** and **CANIF2x**), which are used to access the message objects in the Message RAM. The CAN controller coordinates transfers to and from the Message RAM to and from the registers. The two sets are independent and identical and can be used to queue transactions. Generally, one interface is used to transmit data and one is used to receive data.

Once the CAN module is initialized and the INIT bit in the **CANCTL** register is cleared, the CAN module synchronizes itself to the CAN bus and starts the message transfer. As each message is received, it goes through the message handler's filtering process, and if it passes through the filter, is stored in the message object specified by the MNUM bit in the **CAN IFn Command Request (CANIFnCRQ)** register. The whole message (including all arbitration bits, data-length code, and eight data bytes) is stored in the message object. If the Identifier Mask (the MSK bits in the **CAN IFn Mask 1** and **CAN IFn Mask 2 (CANIFnMSKn)** registers) is used, the arbitration bits that are masked to "don't care" may be overwritten in the message object.

The CPU may read or write each message at any time via the CAN Interface Registers. The message handler guarantees data consistency in case of concurrent accesses.

The transmission of message objects is under the control of the software that is managing the CAN hardware. These can be message objects used for one-time data transfers, or permanent message objects used to respond in a more periodic manner. Permanent message objects have all arbitration and control set up, and only the data bytes are updated. At the start of transmission, the appropriate TXRQST bit in the **CAN Transmission Request n (CANTXRQn)** register and the NEWDAT bit in the **CAN New Data n (CANNWDAn)** register are set. If several transmit messages are assigned to the same message object (when the number of message objects is not sufficient), the whole message object has to be configured before the transmission of this message is requested.

The transmission of any number of message objects may be requested at the same time; they are transmitted according to their internal priority, which is based on the message identifier (MNUM) for the message object, with 1 being the highest priority and 32 being the lowest priority. Messages may be updated or set to not valid any time, even when their requested transmission is still pending. The old data is discarded when a message is updated before its pending transmission has started. Depending on the configuration of the message object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

Transmission can be automatically started by the reception of a matching remote frame. To enable this mode, set the RMTEN bit in the **CAN IFn Message Control (CANIFnMCTL)** register. A matching received remote frame causes the TXRQST bit to be set and the message object automatically transfers its data or generates an interrupt indicating a remote frame was requested. This can be strictly a single message identifier, or it can be a range of values specified in the message object. The CAN mask registers, **CANIFnMSKn**, configure which groups of frames are identified as remote frame requests. The UMASK bit in the **CANIFnMCTL** register enables the MSK bits in the **CANIFnMSKn** register to filter which frames are identified as a remote frame request. The MXTD bit in the **CANIFnMSK2** register should be set if a remote frame request is expected to be triggered by 29-bit extended identifiers.

### 16.3.3 Transmitting Message Objects

If the internal transmit shift register of the CAN module is ready for loading, and if there is no data transfer occurring between the CAN Interface Registers and message RAM, the valid message object with the highest priority that has a pending transmission request is loaded into the transmit shift register by the message handler and the transmission is started. The message object's NEWDAT bit in the **CANNWDAn** register is cleared. After a successful transmission, and if no new data was written to the message object since the start of the transmission, the TXRQST bit in the **CANTXRQn** register is cleared. If the CAN controller is set up to interrupt upon a successful transmission of a message object, (the TXIE bit in the **CAN IFn Message Control (CANIFnMCTL)** register is set), the INTPND bit in the **CANIFnMCTL** register is set after a successful transmission. If the CAN module has lost the arbitration or if an error occurred during the transmission, the message is re-transmitted as soon as the CAN bus is free again. If, meanwhile, the transmission of a message with higher priority has been requested, the messages are transmitted in the order of their priority.

### 16.3.4 Configuring a Transmit Message Object

The following steps illustrate how to configure a transmit message object.

1. In the **CAN IFn Command Mask (CANIFnCMASK)** register:

   - Set the WRNRD bit to specify a write to the **CANIFnCMASK** register; specify whether to transfer the IDMASK, DIR, and MXTD of the message object into the **CAN IFn** registers using the MASK bit
   - Specify whether to transfer the ID, DIR, XTD, and MSGVAL of the message object into the interface registers using the ARB bit

- Specify whether to transfer the control bits into the interface registers using the CONTROL bit
- Specify whether to clear the INTPND bit in the **CANIFnMCTL** register using the CLRINTPND bit
- Specify whether to clear the NEWDAT bit in the **CANNWDAn** register using the NEWDAT bit
- Specify which bits to transfer using the DATAA and DATAB bits

2. In the **CANIFnMSK1** register, use the MSK[15:0] bits to specify which of the bits in the 29-bit or 11-bit message identifier are used for acceptance filtering. Note that MSK[15:0] in this register are used for bits [15:0] of the 29-bit message identifier and are not used for an 11-bit identifier. A value of 0x00 enables all messages to pass through the acceptance filtering. Also note that in order for these bits to be used for acceptance filtering, they must be enabled by setting the UMASK bit in the **CANIFnMCTL** register.
3. In the **CANIFnMSK2** register, use the MSK[12:0] bits to specify which of the bits in the 29-bit or 11-bit message identifier are used for acceptance filtering. Note that MSK[12:0] are used for bits [28:16] of the 29-bit message identifier; whereas MSK[12:2] are used for bits [10:0] of the 11-bit message identifier. Use the MXTD and MDIR bits to specify whether to use XTD and DIR for acceptance filtering. A value of 0x00 enables all messages to pass through the acceptance filtering. Also note that in order for these bits to be used for acceptance filtering, they must be enabled by setting the UMASK bit in the **CANIFnMCTL** register.
4. For a 29-bit identifier, configure ID[15:0] in the **CANIFnARB1** register to are used for bits [15:0] of the message identifier and ID[12:0] in the **CANIFnARB2** register to are used for bits [28:16] of the message identifier. Set the XTD bit to indicate an extended identifier; set the DIR bit to indicate transmit; and set the MSGVAL bit to indicate that the message object is valid.
5. For an 11-bit identifier, disregard the **CANIFnARB1** register and configure ID[12:2] in the **CANIFnARB2** register to are used for bits [10:0] of the message identifier. Clear the XTD bit to indicate a standard identifier; set the DIR bit to indicate transmit; and set the MSGVAL bit to indicate that the message object is valid.
6. In the **CANIFnMCTL** register:
   - Optionally set the UMASK bit to enable the mask (MSK, MXTD, and MDIR specified in the **CANIFnMSK1** and **CANIFnMSK2** registers) for acceptance filtering
   - Optionally set the TXIE bit to enable the INTPND bit to be set after a successful transmission
   - Optionally set the RMTEN bit to enable the TXRQST bit to be set upon the reception of a matching remote frame allowing automatic transmission
   - Set the EOB bit for a single message object;
   - Set the DLC[3:0] field to specify the size of the data frame. Take care during this configuration not to set the NEWDAT, MSGLST, INTPND or TXRQST bits.
7. Load the data to be transmitted into the CAN IFn Data **(CANIFnDA1, CANIFnDA2, CANIFnDB1, CANIFnDB2)** or **(CANIFnDATAA and CANIFnDATAB)** registers. Byte 0 of the CAN data frame is stored in DATA[7:0] in the **CANIFnDA1** register.

8. Program the number of the message object to be transmitted in the MNUM field in the **CAN IFn Command Request (CANIFnCRQ)** register.

9. When everything is properly configured, set the TXRQST bit in the **CANIFnMCTL** register. Once this bit is set, the message object is available to be transmitted, depending on priority and bus availability. Note that setting the RMTEN bit in the **CANIFnMCTL** register can also start message transmission if a matching remote frame has been received.

### 16.3.5 Updating a Transmit Message Object

The CPU may update the data bytes of a Transmit Message Object any time via the CAN Interface Registers and neither the MSGVAL bit in the **CANIFnARB2** register nor the TXRQST bits in the **CANIFnMCTL** register have to be cleared before the update.

Even if only some of the data bytes are to be updated, all four bytes of the corresponding **CANIFnDAn**/**CANIFnDBn** register have to be valid before the content of that register is transferred to the message object. Either the CPU must write all four bytes into the **CANIFnDAn**/**CANIFnDBn** register or the message object is transferred to the **CANIFnDAn**/**CANIFnDBn** register before the CPU writes the new data bytes.

In order to only update the data in a message object, the WRNRD, DATAA and DATAB bits in the **CANIFnMSKn** register are set, followed by writing the updated data into **CANIFnDA1**, **CANIFnDA2**, **CANIFnDB1**, and **CANIFnDB2** registers, and then the number of the message object is written to the MNUM field in the **CAN IFn Command Request (CANIFnCRQ)** register. To begin transmission of the new data as soon as possible, set the TXRQST bit in the **CANIFnMSKn** register.

To prevent the clearing of the TXRQST bit in the **CANIFnMCTL** register at the end of a transmission that may already be in progress while the data is updated, the NEWDAT and TXRQST bits have to be set at the same time in the **CANIFnMCTL** register. When these bits are set at the same time, NEWDAT is cleared as soon as the new transmission has started.

### 16.3.6 Accepting Received Message Objects

When the arbitration and control field (the ID and XTD bits in the **CANIFnARB2** and the RMTEN and DLC[3:0] bits of the **CANIFnMCTL** register) of an incoming message is completely shifted into the CAN controller, the message handling capability of the controller starts scanning the message RAM for a matching valid message object. To scan the message RAM for a matching message object, the controller uses the acceptance filtering programmed through the mask bits in the **CANIFnMSKn** register and enabled using the UMASK bit in the **CANIFnMCTL** register. Each valid message object, starting with object 1, is compared with the incoming message to locate a matching message object in the message RAM. If a match occurs, the scanning is stopped and the message handler proceeds depending on whether it is a data frame or remote frame that was received.

### 16.3.7 Receiving a Data Frame

The message handler stores the message from the CAN controller receive shift register into the matching message object in the message RAM. The data bytes, all arbitration bits, and the DLC bits are all stored into the corresponding message object. In this manner, the data bytes are connected with the identifier even if arbitration masks are used. The NEWDAT bit of the **CANIFnMCTL** register is set to indicate that new data has been received. The CPU should clear this bit when it reads the message object to indicate to the controller that the message has been received, and the buffer is free to receive more messages. If the CAN controller receives a message and the NEWDAT bit is already set, the MSGLST bit in the **CANIFnMCTL** register is set to indicate that the previous data was lost. If the system requires an interrupt upon successful reception of a frame, the RXIE bit of the **CANIFnMCTL** register should be set. In this case, the INTPND bit of the same register is set,

causing the **CANINT** register to point to the message object that just received a message. The TXRQST bit of this message object should be cleared to prevent the transmission of a remote frame.

### 16.3.8 Receiving a Remote Frame

A remote frame contains no data, but instead specifies which object should be transmitted. When a remote frame is received, three different configurations of the matching message object have to be considered:

| Configuration in CANIFnMCTL | Description |
|---|---|
| ■ DIR = 1 (direction = transmit); programmed in the **CANIFnARB2** register<br>■ RMTEN = 1 (set the TXRQST bit of the **CANIFnMCTL** register at reception of the frame to enable transmission)<br>■ UMASK = 1 or 0 | At the reception of a matching remote frame, the TXRQST bit of this message object is set. The rest of the message object remains unchanged, and the controller automatically transfers the data in the message object as soon as possible. |
| ■ DIR = 1 (direction = transmit); programmed in the **CANIFnARB2** register<br>■ RMTEN = 0 (do not change the TXRQST bit of the **CANIFnMCTL** register at reception of the frame)<br>■ UMASK = 0 (ignore mask in the **CANIFnMSKn** register) | At the reception of a matching remote frame, the TXRQST bit of this message object remains unchanged, and the remote frame is ignored. This remote frame is disabled, the data is not transferred and there is no indication that the remote frame ever happened. |
| ■ DIR = 1 (direction = transmit); programmed in the **CANIFnARB2** register<br>■ RMTEN = 0 (do not change the TXRQST bit of the **CANIFnMCTL** register at reception of the frame)<br>■ UMASK = 1 (use mask (MSK, MXTD, and MDIR in the **CANIFnMSKn** register) for acceptance filtering) | At the reception of a matching remote frame, the TXRQST bit of this message object is cleared. The arbitration and control field (ID + XTD + RMTEN + DLC) from the shift register is stored into the message object in the message RAM and the NEWDAT bit of this message object is set. The data field of the message object remains unchanged; the remote frame is treated similar to a received data frame. This is useful for a remote data request from another CAN device for which the Stellaris controller does not have readily available data. The software must fill the data and answer the frame manually. |

### 16.3.9 Receive/Transmit Priority

The receive/transmit priority for the message objects is controlled by the message number. Message object 1 has the highest priority, while message object 32 has the lowest priority. If more than one transmission request is pending, the message objects are transmitted in order based on the message object with the lowest message number. This should not be confused with the message identifier as that priority is enforced by the CAN bus. This means that if message object 1 and message object 2 both have valid messages that need to be transmitted, message object 1 will always be transmitted first regardless of the message identifier in the message object itself.

### 16.3.10 Configuring a Receive Message Object

The following steps illustrate how to configure a receive message object.

1. Program the **CAN IFn Command Mask (CANIFnCMASK)** register as described in the "Configuring a Transmit Message Object" on page 617 section, except that the WRNRD bit is set to specify a write to the message RAM.

2. Program the **CANIFnMSK1**and **CANIFnMSK2** registers as described in the "Configuring a Transmit Message Object" on page 617 section to configure which bits are used for acceptance

filtering. Note that in order for these bits to be used for acceptance filtering, they must be enabled by setting the UMASK bit in the **CANIFnMCTL** register.

3. In the **CANIFnMSK2** register, use the MSK[12:0] bits to specify which of the bits in the 29-bit or 11-bit message identifier are used for acceptance filtering. Note that MSK[12:0] are used for bits [28:16] of the 29-bit message identifier; whereas MSK[12:2] are used for bits [10:0] of the 11-bit message identifier. Use the MXTD and MDIR bits to specify whether to use XTD and DIR for acceptance filtering. A value of 0x00 enables all messages to pass through the acceptance filtering. Also note that in order for these bits to be used for acceptance filtering, they must be enabled by setting the UMASK bit in the **CANIFnMCTL** register.

4. Program the **CANIFnARB1** and **CANIFnARB2** registers as described in the "Configuring a Transmit Message Object" on page 617 section to program XTD and ID bits for the message identifier to be received; set the MSGVAL bit to indicate a valid message; and clear the DIR bit to specify receive.

5. In the **CANIFnMCTL** register:

   - Optionally set the UMASK bit to enable the mask (MSK, MXTD, and MDIR specified in the **CANIFnMSK1** and **CANIFnMSK2** registers) for acceptance filtering
   - Optionally set the RXIE bit to enable the INTPND bit to be set after a successful reception
   - Clear the RMTEN bit to leave the TXRQST bit unchanged
   - Set the EOB bit for a single message object
   - Set the DLC[3:0] field to specify the size of the data frame

   Take care during this configuration not to set the NEWDAT, MSGLST, INTPND or TXRQST bits.

6. Program the number of the message object to be received in the MNUM field in the **CAN IFn Command Request (CANIFnCRQ)** register. Reception of the message object begins as soon as a matching frame is available on the CAN bus.

When the message handler stores a data frame in the message object, it stores the received Data Length Code and eight data bytes in the **CANIFnDA1**, **CANIFnDA2**, **CANIFnDB1**, and **CANIFnDB2** register. Byte 0 of the CAN data frame is stored in DATA[7:0] in the **CANIFnDA1** register. If the Data Length Code is less than 8, the remaining bytes of the message object are overwritten by unspecified values.

The CAN mask registers can be used to allow groups of data frames to be received by a message object. The CAN mask registers, **CANIFnMSKn**, configure which groups of frames are received by a message object. The UMASK bit in the **CANIFnMCTL** register enables the MSK bits in the **CANIFnMSKn** register to filter which frames are received. The MXTD bit in the **CANIFnMSK2** register should be set if only 29-bit extended identifiers are expected by this message object.

### 16.3.11 Handling of Received Message Objects

The CPU may read a received message any time via the CAN Interface registers because the data consistency is guaranteed by the message handler state machine.

Typically, the CPU first writes 0x007F to the **CANIFnCMSK** register and then writes the number of the message object to the **CANIFnCRQ** register. That combination transfers the whole received message from the message RAM into the Message Buffer registers (**CANIFnMSKn**, **CANIFnARBn**, and **CANIFnMCTL**). Additionally, the NEWDAT and INTPND bits are cleared in the message RAM,

acknowledging that the message has been read and clearing the pending interrupt generated by this message object.

If the message object uses masks for acceptance filtering, the **CANIFnARBn** registers show the full, unmasked ID for the received message.

The NEWDAT bit in the **CANIFnMCTL** register shows whether a new message has been received since the last time this message object was read. The MSGLST bit in the **CANIFnMCTL** register shows whether more than one message has been received since the last time this message object was read. MSGLST is not automatically cleared, and should be cleared by software after reading its status.

Using a remote frame, the CPU may request new data from another CAN node on the CAN bus. Setting the TXRQST bit of a receive object causes the transmission of a remote frame with the receive object's identifier. This remote frame triggers the other CAN node to start the transmission of the matching data frame. If the matching data frame is received before the remote frame could be transmitted, the TXRQST bit is automatically reset. This prevents the possible loss of data when the other device on the CAN bus has already transmitted the data slightly earlier than expected.

#### 16.3.11.1 Configuration of a FIFO Buffer

With the exception of the EOB bit in the **CANIFnMCTL** register, the configuration of receive message objects belonging to a FIFO buffer is the same as the configuration of a single receive message object (see “Configuring a Receive Message Object” on page 620). To concatenate two or more message objects into a FIFO buffer, the identifiers and masks (if used) of these message objects have to be programmed to matching values. Due to the implicit priority of the message objects, the message object with the lowest message object number is the first message object in a FIFO buffer. The EOB bit of all message objects of a FIFO buffer except the last one must be cleared. The EOB bit of the last message object of a FIFO buffer is set, indicating it is the last entry in the buffer.

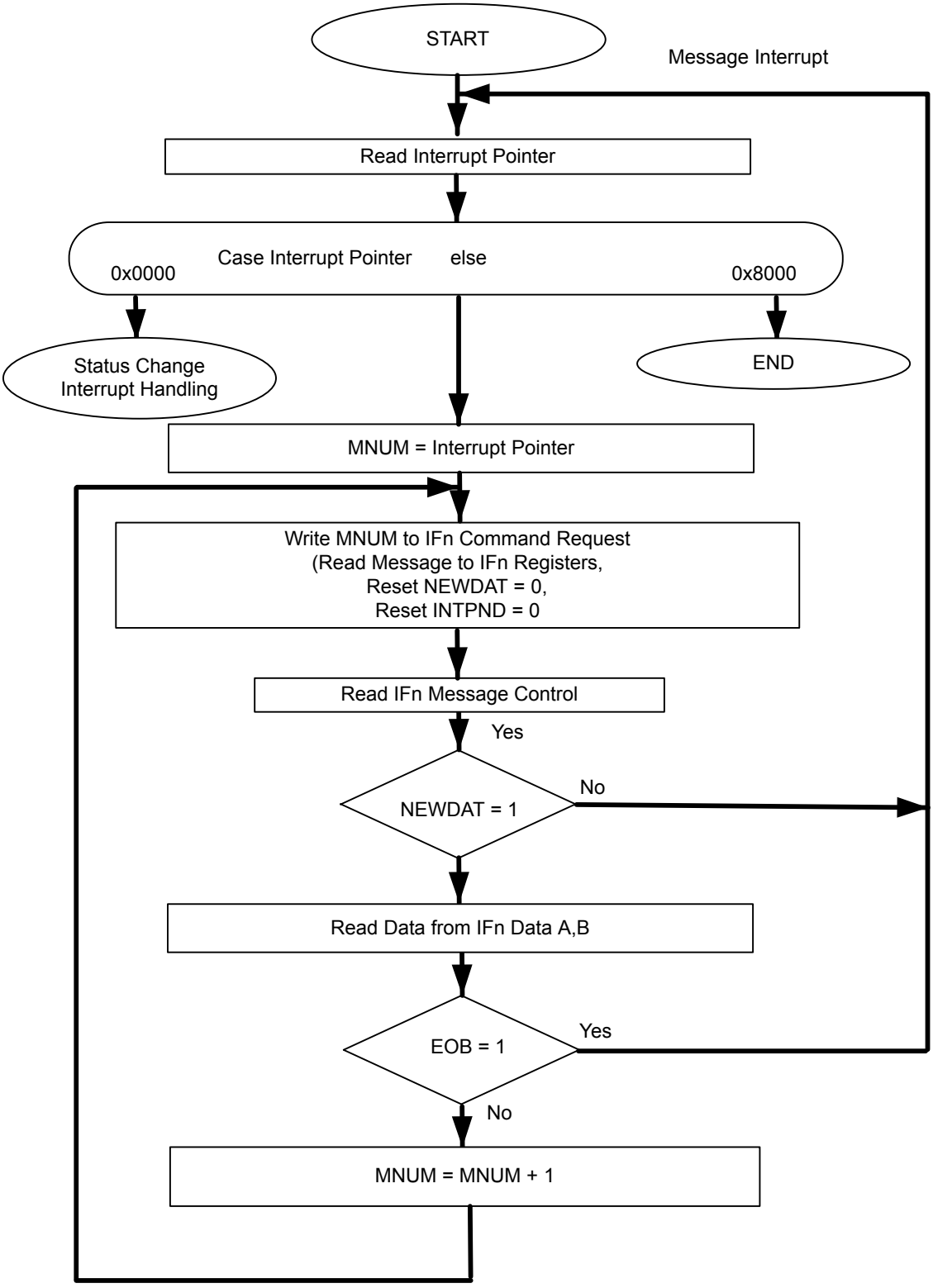
#### 16.3.11.2 Reception of Messages with FIFO Buffers

Received messages with identifiers matching to a FIFO buffer are stored starting with the message object with the lowest message number. When a message is stored into a message object of a FIFO buffer, the NEWDAT of the **CANIFnMCTL** register bit of this message object is set. By setting NEWDAT while EOB is clear, the message object is locked and cannot be written to by the message handler until the CPU has cleared the NEWDAT bit. Messages are stored into a FIFO buffer until the last message object of this FIFO buffer is reached. If none of the preceding message objects has been released by clearing the NEWDAT bit, all further messages for this FIFO buffer will be written into the last message object of the FIFO buffer and therefore overwrite previous messages.

#### 16.3.11.3 Reading from a FIFO Buffer

When the CPU transfers the contents of a message object from a FIFO buffer by writing its number to the **CANIFnCRQ**, the TXRQST and CLRINTPND bits in the **CANIFnCMSK** register should be set such that the NEWDAT and INTPEND bits in the **CANIFnMCTL** register are cleared after the read. The values of these bits in the **CANIFnMCTL** register always reflect the status of the message object before the bits are cleared. To assure the correct function of a FIFO buffer, the CPU should read out the message objects starting with the message object with the lowest message number. When reading from the FIFO buffer, the user should be aware that a new received message is placed in the message object with the lowest message number for which the NEWDAT bit of the **CANIFnMCTL** register. As a result, the order of the received messages in the FIFO is not guaranteed. Figure 16-3 on page 623 shows how a set of message objects which are concatenated to a FIFO Buffer can be handled by the CPU.

**Figure 16-3. Message Objects in a FIFO Buffer**



### 16.3.12 Handling of Interrupts

If several interrupts are pending, the **CAN Interrupt (CANINT)** register points to the pending interrupt with the highest priority, disregarding their chronological order. The status interrupt has the highest priority. Among the message interrupts, the message object's interrupt with the lowest message number has the highest priority. A message interrupt is cleared by clearing the message object's INTPND bit in the **CANIFnMCTL** register or by reading the **CAN Status (CANSTS)** register. The status Interrupt is cleared by reading the **CANSTS** register.

The interrupt identifier INTID in the **CANINT** register indicates the cause of the interrupt. When no interrupt is pending, the register reads as 0x0000. If the value of the INTID field is different from 0, then there is an interrupt pending. If the IE bit is set in the **CANCTL** register, the interrupt line to the CPU is active. The interrupt line remains active until the INTID field is 0, meaning that all interrupt sources have been cleared (the cause of the interrupt is reset), or until IE is cleared, which disables interrupts from the CAN controller.

The INTID field of the **CANINT** register points to the pending message interrupt with the highest interrupt priority. The SIE bit in the **CANCTL** register controls whether a change of the RXOK, TXOK, and LEC bits in the **CANSTS** register can cause an interrupt. The EIE bit in the **CANCTL**register controls whether a change of the BOFF and EWARN bits in the **CANSTS** can cause an interrupt. The IE bit in the **CANCTL** controls whether any interrupt from the CAN controller actually generates an interrupt to the microcontroller's interrupt controller. The **CANINT** register is updated even when the IE bit in the **CANCTL** register is clear, but the interrupt will not be indicated to the CPU.

A value of 0x8000 in the **CANINT** register indicates that an interrupt is pending because the CAN module has updated, but not necessarily changed, the **CANSTS** , indicating that either an error or status interrupt has been generated. A write access to the **CANSTS** register can clear the RXOK, TXOK, and LEC bits in that same register; however, the only way to clear the source of a status interrupt is to read the **CANSTS** register.

There are two ways to determine the source of an interrupt during interrupt handling. The first is to read the INTID bit in the **CANINT** register to determine the highest priority interrupt that is pending, and the second is to read the **CAN Message Interrupt Pending (CANMSGnINT)** register to see all of the message objects that have pending interrupts.

An interrupt service routine reading the message that is the source of the interrupt may read the message and clear the message object's INTPND bit at the same time by setting the CLRINTPND bit in the **CANIFnCMSK** register. Once the INTPND bit has been cleared, the **CANINT** register contains the message number for the next message object with a pending interrupt.

### 16.3.13 Test Mode

A Test Mode is provided, which allows various diagnostics to be performed. Test Mode is entered by setting the TEST bit **CANCTL** register. Once in Test Mode, the TX[1:0], LBACK, SILENT and BASIC bits in the **CAN Test (CANTST)** register can be used to put the CAN controller into the various diagnostic modes. The RX bit in the **CANTST** register allows monitoring of the CANnRX signal. All **CANTST** register functions are disabled when the TEST bit is cleared.

#### 16.3.13.1 Silent Mode

Silent Mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames). The CAN Controller is put in Silent Mode setting the SILENT bit in the **CANTST** register. In Silent Mode, the CAN controller is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the CAN Controller is required to send a dominant bit (ACK bit, overload flag,

or active error flag), the bit is rerouted internally so that the CAN Controller monitors this dominant bit, although the CAN bus remains in recessive state.

### 16.3.13.2 Loopback Mode

Loopback mode is useful for self-test functions. In Loopback Mode, the CAN Controller internally routes the CANnTX signal on to the CANnRX signal and treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) into the message buffer. The CAN Controller is put in Loopback Mode by setting the LBACK bit in the **CANTST** register. To be independent from external stimulation, the CAN Controller ignores acknowledge errors (a recessive bit sampled in the acknowledge slot of a data/remote frame) in Loopback Mode. The actual value of the CANnRX signal is disregarded by the CAN Controller. The transmitted messages can be monitored on the CANnTX signal.

### 16.3.13.3 Loopback Combined with Silent Mode

Loopback Mode and Silent Mode can be combined to allow the CAN Controller to be tested without affecting a running CAN system connected to the CANnTX and CANnRX signals. In this mode, the CANnRX signal is disconnected from the CAN Controller and the CANnTX signal is held recessive. This mode is enabled by setting both the LBACK and SILENT bits in the **CANTST** register.

### 16.3.13.4 Basic Mode

Basic Mode allows the CAN Controller to be operated without the Message RAM. In Basic Mode, The CANIF1 registers are used as the transmit buffer. The transmission of the contents of the IF1 registers is requested by setting the BUSY bit of the **CANIF1CRQ** register. The CANIF1 registers are locked while the BUSY bit is set. The BUSY bit indicates that a transmission is pending. As soon the CAN bus is idle, the CANIF1 registers are loaded into the shift register of the CAN Controller and transmission is started. When the transmission has completed, the BUSY bit is cleared and the locked CANIF1 registers are released. A pending transmission can be aborted at any time by clearing the BUSY bit in the **CANIF1CRQ** register while the CANIF1 registers are locked. If the CPU has cleared the BUSY bit, a possible retransmission in case of lost arbitration or an error is disabled.

The CANIF2 Registers are used as a receive buffer. After the reception of a message, the contents of the shift register is stored into the CANIF2 registers, without any acceptance filtering. Additionally, the actual contents of the shift register can be monitored during the message transfer. Each time a read message object is initiated by setting the BUSY bit of the **CANIF2CRQ** register, the contents of the shift register are stored into the CANIF2 registers.

In Basic Mode, all message-object-related control and status bits and of the control bits of the **CANIFnCMSK** registers are not evaluated. The message number of the **CANIFnCRQ** registers is also not evaluated. In the **CANIF2MCTL** register, the NEWDAT and MSGLST bits retain their function, the DLC[3:0] field shows the received DLC, the other control bits are cleared.

Basic Mode is enabled by setting the BASIC bit in the **CANTST** register.

### 16.3.13.5 Transmit Control

Software can directly override control of the CANnTX signal in four different ways.

- CANnTX is controlled by the CAN Controller
- The sample point is driven on the CANnTX signal to monitor the bit timing
- CANnTX drives a low value

- CANnTX drives a high value

The last two functions, combined with the readable CAN receive pin CANnRX, can be used to check the physical layer of the CAN bus.

The Transmit Control function is enabled by programming the TX[1:0] field in the **CANTST** register. The three test functions for the CANnTX signal interfere with all CAN protocol functions. TX[1:0] must be cleared when CAN message transfer or Loopback Mode, Silent Mode, or Basic Mode are selected.

### 16.3.14 Bit Timing Configuration Error Considerations

Even if minor errors in the configuration of the CAN bit timing do not result in immediate failure, the performance of a CAN network can be reduced significantly. In many cases, the CAN bit synchronization amends a faulty configuration of the CAN bit timing to such a degree that only occasionally an error frame is generated. In the case of arbitration, however, when two or more CAN nodes simultaneously try to transmit a frame, a misplaced sample point may cause one of the transmitters to become error passive. The analysis of such sporadic errors requires a detailed knowledge of the CAN bit synchronization inside a CAN node and of the CAN nodes' interaction on the CAN bus.

### 16.3.15 Bit Time and Bit Rate

The CAN system supports bit rates in the range of lower than 1 Kbps up to 1000 Kbps. Each member of the CAN network has its own clock generator. The timing parameter of the bit time can be configured individually for each CAN node, creating a common bit rate even though the CAN nodes' oscillator periods may be different.

Because of small variations in frequency caused by changes in temperature or voltage and by deteriorating components, these oscillators are not absolutely stable. As long as the variations remain inside a specific oscillator's tolerance range, the CAN nodes are able to compensate for the different bit rates by periodically resynchronizing to the bit stream.

According to the CAN specification, the bit time is divided into four segments (see Figure 16-4 on page 627): the Synchronization Segment, the Propagation Time Segment, the Phase Buffer Segment 1, and the Phase Buffer Segment 2. Each segment consists of a specific, programmable number of time quanta (see Table 16-2 on page 627). The length of the time quantum ($t_q$), which is the basic time unit of the bit time, is defined by the CAN controller's input clock (fsys) and the Baud Rate Prescaler (BRP):

$t_q$ = BRP / fsys

The fsys input clock is the system clock frequency as configured by the **RCC** or **RCC2** registers (see page 194 or page 201).

The Synchronization Segment Sync is that part of the bit time where edges of the CAN bus level are expected to occur; the distance between an edge that occurs outside of Sync and the Sync is called the phase error of that edge.

The Propagation Time Segment Prop is intended to compensate for the physical delay times within the CAN network.

The Phase Buffer Segments Phase1 and Phase2 surround the Sample Point.

The (Re-)Synchronization Jump Width (SJW) defines how far a resynchronization may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.

A given bit rate may be met by different bit-time configurations, but for the proper function of the CAN network, the physical delay times and the oscillator's tolerance range have to be considered.

**Figure 16-4. CAN Bit Time**

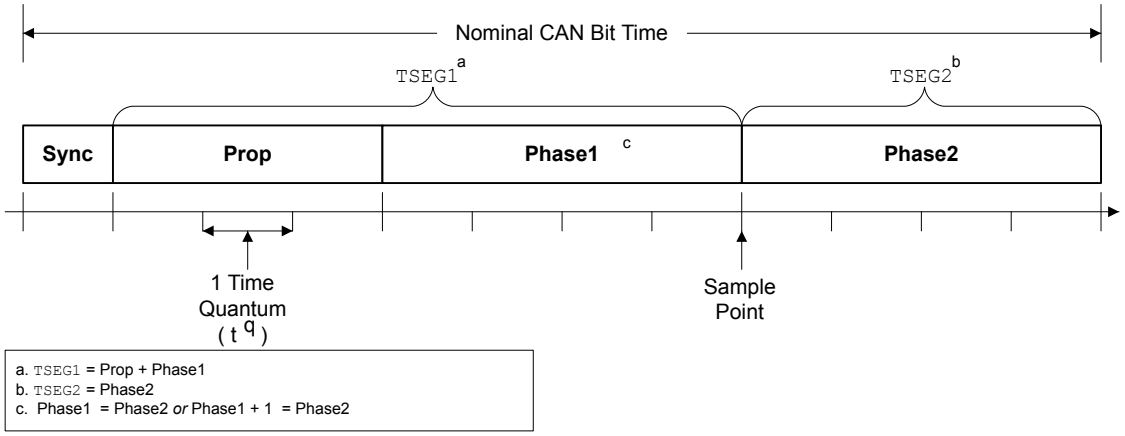



**Table 16-2. CAN Protocol Ranges**[a]

| Parameter | Range | Remark |
|---|---|---|
| BRP | [1 .. 64] | Defines the length of the time quantum $t_q$. The **CANBRPE** register can be used to extend the range to 1024. |
| Sync | 1 $t_q$ | Fixed length, synchronization of bus input to system clock |
| Prop | [1 .. 8] $t_q$ | Compensates for the physical delay times |
| Phase1 | [1 .. 8] $t_q$ | May be lengthened temporarily by synchronization |
| Phase2 | [1 .. 8] $t_q$ | May be shortened temporarily by synchronization |
| SJW | [1 .. 4] $t_q$ | May not be longer than either Phase Buffer Segment |

a. This table describes the minimum programmable ranges required by the CAN protocol.

The bit timing configuration is programmed in two register bytes in the **CANBIT** register. In the **CANBIT** register, the four components TSEG2, TSEG1, SJW, and BRP have to be programmed to a numerical value that is one less than its functional value; so instead of values in the range of [1..n], values in the range of [0..n-1] are programmed. That way, for example, SJW (functional range of [1..4]) is represented by only two bits in the SJW bit field. Table 16-3 shows the relationship between the **CANBIT** register values and the parameters.

**Table 16-3. CANBIT Register Values**

| CANBIT Register Field | Setting |
|---|---|
| TSEG2 | Phase2 - 1 |
| TSEG1 | Prop + Phase1 - 1 |
| SJW | SJW - 1 |
| BRP | BRP |

Therefore, the length of the bit time is (programmed values):

[TSEG1 + TSEG2 + 3] × $t_q$

or (functional values):

[Sync + Prop + Phase1 + Phase2] × $t_q$

The data in the **CANBIT** register is the configuration input of the CAN protocol controller. The baud rate prescaler (configured by the BRP field) defines the length of the time quantum, the basic time unit of the bit time; the bit timing logic (configured by TSEG1, TSEG2, and SJW) defines the number of time quanta in the bit time.

The processing of the bit time, the calculation of the position of the sample point, and occasional synchronizations are controlled by the CAN controller and are evaluated once per time quantum.

The CAN controller translates messages to and from frames. In addition, the controller generates and discards the enclosing fixed format bits, inserts and extracts stuff bits, calculates and checks the CRC code, performs the error management, and decides which type of synchronization is to be used. The bit value is received or transmitted at the sample point. The information processing time (IPT) is the time after the sample point needed to calculate the next bit to be transmitted on the CAN bus. The IPT includes any of the following: retrieving the next data bit, handling a CRC bit, determining if bit stuffing is required, generating an error flag or simply going idle.

The IPT is application-specific but may not be longer than 2 $t_q$; the CAN's IPT is 0 $t_q$. Its length is the lower limit of the programmed length of Phase2. In case of synchronization, Phase2 may be shortened to a value less than IPT, which does not affect bus timing.

### 16.3.16 Calculating the Bit Timing Parameters

Usually, the calculation of the bit timing configuration starts with a required bit rate or bit time. The resulting bit time (1/bit rate) must be an integer multiple of the system clock period.

The bit time may consist of 4 to 25 time quanta. Several combinations may lead to the required bit time, allowing iterations of the following steps.

The first part of the bit time to be defined is Prop. Its length depends on the delay times measured in the system. A maximum bus length as well as a maximum node delay has to be defined for expandable CAN bus systems. The resulting time for Prop is converted into time quanta (rounded up to the nearest integer multiple of $t_q$).

Sync is 1 $t_q$ long (fixed), which leaves (bit time - Prop - 1) $t_q$ for the two Phase Buffer Segments. If the number of remaining $t_q$ is even, the Phase Buffer Segments have the same length, that is, Phase2 = Phase1, else Phase2 = Phase1 + 1.

The minimum nominal length of Phase2 has to be regarded as well. Phase2 may not be shorter than the CAN controller's Information Processing Time, which is, depending on the actual implementation, in the range of [0..2] $t_q$.

The length of the synchronization jump width is set to the least of 4, Phase1 or Phase2.

The oscillator tolerance range necessary for the resulting configuration is calculated by the formula given below:

$$(1 - df) \times fnom \leq fosc \leq (1 + df) \times fnom$$

where:

- df = Maximum tolerance of oscillator frequency
- fosc = Actual oscillator frequency

- fnom = Nominal oscillator frequency

Maximum frequency tolerance must take into account the following formulas:

$$df \leq \frac{(Phase\_seg1, Phase\_seg2)\,\min}{2 \times (13 \times tbit - Phase\_Seg2)}$$

$$df\ \max = 2 \times df \times fnom$$

where:

- Phase1 and Phase2 are from Table 16-2 on page 627
- tbit = Bit Time
- dfmax = Maximum difference between two oscillators

If more than one configuration is possible, that configuration allowing the highest oscillator tolerance range should be chosen.

CAN nodes with different system clocks require different configurations to come to the same bit rate. The calculation of the propagation time in the CAN network, based on the nodes with the longest delay times, is done once for the whole network.

The CAN system's oscillator tolerance range is limited by the node with the lowest tolerance range.

The calculation may show that bus length or bit rate have to be decreased or that the oscillator frequencies' stability has to be increased in order to find a protocol-compliant configuration of the CAN bit timing.

### 16.3.16.1 Example for Bit Timing at High Baud Rate

In this example, the frequency of CAN clock is 25 MHz, and the bit rate is 1 Mbps.

```
bit time = 1 µs = n * tq = 5 * tq
tq = 200 ns
tq = (Baud rate Prescaler)/CAN Clock
Baud rate Prescaler = tq * CAN Clock
Baud rate Prescaler = 200E-9 * 25E6 = 5

tSync = 1 * tq = 200 ns                \\fixed at 1 time quanta

delay of bus driver 50 ns
delay of receiver circuit 30 ns
delay of bus line (40m) 220 ns
tProp 400 ns = 2 * tq                  \\400 is next integer multiple of tq

bit time = tSync + tTSeg1 + tTSeg2 = 5 * tq
bit time = tSync + tProp + tPhase 1 + tPhase2
tPhase 1 + tPhase2 = bit time - tSync - tProp
tPhase 1 + tPhase2 = (5 * tq) - (1 * tq) - (2 * tq)
tPhase 1 + tPhase2 = 2 * tq
```

```
tPhase1 = 1 * tq
tPhase2 = 1 * tq                          \\tPhase2 = tPhase1

tTSeg1 = tProp + tPhase1
tTSeg1 = (2 * tq) + (1 * tq)
tTSeg1 = 3 * tq

tTSeg2 = tPhase2
tTSeg2 = (Information Processing Time + 1) * tq
tTSeg2 = 1 * tq                           \\Assumes IPT=0

tSJW = 1 * tq                             \\Least of 4, Phase1 and Phase2
```

In the above example, the bit field values for the **CANBIT** register are:

| | |
|---|---|
| TSEG2 | = TSeg2 -1<br>= 1-1<br>= 0 |
| TSEG1 | = TSeg1 -1<br>= 3-1<br>= 2 |
| SJW | = SJW -1<br>= 1-1<br>= 0 |
| BRP | = Baud rate prescaler - 1<br>= 5-1<br>=4 |

The final value programmed into the **CANBIT** register = 0x0204.

### 16.3.16.2 Example for Bit Timing at Low Baud Rate

In this example, the frequency of the CAN clock is 50 MHz, and the bit rate is 100 Kbps.

```
bit time = 10 µs = n * tq = 10 * tq
tq = 1 µs
tq = (Baud rate Prescaler)/CAN Clock
Baud rate Prescaler = tq * CAN Clock
Baud rate Prescaler = 1E-6 * 50E6 = 50

tSync = 1 * tq = 1 µs                     \\fixed at 1 time quanta

delay of bus driver 200 ns
delay of receiver circuit 80 ns
delay of bus line (40m) 220 ns
tProp 1 µs = 1 * tq                       \\1 µs is next integer multiple of tq

bit time = tSync + tTSeg1 + tTSeg2 = 10 * tq
bit time = tSync + tProp + tPhase 1 + tPhase2
tPhase 1 + tPhase2 = bit time - tSync - tProp
tPhase 1 + tPhase2 = (10 * tq) - (1 * tq)  - (1 * tq)
tPhase 1 + tPhase2 = 8 * tq
```

```
tPhase1 = 4 * tq
tPhase2 = 4 * tq                        \\tPhase1 = tPhase2

tTSeg1 = tProp + tPhase1
tTSeg1 = (1 * tq) + (4 * tq)
tTSeg1 = 5 * tq
tTSeg2 = tPhase2
tTSeg2 = (Information Processing Time + 4) × tq
tTSeg2 = 4 * tq                         \\Assumes IPT=0

tSJW = 4 * tq                           \\Least of 4, Phase1, and Phase2
```

| | |
|---|---|
| TSEG2 | = TSeg2 -1<br>= 4-1<br>= 3 |
| TSEG1 | = TSeg1 -1<br>= 5-1<br>= 4 |
| SJW | = SJW -1<br>= 4-1<br>= 3 |
| BRP | = Baud rate prescaler - 1<br>= 50-1<br>=49 |

The final value programmed into the **CANBIT** register = 0x34F1.

## 16.4 Register Map

Table 16-4 on page 631 lists the registers. All addresses given are relative to the CAN base address of:

- CAN0: 0x4004.0000

Note that the CAN module clock must be enabled before the registers can be programmed (see page 217). There must be a delay of 3 system clocks after the CAN module clock is enabled before any CAN module registers are accessed.

**Table 16-4. CAN Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | CANCTL | R/W | 0x0000.0001 | CAN Control | 634 |
| 0x004 | CANSTS | R/W | 0x0000.0000 | CAN Status | 636 |
| 0x008 | CANERR | RO | 0x0000.0000 | CAN Error Counter | 638 |
| 0x00C | CANBIT | R/W | 0x0000.2301 | CAN Bit Timing | 639 |
| 0x010 | CANINT | RO | 0x0000.0000 | CAN Interrupt | 640 |
| 0x014 | CANTST | R/W | 0x0000.0000 | CAN Test | 641 |
| 0x018 | CANBRPE | R/W | 0x0000.0000 | CAN Baud Rate Prescaler Extension | 643 |

**Table 16-4. CAN Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x020 | CANIF1CRQ | R/W | 0x0000.0001 | CAN IF1 Command Request | 644 |
| 0x024 | CANIF1CMSK | R/W | 0x0000.0000 | CAN IF1 Command Mask | 645 |
| 0x028 | CANIF1MSK1 | R/W | 0x0000.FFFF | CAN IF1 Mask 1 | 647 |
| 0x02C | CANIF1MSK2 | R/W | 0x0000.FFFF | CAN IF1 Mask 2 | 648 |
| 0x030 | CANIF1ARB1 | R/W | 0x0000.0000 | CAN IF1 Arbitration 1 | 649 |
| 0x034 | CANIF1ARB2 | R/W | 0x0000.0000 | CAN IF1 Arbitration 2 | 650 |
| 0x038 | CANIF1MCTL | R/W | 0x0000.0000 | CAN IF1 Message Control | 652 |
| 0x03C | CANIF1DA1 | R/W | 0x0000.0000 | CAN IF1 Data A1 | 654 |
| 0x040 | CANIF1DA2 | R/W | 0x0000.0000 | CAN IF1 Data A2 | 654 |
| 0x044 | CANIF1DB1 | R/W | 0x0000.0000 | CAN IF1 Data B1 | 654 |
| 0x048 | CANIF1DB2 | R/W | 0x0000.0000 | CAN IF1 Data B2 | 654 |
| 0x080 | CANIF2CRQ | R/W | 0x0000.0001 | CAN IF2 Command Request | 644 |
| 0x084 | CANIF2CMSK | R/W | 0x0000.0000 | CAN IF2 Command Mask | 645 |
| 0x088 | CANIF2MSK1 | R/W | 0x0000.FFFF | CAN IF2 Mask 1 | 647 |
| 0x08C | CANIF2MSK2 | R/W | 0x0000.FFFF | CAN IF2 Mask 2 | 648 |
| 0x090 | CANIF2ARB1 | R/W | 0x0000.0000 | CAN IF2 Arbitration 1 | 649 |
| 0x094 | CANIF2ARB2 | R/W | 0x0000.0000 | CAN IF2 Arbitration 2 | 650 |
| 0x098 | CANIF2MCTL | R/W | 0x0000.0000 | CAN IF2 Message Control | 652 |
| 0x09C | CANIF2DA1 | R/W | 0x0000.0000 | CAN IF2 Data A1 | 654 |
| 0x0A0 | CANIF2DA2 | R/W | 0x0000.0000 | CAN IF2 Data A2 | 654 |
| 0x0A4 | CANIF2DB1 | R/W | 0x0000.0000 | CAN IF2 Data B1 | 654 |
| 0x0A8 | CANIF2DB2 | R/W | 0x0000.0000 | CAN IF2 Data B2 | 654 |
| 0x100 | CANTXRQ1 | RO | 0x0000.0000 | CAN Transmission Request 1 | 655 |
| 0x104 | CANTXRQ2 | RO | 0x0000.0000 | CAN Transmission Request 2 | 655 |
| 0x120 | CANNWDA1 | RO | 0x0000.0000 | CAN New Data 1 | 656 |
| 0x124 | CANNWDA2 | RO | 0x0000.0000 | CAN New Data 2 | 656 |
| 0x140 | CANMSG1INT | RO | 0x0000.0000 | CAN Message 1 Interrupt Pending | 657 |
| 0x144 | CANMSG2INT | RO | 0x0000.0000 | CAN Message 2 Interrupt Pending | 657 |
| 0x160 | CANMSG1VAL | RO | 0x0000.0000 | CAN Message 1 Valid | 658 |
| 0x164 | CANMSG2VAL | RO | 0x0000.0000 | CAN Message 2 Valid | 658 |

## 16.5 CAN Register Descriptions

The remainder of this section lists and describes the CAN registers, in numerical order by address offset. There are two sets of Interface Registers that are used to access the Message Objects in the Message RAM: **CANIF1x** and **CANIF2x**. The function of the two sets are identical and are used to queue transactions.

## Register 1: CAN Control (CANCTL), offset 0x000

This control register initializes the module and enables test mode and interrupts.

The bus-off recovery sequence (see CAN Specification Rev. 2.0) cannot be shortened by setting or clearing INIT. If the device goes bus-off, it sets INIT, stopping all bus activities. Once INIT has been cleared by the CPU, the device then waits for 129 occurrences of Bus Idle (129 * 11 consecutive High bits) before resuming normal operations. At the end of the bus-off recovery sequence, the Error Management Counters are reset.

During the waiting time after INIT is cleared, each time a sequence of 11 High bits has been monitored, a BITERROR0 code is written to the **CANSTS** register (the LEC field = 0x5), enabling the CPU to readily check whether the CAN bus is stuck Low or continuously disturbed, and to monitor the proceeding of the bus-off recovery sequence.

CAN Control (CANCTL)

CAN0 base: 0x4004.0000
Offset 0x000
Type R/W, reset 0x0000.0001

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | | | | | | | | | | | |
| RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | | | | TEST | CCE | DAR | reserved | EIE | SIE | IE | INIT |
| RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | RO | R/W | R/W | R/W | R/W |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7 | TEST | R/W | 0 | Test Mode Enable<br>0: Normal operation<br>1: Test mode |
| 6 | CCE | R/W | 0 | Configuration Change Enable<br>0: Do not allow write access to the **CANBIT** register.<br>1: Allow write access to the **CANBIT** register if the INIT bit is 1. |
| 5 | DAR | R/W | 0 | Disable Automatic-Retransmission<br>0: Auto-retransmission of disturbed messages is enabled.<br>1: Auto-retransmission is disabled. |
| 4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EIE | R/W | 0 | Error Interrupt Enable<br>0: Disabled. No error status interrupt is generated.<br>1: Enabled. A change in the BOFF or EWARN bits in the **CANSTS** register generates an interrupt. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | SIE | R/W | 0 | Status Interrupt Enable<br>0: Disabled. No status interrupt is generated.<br>1: Enabled. An interrupt is generated when a message has successfully been transmitted or received, or a CAN bus error has been detected. A change in the TXOK, RXOK or LEC bits in the **CANSTS** register generates an interrupt. |
| 1 | IE | R/W | 0 | CAN Interrupt Enable<br>0: Interrupts disabled.<br>1: Interrupts enabled. |
| 0 | INIT | R/W | 1 | Initialization<br>0: Normal operation.<br>1: Initialization started. |

## Register 2: CAN Status (CANSTS), offset 0x004

**Important:** This register is read-sensitive. See the register description for details.

The status register contains information for interrupt servicing such as Bus-Off, error count threshold, and error types.

The LEC field holds the code that indicates the type of the last error to occur on the CAN bus. This field is cleared when a message has been transferred (reception or transmission) without error. The unused error code 7 may be written by the CPU to manually set this field to an invalid error so that it can be checked for a change later.

An error interrupt is generated by the BOFF and EWARN bits and a status interrupt is generated by the RXOK, TXOK, and LEC bits, if the corresponding enable bits in the **CAN Control (CANCTL)** register are set. A change of the EPASS bit or a write to the RXOK, TXOK, or LEC bits does not generate an interrupt.

Reading the **CAN Status (CANSTS)** register clears the **CAN Interrupt (CANINT)** register, if it is pending.

CAN Status (CANSTS)

CAN0 base: 0x4004.0000
Offset 0x004
Type R/W, reset 0x0000.0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | | | | | | | | | | | |
| Type RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | | | | BOFF | EWARN | EPASS | RXOK | TXOK | LEC | | |
| Type RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W |
| Reset 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7 | BOFF | RO | 0 | Bus-Off Status<br>0: CAN controller is not in bus-off state.<br>1: CAN controller is in bus-off state. |
| 6 | EWARN | RO | 0 | Warning Status<br>0: Both error counters are below the error warning limit of 96.<br>1: At least one of the error counters has reached the error warning limit of 96. |
| 5 | EPASS | RO | 0 | Error Passive<br>0: The CAN module is in the Error Active state, that is, the receive or transmit error count is less than or equal to 127.<br>1: The CAN module is in the Error Passive state, that is, the receive or transmit error count is greater than 127. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | RXOK | R/W | 0 | Received a Message Successfully<br>0: Since this bit was last cleared, no message has been successfully received.<br>1: Since this bit was last cleared, a message has been successfully received, independent of the result of the acceptance filtering.<br>This bit is never cleared by the CAN module. |
| 3 | TXOK | R/W | 0 | Transmitted a Message Successfully<br>0: Since this bit was last cleared, no message has been successfully transmitted.<br>1: Since this bit was last cleared, a message has been successfully transmitted error-free and acknowledged by at least one other node.<br>This bit is never cleared by the CAN module. |
| 2:0 | LEC | R/W | 0x0 | Last Error Code<br>This is the type of the last error to occur on the CAN bus. |

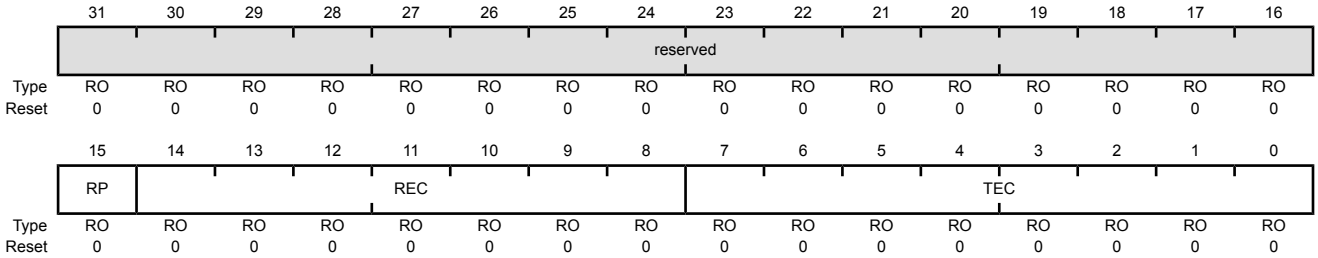
| Value | Definition |
|---|---|
| 0x0 | No Error |
| 0x1 | Stuff Error<br>More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed. |
| 0x2 | Format Error<br>A fixed format part of the received frame has the wrong format. |
| 0x3 | ACK Error<br>The message transmitted was not acknowledged by another node. |
| 0x4 | Bit 1 Error<br>When a message is transmitted, the CAN controller monitors the data lines to detect any conflicts. When the arbitration field is transmitted, data conflicts are a part of the arbitration protocol. When other frame fields are transmitted, data conflicts are considered errors.<br>A Bit 1 Error indicates that the device wanted to send a High level (logical 1) but the monitored bus value was Low (logical 0). |
| 0x5 | Bit 0 Error<br>A Bit 0 Error indicates that the device wanted to send a Low level (logical 0), but the monitored bus value was High (logical 1).<br>During bus-off recovery, this status is set each time a sequence of 11 High bits has been monitored. This enables the CPU to monitor the proceeding of the bus-off recovery sequence without any disturbances to the bus. |
| 0x6 | CRC Error<br>The CRC checksum was incorrect in the received message, indicating that the calculated value received did not match the calculated CRC of the data. |
| 0x7 | No Event<br>When the LEC bit shows this value, no CAN bus event was detected since the CPU wrote this value to LEC. |

## Register 3: CAN Error Counter (CANERR), offset 0x008

This register contains the error counter values, which can be used to analyze the cause of an error.

CAN Error Counter (CANERR)

CAN0 base: 0x4004.0000
Offset 0x008
Type RO, reset 0x0000.0000



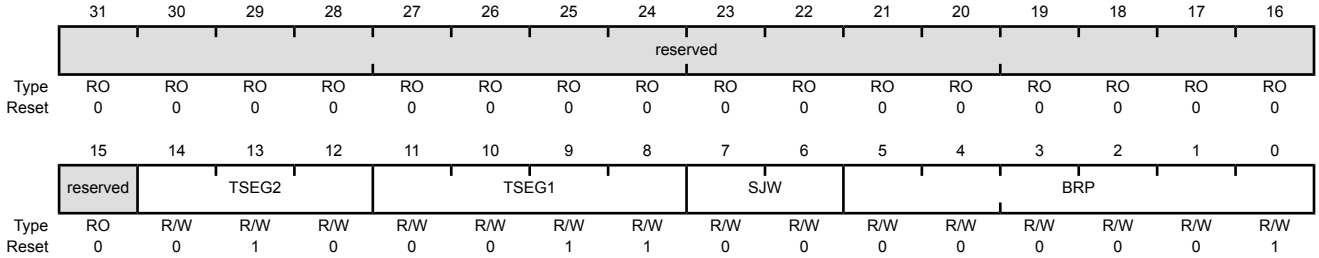

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15 | RP | RO | 0 | Received Error Passive<br>0: The Receive Error counter is below the Error Passive level (127 or less).<br>1: The Receive Error counter has reached the Error Passive level (128 or greater). |
| 14:8 | REC | RO | 0x00 | Receive Error Counter<br>State of the receiver error counter (0 to 127). |
| 7:0 | TEC | RO | 0x00 | Transmit Error Counter<br>State of the transmit error counter (0 to 255). |

## Register 4: CAN Bit Timing (CANBIT), offset 0x00C

This register is used to program the bit width and bit quantum. Values are programmed to the system clock frequency. This register is write-enabled by setting the CCE and INIT bits in the **CANCTL** register. See “Bit Time and Bit Rate” on page 626 for more information.

CAN Bit Timing (CANBIT)

CAN0 base: 0x4004.0000
Offset 0x00C
Type R/W, reset 0x0000.2301




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:15 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14:12 | TSEG2 | R/W | 0x2 | Time Segment after Sample Point<br>0x00-0x07: The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.<br>So, for example, a reset value of 0x2 defines that there is 3 (2+1) bit time quanta defined for Phase_Seg2 (see Figure 16-4 on page 627). The bit time quanta is defined by the BRP field. |
| 11:8 | TSEG1 | R/W | 0x3 | Time Segment Before Sample Point<br>0x00-0x0F: The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.<br>So, for example, the reset value of 0x3 defines that there is 4 (3+1) bit time quanta defined for Phase_Seg1 (see Figure 16-4 on page 627). The bit time quanta is define by the BRP field. |
| 7:6 | SJW | R/W | 0x0 | (Re)Synchronization Jump Width<br>0x00-0x03: The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.<br>During the start of frame (SOF), if the CAN controller detects a phase error (misalignment), it can adjust the length of TSEG2 or TSEG1 by the value in SJW. So the reset value of 0 adjusts the length by 1 bit time quanta. |
| 5:0 | BRP | R/W | 0x1 | Baud Rate Prescaler<br>The value by which the oscillator frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quantum.<br>0x00-0x03F: The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.<br>BRP defines the number of CAN clock periods that make up 1 bit time quanta, so the reset value is 2 bit time quanta (1+1).<br>The **CANBRPE** register can be used to further divide the bit time. |

## Register 5: CAN Interrupt (CANINT), offset 0x010

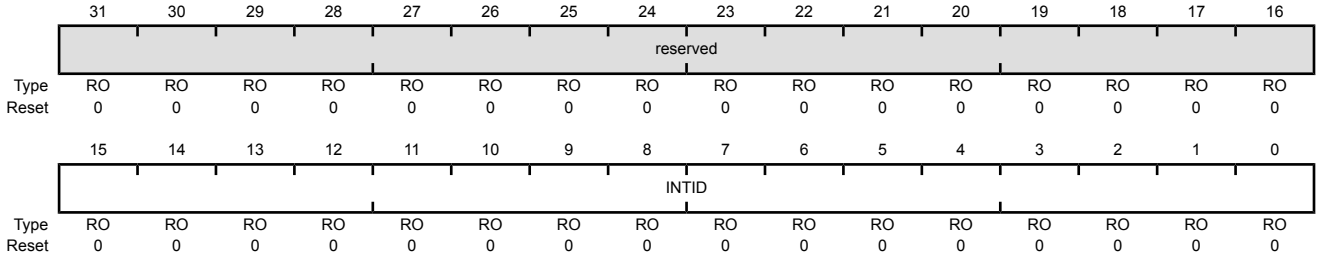
This register indicates the source of the interrupt.

If several interrupts are pending, the **CAN Interrupt (CANINT)** register points to the pending interrupt with the highest priority, disregarding the order in which the interrupts occurred. An interrupt remains pending until the CPU has cleared it. If the `INTID` field is not 0x0000 (the default) and the `IE` bit in the **CANCTL** register is set, the interrupt is active. The interrupt line remains active until the `INTID` field is cleared by reading the **CANSTS** register, or until the `IE` bit in the **CANCTL** register is cleared.

**Note:** Reading the **CAN Status (CANSTS)** register clears the **CAN Interrupt (CANINT)** register, if it is pending.

CAN Interrupt (CANINT)

CAN0 base: 0x4004.0000
Offset 0x010
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | INTID | RO | 0x0000 | Interrupt Identifier<br>The number in this field indicates the source of the interrupt. |

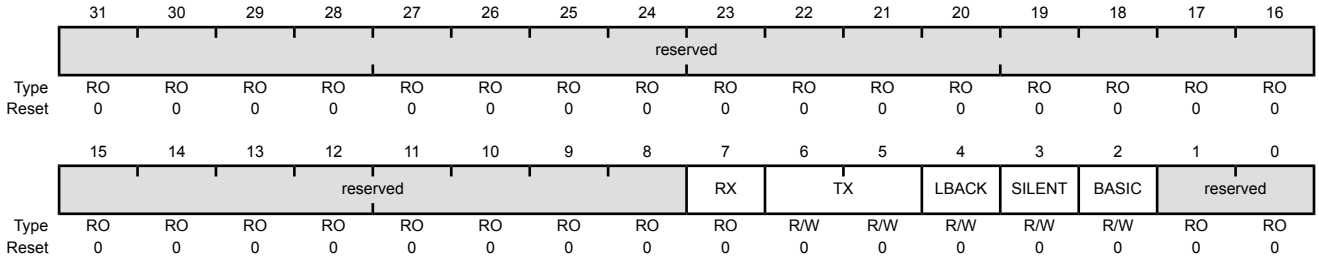
| Value | Definition |
|---|---|
| 0x0000 | No interrupt pending |
| 0x0001-0x0020 | Number of the message object that caused the interrupt |
| 0x0021-0x7FFF | Reserved |
| 0x8000 | Status Interrupt |
| 0x8001-0xFFFF | Reserved |

## Register 6: CAN Test (CANTST), offset 0x014

This is the test mode register for self-test and external pin access. It is write-enabled by setting the TEST bit in the **CANCTL** register. Different test functions may be combined, however, CAN transfers will be affected if the TX bits in this register are not zero.

CAN Test (CANTST)

CAN0 base: 0x4004.0000
Offset 0x014
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7 | RX | RO | 0 | Receive Observation<br>Displays the value on the CANnRx pin. |
| 6:5 | TX | R/W | 0x0 | Transmit Control<br>Overrides control of the CANnTx pin. |

| Value | Description |
|---|---|
| 0x0 | CAN Module Control<br>CANnTx is controlled by the CAN module; default operation |
| 0x1 | Sample Point<br>The sample point is driven on the CANnTx signal. This mode is useful to monitor bit timing. |
| 0x2 | Driven Low<br>CANnTx drives a low value. This mode is useful for checking the physical layer of the CAN bus. |
| 0x3 | Driven High<br>CANnTx drives a high value. This mode is useful for checking the physical layer of the CAN bus. |

| | | | | |
|---|---|---|---|---|
| 4 | LBACK | R/W | 0 | Loopback Mode<br>0: Disabled.<br>1: Enabled. In loopback mode, the data from the transmitter is routed into the receiver. Any data on the receive input is ignored. |

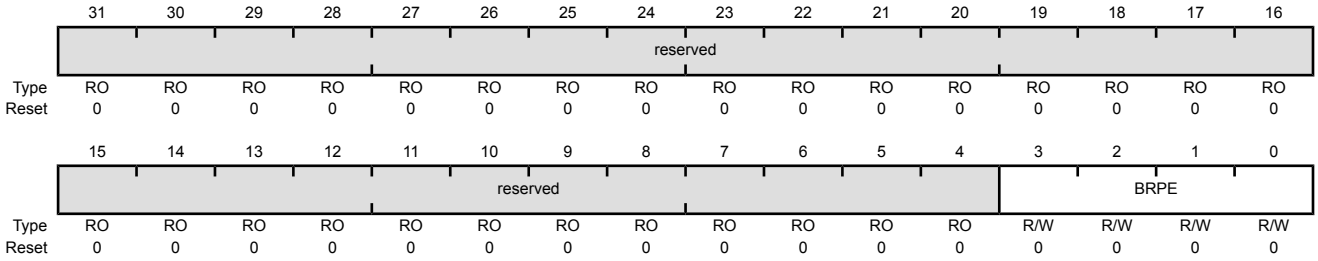
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | SILENT | R/W | 0 | Silent Mode<br>Do not transmit data; monitor the bus. Also known as Bus Monitor mode.<br>0: Disabled.<br>1: Enabled. |
| 2 | BASIC | R/W | 0 | Basic Mode<br>0: Disabled.<br>1: Use **CANIF1** registers as transmit buffer, and use **CANIF2** registers as receive buffer. |
| 1:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 7: CAN Baud Rate Prescaler Extension (CANBRPE), offset 0x018

This register is used to further divide the bit time set with the BRP bit in the **CANBIT** register. It is write-enabled by setting the CCE bit in the **CANCTL** register.

CAN Baud Rate Prescaler Extension (CANBRPE)

CAN0 base: 0x4004.0000
Offset 0x018
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3:0 | BRPE | R/W | 0x0 | Baud Rate Prescaler Extension<br>0x00-0x0F: Extend the BRP bit in the **CANBIT** register to values up to 1023. The actual interpretation by the hardware is one more than the value programmed by BRPE (MSBs) and BRP (LSBs). |

## Register 8: CAN IF1 Command Request (CANIF1CRQ), offset 0x020

## Register 9: CAN IF2 Command Request (CANIF2CRQ), offset 0x080

A message transfer is started as soon as there is a write of the message object number to the MNUM field when the TXRQST bit in the **CANIF1MCTL** register is set. With this write operation, the BUSY bit is automatically set to indicate that a transfer between the CAN Interface Registers and the internal message RAM is in progress. After a wait time of 3 to 6 CAN_CLK periods, the transfer between the interface register and the message RAM completes, which then clears the BUSY bit.

CAN IF1 Command Request (CANIF1CRQ)

CAN0 base: 0x4004.0000
Offset 0x020
Type R/W, reset 0x0000.0001

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BUSY | reserved | | | | | | | | | MNUM | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15 | BUSY | RO | 0 | Busy Flag<br>0: Cleared when read/write action has finished.<br>1: Set when a write occurs to the message number in this register. |
| 14:6 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:0 | MNUM | R/W | 0x01 | Message Number<br>Selects one of the 32 message objects in the message RAM for data transfer. The message objects are numbered from 1 to 32. |

| Value | Description |
|---|---|
| 0x00 | Reserved<br>0 is not a valid message number; it is interpreted as 0x20, or object 32. |
| 0x01-0x20 | Message Number<br>Indicates specified message object 1 to 32. |
| 0x21-0x3F | Reserved<br>Not a valid message number; values are shifted and it is interpreted as 0x01-0x1F. |

## Register 10: CAN IF1 Command Mask (CANIF1CMSK), offset 0x024

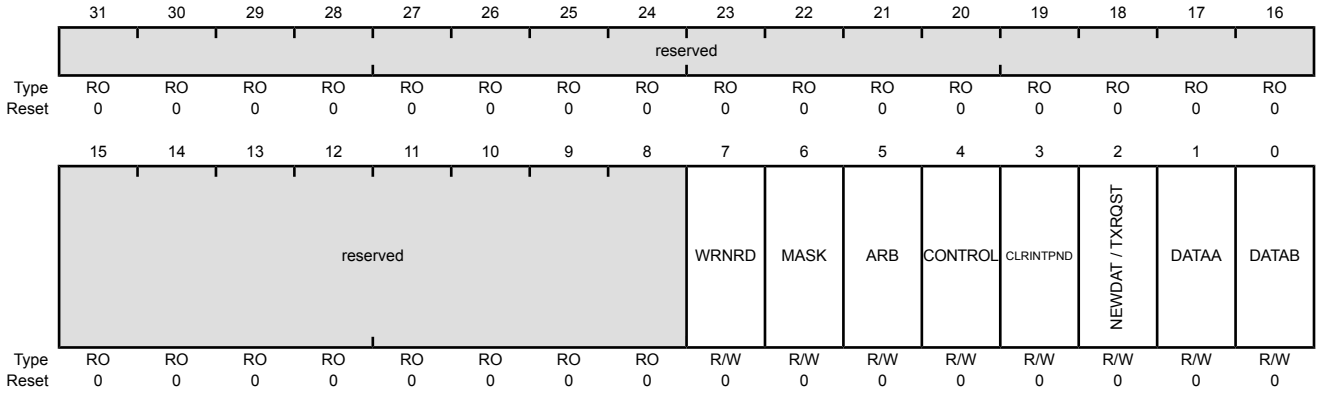
## Register 11: CAN IF2 Command Mask (CANIF2CMSK), offset 0x084

Reading the Command Mask registers provides status for various functions. Writing to the Command Mask registers specifies the transfer direction and selects which buffer registers are the source or target of the data transfer.

Note that when a read from the message object buffer occurs when the WRNRD bit is clear and the CLRINTPND and/or NEWDAT bits are set, the interrupt pending and/or new data flags in the message object buffer are cleared.

CAN IF1 Command Mask (CANIF1CMSK)

CAN0 base: 0x4004.0000
Offset 0x024
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7 | WRNRD | R/W | 0 | Write, Not Read<br>Transfer the message object address specified by the **CAN Command Request (CANIFnCRQ)** register to the CAN message buffer registers.<br>**Note:** Interrupt pending and new data conditions in the message buffer can be cleared by reading from the buffer (WRNRD = 0) when the CLRINTPND and/or NEWDAT bits are set. |
| 6 | MASK | R/W | 0 | Access Mask Bits<br>0: Mask bits unchanged.<br>1: Transfer IDMASK + DIR + MXTD of the message object into the Interface registers. |
| 5 | ARB | R/W | 0 | Access Arbitration Bits<br>0: Arbitration bits unchanged.<br>1: Transfer ID + DIR + XTD + MSGVAL of the message object into the Interface registers. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | CONTROL | R/W | 0 | Access Control Bits<br>0: Control bits unchanged.<br>1: Transfer control bits from the **CANIFnMCTL** register into the Interface registers. |
| 3 | CLRINTPND | R/W | 0 | Clear Interrupt Pending Bit<br>If WRNRD is set, this bit controls whether the INTPND bit in the **CANIFnMCTL** register is changed.<br>0: The INTPND bit in the message object remains unchanged.<br>1: The INTPND bit is cleared in the message object.<br>If WRNRD is clear and this bit is clear, the interrupt pending status is transferred from the message buffer into the **CANIFnMCTL** register.<br>If WRNRD is clear and this bit is set, the interrupt pending status is cleared in the message buffer. Note that the value of this bit that is transferred to the **CANIFnMCTL** register always reflects the status of the bits before clearing. |
| 2 | NEWDAT / TXRQST | R/W | 0 | NEWDAT / TXRQST Bit<br>If WRNRD is set, this bit can act as a TXRQST bit and request a transmission. Note that when this bit is set, the TXRQST bit in the **CANIFnMCTL** register is ignored.<br>0: Transmission is not requested<br>1: Begin a transmission<br>If WRNRD is clear and this bit is clear, the value of the new data status is transferred from the message buffer into the **CANIFnMCTL** register.<br>If WRNRD is clear and this bit is set, the new data status is cleared in the message buffer. Note that the value of this bit that is transferred to the **CANIFnMCTL** register always reflects the status of the bits before clearing. |
| 1 | DATAA | R/W | 0 | Access Data Byte 0 to 3<br>When WRNRD = 1:<br>0: Data bytes 0-3 are unchanged.<br>1: Transfer data bytes 0-3 in message object to **CANIFnDA1** and **CANIFnDA2**.<br>When WRNRD = 0:<br>0: Data bytes 0-3 are unchanged.<br>1: Transfer data bytes 0-3 in **CANIFnDA1** and **CANIFnDA2** to the message object. |
| 0 | DATAB | R/W | 0 | Access Data Byte 4 to 7<br>When WRNRD = 1:<br>0: Data bytes 4-7 are unchanged.<br>1: Transfer data bytes 4-7 in message object to **CANIFnDB1** and **CANIFnDB2**.<br>When WRNRD = 0:<br>0: Data bytes 4-7 are unchanged.<br>1: Transfer data bytes 4-7 in **CANIFnDB1** and **CANIFnDB2** to the message object. |

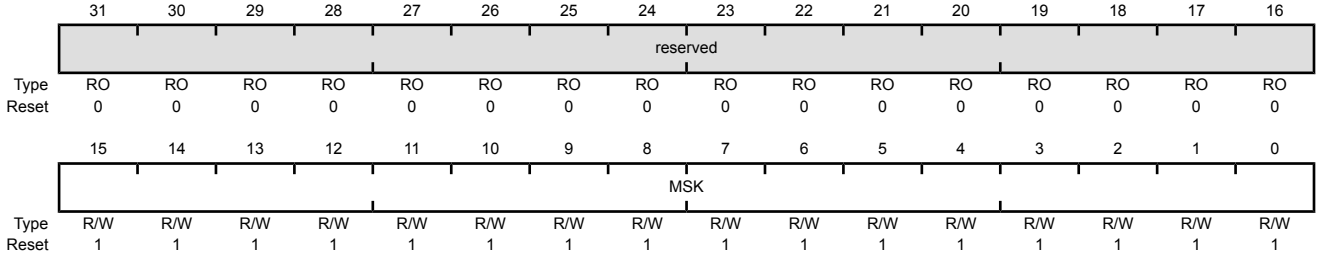
## Register 12: CAN IF1 Mask 1 (CANIF1MSK1), offset 0x028

## Register 13: CAN IF2 Mask 1 (CANIF2MSK1), offset 0x088

The mask information provided in this register accompanies the data (**CANIFnDAn**), arbitration information (**CANIFnARBn**), and control information (**CANIFnMCTL**) to the message object in the message RAM. The mask is used with the ID bit in the **CANIFnARBn** register for acceptance filtering. Additional mask information is contained in the **CANIFnMSK2** register.

CAN IF1 Mask 1 (CANIF1MSK1)

CAN0 base: 0x4004.0000
Offset 0x028
Type R/W, reset 0x0000.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | MSK | R/W | 0xFFFF | Identifier Mask<br>When using a 29-bit identifier, these bits are used for bits [15:0] of the ID. The MSK field in the **CANIFnMSK2** register are used for bits [28:16] of the ID. When using an 11-bit identifier, these bits are ignored.<br>0: The corresponding identifier field (ID) in the message object cannot inhibit the match in acceptance filtering.<br>1: The corresponding identifier field (ID) is used for acceptance filtering. |

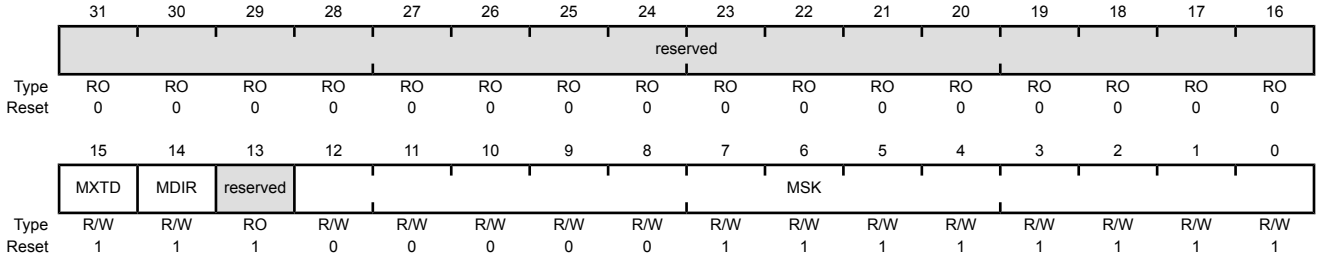
## Register 14: CAN IF1 Mask 2 (CANIF1MSK2), offset 0x02C

## Register 15: CAN IF2 Mask 2 (CANIF2MSK2), offset 0x08C

This register holds extended mask information that accompanies the **CANIFnMSK1** register.

CAN IF1 Mask 2 (CANIF1MSK2)

CAN0 base: 0x4004.0000
Offset 0x02C
Type R/W, reset 0x0000.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15 | MXTD | R/W | 0x1 | Mask Extended Identifier<br>0: The extended identifier bit (XTD in the **CANIFnARB2** register) has no effect on the acceptance filtering.<br>1: The extended identifier bit XTD is used for acceptance filtering. |
| 14 | MDIR | R/W | 0x1 | Mask Message Direction<br>0: The message direction bit (DIR in the **CANIFnARB2** register) has no effect for acceptance filtering.<br>1: The message direction bit DIR is used for acceptance filtering. |
| 13 | reserved | RO | 0x1 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12:0 | MSK | R/W | 0xFF | Identifier Mask<br>When using a 29-bit identifier, these bits are used for bits [28:16] of the ID. The MSK field in the **CANIFnMSK1** register are used for bits [15:0] of the ID. When using an 11-bit identifier, MSK[12:2] are used for bits [10:0] of the ID.<br>0: The corresponding identifier field (ID) in the message object cannot inhibit the match in acceptance filtering.<br>1: The corresponding identifier field (ID) is used for acceptance filtering. |

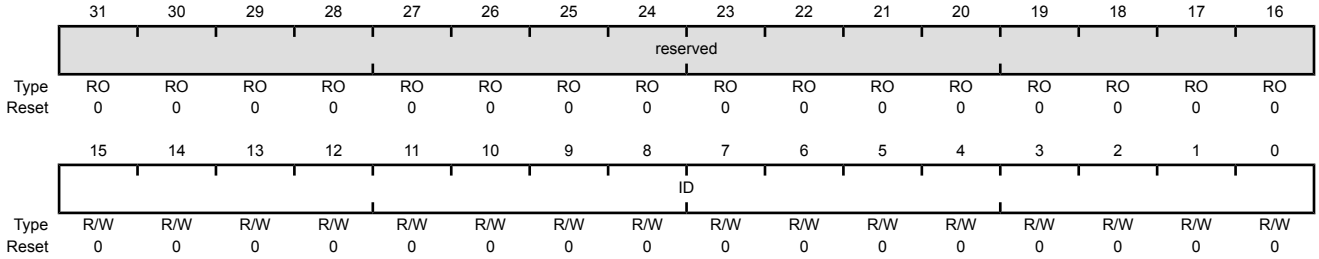
## Register 16: CAN IF1 Arbitration 1 (CANIF1ARB1), offset 0x030

## Register 17: CAN IF2 Arbitration 1 (CANIF2ARB1), offset 0x090

These registers hold the identifiers for acceptance filtering.

CAN IF1 Arbitration 1 (CANIF1ARB1)

CAN0 base: 0x4004.0000
Offset 0x030
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | ID | R/W | 0x0000 | Message Identifier<br>This bit field is used with the `ID` field in the **CANIFnARB2** register to create the message identifier.<br>When using a 29-bit identifier, bits 15:0 of the **CANIFnARB1** register are [15:0] of the ID, while bits 12:0 of the **CANIFnARB2** register are [28:16] of the ID.<br>When using an 11-bit identifier, these bits are not used. |

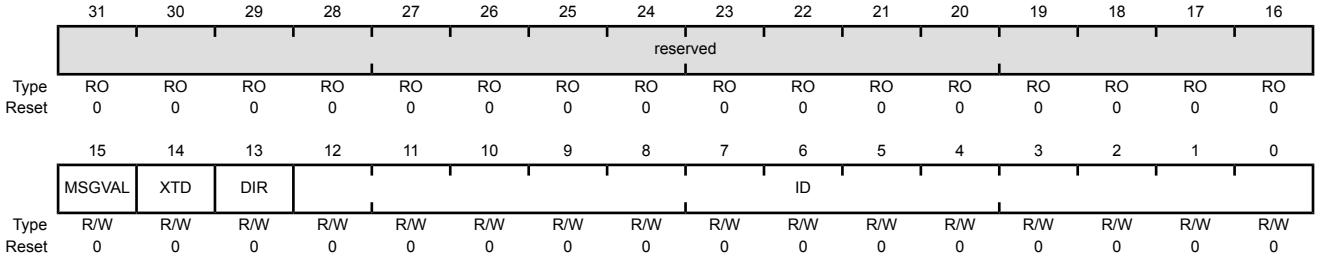
## Register 18: CAN IF1 Arbitration 2 (CANIF1ARB2), offset 0x034

## Register 19: CAN IF2 Arbitration 2 (CANIF2ARB2), offset 0x094

These registers hold information for acceptance filtering.

CAN IF1 Arbitration 2 (CANIF1ARB2)

CAN0 base: 0x4004.0000
Offset 0x034
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15 | MSGVAL | R/W | 0 | Message Valid<br>0: The message object is ignored by the message handler.<br>1: The message object is configured and ready to be considered by the message handler within the CAN controller.<br>All unused message objects should have this bit cleared during initialization and before clearing the `INIT` bit in the **CANCTL** register. The `MSGVAL` bit must also be cleared before any of the following bits are modified or if the message object is no longer required: the `ID` fields in the **CANIFnARBn** registers, the `XTD` and `DIR` bits in the **CANIFnARB2** register, or the `DLC` field in the **CANIFnMCTL** register. |
| 14 | XTD | R/W | 0 | Extended Identifier<br>0: An 11-bit Standard Identifier is used for this message object.<br>1: A 29-bit Extended Identifier is used for this message object. |
| 13 | DIR | R/W | 0 | Message Direction<br>0: Receive. When the `TXRQST` bit in the **CANIFnMCTL** register is set, a remote frame with the identifier of this message object is received. On reception of a data frame with matching identifier, that message is stored in this message object.<br>1: Transmit. When the `TXRQST` bit in the **CANIFnMCTL** register is set, the respective message object is transmitted as a data frame. On reception of a remote frame with matching identifier, the `TXRQST` bit of this message object is set (if `RMTEN`=1). |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 12:0 | ID | R/W | 0x000 | Message Identifier<br>This bit field is used with the `ID` field in the **CANIFnARB2** register to create the message identifier.<br>When using a 29-bit identifier, `ID[15:0]` of the **CANIFnARB1** register are [15:0] of the ID, while these bits, `ID[12:0]`, are [28:16] of the ID.<br>When using an 11-bit identifier, `ID[12:2]` are used for bits [10:0] of the ID. The `ID` field in the **CANIFnARB1** register is ignored. |

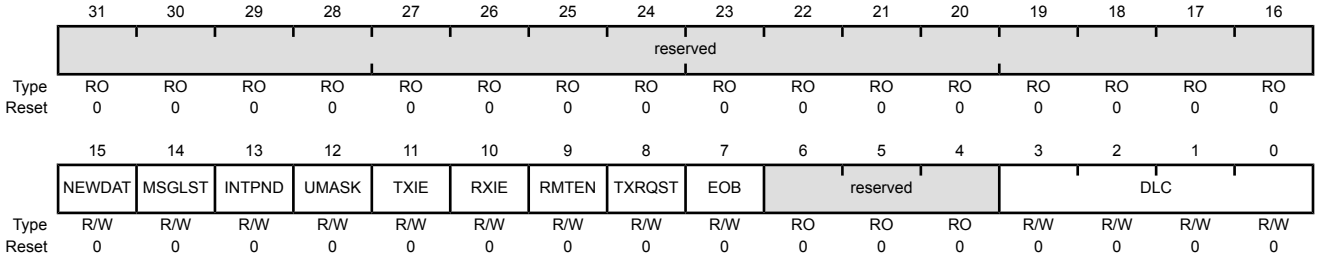
## Register 20: CAN IF1 Message Control (CANIF1MCTL), offset 0x038

## Register 21: CAN IF2 Message Control (CANIF2MCTL), offset 0x098

This register holds the control information associated with the message object to be sent to the Message RAM.

CAN IF1 Message Control (CANIF1MCTL)

CAN0 base: 0x4004.0000
Offset 0x038
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15 | NEWDAT | R/W | 0 | New Data<br>0: No new data has been written into the data portion of this message object by the message handler since the last time this flag was cleared by the CPU.<br>1: The message handler or the CPU has written new data into the data portion of this message object. |
| 14 | MSGLST | R/W | 0 | Message Lost<br>0 : No message was lost since the last time this bit was cleared by the CPU.<br>1: The message handler stored a new message into this object when NEWDAT was set; the CPU has lost a message.<br>This bit is only valid for message objects when the DIR bit in the **CANIFnARB2** register clear (receive). |
| 13 | INTPND | R/W | 0 | Interrupt Pending<br>0: This message object is not the source of an interrupt.<br>1: This message object is the source of an interrupt. The interrupt identifier in the **CANINT** register points to this message object if there is not another interrupt source with a higher priority. |
| 12 | UMASK | R/W | 0 | Use Acceptance Mask<br>0: Mask ignored.<br>1: Use mask (MSK, MXTD, and MDIR bits in the **CANIFnMSKn** registers) for acceptance filtering. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 11 | TXIE | R/W | 0 | Transmit Interrupt Enable<br>0: The INTPND bit in the **CANIFnMCTL** register is unchanged after a successful transmission of a frame.<br>1: The INTPND bit in the **CANIFnMCTL** register is set after a successful transmission of a frame. |
| 10 | RXIE | R/W | 0 | Receive Interrupt Enable<br>0: The INTPND bit in the **CANIFnMCTL** register is unchanged after a successful reception of a frame.<br>1: The INTPND bit in the **CANIFnMCTL** register is set after a successful reception of a frame. |
| 9 | RMTEN | R/W | 0 | Remote Enable<br>0: At the reception of a remote frame, the TXRQST bit in the **CANIFnMCTL** register is left unchanged.<br>1: At the reception of a remote frame, the TXRQST bit in the **CANIFnMCTL** register is set. |
| 8 | TXRQST | R/W | 0 | Transmit Request<br>0: This message object is not waiting for transmission.<br>1: The transmission of this message object is requested and is not yet done. |
| 7 | EOB | R/W | 0 | End of Buffer<br>0: Message object belongs to a FIFO Buffer and is not the last message object of that FIFO Buffer.<br>1: Single message object or last message object of a FIFO Buffer.<br>This bit is used to concatenate two or more message objects (up to 32) to build a FIFO buffer. For a single message object (thus not belonging to a FIFO buffer), this bit must be set. |
| 6:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3:0 | DLC | R/W | 0x0 | Data Length Code<br>Value: 0x0-0x8 — Description: Specifies the number of bytes in the data frame.<br>Value: 0x9-0xF — Description: Defaults to a data frame with 8 bytes.<br>The DLC field in the **CANIFnMCTL** register of a message object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the message handler stores a data frame, it writes DLC to the value given by the received message. |

### Register 22: CAN IF1 Data A1 (CANIF1DA1), offset 0x03C

### Register 23: CAN IF1 Data A2 (CANIF1DA2), offset 0x040

### Register 24: CAN IF1 Data B1 (CANIF1DB1), offset 0x044

### Register 25: CAN IF1 Data B2 (CANIF1DB2), offset 0x048

### Register 26: CAN IF2 Data A1 (CANIF2DA1), offset 0x09C

### Register 27: CAN IF2 Data A2 (CANIF2DA2), offset 0x0A0

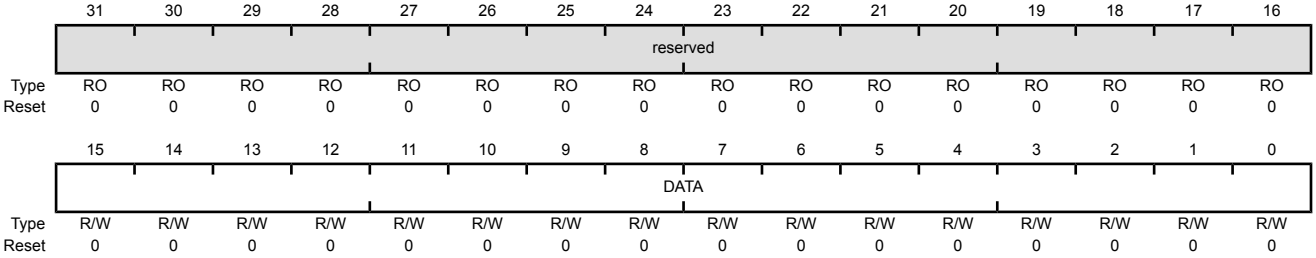
### Register 28: CAN IF2 Data B1 (CANIF2DB1), offset 0x0A4

### Register 29: CAN IF2 Data B2 (CANIF2DB2), offset 0x0A8

These registers contain the data to be sent or that has been received. In a CAN data frame, data byte 0 is the first byte to be transmitted or received and data byte 7 is the last byte to be transmitted or received. In CAN's serial bit stream, the MSB of each byte is transmitted first.

CAN IF1 Data A1 (CANIF1DA1)

CAN0 base: 0x4004.0000
Offset 0x03C
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | DATA | R/W | 0x0000 | Data<br>The **CANIFnDA1** registers contain data bytes 1 and 0; **CANIFnDA2** data bytes 3 and 2; **CANIFnDB1** data bytes 5 and 4; and **CANIFnDB2** data bytes 7 and 6. |

## Register 30: CAN Transmission Request 1 (CANTXRQ1), offset 0x100

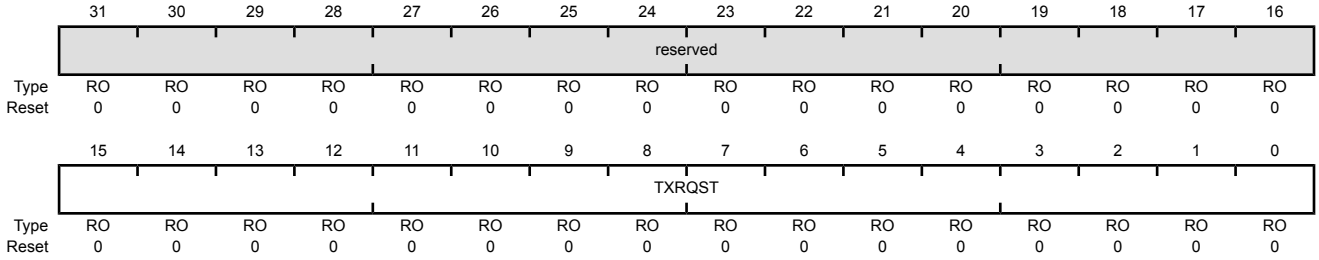
## Register 31: CAN Transmission Request 2 (CANTXRQ2), offset 0x104

The **CANTXRQ1** and **CANTXRQ2** registers hold the TXRQST bits of the 32 message objects. By reading out these bits, the CPU can check which message object has a transmission request pending. The TXRQST bit of a specific message object can be changed by three sources: (1) the CPU via the **CANIFnMCTL** register, (2) the message handler state machine after the reception of a remote frame, or (3) the message handler state machine after a successful transmission.

The **CANTXRQ1** register contains the TXRQST bits of the first 16 message objects in the message RAM; the **CANTXRQ2** register contains the TXRQST bits of the second 16 message objects.

CAN Transmission Request 1 (CANTXRQ1)

CAN0 base: 0x4004.0000
Offset 0x100
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | TXRQST | RO | 0x0000 | Transmission Request Bits<br>0: The corresponding message object is not waiting for transmission.<br>1: The transmission of the corresponding message object is requested and is not yet done. |

## Register 32: CAN New Data 1 (CANNWDA1), offset 0x120

## Register 33: CAN New Data 2 (CANNWDA2), offset 0x124

The **CANNWDA1** and **CANNWDA2** registers hold the NEWDAT bits of the 32 message objects. By reading these bits, the CPU can check which message object has its data portion updated. The NEWDAT bit of a specific message object can be changed by three sources: (1) the CPU via the **CANIFnMCTL** register, (2) the message handler state machine after the reception of a data frame, or (3) the message handler state machine after a successful transmission.

The **CANNWDA1** register contains the NEWDAT bits of the first 16 message objects in the message RAM; the **CANNWDA2** register contains the NEWDAT bits of the second 16 message objects.

CAN New Data 1 (CANNWDA1)

CAN0 base: 0x4004.0000
Offset 0x120
Type RO, reset 0x0000.0000

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NEWDAT | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | NEWDAT | RO | 0x0000 | New Data Bits<br>0: No new data has been written into the data portion of the corresponding message object by the message handler since the last time this flag was cleared by the CPU.<br>1: The message handler or the CPU has written new data into the data portion of the corresponding message object. |

## Register 34: CAN Message 1 Interrupt Pending (CANMSG1INT), offset 0x140

## Register 35: CAN Message 2 Interrupt Pending (CANMSG2INT), offset 0x144

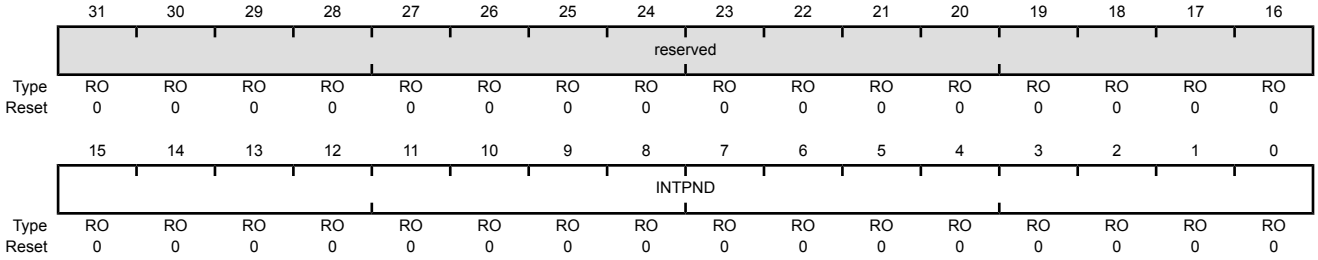
The **CANMSG1INT** and **CANMSG2INT** registers hold the INTPND bits of the 32 message objects. By reading these bits, the CPU can check which message object has an interrupt pending. The INTPND bit of a specific message object can be changed through two sources: (1) the CPU via the **CANIFnMCTL** register, or (2) the message handler state machine after the reception or transmission of a frame.

This field is also encoded in the **CANINT** register.

The **CANMSG1INT** register contains the INTPND bits of the first 16 message objects in the message RAM; the **CANMSG2INT** register contains the INTPND bits of the second 16 message objects.

CAN Message 1 Interrupt Pending (CANMSG1INT)

CAN0 base: 0x4004.0000
Offset 0x140
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | INTPND | RO | 0x0000 | Interrupt Pending Bits<br>0: The corresponding message object is not the source of an interrupt.<br>1: The corresponding message object is the source of an interrupt. |

## Register 36: CAN Message 1 Valid (CANMSG1VAL), offset 0x160

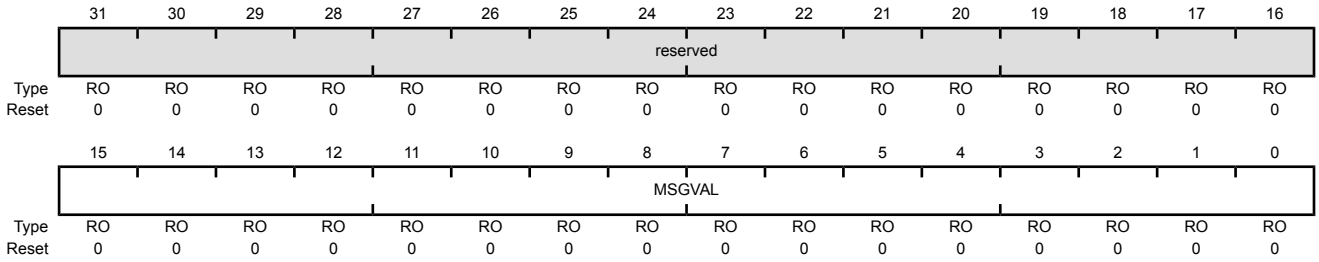
## Register 37: CAN Message 2 Valid (CANMSG2VAL), offset 0x164

The **CANMSG1VAL** and **CANMSG2VAL** registers hold the MSGVAL bits of the 32 message objects. By reading these bits, the CPU can check which message object is valid. The message value of a specific message object can be changed with the **CANIFnMCTL** register.

The **CANMSG1VAL** register contains the MSGVAL bits of the first 16 message objects in the message RAM; the **CANMSG2VAL** register contains the MSGVAL bits of the second 16 message objects in the message RAM.

CAN Message 1 Valid (CANMSG1VAL)

CAN0 base: 0x4004.0000
Offset 0x160
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | MSGVAL | RO | 0x0000 | Message Valid Bits<br>0: The corresponding message object is not configured and is ignored by the message handler.<br>1: The corresponding message object is configured and should be considered by the message handler. |

# 17 Universal Serial Bus (USB) Controller

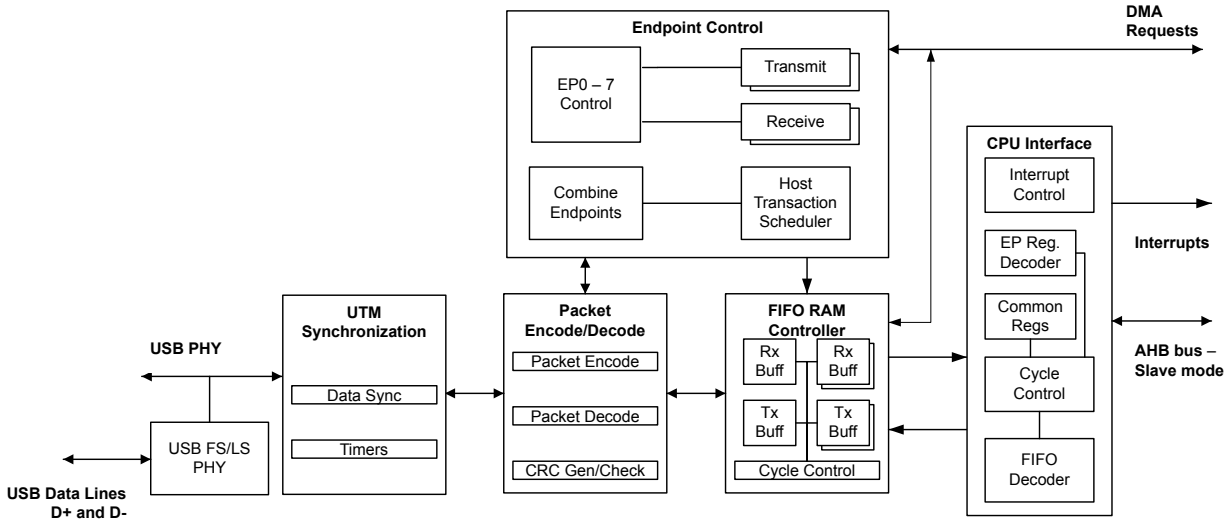
The Stellaris® USB controller operates as a full-speed or low-speed function controller during point-to-point or multipoint (hub) communications with USB functions. The controller complies with the USB 2.0 standard, which includes SUSPEND and RESUME signaling. Eight endpoints including two hard-wired for control transfers (one endpoint for IN and one endpoint for OUT) plus six endpoints defined by firmware along with a dynamic sizable FIFO support multiple packet queueing. µDMA access to the FIFO allows minimal interference from system software. The controller has the capability to access an external power regulator through a power enable pad output (USB0EPEN) and power fault detect pad input (USB0PFLT).

The Stellaris USB module has the following features:

- Standards-based
- USB 2.0 full-speed (12 Mbps) and low-speed (1.5 Mbps) operation
- USB Device or Host mode
- Integrated PHY
- 4 transfer types: Control, Interrupt, Bulk, and Isochronous
- 8 endpoints
  - 1 dedicated control IN endpoint and 1 dedicated control OUT endpoint
  - 3 configurable IN endpoints and 3 configurable OUT endpoints
- 2 KB dedicated endpoint memory
  - Direct memory access (DMA)
  - One endpoint may be defined for double-buffered 1023-byte isochronous packet size

## 17.1 Block Diagram

**Figure 17-1. USB Module Block Diagram**




## 17.2 Signal Description

Some USB controller signals are alternate functions for some GPIO signals and default to be GPIO signals at reset. The column in the table below titled "Pin Assignment" lists the possible GPIO pin placements for these USB signals. The AFSEL bit in the **GPIO Alternate Function Select (GPIOAFSEL)** register (page 371) should be set to choose the USB function. The USB0VBUS and USB0ID signals are configured by clearing the appropriate DEN bit in the **GPIO Digital Enable (GPIODEN)** register. For more information on configuring GPIOs, see “General-Purpose Input/Outputs (GPIOs)” on page 351. The remaining signals (with the word "fixed" in the Pin Assignment column) have a fixed pin assignment and function.

**Table 17-1. USB Signals (64LQFP)**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| USB0DM | 45 | I/O | Analog | Bidirectional differential data pin (D- per USB specification) for USB0. |
| USB0DP | 46 | I/O | Analog | Bidirectional differential data pin (D+ per USB specification) for USB0. |
| USB0EPEN | 14 | O | TTL | Optionally used in Host mode to control an external power source to supply power to the USB bus. |
| USB0PFLT | 15 | I | TTL | Optionally used in Host mode by an external power source to indicate an error state by that power source. |
| USB0RBIAS | 48 | O | Analog | 9.1-kΩ resistor (1% precision) used internally for USB analog circuitry. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 17.3 Functional Description

**Note:** A 9.1-kΩ resistor should be connected between the USB0RBIAS and ground. The 9.1-kΩ resistor should have a 1% tolerance and should be located in close proximity to the

USB0RBIAS pin. Power dissipation in the resistor is low, so a chip resistor of any geometry may be used.

The Stellaris USB controller provides the ability for the controller to choose between Host controller and Device controller functionality. The USB controller requires both A and B connectors in the system to provide Host or Device connectivity. If both connectors are present, the controller provides external signals to enable or disable power to the USB0VBUS pin on the USB connector when not in use. The controller can only be used in Host or Device mode and cannot be used in both modes simultaneously. However, the controller can be manually switched at run time if the system requires both Host and Device functionality.

**Note:** When USB is used in the system, the minimum system frequency is 30 MHz.

### 17.3.1 Operation as a Device

This section describes the Stellaris USB controller's actions when it is being used as a USB Device. Before the USB controller's operating mode is changed from Device to Host or Host to Device, software must reset the USB controller by setting the USB0 bit in the **Software Reset Control 2 (SRCR2)** register (see page 237). IN endpoints, OUT endpoints, entry into and exit from SUSPEND mode, and recognition of Start of Frame (SOF) are all described.

When in Device mode, IN transactions are controlled by an endpoint’s transmit interface and use the transmit endpoint registers for the given endpoint. OUT transactions are handled with an endpoint's receive interface and use the receive endpoint registers for the given endpoint.

When configuring the size of the FIFOs for endpoints, take into account the maximum packet size for an endpoint.

- **Bulk.** Bulk endpoints should be the size of the maximum packet (up to 64 bytes) or twice the maximum packet size if double buffering is used (described further in the following section).
- **Interrupt.** Interrupt endpoints should be the size of the maximum packet (up to 64 bytes) or twice the maximum packet size if double buffering is used.
- **Isochronous.** Isochronous endpoints are more flexible and can be up to 1023 bytes.
- **Control.** It is also possible to specify a separate control endpoint for a USB Device. However, in most cases the USB Device should use the dedicated control endpoint on the USB controller’s endpoint 0.

#### 17.3.1.1 Endpoints

When operating as a Device, the USB controller provides two dedicated control endpoints (IN and OUT) and six configurable endpoints (3 IN and 3 OUT) that can be used for communications with a Host controller. The endpoint number and direction associated with an endpoint is directly related to its register designation. For example, when the Host is transmitting to endpoint 1, all configuration and data is in the endpoint 1 transmit register interface.

Endpoint 0 is a dedicated control endpoint used for all control transactions to endpoint 0 during enumeration or when any other control requests are made to endpoint 0. Endpoint 0 uses the first 64 bytes of the USB controller's FIFO RAM as a shared memory for both IN and OUT transactions.

The remaining six endpoints can be configured as control, bulk, interrupt, or isochronous endpoints. They should be treated as three configurable IN and three configurable OUT endpoints. The endpoint pairs are not required to have the same type for their IN and OUT endpoint configuration. For example, the OUT portion of an endpoint pair could be a bulk endpoint, while the IN portion of that

endpoint pair could be an interrupt endpoint. The address and size of the FIFOs attached to each endpoint can be modified to fit the application's needs.

#### 17.3.1.2 IN Transactions as a Device

When operating as a USB Device, data for IN transactions is handled through the FIFOs attached to the transmit endpoints. The sizes of the FIFOs for the three configurable IN endpoints are determined by the **USB Transmit FIFO Start Address (USBTXFIFOADD)** register. The maximum size of a data packet that may be placed in a transmit endpoint's FIFO for transmission is programmable and is determined by the value written to the **USB Maximum Transmit Data Endpoint n (USBTXMAXPn)** register for that endpoint. The endpoint's FIFO can also be configured to use double-packet or single-packet buffering. When double-packet buffering is enabled, two data packets can be buffered in the FIFO, which also requires that the FIFO is at least two packets in size. When double-packet buffering is disabled, only one packet can be buffered, even if the packet size is less than half the FIFO size.

**Note:** The maximum packet size set for any endpoint must not exceed the FIFO size. The **USBTXMAXPn** register should not be written to while data is in the FIFO as unexpected results may occur.

##### *Single-Packet Buffering*

If the size of the transmit endpoint's FIFO is less than twice the maximum packet size for this endpoint (as set in the **USB Transmit Dynamic FIFO Sizing (USBTXFIFOSZ)** register), only one packet can be buffered in the FIFO and single-packet buffering is required. When each packet is completely loaded into the transmit FIFO, the TXRDY bit in the **USB Transmit Control and Status Endpoint n Low (USBTXCSRLn)** register must be set. If the AUTOSET bit in the **USB Transmit Control and Status Endpoint n High (USBTXCSRHn)** register is set, the TXRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, the TXRDY bit must be set manually. When the TXRDY bit is set, either manually or automatically, the packet is ready to be sent. When the packet has been successfully sent, both TXRDY and FIFONE are cleared, and the appropriate transmit endpoint interrupt signaled. At this point, the next packet can be loaded into the FIFO.

##### *Double-Packet Buffering*

If the size of the transmit endpoint's FIFO is at least twice the maximum packet size for this endpoint, two packets can be buffered in the FIFO and double-packet buffering is allowed. As each packet is loaded into the transmit FIFO, the TXRDY bit in the **USBTXCSRLn** register must be set. If the AUTOSET bit in the **USBTXCSRHn** register is set, the TXRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, TXRDY must be set manually. When the TXRDY bit is set, either manually or automatically, the packet is ready to be sent. After the first packet is loaded, TXRDY is immediately cleared and an interrupt is generated. A second packet can now be loaded into the transmit FIFO and TXRDY set again (either manually or automatically if the packet is the maximum size). At this point, both packets are ready to be sent. After each packet has been successfully sent, TXRDY is automatically cleared and the appropriate transmit endpoint interrupt signaled to indicate that another packet can now be loaded into the transmit FIFO. The state of the FIFONE bit in the **USBTXCSRLn** register at this point indicates how many packets may be loaded. If the FIFONE bit is set, then another packet is in the FIFO and only one more packet can be loaded. If the FIFONE bit is clear, then no packets are in the FIFO and two more packets can be loaded.

**Note:** Double-packet buffering is disabled if an endpoint's corresponding EPn bit is set in the **USB Transmit Double Packet Buffer Disable (USBTXDPKTBUFDIS)** register. This bit is set by default, so it must be cleared to enable double-packet buffering.

#### 17.3.1.3 OUT Transactions as a Device

When in Device mode, OUT transactions are handled through the USB controller receive FIFOs. The sizes of the receive FIFOs for the three configurable OUT endpoints are determined by the **USB Receive FIFO Start Address (USBRXFIFOADD)** register. The maximum amount of data received by an endpoint in any packet is determined by the value written to the **USB Maximum Receive Data Endpoint n (USBRXMAXPn)** register for that endpoint. When double-packet buffering is enabled, two data packets can be buffered in the FIFO. When double-packet buffering is disabled, only one packet can be buffered even if the packet is less than half the FIFO size.

**Note:** In all cases, the maximum packet size must not exceed the FIFO size.

***Single-Packet Buffering***

If the size of the receive endpoint FIFO is less than twice the maximum packet size for an endpoint, only one data packet can be buffered in the FIFO and single-packet buffering is required. When a packet is received and placed in the receive FIFO, the RXRDY and FULL bits in the **USB Receive Control and Status Endpoint n Low (USBRXCSRLn)** register are set and the appropriate receive endpoint is signaled, indicating that a packet can now be unloaded from the FIFO. After the packet has been unloaded, the RXRDY bit must be cleared in order to allow further packets to be received. This action also generates the acknowledge signaling to the Host controller. If the AUTOCL bit in the **USB Receive Control and Status Endpoint n High (USBRXCSRHn)** register is set and a maximum-sized packet is unloaded from the FIFO, the RXRDY and FULL bits are cleared automatically. For packet sizes less than the maximum, RXRDY must be cleared manually.

***Double-Packet Buffering***

If the size of the receive endpoint FIFO is at least twice the maximum packet size for the endpoint, two data packets can be buffered and double-packet buffering can be used. When the first packet is received and loaded into the receive FIFO, the RXRDY bit in the **USBRXCSRLn** register is set and the appropriate receive endpoint interrupt is signaled to indicate that a packet can now be unloaded from the FIFO.

**Note:** The FULL bit in **USBRXCSRLn** is not set when the first packet is received. It is only set if a second packet is received and loaded into the receive FIFO.

After each packet has been unloaded, the RXRDY bit must be cleared to allow further packets to be received. If the AUTOCL bit in the **USBRXCSRHn** register is set and a maximum-sized packet is unloaded from the FIFO, the RXRDY bit is cleared automatically. For packet sizes less than the maximum, RXRDY must be cleared manually. If the FULL bit is set when RXRDY is cleared, the USB controller first clears the FULL bit, then sets RXRDY again to indicate that there is another packet waiting in the FIFO to be unloaded.

**Note:** Double-packet buffering is disabled if an endpoint's corresponding EPn bit is set in the **USB Receive Double Packet Buffer Disable (USBRXDPKTBUFDIS)** register. This bit is set by default, so it must be cleared to enable double-packet buffering.

#### 17.3.1.4 Scheduling

The Device has no control over the scheduling of transactions as scheduling is determined by the Host controller. The Stellaris USB controller can set up a transaction at any time. The USB controller waits for the request from the Host controller and generates an interrupt when the transaction is complete or if it was terminated due to some error. If the Host controller makes a request and the Device controller is not ready, the USB controller sends a busy response (NAK) to all requests until it is ready.

#### 17.3.1.5 Additional Actions

The USB controller responds automatically to certain conditions on the USB bus or actions by the Host controller such as when the USB controller automatically stalls a control transfer or unexpected zero length OUT data packets.

***Stalled Control Transfer***

The USB controller automatically issues a STALL handshake to a control transfer under the following conditions:

1. The Host sends more data during an OUT data phase of a control transfer than was specified in the Device request during the SETUP phase. This condition is detected by the USB controller when the Host sends an OUT token (instead of an IN token) after the last OUT packet has been unloaded and the DATAEND bit in the **USB Control and Status Endpoint 0 Low (USBCSRL0)** register has been set.

2. The Host requests more data during an IN data phase of a control transfer than was specified in the Device request during the SETUP phase. This condition is detected by the USB controller when the Host sends an IN token (instead of an OUT token) after the CPU has cleared TXRDY and set DATAEND in response to the ACK issued by the Host to what should have been the last packet.

3. The Host sends more than **USBRXMAXPn** bytes of data with an OUT data token.

4. The Host sends more than a zero length data packet for the OUT STATUS phase.

***Zero Length OUT Data Packets***

A zero-length OUT data packet is used to indicate the end of a control transfer. In normal operation, such packets should only be received after the entire length of the Device request has been transferred.

However, if the Host sends a zero-length OUT data packet before the entire length of Device request has been transferred, it is signaling the premature end of the transfer. In this case, the USB controller automatically flushes any IN token ready for the data phase from the FIFO and sets the DATAEND bit in the **USBCSRL0** register.

***Setting the Device Address***

When a Host is attempting to enumerate the USB Device, it requests that the Device change its address from zero to some other value. The address is changed by writing the value that the Host requested to the **USB Device Functional Address (USBFADDR)** register. However, care should be taken when writing to **USBFADDR** to avoid changing the address before the transaction is complete. This register should only be set after the SET_ADDRESS command is complete. Like all control transactions, the transaction is only complete after the Device has left the STATUS phase. In the case of a SET_ADDRESS command, the transaction is completed by responding to the IN request from the Host with a zero-byte packet. Once the Device has responded to the IN request, the **USBFADDR** register should be programmed to the new value as soon as possible to avoid missing any new commands sent to the new address.

**Note:** If the **USBFADDR** register is set to the new value as soon as the Device receives the OUT transaction with the SET_ADDRESS command in the packet, it changes the address during the control transfer. In this case, the Device does not receive the IN request that allows the USB transaction to exit the STATUS phase of the control transfer because it is sent to the

old address. As a result, the Host does not get a response to the IN request, and the Host fails to enumerate the Device.

#### 17.3.1.6 Device Mode SUSPEND

When no activity has occurred on the USB bus for 3 ms, the USB controller automatically enters SUSPEND mode. If the SUSPEND interrupt has been enabled in the **USB Interrupt Enable (USBIE)** register, an interrupt is generated at this time. When in SUSPEND mode, the PHY also goes into SUSPEND mode. When RESUME signaling is detected, the USB controller exits SUSPEND mode and takes the PHY out of SUSPEND. If the RESUME interrupt is enabled, an interrupt is generated. The USB controller can also be forced to exit SUSPEND mode by setting the RESUME bit in the **USB Power (USBPOWER)** register. When this bit is set, the USB controller exits SUSPEND mode and drives RESUME signaling onto the bus. The RESUME bit must be cleared after 10 ms (a maximum of 15 ms) to end RESUME signaling.

To meet USB power requirements, the controller can be put into Deep Sleep mode which keeps the controller in a static state. The USB controller is not able to Hibernate because all the internal states are lost as a result.

#### 17.3.1.7 Start-of-Frame

When the USB controller is operating in Device mode, it receives a Start-Of-Frame (SOF) packet from the Host once every millisecond. When the SOF packet is received, the 11-bit frame number contained in the packet is written into the **USB Frame Value (USBFRAME)** register, and an SOF interrupt is also signaled and can be handled by the application. Once the USB controller has started to receive SOF packets, it expects one every millisecond. If no SOF packet is received after 1.00358 ms, the packet is assumed to have been lost, and the **USBFRAME** register is not updated. The USB controller continues and resynchronizes these pulses to the received SOF packets when these packets are successfully received again.

#### 17.3.1.8 USB RESET

When the USB controller is in Device mode and a RESET condition is detected on the USB bus, the USB controller automatically performs the following actions:

- Clears the **USBFADDR** register.
- Clears the **USB Endpoint Index (USBEPIDX)** register.
- Flushes all endpoint FIFOs.
- Clears all control/status registers.
- Enables all endpoint interrupts.
- Generates a RESET interrupt.

When the application software driving the USB controller receives a RESET interrupt, any open pipes are closed and the USB controller waits for bus enumeration to begin.

#### 17.3.1.9 Connect/Disconnect

The USB controller connection to the USB bus is handled by software. The USB PHY can be switched between normal mode and non-driving mode by setting or clearing the SOFTCONN bit of the **USBPOWER** register. When the SOFTCONN bit is set, the PHY is placed in its normal mode,

and the USB0DP/USB0DM lines of the USB bus are enabled. At the same time, the USB controller is placed into a state, in which it does not respond to any USB signaling except a USB RESET.

When the SOFTCONN bit is cleared, the PHY is put into non-driving mode, USB0DP and USB0DM are tristated, and the USB controller appears to other devices on the USB bus as if it has been disconnected. The non-driving mode is the default so the USB controller appears disconnected until the SOFTCONN bit has been set. The application software can then choose when to set the PHY into its normal mode. Systems with a lengthy initialization procedure may use this to ensure that initialization is complete, and the system is ready to perform enumeration before connecting to the USB bus. Once the SOFTCONN bit has been set, the USB controller can be disconnected by clearing this bit.

When the USB controller is acting as a self-powered Device, a GPIO input or analog comparator input must be connected to VBUS and configured to generate an interrupt when the VBUS level drops. This interrupt is used to disable the pullup resistor on the USB0DP signal.

**Note:** The USB controller does not generate an interrupt when the Device is connected to the Host. However, an interrupt is generated when the Host terminates a session.

### 17.3.2 Operation as a Host

When the Stellaris USB controller is operating in Host mode, it can either be used for point-to-point communications with another USB device or, when attached to a hub, for communication with multiple devices. Before the USB controller's operating mode is changed from Host to Device or Device to Host, software must reset the USB controller by setting the USB0 bit in the **Software Reset Control 2 (SRCR2)** register (see page 237). Full-speed and low-speed USB devices are supported, both for point-to-point communication and for operation through a hub. The USB controller automatically carries out the necessary transaction translation needed to allow a low-speed or full-speed device to be used with a USB 2.0 hub. Control, bulk, isochronous, and interrupt transactions are supported. This section describes the USB controller's actions when it is being used as a USB Host. Configuration of IN endpoints, OUT endpoints, entry into and exit from SUSPEND mode, and RESET are all described.

When in Host mode, IN transactions are controlled by an endpoint’s receive interface. All IN transactions use the receive endpoint registers and all OUT endpoints use the transmit endpoint registers for a given endpoint. As in Device mode, the FIFOs for endpoints should take into account the maximum packet size for an endpoint.

- **Bulk.** Bulk endpoints should be the size of the maximum packet (up to 64 bytes) or twice the maximum packet size if double buffering is used (described further in the following section).
- **Interrupt.** Interrupt endpoints should be the size of the maximum packet (up to 64 bytes) or twice the maximum packet size if double buffering is used.
- **Isochronous.** Isochronous endpoints are more flexible and can be up to 1023 bytes.
- **Control.** It is also possible to specify a separate control endpoint to communicate with a Device. However, in most cases the USB controller should use the dedicated control endpoint to communicate with a Device’s endpoint 0.

#### 17.3.2.1 Endpoints

The endpoint registers are used to control the USB endpoint interfaces which communicate with Device(s) that are connected. The endpoints consist of a dedicated control IN endpoint, a dedicated control OUT endpoint, three configurable OUT endpoints, and three configurable IN endpoints.

The dedicated control interface can only be used for control transactions to endpoint 0 of Devices. These control transactions are used during enumeration or other control functions that communicate using endpoint 0 of Devices. This control endpoint shares the first 64 bytes of the USB controller's FIFO RAM for IN and OUT transactions. The remaining IN and OUT interfaces can be configured to communicate with control, bulk, interrupt, or isochronous Device endpoints.

These USB interfaces can be used to simultaneously schedule as many as three independent OUT and three independent IN transactions to any endpoints on any Device. The IN and OUT controls are paired in three sets of registers. However, they can be configured to communicate with different types of endpoints and different endpoints on Devices. For example, the first pair of endpoint controls can be split so that the OUT portion is communicating with a Device's bulk OUT endpoint 1, while the IN portion is communicating with a Device's interrupt IN endpoint 2.

Before accessing any Device, whether for point-to-point communications or for communications via a hub, the relevant **USB Receive Functional Address Endpoint n (USBRXFUNCADDRn)** or **USB Transmit Functional Address Endpoint n (USBTXFUNCADDRn)** registers must be set for each receive or transmit endpoint to record the address of the Device being accessed.

The USB controller also supports connections to Devices through a USB hub by providing a register that specifies the hub address and port of each USB transfer. The FIFO address and size are customizable and can be specified for each USB IN and OUT transfer. Customization includes allowing one FIFO per transaction, sharing a FIFO across transactions, and allowing for double-buffered FIFOs.

#### 17.3.2.2 IN Transactions as a Host

IN transactions are handled in a similar manner to the way in which OUT transactions are handled when the USB controller is in Device mode except that the transaction first must be initiated by setting the REQPKT bit in the **USBCSRL0** register, indicating to the transaction scheduler that there is an active transaction on this endpoint. The transaction scheduler then sends an IN token to the target Device. When the packet is received and placed in the receive FIFO, the RXRDY bit in the **USBCSRL0** register is set, and the appropriate receive endpoint interrupt is signaled to indicate that a packet can now be unloaded from the FIFO.

When the packet has been unloaded, RXRDY must be cleared. The AUTOCL bit in the **USBRXCSRHn** register can be used to have RXRDY automatically cleared when a maximum-sized packet has been unloaded from the FIFO. The AUTORQ bit in **USBRXCSRHn** causes the REQPKT bit to be automatically set when the RXRDY bit is cleared. The AUTOCL and AUTORQ bits can be used with µDMA accesses to perform complete bulk transfers without main processor intervention. When the RXRDY bit is cleared, the controller sends an acknowledge to the Device. When there is a known number of packets to be transferred, the **USB Request Packet Count in Block Transfer Endpoint n (USBRQPKTCOUNTn)** register associated with the endpoint should be configured to the number of packets to be transferred. The USB controller decrements the value in the **USBRQPKTCOUNTn** register following each request. When the **USBRQPKTCOUNTn** value decrements to 0, the AUTORQ bit is cleared to prevent any further transactions being attempted. For cases where the size of the transfer is unknown, **USBRQPKTCOUNTn** should be cleared. AUTORQ then remains set until cleared by the reception of a short packet (that is, less than the MAXLOAD value in the **USBRXMAXPn** register) such as may occur at the end of a bulk transfer.

If the Device responds to a bulk or interrupt IN token with a NAK, the USB Host controller keeps retrying the transaction until any NAK Limit that has been set has been reached. If the target Device responds with a STALL, however, the USB Host controller does not retry the transaction but sets the STALLED bit in the **USBCSRL0** register. If the target Device does not respond to the IN token within the required time, or the packet contained a CRC or bit-stuff error, the USB Host controller

retries the transaction. If after three attempts the target Device has still not responded, the USB Host controller clears the REQPKT bit and sets the ERROR bit in the **USBCSRL0** register.

#### 17.3.2.3 OUT Transactions as a Host

OUT transactions are handled in a similar manner to the way in which IN transactions are handled when the USB controller is in Device mode. The TXRDY bit in the **USBTXCSRLn** register must be set as each packet is loaded into the transmit FIFO. Again, setting the AUTOSET bit in the **USBTXCSRHn** register automatically sets TXRDY when a maximum-sized packet has been loaded into the FIFO. Furthermore, AUTOSET can be used with the µDMA controller to perform complete bulk transfers without software intervention.

If the target Device responds to the OUT token with a NAK, the USB Host controller keeps retrying the transaction until the NAK Limit that has been set has been reached. However, if the target Device responds with a STALL, the USB controller does not retry the transaction but interrupts the main processor by setting the STALLED bit in the **USBTXCSRLn** register. If the target Device does not respond to the OUT token within the required time, or the packet contained a CRC or bit-stuff error, the USB Host controller retries the transaction. If after three attempts the target Device has still not responded, the USB controller flushes the FIFO and sets the ERROR bit in the **USBTXCSRLn** register.

#### 17.3.2.4 Transaction Scheduling

Scheduling of transactions is handled automatically by the USB Host controller. The Host controller allows configuration of the endpoint communication scheduling based on the type of endpoint transaction. Interrupt transactions can be scheduled to occur in the range of every frame to every 255 frames in 1 frame increments. Bulk endpoints do not allow scheduling parameters, but do allow for a NAK timeout in the event an endpoint on a Device is not responding. Isochronous endpoints can be scheduled from every frame to every $2^{16}$ frames, in powers of 2.

The USB controller maintains a frame counter. If the target Device is a full-speed device, the USB controller automatically sends an SOF packet at the start of each frame and increments the frame counter. If the target Device is a low-speed device, a *K* state is transmitted on the bus to act as a *keep-alive* to stop the low-speed device from going into SUSPEND mode.

After the SOF packet has been transmitted, the USB Host controller cycles through all the configured endpoints looking for active transactions. An active transaction is defined as a receive endpoint for which the REQPKT bit is set or a transmit endpoint for which the TXRDY bit and/or the FIFONE bit is set.

An isochronous or interrupt transaction is started if the transaction is found on the first scheduler cycle of a frame and if the interval counter for that endpoint has counted down to zero. As a result, only one interrupt or isochronous transaction occurs per endpoint every n frames, where n is the interval set via the **USB Host Transmit Interval Endpoint n (USBTXINTERVALn)** or **USB Host Receive Interval Endpoint n (USBRXINTERVALn)** register for that endpoint.

An active bulk transaction starts immediately, provided sufficient time is left in the frame to complete the transaction before the next SOF packet is due. If the transaction must be retried (for example, because a NAK was received or the target Device did not respond), then the transaction is not retried until the transaction scheduler has first checked all the other endpoints for active transactions. This process ensures that an endpoint that is sending a lot of NAKs does not block other transactions on the bus. The controller also allows the user to specify a limit to the length of time for NAKs to be received from a target Device before the endpoint times out.

#### 17.3.2.5 USB Hubs

The following setup requirements apply to the USB Host controller only if it is used with a USB hub. When a full- or low-speed Device is connected to the USB controller via a USB 2.0 hub, details of the hub address and the hub port also must be recorded in the corresponding **USB Receive Hub Address Endpoint n (USBRXHUBADDRn)** and **USB Receive Hub Port Endpoint n (USBRXHUBPORTn)** or the **USB Transmit Hub Address Endpoint n (USBTXHUBADDRn)** and **USB Transmit Hub Port Endpoint n (USBTXHUBPORTn)** registers. In addition, the speed at which the Device operates (full or low) must be recorded in the **USB Type Endpoint 0 (USBTYPE0)** (endpoint 0), **USB Host Configure Transmit Type Endpoint n (USBTXTYPEn)**, or **USB Host Configure Receive Type Endpoint n (USBRXTYPEn)** registers for each endpoint that is accessed by the Device.

For hub communications, the settings in these registers record the current allocation of the endpoints to the attached USB Devices. To maximize the number of Devices supported, the USB Host controller allows this allocation to be changed dynamically by simply updating the address and speed information recorded in these registers. Any changes in the allocation of endpoints to Device functions must be made following the completion of any on-going transactions on the endpoints affected.

#### 17.3.2.6 Babble

The USB Host controller does not start a transaction until the bus has been inactive for at least the minimum inter-packet delay. The controller also does not start a transaction unless it can be finished before the end of the frame. If the bus is still active at the end of a frame, then the USB Host controller assumes that the target Device to which it is connected has malfunctioned, and the USB controller suspends all transactions and generates a babble interrupt.

#### 17.3.2.7 Host SUSPEND

If the SUSPEND bit in the **USBPOWER** register is set, the USB Host controller completes the current transaction then stops the transaction scheduler and frame counter. No further transactions are started and no SOF packets are generated.

To exit SUSPEND mode, set the RESUME bit and clear the SUSPEND bit. While the RESUME bit is set, the USB Host controller generates RESUME signaling on the bus. After 20 ms, the RESUME bit must be cleared, at which point the frame counter and transaction scheduler start. The Host supports the detection of a remote wake-up.

#### 17.3.2.8 USB RESET

If the RESET bit in the **USBPOWER** register is set, the USB Host controller generates USB RESET signaling on the bus. The RESET bit must be set for at least 20 ms to ensure correct resetting of the target Device. After the CPU has cleared the bit, the USB Host controller starts its frame counter and transaction scheduler.

#### 17.3.2.9 Connect/Disconnect

A session is started by setting the SESSION bit in the **USB Device Control (USBDEVCTL)** register, enabling the USB controller to wait for a Device to be connected. When a Device is detected, a connect interrupt is generated. The speed of the Device that has been connected can be determined by reading the **USBDEVCTL** register where the FSDEV bit is set for a full-speed Device, and the LSDEV bit is set for a low-speed Device. The USB controller must generate a RESET to the Device, and then the USB Host controller can begin Device enumeration. If the Device is disconnected while a session is in progress, a disconnect interrupt is generated.

### 17.3.3 DMA Operation

The USB peripheral provides an interface connected to the µDMA controller. The µDMA operation of the USB is enabled through the **USBTXCSRHn** and **USBRXCSRHn** registers, for the TX and RX channels respectively. When µDMA operation is enabled, the USB asserts a µDMA request on the enabled receive or transmit channel when the associated FIFO can transfer data. When either FIFO can transfer data, the burst request for that channel is asserted. The µDMA channel must be configured to operate in Basic mode, and the size of the µDMA transfer must be restricted to whole multiples of the size of the USB FIFO. Both read and write transfers of the USB FIFOs using µDMA must be configured in this manner. For example, if the USB endpoint is configured with a FIFO size of 64 bytes, the µDMA channel can be used to transfer 64 bytes to or from the endpoint FIFO. If the number of bytes to transfer is less than 64, then a programmed I/O method must be used to copy the data to or from the FIFO.

If the DMAMOD bit in the **USBTXCSRHn**/**USBRXCSRHn** register is clear, an interrupt is generated after every packet is transferred, but the µDMA continues transferring data. If the DMAMOD bit is set, an interrupt is generated only when the entire µDMA transfer is complete. The interrupt occurs on the USB interrupt vector. Therefore, if interrupts are used for USB operation and the µDMA is enabled, the USB interrupt handler must be designed to handle the µDMA completion interrupt.

Care must be taken when using the µDMA to unload the receive FIFO as data is read from the receive FIFO in 4 byte chunks regardless of the value off the MAXLOAD field in the **USBRXCSRHn** register. The RXRDY bit is cleared as follows.

**Table 17-2. Remainder (MAXLOAD/4)**

| Value | Description |
|---|---|
| 0 | MAXLOAD = 64 bytes |
| 1 | MAXLOAD = 61 bytes |
| 2 | MAXLOAD = 62 bytes |
| 3 | MAXLOAD = 63 bytes |

**Table 17-3. Actual Bytes Read**

| Value | Description |
|---|---|
| 0 | MAXLOAD |
| 1 | MAXLOAD+3 |
| 2 | MAXLOAD+2 |
| 3 | MAXLOAD+1 |

**Table 17-4. Packet Sizes That Clear RXRDY**

| Value | Description |
|---|---|
| 0 | MAXLOAD, MAXLOAD-1, MAXLOAD-2, MAXLOAD-3 |
| 1 | MAXLOAD |
| 2 | MAXLOAD, MAXLOAD-1 |
| 3 | MAXLOAD, MAXLOAD-1, MAXLOAD-2 |

To enable DMA operation for the endpoint receive channel, the DMAEN bit of the **USBRXCSRHn** register should be set. To enable DMA operation for the endpoint transmit channel, the DMAEN bit of the **USBTXCSRHn** register must be set.

See “Micro Direct Memory Access (µDMA)” on page 290 for more details about programming the µDMA controller.

## 17.4 Initialization and Configuration

To use the USB Controller, the peripheral clock must be enabled via the **RCGC2** register (see page 229). In addition, the clock to the appropriate GPIO module must be enabled via the **RCGC2** register in the System Control module (see page 229). To find out which GPIO port to enable, refer to Table 19-3 on page 760.

The initial configuration in all cases requires that the processor enable the USB controller and USB controller’s physical layer interface (PHY) before setting any registers. The next step is to enable the USB PLL so that the correct clocking is provided to the PHY. To ensure that voltage is not supplied to the bus incorrectly, the external power control signal, USB0EPEN, should be negated on start up by configuring the USB0EPEN and USB0PFLT pins to be controlled by the USB controller and not exhibit their default GPIO behavior.

The USB controller provides a method to set the current operating mode of the USB controller. This register should be written with the desired default mode so that the controller can respond to external USB events.

### 17.4.1 Pin Configuration

When using the Device controller portion of the USB controller in a system that also provides Host functionality, the power to VBUS must be disabled to allow the external Host controller to supply power. Usually, the USB0EPEN signal is used to control the external regulator and should be negated to avoid having two devices driving the USB0VBUS power pin on the USB connector.

When the USB controller is acting as a Host, it is in control of two signals that are attached to an external voltage supply that provides power to VBUS. The Host controller uses the USB0EPEN signal to enable or disable power to the USB0VBUS pin on the USB connector. An input pin, USB0PFLT, provides feedback when there has been a power fault on VBUS. The USB0PFLT signal can be configured to either automatically negate the USB0EPEN signal to disable power, and/or it can generate an interrupt to the interrupt controller to allow software to handle the power fault condition. The polarity and actions related to both USB0EPEN and USB0PFLT are fully configurable in the USB controller. The controller also provides interrupts on Device insertion and removal to allow the Host controller code to respond to these external events.

### 17.4.2 Endpoint Configuration

To start communication in Host or Device mode, the endpoint registers must first be configured. In Host mode, this configuration establishes a connection between an endpoint register and an endpoint on a Device. In Device mode, an endpoint must be configured before enumerating to the Host controller.

In both cases, the endpoint 0 configuration is limited because it is a fixed-function, fixed-FIFO-size endpoint. In Device and Host modes, the endpoint requires little setup but does require a software-based state machine to progress through the setup, data, and status phases of a standard control transaction. In Device mode, the configuration of the remaining endpoints is done once before enumerating and then only changed if an alternate configuration is selected by the Host controller. In Host mode, the endpoints must be configured to operate as control, bulk, interrupt or isochronous mode. Once the type of endpoint is configured, a FIFO area must be assigned to each endpoint. In the case of bulk, control and interrupt endpoints, each has a maximum of 64 bytes per transaction. Isochronous endpoints can have packets with up to 1023 bytes per packet. In either

mode, the maximum packet size for the given endpoint must be set prior to sending or receiving data.

Configuring each endpoint's FIFO involves reserving a portion of the overall USB FIFO RAM to each endpoint. The total FIFO RAM available is 4 Kbytes with the first 64 bytes reserved for endpoint 0. The endpoint's FIFO must be at least as large as the maximum packet size. The FIFO can also be configured as a double-buffered FIFO so that interrupts occur at the end of each packet and allow filling the other half of the FIFO.

If operating as a Device, the USB Device controller's soft connect must be enabled when the Device is ready to start communications, indicating to the Host controller that the Device is ready to start the enumeration process. If operating as a Host controller, the Device soft connect must be disabled and power must be provided to VBUS via the USB0EPEN signal.

## 17.5 Register Map

Table 17-5 on page 672 lists the registers. All addresses given are relative to the USB base address of 0x4005.0000. Note that the USB controller clock must be enabled before the registers can be programmed (see page 229). There must be a delay of 3 system clocks after the USB module clock is enabled before any USB module registers are accessed.

**Table 17-5. Universal Serial Bus (USB) Controller Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | USBFADDR | R/W | 0x00 | USB Device Functional Address | 676 |
| 0x001 | USBPOWER | R/W | 0x20 | USB Power | 677 |
| 0x002 | USBTXIS | RO | 0x0000 | USB Transmit Interrupt Status | 680 |
| 0x004 | USBRXIS | RO | 0x0000 | USB Receive Interrupt Status | 681 |
| 0x006 | USBTXIE | R/W | 0x000F | USB Transmit Interrupt Enable | 682 |
| 0x008 | USBRXIE | R/W | 0x000E | USB Receive Interrupt Enable | 683 |
| 0x00A | USBIS | RO | 0x00 | USB General Interrupt Status | 684 |
| 0x00B | USBIE | R/W | 0x06 | USB Interrupt Enable | 687 |
| 0x00C | USBFRAME | RO | 0x0000 | USB Frame Value | 690 |
| 0x00E | USBEPIDX | R/W | 0x00 | USB Endpoint Index | 691 |
| 0x00F | USBTEST | R/W | 0x00 | USB Test Mode | 692 |
| 0x020 | USBFIFO0 | R/W | 0x0000.0000 | USB FIFO Endpoint 0 | 694 |
| 0x024 | USBFIFO1 | R/W | 0x0000.0000 | USB FIFO Endpoint 1 | 694 |
| 0x028 | USBFIFO2 | R/W | 0x0000.0000 | USB FIFO Endpoint 2 | 694 |
| 0x02C | USBFIFO3 | R/W | 0x0000.0000 | USB FIFO Endpoint 3 | 694 |
| 0x060 | USBDEVCTL | RO | 0x80 | USB Device Control | 695 |
| 0x062 | USBTXFIFOSZ | R/W | 0x00 | USB Transmit Dynamic FIFO Sizing | 696 |
| 0x063 | USBRXFIFOSZ | R/W | 0x00 | USB Receive Dynamic FIFO Sizing | 696 |
| 0x064 | USBTXFIFOADD | R/W | 0x0000 | USB Transmit FIFO Start Address | 697 |

**Table 17-5. Universal Serial Bus (USB) Controller Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x066 | USBRXFIFOADD | R/W | 0x0000 | USB Receive FIFO Start Address | 697 |
| 0x07A | USBCONTIM | R/W | 0x5C | USB Connect Timing | 698 |
| 0x07D | USBFSEOF | R/W | 0x77 | USB Full-Speed Last Transaction to End of Frame Timing | 699 |
| 0x07E | USBLSEOF | R/W | 0x72 | USB Low-Speed Last Transaction to End of Frame Timing | 700 |
| 0x080 | USBTXFUNCADDR0 | R/W | 0x00 | USB Transmit Functional Address Endpoint 0 | 701 |
| 0x082 | USBTXHUBADDR0 | R/W | 0x00 | USB Transmit Hub Address Endpoint 0 | 702 |
| 0x083 | USBTXHUBPORT0 | R/W | 0x00 | USB Transmit Hub Port Endpoint 0 | 703 |
| 0x088 | USBTXFUNCADDR1 | R/W | 0x00 | USB Transmit Functional Address Endpoint 1 | 701 |
| 0x08A | USBTXHUBADDR1 | R/W | 0x00 | USB Transmit Hub Address Endpoint 1 | 702 |
| 0x08B | USBTXHUBPORT1 | R/W | 0x00 | USB Transmit Hub Port Endpoint 1 | 703 |
| 0x08C | USBRXFUNCADDR1 | R/W | 0x00 | USB Receive Functional Address Endpoint 1 | 704 |
| 0x08E | USBRXHUBADDR1 | R/W | 0x00 | USB Receive Hub Address Endpoint 1 | 705 |
| 0x08F | USBRXHUBPORT1 | R/W | 0x00 | USB Receive Hub Port Endpoint 1 | 706 |
| 0x090 | USBTXFUNCADDR2 | R/W | 0x00 | USB Transmit Functional Address Endpoint 2 | 701 |
| 0x092 | USBTXHUBADDR2 | R/W | 0x00 | USB Transmit Hub Address Endpoint 2 | 702 |
| 0x093 | USBTXHUBPORT2 | R/W | 0x00 | USB Transmit Hub Port Endpoint 2 | 703 |
| 0x094 | USBRXFUNCADDR2 | R/W | 0x00 | USB Receive Functional Address Endpoint 2 | 704 |
| 0x096 | USBRXHUBADDR2 | R/W | 0x00 | USB Receive Hub Address Endpoint 2 | 705 |
| 0x097 | USBRXHUBPORT2 | R/W | 0x00 | USB Receive Hub Port Endpoint 2 | 706 |
| 0x098 | USBTXFUNCADDR3 | R/W | 0x00 | USB Transmit Functional Address Endpoint 3 | 701 |
| 0x09A | USBTXHUBADDR3 | R/W | 0x00 | USB Transmit Hub Address Endpoint 3 | 702 |
| 0x09B | USBTXHUBPORT3 | R/W | 0x00 | USB Transmit Hub Port Endpoint 3 | 703 |
| 0x09C | USBRXFUNCADDR3 | R/W | 0x00 | USB Receive Functional Address Endpoint 3 | 704 |
| 0x09E | USBRXHUBADDR3 | R/W | 0x00 | USB Receive Hub Address Endpoint 3 | 705 |
| 0x09F | USBRXHUBPORT3 | R/W | 0x00 | USB Receive Hub Port Endpoint 3 | 706 |
| 0x102 | USBCSRL0 | R/W | 0x00 | USB Control and Status Endpoint 0 Low | 708 |
| 0x103 | USBCSRH0 | R/W | 0x00 | USB Control and Status Endpoint 0 High | 712 |
| 0x108 | USBCOUNT0 | RO | 0x00 | USB Receive Byte Count Endpoint 0 | 714 |
| 0x10A | USBTYPE0 | R/W | 0x00 | USB Type Endpoint 0 | 715 |
| 0x10B | USBNAKLMT | R/W | 0x00 | USB NAK Limit | 716 |
| 0x110 | USBTXMAXP1 | R/W | 0x0000 | USB Maximum Transmit Data Endpoint 1 | 707 |

**Table 17-5. Universal Serial Bus (USB) Controller Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x112 | USBTXCSRL1 | R/W | 0x00 | USB Transmit Control and Status Endpoint 1 Low | 717 |
| 0x113 | USBTXCSRH1 | R/W | 0x00 | USB Transmit Control and Status Endpoint 1 High | 721 |
| 0x114 | USBRXMAXP1 | R/W | 0x0000 | USB Maximum Receive Data Endpoint 1 | 725 |
| 0x116 | USBRXCSRL1 | R/W | 0x00 | USB Receive Control and Status Endpoint 1 Low | 726 |
| 0x117 | USBRXCSRH1 | R/W | 0x00 | USB Receive Control and Status Endpoint 1 High | 731 |
| 0x118 | USBRXCOUNT1 | RO | 0x0000 | USB Receive Byte Count Endpoint 1 | 735 |
| 0x11A | USBTXTYPE1 | R/W | 0x00 | USB Host Transmit Configure Type Endpoint 1 | 736 |
| 0x11B | USBTXINTERVAL1 | R/W | 0x00 | USB Host Transmit Interval Endpoint 1 | 737 |
| 0x11C | USBRXTYPE1 | R/W | 0x00 | USB Host Configure Receive Type Endpoint 1 | 738 |
| 0x11D | USBRXINTERVAL1 | R/W | 0x00 | USB Host Receive Polling Interval Endpoint 1 | 739 |
| 0x120 | USBTXMAXP2 | R/W | 0x0000 | USB Maximum Transmit Data Endpoint 2 | 707 |
| 0x122 | USBTXCSRL2 | R/W | 0x00 | USB Transmit Control and Status Endpoint 2 Low | 717 |
| 0x123 | USBTXCSRH2 | R/W | 0x00 | USB Transmit Control and Status Endpoint 2 High | 721 |
| 0x124 | USBRXMAXP2 | R/W | 0x0000 | USB Maximum Receive Data Endpoint 2 | 725 |
| 0x126 | USBRXCSRL2 | R/W | 0x00 | USB Receive Control and Status Endpoint 2 Low | 726 |
| 0x127 | USBRXCSRH2 | R/W | 0x00 | USB Receive Control and Status Endpoint 2 High | 731 |
| 0x128 | USBRXCOUNT2 | RO | 0x0000 | USB Receive Byte Count Endpoint 2 | 735 |
| 0x12A | USBTXTYPE2 | R/W | 0x00 | USB Host Transmit Configure Type Endpoint 2 | 736 |
| 0x12B | USBTXINTERVAL2 | R/W | 0x00 | USB Host Transmit Interval Endpoint 2 | 737 |
| 0x12C | USBRXTYPE2 | R/W | 0x00 | USB Host Configure Receive Type Endpoint 2 | 738 |
| 0x12D | USBRXINTERVAL2 | R/W | 0x00 | USB Host Receive Polling Interval Endpoint 2 | 739 |
| 0x130 | USBTXMAXP3 | R/W | 0x0000 | USB Maximum Transmit Data Endpoint 3 | 707 |
| 0x132 | USBTXCSRL3 | R/W | 0x00 | USB Transmit Control and Status Endpoint 3 Low | 717 |
| 0x133 | USBTXCSRH3 | R/W | 0x00 | USB Transmit Control and Status Endpoint 3 High | 721 |
| 0x134 | USBRXMAXP3 | R/W | 0x0000 | USB Maximum Receive Data Endpoint 3 | 725 |
| 0x136 | USBRXCSRL3 | R/W | 0x00 | USB Receive Control and Status Endpoint 3 Low | 726 |
| 0x137 | USBRXCSRH3 | R/W | 0x00 | USB Receive Control and Status Endpoint 3 High | 731 |
| 0x138 | USBRXCOUNT3 | RO | 0x0000 | USB Receive Byte Count Endpoint 3 | 735 |
| 0x13A | USBTXTYPE3 | R/W | 0x00 | USB Host Transmit Configure Type Endpoint 3 | 736 |
| 0x13B | USBTXINTERVAL3 | R/W | 0x00 | USB Host Transmit Interval Endpoint 3 | 737 |
| 0x13C | USBRXTYPE3 | R/W | 0x00 | USB Host Configure Receive Type Endpoint 3 | 738 |
| 0x13D | USBRXINTERVAL3 | R/W | 0x00 | USB Host Receive Polling Interval Endpoint 3 | 739 |

**Table 17-5. Universal Serial Bus (USB) Controller Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x304 | USBRQPKTCOUNT1 | R/W | 0x0000 | USB Request Packet Count in Block Transfer Endpoint 1 | 740 |
| 0x308 | USBRQPKTCOUNT2 | R/W | 0x0000 | USB Request Packet Count in Block Transfer Endpoint 2 | 740 |
| 0x30C | USBRQPKTCOUNT3 | R/W | 0x0000 | USB Request Packet Count in Block Transfer Endpoint 3 | 740 |
| 0x340 | USBRXDPKTBUFDIS | R/W | 0x0000 | USB Receive Double Packet Buffer Disable | 741 |
| 0x342 | USBTXDPKTBUFDIS | R/W | 0x0000 | USB Transmit Double Packet Buffer Disable | 742 |
| 0x400 | USBEPC | R/W | 0x0000.0000 | USB External Power Control | 743 |
| 0x404 | USBEPCRIS | RO | 0x0000.0000 | USB External Power Control Raw Interrupt Status | 746 |
| 0x408 | USBEPCIM | R/W | 0x0000.0000 | USB External Power Control Interrupt Mask | 747 |
| 0x40C | USBEPCISC | R/W1C | 0x0000.0000 | USB External Power Control Interrupt Status and Clear | 748 |
| 0x410 | USBDRRIS | RO | 0x0000.0000 | USB Device RESUME Raw Interrupt Status | 749 |
| 0x414 | USBDRIM | R/W | 0x0000.0000 | USB Device RESUME Interrupt Mask | 750 |
| 0x418 | USBDRISC | R/W1C | 0x0000.0000 | USB Device RESUME Interrupt Status and Clear | 751 |
| 0x41C | USBGPCS | R/W | 0x0000.0000 | USB General-Purpose Control and Status | 752 |

## 17.6 Register Descriptions

The LM3S5632 USB controller has Host-or-Device capabilities as specified in the USB0 bit field in the **DC6** register (see page 214).

**Device** This icon indicates that the register is used in Device mode. Some registers are used for both Host and Device mode and may have different bit definitions depending on the mode.

**Host** This icon indicates that the register is used in Host mode. Some registers are used for both Host and Device mode and may have different bit definitions depending on the mode. The USB controller is in Device mode upon reset, so the reset values shown for these registers apply to the Device mode definition.

## Register 1: USB Device Functional Address (USBFADDR), offset 0x000

**Device**

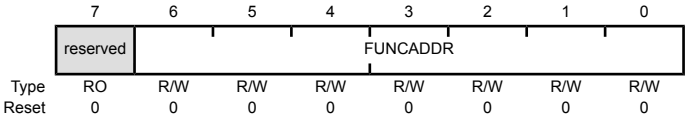
**USBFADDR** is an 8-bit register that contains the 7-bit address of the Device part of the transaction.

When the USB controller is being used in Device mode (the HOST bit in the **USBDEVCTL** register is clear), this register must be written with the address received through a SET_ADDRESS command, which is then used for decoding the function address in subsequent token packets.

**Important:** See the section called “Setting the Device Address” on page 664 for special considerations when writing this register.

USB Device Functional Address (USBFADDR)

Base 0x4005.0000
Offset 0x000
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | FUNCADDR | R/W | 0x00 | Function Address<br>Function Address of Device as received through SET_ADDRESS. |

## Register 2: USB Power (USBPOWER), offset 0x001

**Host**

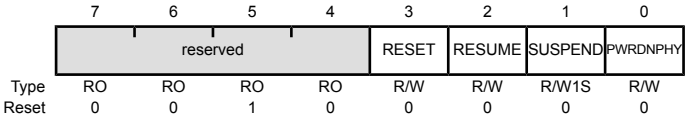
**USBPOWER** is an 8-bit register used for controlling SUSPEND and RESUME signaling and some basic operational aspects of the USB controller.

**Device**

### Host Mode

USB Power (USBPOWER)

Base 0x4005.0000
Offset 0x001
Type R/W, reset 0x20

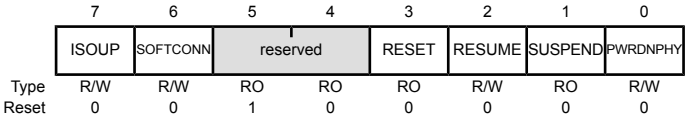



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:4 | reserved | RO | 0x2 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | RESET | R/W | 0 | RESET Signaling<br><br>Value Description<br>1 Enables RESET signaling on the bus.<br>0 Ends RESET signaling on the bus. |
| 2 | RESUME | R/W | 0 | RESUME Signaling<br><br>Value Description<br>1 Enables RESUME signaling when the Device is in SUSPEND mode.<br>0 Ends RESUME signaling on the bus.<br><br>This bit must be cleared by software 20 ms after being set. |
| 1 | SUSPEND | R/W1S | 0 | SUSPEND Mode<br><br>Value Description<br>1 Enables SUSPEND mode.<br>0 No effect. |
| 0 | PWRDNPHY | R/W | 0 | Power Down PHY<br><br>Value Description<br>1 Powers down the internal USB PHY.<br>0 No effect. |

## Device Mode

### USB Power (USBPOWER)

Base 0x4005.0000
Offset 0x001
Type R/W, reset 0x20




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | ISOUP | R/W | 0 | Isochronous Update<br><br>Value Description<br>1 The USB controller waits for an SOF token from the time the TXRDY bit is set in the **USBTXCSRLn** register before sending the packet. If an IN token is received before an SOF token, then a zero-length data packet is sent.<br>0 No effect.<br><br>**Note:** This bit is only valid for isochronous transfers. |
| 6 | SOFTCONN | R/W | 0 | Soft Connect/Disconnect<br><br>Value Description<br>1 The USB D+/D- lines are enabled.<br>0 The USB D+/D- lines are tri-stated. |
| 5:4 | reserved | RO | 0x2 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | RESET | RO | 0 | RESET Signaling<br><br>Value Description<br>1 RESET signaling is present on the bus.<br>0 RESET signaling is not present on the bus. |
| 2 | RESUME | R/W | 0 | RESUME Signaling<br><br>Value Description<br>1 Enables RESUME signaling when the Device is in SUSPEND mode.<br>0 Ends RESUME signaling on the bus.<br><br>This bit must be cleared by software 10 ms (a maximum of 15 ms) after being set. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | SUSPEND | RO | 0 | SUSPEND Mode<br><br>Value Description<br>1 The USB controller is in SUSPEND mode.<br>0 This bit is cleared when software reads the interrupt register or sets the RESUME bit above. |
| 0 | PWRDNPHY | R/W | 0 | Power Down PHY<br><br>Value Description<br>1 Powers down the internal USB PHY.<br>0 No effect. |

## Register 3: USB Transmit Interrupt Status (USBTXIS), offset 0x002

**Important:** This register is read-sensitive. See the register description for details.

**Host**

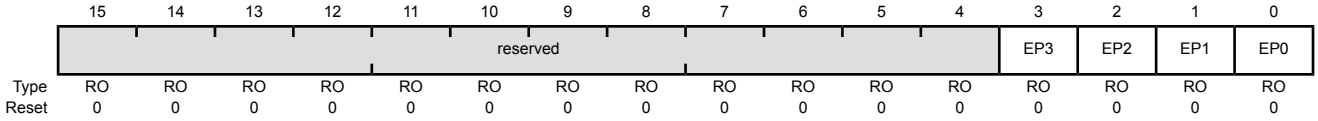
**Device**

**USBTXIS** is a 16-bit read-only register that indicates which interrupts are currently active for endpoint 0 and the transmit endpoints 1–3. The meaning of the EPn bits in this register are based on the mode of the device. The EP1, EP2 and EP3 bits always indicate that the USB controller is sending data; however, in Host mode, the bits refer to OUT endpoints; while in Device mode, the bits refer to IN endpoints. The EP0 bit is special in Host and Device modes and indicates that either a control IN or control OUT endpoint has generated an interrupt.

**Note:** Bits relating to endpoints that have not been configured always return 0. Note also that all active interrupts are cleared when this register is read.

USB Transmit Interrupt Status (USBTXIS)

Base 0x4005.0000
Offset 0x002
Type RO, reset 0x0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EP3 | RO | 0 | TX Endpoint 3 Interrupt<br>Value Description<br>0 No interrupt.<br>1 The Endpoint 3 transmit interrupt is asserted. |
| 2 | EP2 | RO | 0 | TX Endpoint 2 Interrupt<br>Same description as EP15. |
| 1 | EP1 | RO | 0 | TX Endpoint 1 Interrupt<br>Same description as EP15. |
| 0 | EP0 | RO | 0 | TX and RX Endpoint 0 Interrupt<br>Value Description<br>0 No interrupt.<br>1 The Endpoint 0 transmit and receive interrupt is asserted. |

## Register 4: USB Receive Interrupt Status (USBRXIS), offset 0x004

**Important:** This register is read-sensitive. See the register description for details.

**Host**

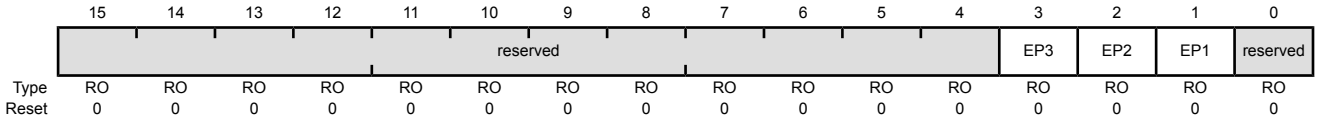
**USBRXIS** is a 16-bit read-only register that indicates which of the interrupts for receive endpoints 1–3 are currently active.

**Device**

**Note:** Bits relating to endpoints that have not been configured always return 0. Note also that all active interrupts are cleared when this register is read.

USB Receive Interrupt Status (USBRXIS)

Base 0x4005.0000
Offset 0x004
Type RO, reset 0x0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EP3 | RO | 0 | RX Endpoint 3 Interrupt<br>Value Description<br>0 No interrupt.<br>1 The Endpoint 3 receive interrupt is asserted. |
| 2 | EP2 | RO | 0 | RX Endpoint 2 Interrupt<br>Same description as EP3. |
| 1 | EP1 | RO | 0 | RX Endpoint 1 Interrupt<br>Same description as EP3. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 5: USB Transmit Interrupt Enable (USBTXIE), offset 0x006

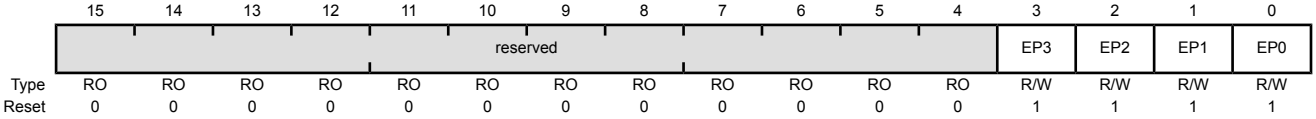
**Host**

**Device**

**USBTXIE** is a 16-bit register that provides interrupt enable bits for the interrupts in the **USBTXIS** register. When a bit is set, the USB interrupt is asserted to the interrupt controller when the corresponding interrupt bit in the **USBTXIS** register is set. When a bit is cleared, the interrupt in the **USBTXIS** register is still set but the USB interrupt to the interrupt controller is not asserted. On reset, the bits corresponding to endpoint 0 and transmit endpoints 1-3 are set to 1, while the remaining bits are set to 0.

USB Transmit Interrupt Enable (USBTXIE)

Base 0x4005.0000
Offset 0x006
Type R/W, reset 0x000F




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EP3 | R/W | 1 | TX Endpoint 3 Interrupt Enable<br>Value Description<br>1 An interrupt is sent to the interrupt controller when the EP3 bit in the **USBTXIS** register is set.<br>0 The EP3 transmit interrupt is suppressed and not sent to the interrupt controller. |
| 2 | EP2 | R/W | 1 | TX Endpoint 2 Interrupt Enable<br>Same description as EP3. |
| 1 | EP1 | R/W | 1 | TX Endpoint 1 Interrupt Enable<br>Same description as EP3. |
| 0 | EP0 | R/W | 1 | TX and RX Endpoint 0 Interrupt Enable<br>Value Description<br>1 An interrupt is sent to the interrupt controller when the EP0 bit in the **USBTXIS** register is set.<br>0 The EP0 transmit interrupt is suppressed and not sent to the interrupt controller. |

## Register 6: USB Receive Interrupt Enable (USBRXIE), offset 0x008

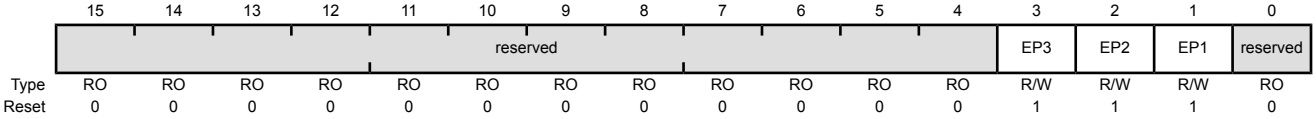
**Host**

**Device**

**USBRXIE** is a 16-bit register that provides interrupt enable bits for the interrupts in the **USBRXIS** register. When a bit is set, the USB interrupt is asserted to the interrupt controller when the corresponding interrupt bit in the **USBRXIS** register is set. When a bit is cleared, the interrupt in the **USBRXIS** register is still set but the USB interrupt to the interrupt controller is not asserted. On reset, the bits corresponding to receive endpoints 1-3 are set to 1, while the remaining bits are set to 0.

USB Receive Interrupt Enable (USBRXIE)

Base 0x4005.0000
Offset 0x008
Type R/W, reset 0x000E




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EP3 | R/W | 1 | RX Endpoint 3 Interrupt Enable |
| 2 | EP2 | R/W | 1 | RX Endpoint 2 Interrupt Enable |
| 1 | EP1 | R/W | 1 | RX Endpoint 1 Interrupt Enable |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 7: USB General Interrupt Status (USBIS), offset 0x00A

**Important:** This register is read-sensitive. See the register description for details.

**Host**

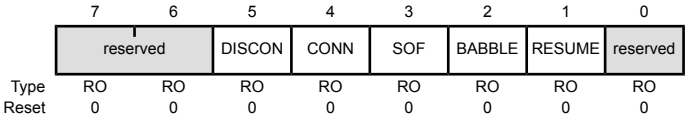
**Device**

**USBIS** is an 8-bit read-only register that indicates which USB interrupts are currently active. All active interrupts are cleared when this register is read.

### Host Mode

USB General Interrupt Status (USBIS)

Base 0x4005.0000
Offset 0x00A
Type RO, reset 0x00



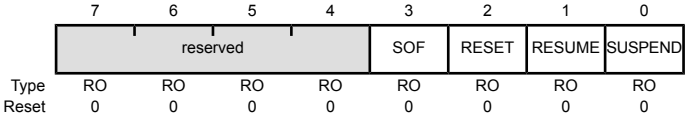

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:6 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5 | DISCON | RO | 0 | Session Disconnect<br>Value Description<br>1 A Device disconnect has been detected.<br>0 No interrupt. |
| 4 | CONN | RO | 0 | Session Connect<br>Value Description<br>1 A Device connection has been detected.<br>0 No interrupt. |
| 3 | SOF | RO | 0 | Start of Frame<br>Value Description<br>1 A new frame has started.<br>0 No interrupt. |
| 2 | BABBLE | RO | 0 | Babble Detected<br>Value Description<br>1 Babble has been detected. This interrupt is active only after the first SOF has been sent.<br>0 No interrupt. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | RESUME | RO | 0 | RESUME Signaling Detected<br><br>Value Description<br>1 RESUME signaling has been detected on the bus while the USB controller is in SUSPEND mode.<br>0 No interrupt.<br><br>This interrupt can only be used if the USB controller's system clock is enabled. If the user disables the clock programming, the **USBDRRIS**, **USBDRIM**, and **USBDRISC** registers should be used. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Device Mode

### USB General Interrupt Status (USBIS)

Base 0x4005.0000
Offset 0x00A
Type RO, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | SOF | RO | 0 | Start of Frame<br><br>Value Description<br>1 A new frame has started.<br>0 No interrupt. |
| 2 | RESET | RO | 0 | RESET Signaling Detected<br><br>Value Description<br>1 RESET signaling has been detected on the bus.<br>0 No interrupt. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | RESUME | RO | 0 | RESUME Signaling Detected<br><br>Value Description<br>1 RESUME signaling has been detected on the bus while the USB controller is in SUSPEND mode.<br>0 No interrupt.<br><br>This interrupt can only be used if the USB controller's system clock is enabled. If the user disables the clock programming, the **USBDRRIS**, **USBDRIM**, and **USBDRISC** registers should be used. |
| 0 | SUSPEND | RO | 0 | SUSPEND Signaling Detected<br><br>Value Description<br>1 SUSPEND signaling has been detected on the bus.<br>0 No interrupt. |

## Register 8: USB Interrupt Enable (USBIE), offset 0x00B

**Host**

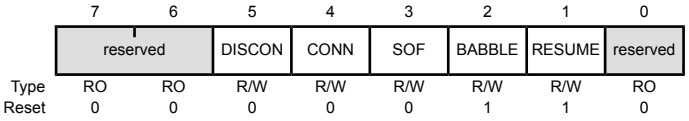
**Device**

**USBIE** is an 8-bit register that provides interrupt enable bits for each of the interrupts in **USBIS**. At reset interrupts 1 and 2 are enabled in Device mode.

### Host Mode

USB Interrupt Enable (USBIE)

Base 0x4005.0000
Offset 0x00B
Type R/W, reset 0x06




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:6 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5 | DISCON | R/W | 0 | Enable Disconnect Interrupt<br>Value Description<br>1 An interrupt is sent to the interrupt controller when the DISCON bit in the **USBIS** register is set.<br>0 The DISCON interrupt is suppressed and not sent to the interrupt controller. |
| 4 | CONN | R/W | 0 | Enable Connect Interrupt<br>Value Description<br>1 An interrupt is sent to the interrupt controller when the CONN bit in the **USBIS** register is set.<br>0 The CONN interrupt is suppressed and not sent to the interrupt controller. |
| 3 | SOF | R/W | 0 | Enable Start-of-Frame Interrupt<br>Value Description<br>1 An interrupt is sent to the interrupt controller when the SOF bit in the **USBIS** register is set.<br>0 The SOF interrupt is suppressed and not sent to the interrupt controller. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | BABBLE | R/W | 1 | Enable Babble Interrupt<br>Value Description<br>1 An interrupt is sent to the interrupt controller when the BABBLE bit in the **USBIS** register is set.<br>0 The BABBLE interrupt is suppressed and not sent to the interrupt controller. |
| 1 | RESUME | R/W | 1 | Enable RESUME Interrupt<br>Value Description<br>1 An interrupt is sent to the interrupt controller when the RESUME bit in the **USBIS** register is set.<br>0 The RESUME interrupt is suppressed and not sent to the interrupt controller. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Device Mode

### USB Interrupt Enable (USBIE)

Base 0x4005.0000
Offset 0x00B
Type R/W, reset 0x06

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | reserved | | DISCON | reserved | SOF | RESET | RESUME | SUSPEND |
| Type | RO | RO | R/W | RO | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:6 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5 | DISCON | R/W | 0 | Enable Disconnect Interrupt<br>Value Description<br>1 An interrupt is sent to the interrupt controller when the DISCON bit in the **USBIS** register is set.<br>0 The DISCON interrupt is suppressed and not sent to the interrupt controller. |
| 4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | SOF | R/W | 0 | Enable Start-of-Frame Interrupt<br><br>Value Description<br>1 An interrupt is sent to the interrupt controller when the SOF bit in the **USBIS** register is set.<br>0 The SOF interrupt is suppressed and not sent to the interrupt controller. |
| 2 | RESET | R/W | 1 | Enable RESET Interrupt<br><br>Value Description<br>1 An interrupt is sent to the interrupt controller when the RESET bit in the **USBIS** register is set.<br>0 The RESET interrupt is suppressed and not sent to the interrupt controller. |
| 1 | RESUME | R/W | 1 | Enable RESUME Interrupt<br><br>Value Description<br>1 An interrupt is sent to the interrupt controller when the RESUME bit in the **USBIS** register is set.<br>0 The RESUME interrupt is suppressed and not sent to the interrupt controller. |
| 0 | SUSPEND | R/W | 0 | Enable SUSPEND Interrupt<br><br>Value Description<br>1 An interrupt is sent to the interrupt controller when the SUSPEND bit in the **USBIS** register is set.<br>0 The SUSPEND interrupt is suppressed and not sent to the interrupt controller. |

## Register 9: USB Frame Value (USBFRAME), offset 0x00C

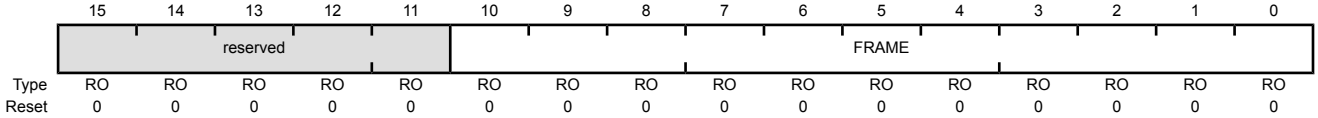
**Host**

**Device**

**USBFRAME** is a 16-bit read-only register that holds the last received frame number.

USB Frame Value (USBFRAME)

Base 0x4005.0000
Offset 0x00C
Type RO, reset 0x0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10:0 | FRAME | RO | 0x00 | Frame Number |

## Register 10: USB Endpoint Index (USBEPIDX), offset 0x00E

**Host**

**Device**

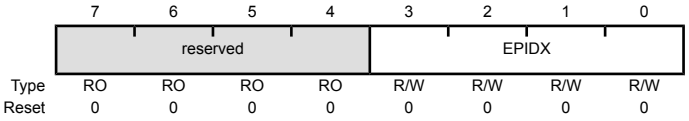
Each endpoint's buffer can be accessed by configuring a FIFO size and starting address. The **USBEPIDX** 16-bit register is used with the **USBTXFIFOSZ**, **USBRXFIFOSZ**, **USBTXFIFOADD**, and **USBRXFIFOADD** registers.

USB Endpoint Index (USBEPIDX)

Base 0x4005.0000

Offset 0x00E

Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3:0 | EPIDX | R/W | 0x0 | Endpoint Index<br>This bit field configures which endpoint is accessed when reading or writing to one of the USB controller's indexed registers. A value of 0x0 corresponds to Endpoint 0 and a value of 0x3 corresponds to Endpoint 3. |

## Register 11: USB Test Mode (USBTEST), offset 0x00F

**Host**

**Device**

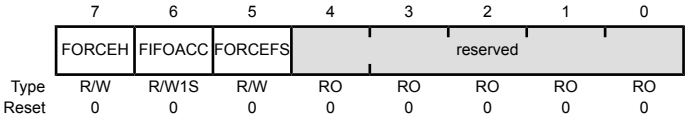
**USBTEST** is an 8-bit register that is primarily used to put the USB controller into one of the four test modes for operation described in the *USB 2.0 Specification*, in response to a SET FEATURE: USBTESTMODE command. This register is not used in normal operation.

**Note:** Only one of these bits should be set at any time.

### Host Mode

USB Test Mode (USBTEST)

Base 0x4005.0000
Offset 0x00F
Type R/W, reset 0x00



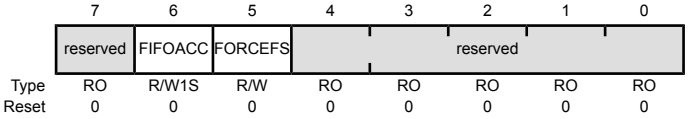

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | FORCEH | R/W | 0 | Force Host Mode<br><br>Value Description<br>1 Forces the USB controller to enter Host mode when the SESSION bit is set, regardless of whether the USB controller is connected to any peripheral. The state of the USB0DP and USB0DM signals is ignored. The USB controller then remains in Host mode until the SESSION bit is cleared, even if a Device is disconnected. If the FORCEH bit remains set, the USB controller re-enters Host mode the next time the SESSION bit is set.<br>0 No effect.<br><br>While in this mode, status of the bus connection may be read using the DEV bit of the **USBDEVCTL** register. The operating speed is determined from the FORCEFS bit. |
| 6 | FIFOACC | R/W1S | 0 | FIFO Access<br><br>Value Description<br>1 Transfers the packet in the endpoint 0 transmit FIFO to the endpoint 0 receive FIFO.<br>0 No effect.<br><br>This bit is cleared automatically. |
| 5 | FORCEFS | R/W | 0 | Force Full-Speed Mode<br><br>Value Description<br>1 Forces the USB controller into Full-Speed mode upon receiving a USB RESET.<br>0 The USB controller operates at Low Speed. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Device Mode

### USB Test Mode (USBTEST)

Base 0x4005.0000
Offset 0x00F
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | FIFOACC | R/W1S | 0 | FIFO Access<br>Value Description<br>1 Transfers the packet in the endpoint 0 transmit FIFO to the endpoint 0 receive FIFO.<br>0 No effect.<br>This bit is cleared automatically. |
| 5 | FORCEFS | R/W | 0 | Force Full-Speed Mode<br>Value Description<br>1 Forces the USB controller into Full-Speed mode upon receiving a USB RESET.<br>0 The USB controller operates at Low Speed. |
| 4:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 12: USB FIFO Endpoint 0 (USBFIFO0), offset 0x020

## Register 13: USB FIFO Endpoint 1 (USBFIFO1), offset 0x024

## Register 14: USB FIFO Endpoint 2 (USBFIFO2), offset 0x028

## Register 15: USB FIFO Endpoint 3 (USBFIFO3), offset 0x02C

**Important:** This register is read-sensitive. See the register description for details.

**Host**

**Device**

These 32-bit registers provide an address for CPU access to the FIFOs for each endpoint. Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

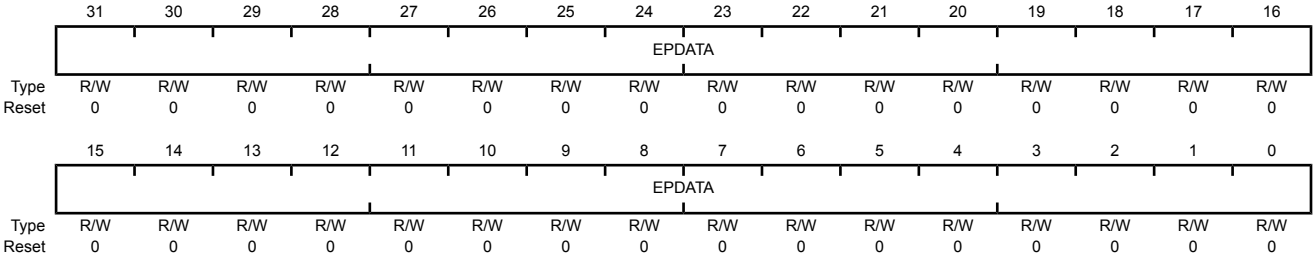
Transfers to and from FIFOs may be 8-bit, 16-bit or 32-bit as required, and any combination of accesses is allowed provided the data accessed is contiguous. All transfers associated with one packet must be of the same width so that the data is consistently byte-, halfword- or word-aligned. However, the last transfer may contain fewer bytes than the previous transfers in order to complete an odd-byte or odd-word transfer.

Depending on the size of the FIFO and the expected maximum packet size, the FIFOs support either single-packet or double-packet buffering (see the section called “Single-Packet Buffering” on page 663). Burst writing of multiple packets is not supported as flags must be set after each packet is written.

Following a STALL response or a transmit error on endpoint 1–3, the associated FIFO is completely flushed.

USB FIFO Endpoint 0 (USBFIFO0)

Base 0x4005.0000
Offset 0x020
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | EPDATA | R/W | 0x0000.0000 | Endpoint Data<br>Writing to this register loads the data into the Transmit FIFO and reading unloads data from the Receive FIFO. |

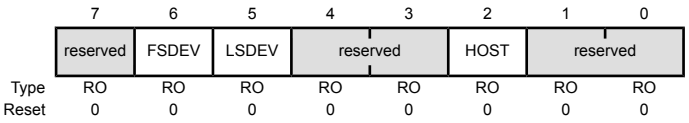
## Register 16: USB Device Control (USBDEVCTL), offset 0x060

**Host**

**USBDEVCTL** provides the status information for the current operating mode (Host or Device) of the USB controller. If the USB controller is in Host mode, this register also indicates if a full- or low-speed Device has been connected.

USB Device Control (USBDEVCTL)

Base 0x4005.0000
Offset 0x060
Type RO, reset 0x80




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | FSDEV | RO | 0 | Full-Speed Device Detected<br>Value Description<br>0 A full-speed Device has not been detected on the port.<br>1 A full-speed Device has been detected on the port. |
| 5 | LSDEV | RO | 0 | Low-Speed Device Detected<br>Value Description<br>0 A low-speed Device has not been detected on the port.<br>1 A low-speed Device has been detected on the port. |
| 4:3 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | HOST | RO | 0 | Host Mode<br>Value Description<br>0 The USB controller is acting as a Device.<br>1 The USB controller is acting as a Host.<br>**Note:** This value is only valid while a session is in progress. |
| 1:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 17: USB Transmit Dynamic FIFO Sizing (USBTXFIFOSZ), offset 0x062

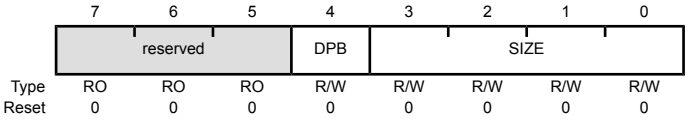
## Register 18: USB Receive Dynamic FIFO Sizing (USBRXFIFOSZ), offset 0x063

**Host**

These 8-bit registers allow the selected TX/RX endpoint FIFOs to be dynamically sized. **USBEPIDX** is used to configure each transmit endpoint's FIFO size.

**Device**

USB Transmit Dynamic FIFO Sizing (USBTXFIFOSZ)
Base 0x4005.0000
Offset 0x062
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:5 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | DPB | R/W | 0 | Double Packet Buffer Support<br>Value Description<br>0 Only single-packet buffering is supported.<br>1 Double-packet buffering is supported. |
| 3:0 | SIZE | R/W | 0x0 | Max Packet Size<br>Maximum packet size to be allowed.<br>If DPB = 0, the FIFO also is this size; if DPB = 1, the FIFO is twice this size.<br>Value Packet Size (Bytes)<br>0x0 8<br>0x1 16<br>0x2 32<br>0x3 64<br>0x4 128<br>0x5 256<br>0x6 512<br>0x7 1024<br>0x8 2048<br>0x9-0xF Reserved |

## Register 19: USB Transmit FIFO Start Address (USBTXFIFOADD), offset 0x064

## Register 20: USB Receive FIFO Start Address (USBRXFIFOADD), offset 0x066

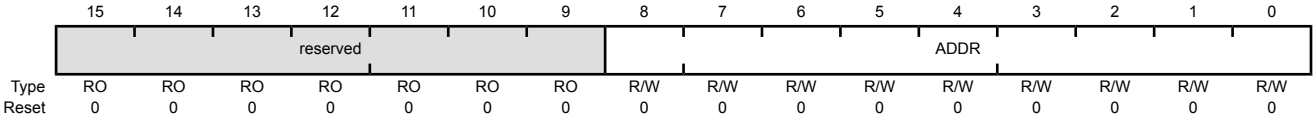
**Host**

**USBTXFIFOADD** and **USBRXFIFOADD** are 16-bit registers that control the start address of the selected transmit and receive endpoint FIFOs.

**Device**

USB Transmit FIFO Start Address (USBTXFIFOADD)

Base 0x4005.0000
Offset 0x064
Type R/W, reset 0x0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:9 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 8:0 | ADDR | R/W | 0x00 | Transmit/Receive Start Address<br>Start address of the endpoint FIFO. |

| Value | Start Address |
|---|---|
| 0x0 | 0 |
| 0x1 | 8 |
| 0x2 | 16 |
| 0x3 | 24 |
| 0x4 | 32 |
| 0x5 | 40 |
| 0x6 | 48 |
| 0x7 | 56 |
| 0x8 | 64 |
| ... | ... |
| 0x1FF | 4095 |

## Register 21: USB Connect Timing (USBCONTIM), offset 0x07A

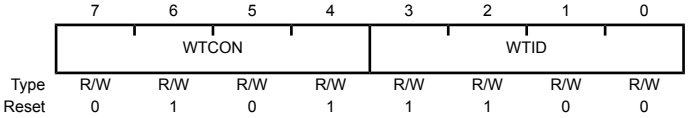
**Host**

This 8-bit configuration register specifies connection delay.

**Device**

USB Connect Timing (USBCONTIM)

Base 0x4005.0000
Offset 0x07A
Type R/W, reset 0x5C




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:4 | WTCON | R/W | 0x5 | Connect Wait<br>This field configures the wait required to allow for the user's connect/disconnect filter, in units of 533.3 ns. The default corresponds to 2.667 µs. |
| 3:0 | WTID | R/W | 0xC | Wait ID<br>This field configures the delay required from the enable of the ID detection to when the ID value is valid, in units of 4.369 ms. The default corresponds to 52.43 ms. |

## Register 22: USB Full-Speed Last Transaction to End of Frame Timing (USBFSEOF), offset 0x07D

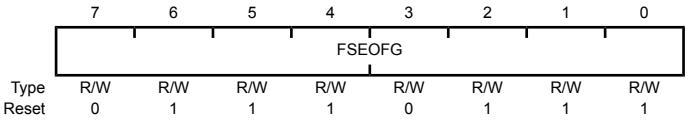
**Host**

This 8-bit configuration register specifies the minimum time gap allowed between the start of the last transaction and the EOF for full-speed transactions.

**Device**

USB Full-Speed Last Transaction to End of Frame Timing (USBFSEOF)

Base 0x4005.0000
Offset 0x07D
Type R/W, reset 0x77




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:0 | FSEOFG | R/W | 0x77 | Full-Speed End-of-Frame Gap<br>This field is used during full-speed transactions to configure the gap between the last transaction and the End-of-Frame (EOF), in units of 533.3 ns. The default corresponds to 63.46 µs. |

## Register 23: USB Low-Speed Last Transaction to End of Frame Timing (USBLSEOF), offset 0x07E

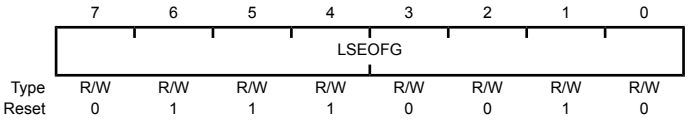
**Host**

This 8-bit configuration register specifies the minimum time gap that is to be allowed between the start of the last transaction and the EOF for low-speed transactions.

**Device**

USB Low-Speed Last Transaction to End of Frame Timing (USBLSEOF)

Base 0x4005.0000
Offset 0x07E
Type R/W, reset 0x72




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:0 | LSEOFG | R/W | 0x72 | Low-Speed End-of-Frame Gap<br>This field is used during low-speed transactions to set the gap between the last transaction and the End-of-Frame (EOF), in units of 1.067 µs. The default corresponds to 121.6 µs. |

## Register 24: USB Transmit Functional Address Endpoint 0 (USBTXFUNCADDR0), offset 0x080

## Register 25: USB Transmit Functional Address Endpoint 1 (USBTXFUNCADDR1), offset 0x088

## Register 26: USB Transmit Functional Address Endpoint 2 (USBTXFUNCADDR2), offset 0x090

## Register 27: USB Transmit Functional Address Endpoint 3 (USBTXFUNCADDR3), offset 0x098

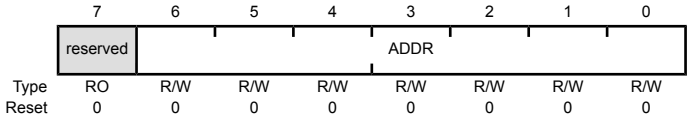
**Host**

**USBTXFUNCADDRn** is an 8-bit read/write register that records the address of the target function to be accessed through the associated endpoint (EPn). **USBTXFUNCADDRn** must be defined for each transmit endpoint that is used.

**Note:** **USBTXFUNCADDR0** is used for both receive and transmit for endpoint 0.

USB Transmit Functional Address Endpoint 0 (USBTXFUNCADDR0)

Base 0x4005.0000
Offset 0x080
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | ADDR | R/W | 0x00 | Device Address<br>Specifies the USB bus address for the target Device. |

### Register 28: USB Transmit Hub Address Endpoint 0 (USBTXHUBADDR0), offset 0x082

### Register 29: USB Transmit Hub Address Endpoint 1 (USBTXHUBADDR1), offset 0x08A

### Register 30: USB Transmit Hub Address Endpoint 2 (USBTXHUBADDR2), offset 0x092

### Register 31: USB Transmit Hub Address Endpoint 3 (USBTXHUBADDR3), offset 0x09A

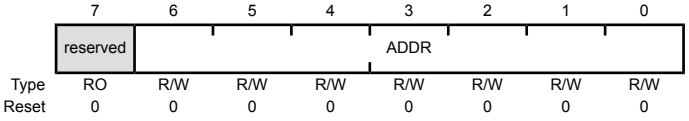
**Host**

**USBTXHUBADDRn** is an 8-bit read/write register that, like **USBTXHUBPORTn**, only must be written when a USB Device is connected to transmit endpoint EPn via a USB 2.0 hub. This register records the address of the USB 2.0 hub through which the target associated with the endpoint is accessed.

**Note:** **USBTXHUBADDR0** is used for both receive and transmit for endpoint 0.

USB Transmit Hub Address Endpoint 0 (USBTXHUBADDR0)

Base 0x4005.0000
Offset 0x082
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | ADDR | R/W | 0x00 | Hub Address<br>This field specifies the USB bus address for the USB 2.0 hub. |

## Register 32: USB Transmit Hub Port Endpoint 0 (USBTXHUBPORT0), offset 0x083

## Register 33: USB Transmit Hub Port Endpoint 1 (USBTXHUBPORT1), offset 0x08B

## Register 34: USB Transmit Hub Port Endpoint 2 (USBTXHUBPORT2), offset 0x093

## Register 35: USB Transmit Hub Port Endpoint 3 (USBTXHUBPORT3), offset 0x09B

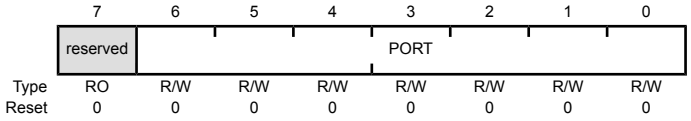
**Host**

**USBTXHUBPORTn** is an 8-bit read/write register that, like **USBTXHUBADDRn**, only must be written when a full- or low-speed Device is connected to transmit endpoint EPn via a USB 2.0 hub. This register records the port of the USB 2.0 hub through which the target associated with the endpoint is accessed.

**Note:** **USBTXHUBPORT0** is used for both receive and transmit for endpoint 0.

USB Transmit Hub Port Endpoint 0 (USBTXHUBPORT0)

Base 0x4005.0000
Offset 0x083
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | PORT | R/W | 0x00 | Hub Port<br>This field specifies the USB hub port number. |

## Register 36: USB Receive Functional Address Endpoint 1 (USBRXFUNCADDR1), offset 0x08C

## Register 37: USB Receive Functional Address Endpoint 2 (USBRXFUNCADDR2), offset 0x094

## Register 38: USB Receive Functional Address Endpoint 3 (USBRXFUNCADDR3), offset 0x09C

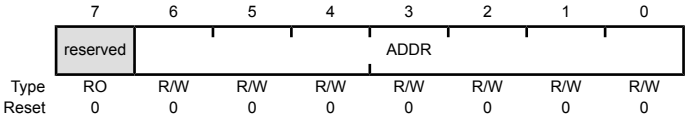
**Host**

**USBRXFUNCADDRn** is an 8-bit read/write register that records the address of the target function accessed through the associated endpoint (EPn). **USBRXFUNCADDRn** must be defined for each receive endpoint that is used.

**Note:** **USBTXFUNCADDR0** is used for both receive and transmit for endpoint 0.

USB Receive Functional Address Endpoint 1 (USBRXFUNCADDR1)

Base 0x4005.0000
Offset 0x08C
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | ADDR | R/W | 0x00 | Device Address<br>This field specifies the USB bus address for the target Device. |

## Register 39: USB Receive Hub Address Endpoint 1 (USBRXHUBADDR1), offset 0x08E

## Register 40: USB Receive Hub Address Endpoint 2 (USBRXHUBADDR2), offset 0x096

## Register 41: USB Receive Hub Address Endpoint 3 (USBRXHUBADDR3), offset 0x09E

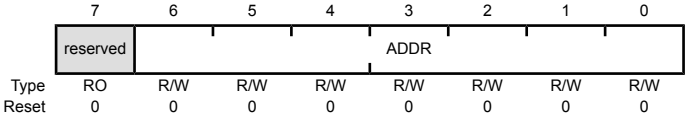
**Host**

**USBRXHUBADDRn** is an 8-bit read/write register that, like **USBRXHUBPORTn**, only must be written when a full- or low-speed Device is connected to receive endpoint EPn via a USB 2.0 hub. This register records the address of the USB 2.0 hub through which the target associated with the endpoint is accessed.

**Note:** **USBTXHUBADDR0** is used for both receive and transmit for endpoint 0.

USB Receive Hub Address Endpoint 1 (USBRXHUBADDR1)

Base 0x4005.0000
Offset 0x08E
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | ADDR | R/W | 0x00 | Hub Address<br>This field specifies the USB bus address for the USB 2.0 hub. |

## Register 42: USB Receive Hub Port Endpoint 1 (USBRXHUBPORT1), offset 0x08F

## Register 43: USB Receive Hub Port Endpoint 2 (USBRXHUBPORT2), offset 0x097

## Register 44: USB Receive Hub Port Endpoint 3 (USBRXHUBPORT3), offset 0x09F

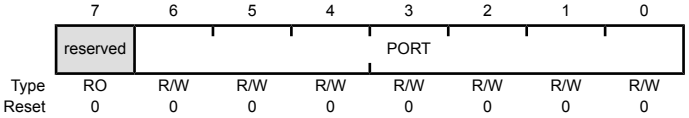
**Host**

**USBRXHUBPORTn** is an 8-bit read/write register that, like **USBRXHUBADDRn**, only must be written when a full- or low-speed Device is connected to receive endpoint EPn via a USB 2.0 hub. This register records the port of the USB 2.0 hub through which the target associated with the endpoint is accessed.

**Note:** **USBTXHUBPORT0** is used for both receive and transmit for endpoint 0.

USB Receive Hub Port Endpoint 1 (USBRXHUBPORT1)

Base 0x4005.0000
Offset 0x08F
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | PORT | R/W | 0x00 | Hub Port<br>This field specifies the USB hub port number. |

## Register 45: USB Maximum Transmit Data Endpoint 1 (USBTXMAXP1), offset 0x110

## Register 46: USB Maximum Transmit Data Endpoint 2 (USBTXMAXP2), offset 0x120

## Register 47: USB Maximum Transmit Data Endpoint 3 (USBTXMAXP3), offset 0x130

**Host**

The **USBTXMAXPn** 16-bit register defines the maximum amount of data that can be transferred through the transmit endpoint in a single operation.

**Device**

Bits 10:0 define (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes but is subject to the constraints placed by the *USB Specification* on packet sizes for bulk, interrupt and isochronous transfers in full-speed operation.

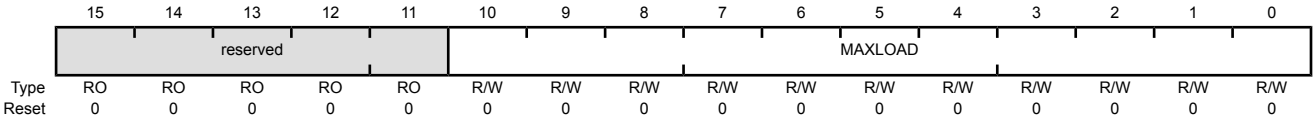
The total amount of data represented by the value written to this register must not exceed the FIFO size for the transmit endpoint, and must not exceed half the FIFO size if double-buffering is required.

If this register is changed after packets have been sent from the endpoint, the transmit endpoint FIFO must be completely flushed (using the FLUSH bit in **USBTXCSRLn**) after writing the new value to this register.

**Note:** **USBTXMAXPn** must be set to an even number of bytes for proper interrupt generation in µDMA Basic Mode.

USB Maximum Transmit Data Endpoint 1 (USBTXMAXP1)

Base 0x4005.0000
Offset 0x110
Type R/W, reset 0x0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:11 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10:0 | MAXLOAD | R/W | 0x000 | Maximum Payload<br>This field specifies the maximum payload in bytes per transaction. |

## Register 48: USB Control and Status Endpoint 0 Low (USBCSRL0), offset 0x102

**Host**

**Device**

**USBCSRL0** is an 8-bit register that provides control and status bits for endpoint 0.

### Host Mode

USB Control and Status Endpoint 0 Low (USBCSRL0)

Base 0x4005.0000
Offset 0x102
Type R/W, reset 0x00

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | NAKTO | STATUS | REQPKT | ERROR | SETUP | STALLED | TXRDY | RXRDY |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | NAKTO | R/W | 0 | NAK Timeout<br><br>Value Description<br>0 No timeout.<br>1 Indicates that endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the **USBNAKLMT** register.<br><br>Software must clear this bit to allow the endpoint to continue. |
| 6 | STATUS | R/W | 0 | STATUS Packet<br><br>Value Description<br>0 No transaction.<br>1 Initiates a STATUS stage transaction. This bit must be set at the same time as the TXRDY or REQPKT bit is set.<br><br>Setting this bit ensures that the DT bit is set in the **USBCSRH0** register so that a DATA1 packet is used for the STATUS stage transaction.<br>This bit is automatically cleared when the STATUS stage is over. |
| 5 | REQPKT | R/W | 0 | Request Packet<br><br>Value Description<br>0 No request.<br>1 Requests an IN transaction.<br><br>This bit is cleared when the RXRDY bit is set. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | ERROR | R/W | 0 | Error<br><br>Value Description<br>0 No error.<br>1 Three attempts have been made to perform a transaction with no response from the peripheral. The EP0 bit in the **USBTXIS** register is also set in this situation.<br><br>Software must clear this bit. |
| 3 | SETUP | R/W | 0 | Setup Packet<br><br>Value Description<br>0 Sends an OUT token.<br>1 Sends a SETUP token instead of an OUT token for the transaction. This bit should be set at the same time as the TXRDY bit is set.<br><br>Setting this bit always clears the DT bit in the **USBCSRH0** register to send a DATA0 packet. |
| 2 | STALLED | R/W | 0 | Endpoint Stalled<br><br>Value Description<br>0 No handshake has been received.<br>1 A STALL handshake has been received.<br><br>Software must clear this bit. |
| 1 | TXRDY | R/W | 0 | Transmit Packet Ready<br><br>Value Description<br>0 No transmit packet is ready.<br>1 Software sets this bit after loading a data packet into the TX FIFO. The EP0 bit in the **USBTXIS** register is also set in this situation.<br>If both the TXRDY and SETUP bits are set, a setup packet is sent. If just TXRDY is set, an OUT packet is sent.<br><br>This bit is cleared automatically when the data packet has been transmitted. |
| 0 | RXRDY | R/W | 0 | Receive Packet Ready<br><br>Value Description<br>0 No received packet has been received.<br>1 Indicates that a data packet has been received in the RX FIFO. The EP0 bit in the **USBTXIS** register is also set in this situation.<br><br>Software must clear this bit after the packet has been read from the FIFO to acknowledge that the data has been read from the FIFO. |

## Device Mode

### USB Control and Status Endpoint 0 Low (USBCSRL0)

Base 0x4005.0000
Offset 0x102
Type R/W, reset 0x00

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SETENDC | RXRDYC | STALL | SETEND | DATAEND | STALLED | TXRDY | RXRDY |
| Type | W1C | W1C | W1C | RO | W1C | R/W | R/W | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | SETENDC | W1C | 0 | Setup End Clear<br>Writing a 1 to this bit clears the SETEND bit. |
| 6 | RXRDYC | W1C | 0 | RXRDY Clear<br>Writing a 1 to this bit clears the RXRDY bit. |
| 5 | STALL | W1C | 0 | Send Stall<br>Value Description<br>0 No effect.<br>1 Terminates the current transaction and transmits the STALL handshake.<br>This bit is cleared automatically after the STALL handshake is transmitted. |
| 4 | SETEND | RO | 0 | Setup End<br>Value Description<br>0 A control transaction has not ended or ended after the DATAEND bit was set.<br>1 A control transaction has ended before the DATAEND bit has been set. The EP0 bit in the **USBTXIS** register is also set in this situation.<br>This bit is cleared by writing a 1 to the SETENDC bit. |
| 3 | DATAEND | W1C | 0 | Data End<br>Value Description<br>0 No effect.<br>1 Set this bit in the following situations:<br>■ When setting TXRDY for the last data packet<br>■ When clearing RXRDY after unloading the last data packet<br>■ When setting TXRDY for a zero-length data packet<br>This bit is cleared automatically. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | STALLED | R/W | 0 | Endpoint Stalled<br><br>Value Description<br>0 A STALL handshake has not been transmitted.<br>1 A STALL handshake has been transmitted.<br><br>Software must clear this bit. |
| 1 | TXRDY | R/W | 0 | Transmit Packet Ready<br><br>Value Description<br>0 No transmit packet is ready.<br>1 Software sets this bit after loading an IN data packet into the TX FIFO. The EP0 bit in the **USBTXIS** register is also set in this situation.<br><br>This bit is cleared automatically when the data packet has been transmitted. |
| 0 | RXRDY | RO | 0 | Receive Packet Ready<br><br>Value Description<br>0 No data packet has been received.<br>1 A data packet has been received. The EP0 bit in the **USBTXIS** register is also set in this situation.<br><br>This bit is cleared by writing a 1 to the RXRDYC bit. |

## Register 49: USB Control and Status Endpoint 0 High (USBCSRH0), offset 0x103

**Host**

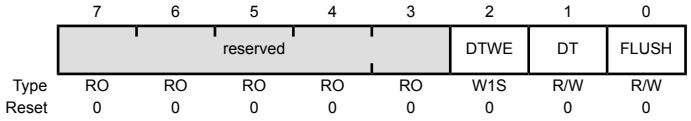
**Device**

**USBSR0H** is an 8-bit register that provides control and status bits for endpoint 0.

### Host Mode

USB Control and Status Endpoint 0 High (USBCSRH0)

Base 0x4005.0000
Offset 0x103
Type R/W, reset 0x00



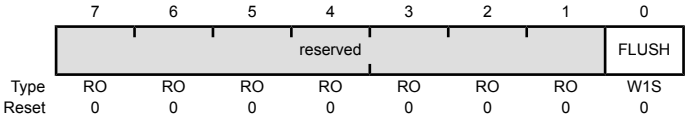

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:3 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | DTWE | W1S | 0 | Data Toggle Write Enable<br>Value Description<br>0 The DT bit cannot be written.<br>1 Enables the current state of the endpoint 0 data toggle to be written (see DT bit).<br>This bit is automatically cleared once the new value is written. |
| 1 | DT | R/W | 0 | Data Toggle<br>When read, this bit indicates the current state of the endpoint 0 data toggle.<br>If DTWE is set, this bit may be written with the required setting of the data toggle. If DTWE is Low, this bit cannot be written. Care should be taken when writing to this bit as it should only be changed to RESET USB endpoint 0. |
| 0 | FLUSH | R/W | 0 | Flush FIFO<br>Value Description<br>0 No effect.<br>1 Flushes the next packet to be transmitted/read from the endpoint 0 FIFO. The FIFO pointer is reset and the TXRDY/RXRDY bit is cleared.<br>This bit is automatically cleared after the flush is performed. |

**Important:** This bit should only be set when TXRDY is clear and RXRDY is set. At other times, it may cause data to be corrupted.

### Device Mode

USB Control and Status Endpoint 0 High (USBCSRH0)

Base 0x4005.0000
Offset 0x103
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | FLUSH | W1S | 0 | Flush FIFO<br><br>Value Description<br>0 No effect.<br>1 Flushes the next packet to be transmitted/read from the endpoint 0 FIFO. The FIFO pointer is reset and the TXRDY/RXRDY bit is cleared.<br><br>This bit is automatically cleared after the flush is performed. |

**Important:** This bit should only be set when TXRDY is clear and RXRDY is set. At other times, it may cause data to be corrupted.

## Register 50: USB Receive Byte Count Endpoint 0 (USBCOUNT0), offset 0x108

**Host**

**Device**

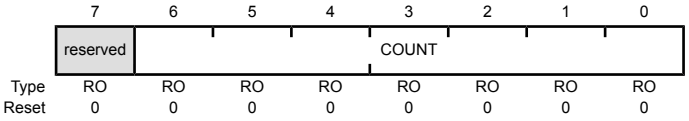
**USBCOUNT0** is an 8-bit read-only register that indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while the RXRDY bit is set.

USB Receive Byte Count Endpoint 0 (USBCOUNT0)

Base 0x4005.0000

Offset 0x108

Type RO, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | COUNT | RO | 0x00 | FIFO Count<br>COUNT is a read-only value that indicates the number of received data bytes in the endpoint 0 FIFO. |

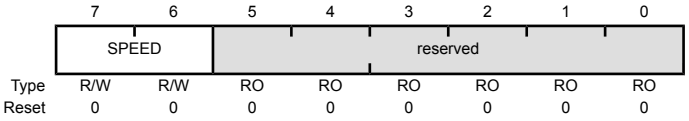
## Register 51: USB Type Endpoint 0 (USBTYPE0), offset 0x10A

**Host**

This is an 8-bit register that must be written with the operating speed of the targeted Device being communicated with using endpoint 0.

USB Type Endpoint 0 (USBTYPE0)

Base 0x4005.0000
Offset 0x10A
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:6 | SPEED | R/W | 0x0 | Operating Speed<br>This field specifies the operating speed of the target Device. If selected, the target is assumed to have the same connection speed as the USB controller.<br><br>Value / Description<br>0x0 - 0x1 Reserved<br>0x2 Full<br>0x3 Low |
| 5:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 52: USB NAK Limit (USBNAKLMT), offset 0x10B

**Host**

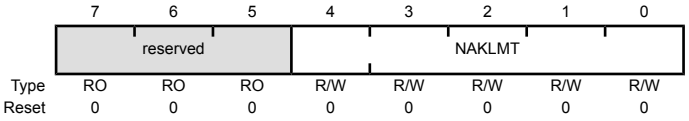
**USBNAKLMT** is an 8-bit register that sets the number of frames after which endpoint 0 should time out on receiving a stream of NAK responses. (Equivalent settings for other endpoints can be made through their **USBTXINTERVALn** and **USBRXINTERVALn** registers.)

The number of frames selected is $2^{(m-1)}$ (where *m* is the value set in the register, with valid values of 2–16). If the Host receives NAK responses from the target for more frames than the number represented by the limit set in this register, the endpoint is halted.

**Note:** A value of 0 or 1 disables the NAK timeout function.

USB NAK Limit (USBNAKLMT)

Base 0x4005.0000
Offset 0x10B
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:5 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4:0 | NAKLMT | R/W | 0x0 | EP0 NAK Limit<br>This field specifies the number of frames after receiving a stream of NAK responses. |

## Register 53: USB Transmit Control and Status Endpoint 1 Low (USBTXCSRL1), offset 0x112

## Register 54: USB Transmit Control and Status Endpoint 2 Low (USBTXCSRL2), offset 0x122

## Register 55: USB Transmit Control and Status Endpoint 3 Low (USBTXCSRL3), offset 0x132

**Host**

**Device**

**USBTXCSRLn** is an 8-bit register that provides control and status bits for transfers through the currently selected transmit endpoint.

### Host Mode

USB Transmit Control and Status Endpoint 1 Low (USBTXCSRL1)

Base 0x4005.0000
Offset 0x112
Type R/W, reset 0x00

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | NAKTO | CLRDT | STALLED | SETUP | FLUSH | ERROR | FIFONE | TXRDY |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | NAKTO | R/W | 0 | NAK Timeout<br>Value Description<br>0 No timeout.<br>1 *Bulk endpoints only:* Indicates that the transmit endpoint is halted following the receipt of NAK responses for longer than the time set by the NAKLMT field in the **USBTXINTERVALn** register. Software must clear this bit to allow the endpoint to continue. |
| 6 | CLRDT | R/W | 0 | Clear Data Toggle<br>Writing a 1 to this bit clears the DT bit in the **USBTXCSRHn** register. |
| 5 | STALLED | R/W | 0 | Endpoint Stalled<br>Value Description<br>0 A STALL handshake has not been received.<br>1 Indicates that a STALL handshake has been received. When this bit is set, any µDMA request that is in progress is stopped, the FIFO is completely flushed, and the TXRDY bit is cleared.<br>Software must clear this bit. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | SETUP | R/W | 0 | Setup Packet |

| Value | Description |
|---|---|
| 0 | No SETUP token is sent. |
| 1 | Sends a SETUP token instead of an OUT token for the transaction. This bit should be set at the same time as the TXRDY bit is set. |

**Note:** Setting this bit also clears the DT bit in the **USBTXCSRHn** register.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | FLUSH | R/W | 0 | Flush FIFO |

| Value | Description |
|---|---|
| 0 | No effect. |
| 1 | Flushes the latest packet from the endpoint transmit FIFO. The FIFO pointer is reset and the TXRDY bit is cleared. The EPn bit in the **USBTXIS** register is also set in this situation. |

This bit may be set simultaneously with the TXRDY bit to abort the packet that is currently being loaded into the FIFO. Note that if the FIFO is double-buffered, FLUSH may have to be set twice to completely clear the FIFO.

**Important:** This bit should only be set when the TXRDY bit is clear. At other times, it may cause data to be corrupted.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | ERROR | R/W | 0 | Error |

| Value | Description |
|---|---|
| 0 | No error. |
| 1 | Three attempts have been made to send a packet and no handshake packet has been received. The TXRDY bit is cleared, the EPn bit in the **USBTXIS** register is set, and the FIFO is completely flushed in this situation. |

Software must clear this bit.

**Note:** This is valid only when the endpoint is operating in Bulk or Interrupt mode.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | FIFONE | R/W | 0 | FIFO Not Empty |

| Value | Description |
|---|---|
| 0 | The FIFO is empty. |
| 1 | At least one packet is in the transmit FIFO. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | TXRDY | R/W | 0 | Transmit Packet Ready |

| Value | Description |
|---|---|
| 0 | No transmit packet is ready. |
| 1 | Software sets this bit after loading a data packet into the TX FIFO. |

This bit is cleared automatically when a data packet has been transmitted. The EPn bit in the **USBTXIS** register is also set at this point. TXRDY is also automatically cleared prior to loading a second packet into a double-buffered FIFO.

## Device Mode

USB Transmit Control and Status Endpoint 1 Low (USBTXCSRL1)

Base 0x4005.0000
Offset 0x112
Type R/W, reset 0x00

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | reserved | CLRDT | STALLED | STALL | FLUSH | UNDRN | FIFONE | TXRDY |
| Type | RO | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | CLRDT | R/W | 0 | Clear Data Toggle<br>Writing a 1 to this bit clears the DT bit in the **USBTXCSRHn** register. |
| 5 | STALLED | R/W | 0 | Endpoint Stalled |

| Value | Description |
|---|---|
| 0 | A STALL handshake has not been transmitted. |
| 1 | A STALL handshake has been transmitted. The FIFO is flushed and the TXRDY bit is cleared. |

Software must clear this bit.

| | | | | |
|---|---|---|---|---|
| 4 | STALL | R/W | 0 | Send STALL |

| Value | Description |
|---|---|
| 0 | No effect. |
| 1 | Issues a STALL handshake to an IN token. |

Software clears this bit to terminate the STALL condition.

**Note:** This bit has no effect in isochronous transfers.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | FLUSH | R/W | 0 | Flush FIFO<br><br>Value Description<br>0 No effect.<br>1 Flushes the latest packet from the endpoint transmit FIFO. The FIFO pointer is reset and the TXRDY bit is cleared. The EPn bit in the **USBTXIS** register is also set in this situation.<br><br>This bit may be set simultaneously with the TXRDY bit to abort the packet that is currently being loaded into the FIFO. Note that if the FIFO is double-buffered, FLUSH may have to be set twice to completely clear the FIFO.<br><br>**Important:** This bit should only be set when the TXRDY bit is clear. At other times, it may cause data to be corrupted. |
| 2 | UNDRN | R/W | 0 | Underrun<br><br>Value Description<br>0 No underrun.<br>1 An IN token has been received when TXRDY is not set.<br><br>Software must clear this bit. |
| 1 | FIFONE | R/W | 0 | FIFO Not Empty<br><br>Value Description<br>0 The FIFO is empty.<br>1 At least one packet is in the transmit FIFO. |
| 0 | TXRDY | R/W | 0 | Transmit Packet Ready<br><br>Value Description<br>0 No transmit packet is ready.<br>1 Software sets this bit after loading a data packet into the TX FIFO.<br><br>This bit is cleared automatically when a data packet has been transmitted. The EPn bit in the **USBTXIS** register is also set at this point. TXRDY is also automatically cleared prior to loading a second packet into a double-buffered FIFO. |

## Register 56: USB Transmit Control and Status Endpoint 1 High (USBTXCSRH1), offset 0x113

## Register 57: USB Transmit Control and Status Endpoint 2 High (USBTXCSRH2), offset 0x123

## Register 58: USB Transmit Control and Status Endpoint 3 High (USBTXCSRH3), offset 0x133

**Host**

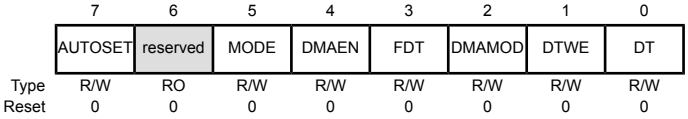
**Device**

**USBTXCSRHn** is an 8-bit register that provides additional control for transfers through the currently selected transmit endpoint.

### Host Mode

USB Transmit Control and Status Endpoint 1 High (USBTXCSRH1)

Base 0x4005.0000
Offset 0x113
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | AUTOSET | R/W | 0 | Auto Set<br>Value Description<br>0 The TXRDY bit must be set manually.<br>1 Enables the TXRDY bit to be automatically set when data of the maximum packet size (value in **USBTXMAXPn**) is loaded into the transmit FIFO. If a packet of less than the maximum packet size is loaded, then the TXRDY bit must be set manually. |
| 6 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5 | MODE | R/W | 0 | Mode<br>Value Description<br>0 Enables the endpoint direction as RX.<br>1 Enables the endpoint direction as TX.<br>**Note:** This bit only has an effect when the same endpoint FIFO is used for both transmit and receive transactions. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | DMAEN | R/W | 0 | DMA Request Enable<br>Value Description<br>0 Disables the µDMA request for the transmit endpoint.<br>1 Enables the µDMA request for the transmit endpoint.<br>**Note:** 3 TX and 3 /RX endpoints can be connected to the µDMA module. If this bit is set for a particular endpoint, the DMAATX, DMABTX, or DMACTX field in the **USB DMA Select (USBDMASEL)** register must be programmed correspondingly. |
| 3 | FDT | R/W | 0 | Force Data Toggle<br>Value Description<br>0 No effect.<br>1 Forces the endpoint DT bit to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This bit can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints. |
| 2 | DMAMOD | R/W | 0 | DMA Request Mode<br>Value Description<br>0 An interrupt is generated after every µDMA packet transfer.<br>1 An interrupt is generated only after the entire µDMA transfer is complete.<br>**Note:** This bit must not be cleared either before or in the same cycle as the above DMAEN bit is cleared. |
| 1 | DTWE | R/W | 0 | Data Toggle Write Enable<br>Value Description<br>0 The DT bit cannot be written.<br>1 Enables the current state of the transmit endpoint data to be written (see DT bit).<br>This bit is automatically cleared once the new value is written. |
| 0 | DT | R/W | 0 | Data Toggle<br>When read, this bit indicates the current state of the transmit endpoint data toggle.<br>If DTWE is High, this bit may be written with the required setting of the data toggle. If DTWE is Low, any value written to this bit is ignored. Care should be taken when writing to this bit as it should only be changed to RESET the transmit endpoint. |

## Device Mode

USB Transmit Control and Status Endpoint 1 High (USBTXCSRH1)

Base 0x4005.0000
Offset 0x113
Type R/W, reset 0x00

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | AUTOSET | ISO | MODE | DMAEN | FDT | DMAMOD | reserved | |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | AUTOSET | R/W | 0 | Auto Set |

| Value | Description |
|---|---|
| 0 | The TXRDY bit must be set manually. |
| 1 | Enables the TXRDY bit to be automatically set when data of the maximum packet size (value in **USBTXMAXPn**) is loaded into the transmit FIFO. If a packet of less than the maximum packet size is loaded, then the TXRDY bit must be set manually. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | ISO | R/W | 0 | Isochronous Transfers |

| Value | Description |
|---|---|
| 0 | Enables the transmit endpoint for bulk or interrupt transfers. |
| 1 | Enables the transmit endpoint for isochronous transfers. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 5 | MODE | R/W | 0 | Mode |

| Value | Description |
|---|---|
| 0 | Enables the endpoint direction as RX. |
| 1 | Enables the endpoint direction as TX. |

**Note:** This bit only has an effect where the same endpoint FIFO is used for both transmit and receive transactions.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | DMAEN | R/W | 0 | DMA Request Enable |

| Value | Description |
|---|---|
| 0 | Disables the µDMA request for the transmit endpoint. |
| 1 | Enables the µDMA request for the transmit endpoint. |

**Note:** 3 TX and 3 RX endpoints can be connected to the µDMA module. If this bit is set for a particular endpoint, the DMAATX, DMABTX, or DMACTX field in the **USB DMA Select (USBDMASEL)** register must be programmed correspondingly.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | FDT | R/W | 0 | Force Data Toggle<br><br>Value Description<br>0 No effect.<br>1 Forces the endpoint DT bit to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This bit can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints. |
| 2 | DMAMOD | R/W | 0 | DMA Request Mode<br><br>Value Description<br>0 An interrupt is generated after every µDMA packet transfer.<br>1 An interrupt is generated only after the entire µDMA transfer is complete.<br><br>**Note:** This bit must not be cleared either before or in the same cycle as the above DMAEN bit is cleared. |
| 1:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 59: USB Maximum Receive Data Endpoint 1 (USBRXMAXP1), offset 0x114

## Register 60: USB Maximum Receive Data Endpoint 2 (USBRXMAXP2), offset 0x124

## Register 61: USB Maximum Receive Data Endpoint 3 (USBRXMAXP3), offset 0x134

**Host**

The **USBRXMAXPn** is a 16-bit register which defines the maximum amount of data that can be transferred through the selected receive endpoint in a single operation.

**Device**

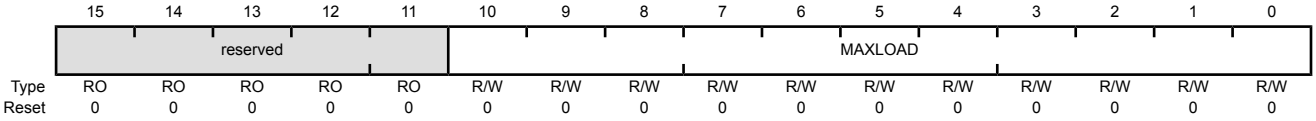
Bits 10:0 define (in bytes) the maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes but is subject to the constraints placed by the *USB Specification* on packet sizes for bulk, interrupt and isochronous transfers in full-speed operations.

The total amount of data represented by the value written to this register must not exceed the FIFO size for the receive endpoint, and must not exceed half the FIFO size if double-buffering is required.

**Note:** **USBRXMAXPn** must be set to an even number of bytes for proper interrupt generation in µDMA Basic mode.

USB Maximum Receive Data Endpoint 1 (USBRXMAXP1)

Base 0x4005.0000
Offset 0x114
Type R/W, reset 0x0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:11 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10:0 | MAXLOAD | R/W | 0x000 | Maximum Payload<br>The maximum payload in bytes per transaction. |

## Register 62: USB Receive Control and Status Endpoint 1 Low (USBRXCSRL1), offset 0x116

## Register 63: USB Receive Control and Status Endpoint 2 Low (USBRXCSRL2), offset 0x126

## Register 64: USB Receive Control and Status Endpoint 3 Low (USBRXCSRL3), offset 0x136

**Host**

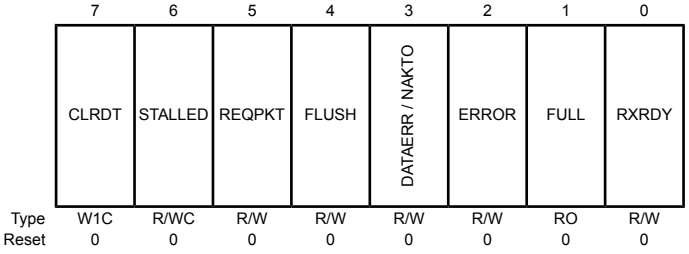
**Device**

**USBRXCSRLn** is an 8-bit register that provides control and status bits for transfers through the currently selected receive endpoint.

### Host Mode

USB Receive Control and Status Endpoint 1 Low (USBRXCSRL1)

Base 0x4005.0000
Offset 0x116
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | CLRDT | W1C | 0 | Clear Data Toggle<br>Writing a 1 to this bit clears the DT bit in the **USBRXCSRHn** register. |
| 6 | STALLED | R/WC | 0 | Endpoint Stalled<br>Value Description<br>0 A STALL handshake has not been received.<br>1 A STALL handshake has been received. The EPn bit in the **USBRXIS** register is also set.<br>Software must clear this bit. |
| 5 | REQPKT | R/W | 0 | Request Packet<br>Value Description<br>0 No request.<br>1 Requests an IN transaction.<br>This bit is cleared when RXRDY is set. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | FLUSH | R/W | 0 | Flush FIFO<br><br>Value Description<br>0 No effect.<br>1 Flushes the next packet to be read from the endpoint receive FIFO. The FIFO pointer is reset and the RXRDY bit is cleared.<br><br>Note that if the FIFO is double-buffered, FLUSH may have to be set twice to completely clear the FIFO.<br><br>**Important:** This bit should only be set when the RXRDY bit is set. At other times, it may cause data to be corrupted. |
| 3 | DATAERR / NAKTO | R/W | 0 | Data Error / NAK Timeout<br><br>Value Description<br>0 Normal operation.<br>1 *Isochronous endpoints only:* Indicates that RXRDY is set and the data packet has a CRC or bit-stuff error. This bit is cleared when RXRDY is cleared.<br>*Bulk endpoints only:* Indicates that the receive endpoint is halted following the receipt of NAK responses for longer than the time set by the NAKLMT field in the **USBRXINTERVALn** register. Software must clear this bit to allow the endpoint to continue. |
| 2 | ERROR | R/W | 0 | Error<br><br>Value Description<br>0 No error.<br>1 Three attempts have been made to receive a packet and no data packet has been received. The EPn bit in the **USBRXIS** register is set in this situation.<br><br>Software must clear this bit.<br><br>**Note:** This bit is only valid when the receive endpoint is operating in Bulk or Interrupt mode. In Isochronous mode, it always returns zero. |
| 1 | FULL | RO | 0 | FIFO Full<br><br>Value Description<br>0 The receive FIFO is not full.<br>1 No more packets can be loaded into the receive FIFO. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | RXRDY | R/W | 0 | Receive Packet Ready |

| Value | Description |
|---|---|
| 0 | No data packet has been received. |
| 1 | A data packet has been received. The EPn bit in the **USBRXIS** register is also set in this situation. |

If the AUTOCLR bit in the **USBRXCSRHn** register is set, then the this bit is automatically cleared when a packet of **USBRXMAXPn** bytes has been unloaded from the receive FIFO. If the AUTOCLR bit is clear, or if packets of less than the maximum packet size are unloaded, then software must clear this bit manually when the packet has been unloaded from the receive FIFO.

## Device Mode

### USB Receive Control and Status Endpoint 1 Low (USBRXCSRL1)

Base 0x4005.0000
Offset 0x116
Type R/W, reset 0x00

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CLRDT | STALLED | STALL | FLUSH | DATAERR | OVER | FULL | RXRDY |
| Type | W1C | R/W | R/W | W1S | RO | R/W | RO | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | CLRDT | W1C | 0 | Clear Data Toggle<br>Writing a 1 to this bit clears the DT bit in the **USBRXCSRHn** register. |
| 6 | STALLED | R/W | 0 | Endpoint Stalled |

| Value | Description |
|---|---|
| 0 | A STALL handshake has not been transmitted. |
| 1 | A STALL handshake has been transmitted. |

Software must clear this bit.

| | | | | |
|---|---|---|---|---|
| 5 | STALL | R/W | 0 | Send STALL |

| Value | Description |
|---|---|
| 0 | No effect. |
| 1 | Issues a STALL handshake. |

Software must clear this bit to terminate the STALL condition.

**Note:** This bit has no effect where the endpoint is being used for isochronous transfers.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | FLUSH | W1S | 0 | Flush FIFO<br><br>Value Description<br>0 No effect.<br>1 Flushes the next packet from the endpoint receive FIFO. The FIFO pointer is reset and the RXRDY bit is cleared.<br><br>The CPU writes a 1 to this bit to flush the next packet to be read from the endpoint receive FIFO. The FIFO pointer is reset and the RXRDY bit is cleared. Note that if the FIFO is double-buffered, FLUSH may have to be set twice to completely clear the FIFO.<br><br>**Important:** This bit should only be set when the RXRDY bit is set. At other times, it may cause data to be corrupted. |
| 3 | DATAERR | RO | 0 | Data Error<br><br>Value Description<br>0 Normal operation.<br>1 Indicates that RXRDY is set and the data packet has a CRC or bit-stuff error.<br><br>This bit is cleared when RXRDY is cleared.<br><br>**Note:** This bit is only valid when the endpoint is operating in Isochronous mode. In Bulk mode, it always returns zero. |
| 2 | OVER | R/W | 0 | Overrun<br><br>Value Description<br>0 No overrun error.<br>1 Indicates that an OUT packet cannot be loaded into the receive FIFO.<br><br>Software must clear this bit.<br><br>**Note:** This bit is only valid when the endpoint is operating in Isochronous mode. In Bulk mode, it always returns zero. |
| 1 | FULL | RO | 0 | FIFO Full<br><br>Value Description<br>0 The receive FIFO is not full.<br>1 No more packets can be loaded into the receive FIFO. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | RXRDY | R/W | 0 | Receive Packet Ready |

| Value | Description |
|---|---|
| 0 | No data packet has been received. |
| 1 | A data packet has been received. The EPn bit in the **USBRXIS** register is also set in this situation. |

If the AUTOCLR bit in the **USBRXCSRHn** register is set, then the this bit is automatically cleared when a packet of **USBRXMAXPn** bytes has been unloaded from the receive FIFO. If the AUTOCLR bit is clear, or if packets of less than the maximum packet size are unloaded, then software must clear this bit manually when the packet has been unloaded from the receive FIFO.

## Register 65: USB Receive Control and Status Endpoint 1 High (USBRXCSRH1), offset 0x117

## Register 66: USB Receive Control and Status Endpoint 2 High (USBRXCSRH2), offset 0x127

## Register 67: USB Receive Control and Status Endpoint 3 High (USBRXCSRH3), offset 0x137

**Host**

**Device**

**USBRXCSRHn** is an 8-bit register that provides additional control and status bits for transfers through the currently selected receive endpoint.

### Host Mode

USB Receive Control and Status Endpoint 1 High (USBRXCSRH1)

Base 0x4005.0000
Offset 0x117
Type R/W, reset 0x00

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | AUTOCL | AUTORQ | DMAEN | PIDERR | DMAMOD | DTWE | DT | reserved |
| Type | R/W | R/W | R/W | RO | R/W | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | AUTOCL | R/W | 0 | Auto Clear |

| Value | Description |
|---|---|
| 0 | No effect. |
| 1 | Enables the RXRDY bit to be automatically cleared when a packet of **USBRXMAXPn** bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, RXRDY must be cleared manually. Care must be taken when using µDMA to unload the receive FIFO as data is read from the receive FIFO in 4 byte chunks regardless of the value of the MAXLOAD field in the **USBRXMAXPn** register, see "DMA Operation" on page 670. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | AUTORQ | R/W | 0 | Auto Request |

| Value | Description |
|---|---|
| 0 | No effect. |
| 1 | Enables the REQPKT bit to be automatically set when the RXRDY bit is cleared. |

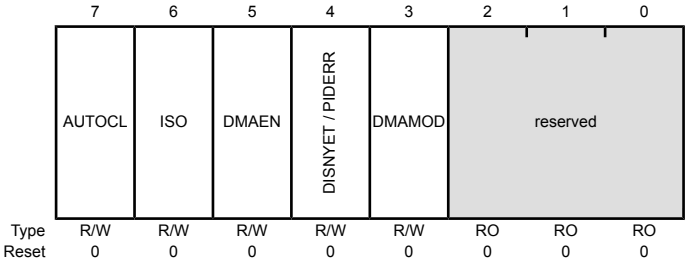
**Note:** This bit is automatically cleared when a short packet is received.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 5 | DMAEN | R/W | 0 | DMA Request Enable<br><br>Value Description<br>0 Disables the µDMA request for the receive endpoint.<br>1 Enables the µDMA request for the receive endpoint.<br><br>**Note:** 3 TX and 3 RX endpoints can be connected to the µDMA module. If this bit is set for a particular endpoint, the DMAARX, DMABRX, or DMACRX field in the **USB DMA Select (USBDMASEL)** register must be programmed correspondingly. |
| 4 | PIDERR | RO | 0 | PID Error<br><br>Value Description<br>0 No error.<br>1 Indicates a PID error in the received packet of an isochronous transaction.<br><br>This bit is ignored in bulk or interrupt transactions. |
| 3 | DMAMOD | R/W | 0 | DMA Request Mode<br><br>Value Description<br>0 An interrupt is generated after every µDMA packet transfer.<br>1 An interrupt is generated only after the entire µDMA transfer is complete.<br><br>**Note:** This bit must not be cleared either before or in the same cycle as the above DMAEN bit is cleared. |
| 2 | DTWE | RO | 0 | Data Toggle Write Enable<br><br>Value Description<br>0 The DT bit cannot be written.<br>1 Enables the current state of the receive endpoint data to be written (see DT bit).<br><br>This bit is automatically cleared once the new value is written. |
| 1 | DT | RO | 0 | Data Toggle<br>When read, this bit indicates the current state of the receive data toggle.<br>If DTWE is High, this bit may be written with the required setting of the data toggle. If DTWE is Low, any value written to this bit is ignored. Care should be taken when writing to this bit as it should only be changed to RESET the receive endpoint. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

### Device Mode

USB Receive Control and Status Endpoint 1 High (USBRXCSRH1)

Base 0x4005.0000
Offset 0x117
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | AUTOCL | R/W | 0 | Auto Clear |

| Value | Description |
|---|---|
| 0 | No effect. |
| 1 | Enables the RXRDY bit to be automatically cleared when a packet of **USBRXMAXPn** bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, RXRDY must be cleared manually. Care must be taken when using µDMA to unload the receive FIFO as data is read from the receive FIFO in 4 byte chunks regardless of the value of the MAXLOAD field in the **USBRXMAXPn** register, see "DMA Operation" on page 670. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | ISO | R/W | 0 | Isochronous Transfers |

| Value | Description |
|---|---|
| 0 | Enables the receive endpoint for isochronous transfers. |
| 1 | Enables the receive endpoint for bulk/interrupt transfers. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 5 | DMAEN | R/W | 0 | DMA Request Enable |

| Value | Description |
|---|---|
| 0 | Disables the µDMA request for the receive endpoint. |
| 1 | Enables the µDMA request for the receive endpoint. |

**Note:** 3 TX and 3 RX endpoints can be connected to the µDMA module. If this bit is set for a particular endpoint, the DMAARX, DMABRX, or DMACRX field in the **USB DMA Select (USBDMASEL)** register must be programmed correspondingly.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | DISNYET / PIDERR | R/W | 0 | Disable NYET / PID Error<br><br>Value Description<br>0 No effect.<br>1 *For bulk or interrupt transactions:* Disables the sending of NYET handshakes. When this bit is set, all successfully received packets are acknowledged, including at the point at which the FIFO becomes full.<br>*For isochronous transactions:* Indicates a PID error in the received packet. |
| 3 | DMAMOD | R/W | 0 | DMA Request Mode<br><br>Value Description<br>0 An interrupt is generated after every µDMA packet transfer.<br>1 An interrupt is generated only after the entire µDMA transfer is complete.<br><br>**Note:** This bit must not be cleared either before or in the same cycle as the above DMAEN bit is cleared. |
| 2:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 68: USB Receive Byte Count Endpoint 1 (USBRXCOUNT1), offset 0x118

## Register 69: USB Receive Byte Count Endpoint 2 (USBRXCOUNT2), offset 0x128

## Register 70: USB Receive Byte Count Endpoint 3 (USBRXCOUNT3), offset 0x138

**Host**

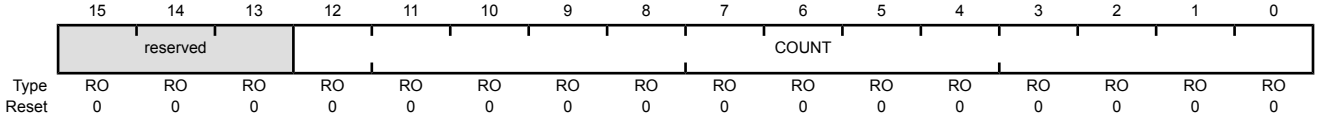
**Note:** The value returned changes as the FIFO is unloaded and is only valid while the RXRDY bit in the **USBRXCSRLn** register is set.

**Device**

**USBRXCOUNTn** is a 16-bit read-only register that holds the number of data bytes in the packet currently in line to be read from the receive FIFO. If the packet is transmitted as multiple bulk packets, the number given is for the combined packet.

USB Receive Byte Count Endpoint 1 (USBRXCOUNT1)

Base 0x4005.0000
Offset 0x118
Type RO, reset 0x0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:13 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12:0 | COUNT | RO | 0x000 | Receive Packet Count<br>Indicates the number of bytes in the receive packet. |

## Register 71: USB Host Transmit Configure Type Endpoint 1 (USBTXTYPE1), offset 0x11A

## Register 72: USB Host Transmit Configure Type Endpoint 2 (USBTXTYPE2), offset 0x12A

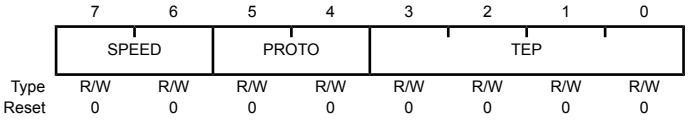
## Register 73: USB Host Transmit Configure Type Endpoint 3 (USBTXTYPE3), offset 0x13A

**Host**

**USBTXTYPEn** is an 8-bit register that must be written with the endpoint number to be targeted by the endpoint, the transaction protocol to use for the currently selected transmit endpoint, and its operating speed.

USB Host Transmit Configure Type Endpoint 1 (USBTXTYPE1)

Base 0x4005.0000
Offset 0x11A
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:6 | SPEED | R/W | 0x0 | Operating Speed<br>This bit field specifies the operating speed of the target Device:<br><br>Value Description<br>0x0 Default<br>The target is assumed to be using the same connection speed as the USB controller.<br>0x1 Reserved<br>0x2 Full<br>0x3 Low |
| 5:4 | PROTO | R/W | 0x0 | Protocol<br>Software must configure this bit field to select the required protocol for the transmit endpoint:<br><br>Value Description<br>0x0 Control<br>0x1 Isochronous<br>0x2 Bulk<br>0x3 Interrupt |
| 3:0 | TEP | R/W | 0x0 | Target Endpoint Number<br>Software must configure this value to the endpoint number contained in the transmit endpoint descriptor returned to the USB controller during Device enumeration. |

## Register 74: USB Host Transmit Interval Endpoint 1 (USBTXINTERVAL1), offset 0x11B

## Register 75: USB Host Transmit Interval Endpoint 2 (USBTXINTERVAL2), offset 0x12B

## Register 76: USB Host Transmit Interval Endpoint 3 (USBTXINTERVAL3), offset 0x13B

**Host**

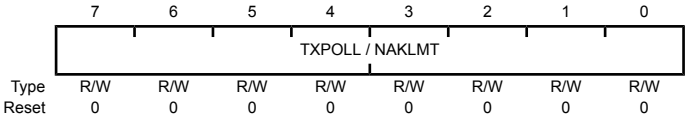
**USBTXINTERVALn** is an 8-bit register that, for interrupt and isochronous transfers, defines the polling interval for the currently selected transmit endpoint. For bulk endpoints, this register defines the number of frames after which the endpoint should time out on receiving a stream of NAK responses.

The **USBTXINTERVALn** register value defines a number of frames, as follows:

| Transfer Type | Speed | Valid values (m) | Interpretation |
|---|---|---|---|
| Interrupt | Low-Speed or Full-Speed | 0x01 – 0xFF | The polling interval is *m* frames. |
| Isochronous | Full-Speed | 0x01 – 0x10 | The polling interval is $2^{(m-1)}$ frames. |
| Bulk | Full-Speed | 0x02 – 0x10 | The NAK Limit is $2^{(m-1)}$ frames. A value of 0 or 1 disables the NAK timeout function. |

USB Host Transmit Interval Endpoint 1 (USBTXINTERVAL1)

Base 0x4005.0000
Offset 0x11B
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:0 | TXPOLL / NAKLMT | R/W | 0x00 | TX Polling / NAK Limit<br>The polling interval for interrupt/isochronous transfers; the NAK limit for bulk transfers. See table above for valid entries; other values are reserved. |

## Register 77: USB Host Configure Receive Type Endpoint 1 (USBRXTYPE1), offset 0x11C

## Register 78: USB Host Configure Receive Type Endpoint 2 (USBRXTYPE2), offset 0x12C

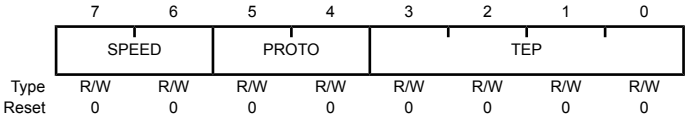
## Register 79: USB Host Configure Receive Type Endpoint 3 (USBRXTYPE3), offset 0x13C

**Host**

**USBRXTYPEn** is an 8-bit register that must be written with the endpoint number to be targeted by the endpoint, the transaction protocol to use for the currently selected receive endpoint, and its operating speed.

USB Host Configure Receive Type Endpoint 1 (USBRXTYPE1)

Base 0x4005.0000
Offset 0x11C
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:6 | SPEED | R/W | 0x0 | Operating Speed<br>This bit field specifies the operating speed of the target Device:<br><br>Value Description<br>0x0 Default<br>The target is assumed to be using the same connection speed as the USB controller.<br>0x1 Reserved<br>0x2 Full<br>0x3 Low |
| 5:4 | PROTO | R/W | 0x0 | Protocol<br>Software must configure this bit field to select the required protocol for the receive endpoint:<br><br>Value Description<br>0x0 Control<br>0x1 Isochronous<br>0x2 Bulk<br>0x3 Interrupt |
| 3:0 | TEP | R/W | 0x0 | Target Endpoint Number<br>Software must set this value to the endpoint number contained in the receive endpoint descriptor returned to the USB controller during Device enumeration. |

## Register 80: USB Host Receive Polling Interval Endpoint 1 (USBRXINTERVAL1), offset 0x11D

## Register 81: USB Host Receive Polling Interval Endpoint 2 (USBRXINTERVAL2), offset 0x12D

## Register 82: USB Host Receive Polling Interval Endpoint 3 (USBRXINTERVAL3), offset 0x13D

**Host**

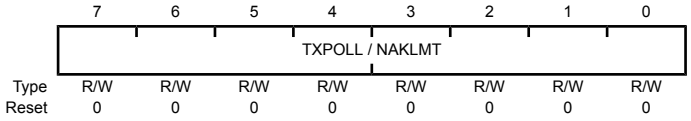
**USBRXINTERVALn** is an 8-bit register that, for interrupt and isochronous transfers, defines the polling interval for the currently selected receive endpoint. For bulk endpoints, this register defines the number of frames after which the endpoint should time out on receiving a stream of NAK responses.

The **USBTXINTERVALn** register value defines a number of frames, as follows:

| Transfer Type | Speed | Valid values (m) | Interpretation |
|---|---|---|---|
| Interrupt | Low-Speed or Full-Speed | 0x01 – 0xFF | The polling interval is *m* frames. |
| Isochronous | Full-Speed | 0x01 – 0x10 | The polling interval is $2^{(m-1)}$ frames. |
| Bulk | Full-Speed | 0x02 – 0x10 | The NAK Limit is $2^{(m-1)}$ frames. A value of 0 or 1 disables the NAK timeout function. |

USB Host Receive Polling Interval Endpoint 1 (USBRXINTERVAL1)

Base 0x4005.0000
Offset 0x11D
Type R/W, reset 0x00




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:0 | TXPOLL / NAKLMT | R/W | 0x00 | RX Polling / NAK Limit<br>The polling interval for interrupt/isochronous transfers; the NAK limit for bulk transfers. See table above for valid entries; other values are reserved. |

## Register 83: USB Request Packet Count in Block Transfer Endpoint 1 (USBRQPKTCOUNT1), offset 0x304

## Register 84: USB Request Packet Count in Block Transfer Endpoint 2 (USBRQPKTCOUNT2), offset 0x308

## Register 85: USB Request Packet Count in Block Transfer Endpoint 3 (USBRQPKTCOUNT3), offset 0x30C

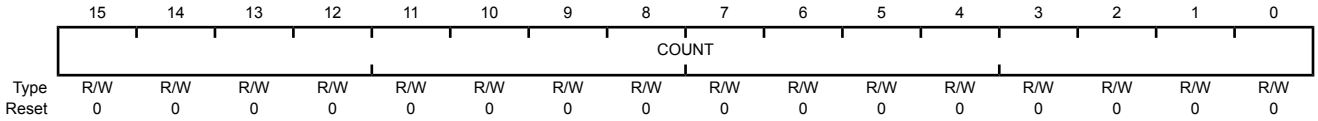
**Host**

This 16-bit read/write register is used in Host mode to specify the number of packets that are to be transferred in a block transfer of one or more bulk packets to receive endpoint n. The USB controller uses the value recorded in this register to determine the number of requests to issue where the AUTORQ bit in the **USBRXCSRHn** register has been set. See “IN Transactions as a Host” on page 667.

**Note:** Multiple packets combined into a single bulk packet within the FIFO count as one packet.

USB Request Packet Count in Block Transfer Endpoint 1 (USBRQPKTCOUNT1)

Base 0x4005.0000
Offset 0x304
Type R/W, reset 0x0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:0 | COUNT | R/W | 0x0000 | Block Transfer Packet Count<br>Sets the number of packets of the size defined by the MAXLOAD bit field that are to be transferred in a block transfer.<br>**Note:** This is only used in Host mode when AUTORQ is set. The bit has no effect in Device mode or when AUTORQ is not set. |

## Register 86: USB Receive Double Packet Buffer Disable (USBRXDPKTBUFDIS), offset 0x340

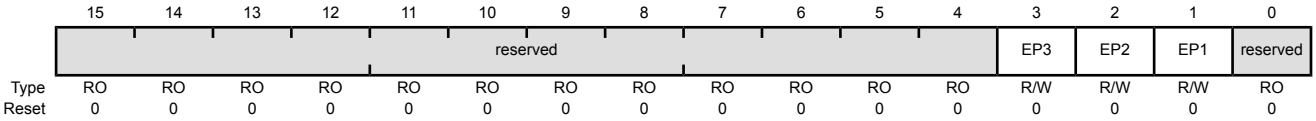
**Host**

**USBRXDPKTBUFDIS** is a 16-bit register that indicates which of the receive endpoints have disabled the double-packet buffer functionality (see the section called “Double-Packet Buffering” on page 663).

**Device**

USB Receive Double Packet Buffer Disable (USBRXDPKTBUFDIS)

Base 0x4005.0000
Offset 0x340
Type R/W, reset 0x0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EP3 | R/W | 0 | EP3 RX Double-Packet Buffer Disable |
| 2 | EP2 | R/W | 0 | EP2 RX Double-Packet Buffer Disable |
| 1 | EP1 | R/W | 0 | EP1 RX Double-Packet Buffer Disable |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 87: USB Transmit Double Packet Buffer Disable (USBTXDPKTBUFDIS), offset 0x342

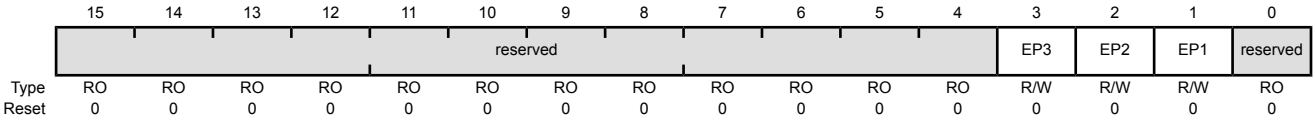
**Host**

**USBTXDPKTBUFDIS** is a 16-bit register that indicates which of the transmit endpoints have disabled the double-packet buffer functionality (see the section called “Double-Packet Buffering” on page 662).

**Device**

USB Transmit Double Packet Buffer Disable (USBTXDPKTBUFDIS)

Base 0x4005.0000
Offset 0x342
Type R/W, reset 0x0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | EP3 | R/W | 0 | EP3 TX Double-Packet Buffer Disable |
| 2 | EP2 | R/W | 0 | EP2 TX Double-Packet Buffer Disable |
| 1 | EP1 | R/W | 0 | EP1 TX Double-Packet Buffer Disable |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 88: USB External Power Control (USBEPC), offset 0x400

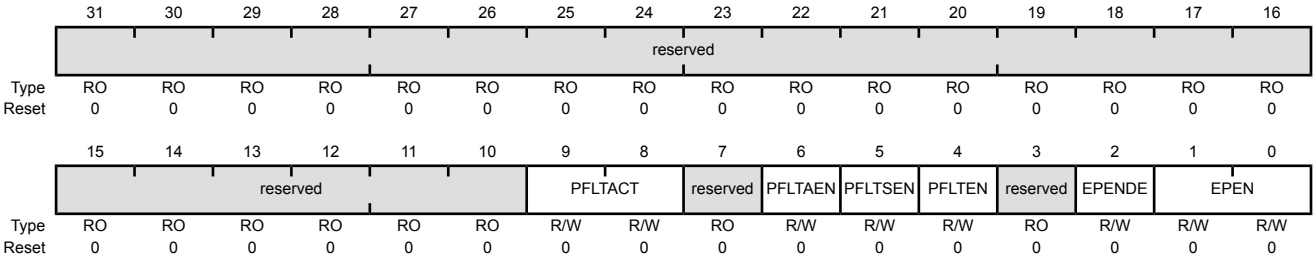
**Host**

**Device**

This 32-bit register specifies the function of the two-pin external power interface (USB0EPEN and USB0PFLT). The assertion of the power fault input may generate an automatic action, as controlled by the hardware configuration registers. The automatic action is necessary because the fault condition may require a response faster than one provided by firmware.

USB External Power Control (USBEPC)

Base 0x4005.0000
Offset 0x400
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | reserved | RO | 0x0000.0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9:8 | PFLTACT | R/W | 0x0 | Power Fault Action<br>This bit field specifies how the USB0EPEN signal is changed when detecting a USB power fault.<br><br>Value Description<br>0x0 Unchanged<br>USB0EPEN is controlled by the combination of the EPEN and EPENDE bits.<br>0x1 Tristate<br>USB0EPEN is undriven (tristate).<br>0x2 Low<br>USB0EPEN is driven Low.<br>0x3 High<br>USB0EPEN is driven High. |
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | PFLTAEN | R/W | 0 | Power Fault Action Enable<br>This bit specifies whether a USB power fault triggers any automatic corrective action regarding the driven state of the USB0EPEN signal.<br><br>Value Description<br>0 Disabled<br>USB0EPEN is controlled by the combination of the EPEN and EPENDE bits.<br>1 Enabled<br>The USB0EPEN output is automatically changed to the state specified by the PFLTACT field. |
| 5 | PFLTSEN | R/W | 0 | Power Fault Sense<br>This bit specifies the logical sense of the USB0PFLT input signal that indicates an error condition.<br>The complementary state is the inactive state.<br><br>Value Description<br>0 Low Fault<br>If USB0PFLT is driven Low, the power fault is signaled internally (if enabled by the PFLTEN bit).<br>1 High Fault<br>If USB0PFLT is driven High, the power fault is signaled internally (if enabled by the PFLTEN bit). |
| 4 | PFLTEN | R/W | 0 | Power Fault Input Enable<br>This bit specifies whether the USB0PFLT input signal is used in internal logic.<br><br>Value Description<br>0 Not Used<br>The USB0PFLT signal is ignored.<br>1 Used<br>The USB0PFLT signal is used internally. |
| 3 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | EPENDE | R/W | 0 | EPEN Drive Enable<br>This bit specifies whether the USB0EPEN signal is driven or undriven (tristate). When driven, the signal value is specified by the EPEN field. When not driven, the EPEN field is ignored and the USB0EPEN signal is placed in a high-impedance state.<br><br>Value Description<br>0 Not Driven<br>The USB0EPEN signal is high impedance.<br>1 Driven<br>The USB0EPEN signal is driven to the logical value specified by the value of the EPEN field.<br><br>The USB0EPEN signal is undriven at reset because the sense of the external power supply enable is unknown. By adding the high-impedance state, system designers may bias the power supply enable to the disabled state using a large resistor (100 kΩ) and later configure and drive the output signal to enable the power supply. |
| 1:0 | EPEN | R/W | 0x0 | External Power Supply Enable Configuration<br>This bit field specifies and controls the logical value driven on the USB0EPEN signal.<br><br>Value Description<br>0x0 Power Enable Active Low<br>The USB0EPEN signal is driven Low if the EPENDE bit is set.<br>0x1 Power Enable Active High<br>The USB0EPEN signal is driven High if the EPENDE bit is set.<br>0x2-0x3 Reserved |

## Register 89: USB External Power Control Raw Interrupt Status (USBEPCRIS), offset 0x404

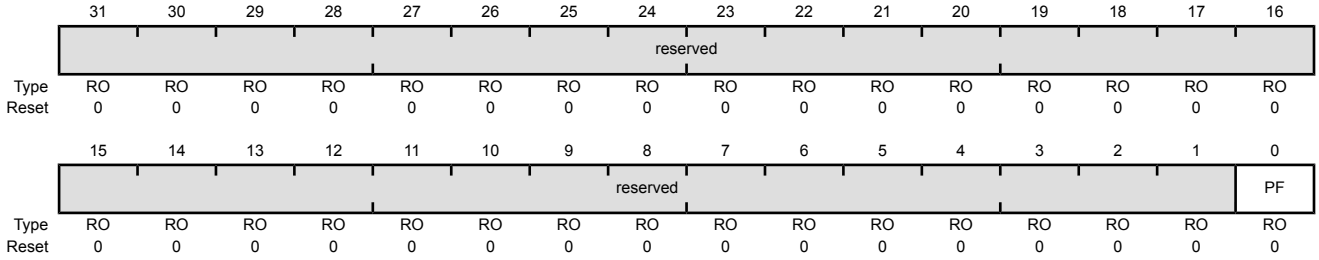
**Host**

This 32-bit register specifies the unmasked interrupt status of the two-pin external power interface.

**Device**

USB External Power Control Raw Interrupt Status (USBEPCRIS)

Base 0x4005.0000
Offset 0x404
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | PF | RO | 0 | USB Power Fault Interrupt Status<br><br>Value Description<br>1 A Power Fault status has been detected.<br>0 An interrupt has not occurred.<br><br>This bit is cleared by writing a 1 to the PF bit in the **USBEPCISC** register. |

## Register 90: USB External Power Control Interrupt Mask (USBEPCIM), offset 0x408

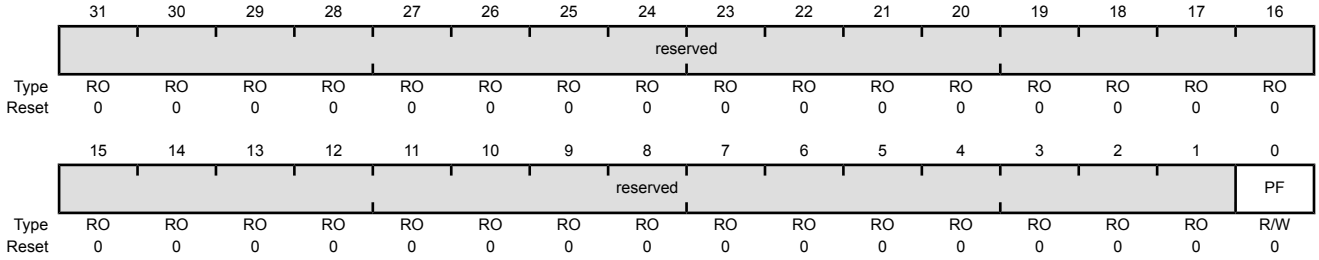
**Host**

This 32-bit register specifies the interrupt mask of the two-pin external power interface.

USB External Power Control Interrupt Mask (USBEPCIM)

**Device**

Base 0x4005.0000
Offset 0x408
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | PF | R/W | 0 | USB Power Fault Interrupt Mask |

| Value | Description |
|---|---|
| 1 | The raw interrupt signal from a detected power fault is sent to the interrupt controller. |
| 0 | A detected power fault does not affect the interrupt status. |

## Register 91: USB External Power Control Interrupt Status and Clear (USBEPCISC), offset 0x40C

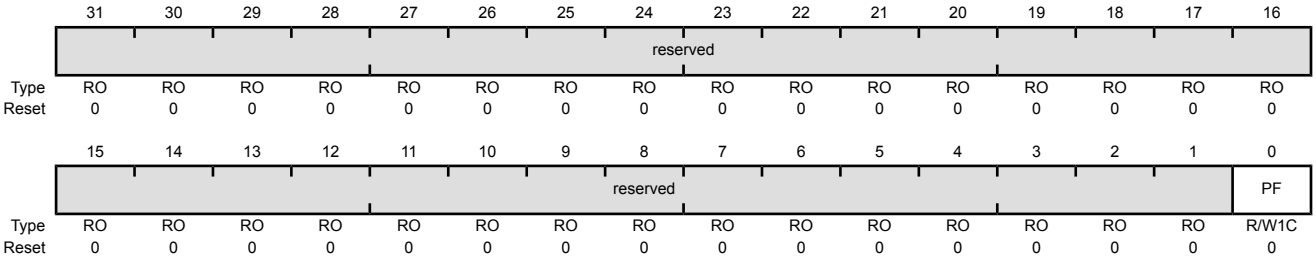
**Host**

This 32-bit register specifies the masked interrupt status of the two-pin external power interface. It also provides a method to clear the interrupt state.

**Device**

USB External Power Control Interrupt Status and Clear (USBEPCISC)

Base 0x4005.0000
Offset 0x40C
Type R/W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | PF | R/W1C | 0 | USB Power Fault Interrupt Status and Clear<br><br>Value Description<br>1 The PF bits in the **USBEPCRIS** and **USBEPCIM** registers are set, providing an interrupt to the interrupt controller.<br>0 No interrupt has occurred or the interrupt is masked.<br><br>This bit is cleared by writing a 1. Clearing this bit also clears the PF bit in the **USBEPCRIS** register. |

## Register 92: USB Device RESUME Raw Interrupt Status (USBDRRIS), offset 0x410

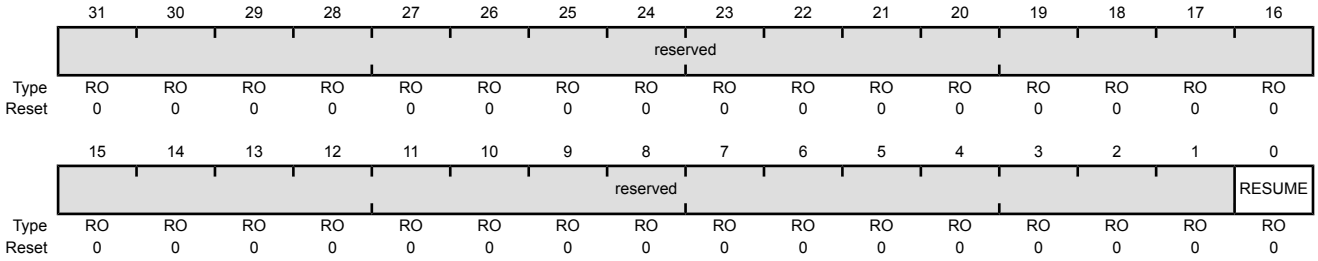
**Host**

The **USBDRRIS** 32-bit register is the raw interrupt status register. On a read, this register gives the current raw status value of the corresponding interrupt prior to masking. A write has no effect.

**Device**

USB Device RESUME Raw Interrupt Status (USBDRRIS)

Base 0x4005.0000
Offset 0x410
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | RESUME | RO | 0 | RESUME Interrupt Status<br><br>Value Description<br>1 A RESUME status has been detected.<br>0 An interrupt has not occurred.<br><br>This bit is cleared by writing a 1 to the RESUME bit in the **USBDRISC** register. |

## Register 93: USB Device RESUME Interrupt Mask (USBDRIM), offset 0x414

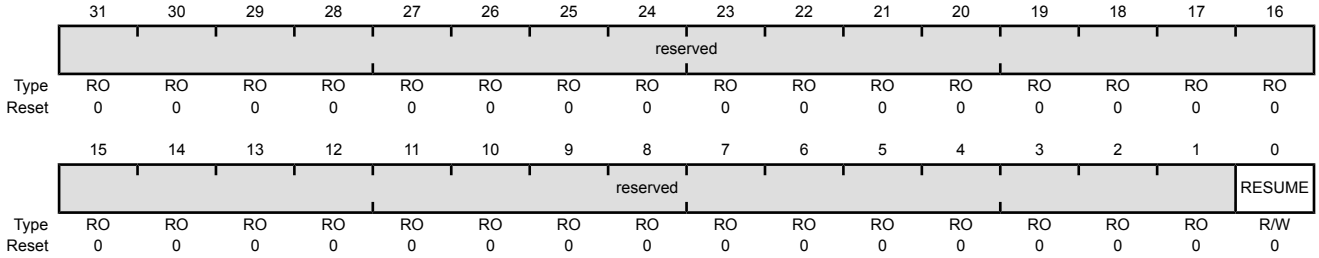
**Host**

**Device**

The **USBDRIM** 32-bit register is the masked interrupt status register. On a read, this register gives the current value of the mask on the corresponding interrupt. Setting a bit sets the mask, preventing the interrupt from being signaled to the interrupt controller. Clearing a bit clears the corresponding mask, enabling the interrupt to be sent to the interrupt controller.

USB Device RESUME Interrupt Mask (USBDRIM)

Base 0x4005.0000
Offset 0x414
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | RESUME | R/W | 0 | RESUME Interrupt Mask |

| Value | Description |
|---|---|
| 1 | The raw interrupt signal from a detected RESUME is sent to the interrupt controller. This bit should only be set when a SUSPEND has been detected (the SUSPEND bit in the **USBIS** register is set). |
| 0 | A detected RESUME does not affect the interrupt status. |

## Register 94: USB Device RESUME Interrupt Status and Clear (USBDRISC), offset 0x418

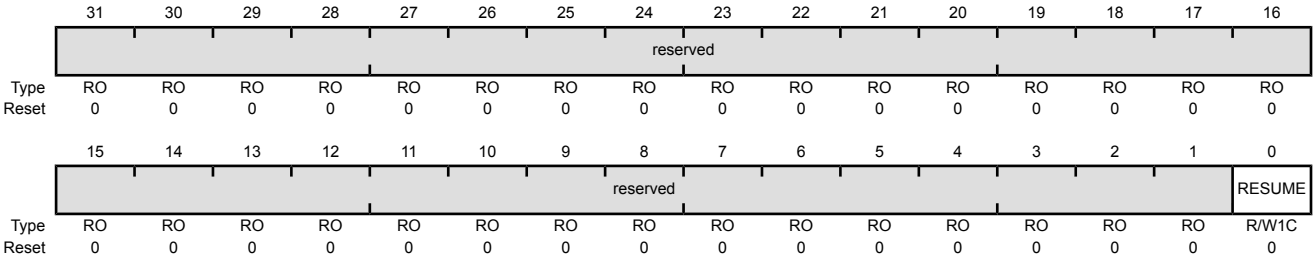
**Host**

The **USBDRISC** 32-bit register is the interrupt clear register. On a write of 1, the corresponding interrupt is cleared. A write of 0 has no effect.

**Device**

USB Device RESUME Interrupt Status and Clear (USBDRISC)

Base 0x4005.0000
Offset 0x418
Type R/W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | RESUME | R/W1C | 0 | RESUME Interrupt Status and Clear<br><br>Value Description<br>1 The RESUME bits in the **USBDRRIS** and **USBDRCIM** registers are set, providing an interrupt to the interrupt controller.<br>0 No interrupt has occurred or the interrupt is masked.<br><br>This bit is cleared by writing a 1. Clearing this bit also clears the RESUME bit in the **USBDRCRIS** register. |

## Register 95: USB General-Purpose Control and Status (USBGPCS), offset 0x41C

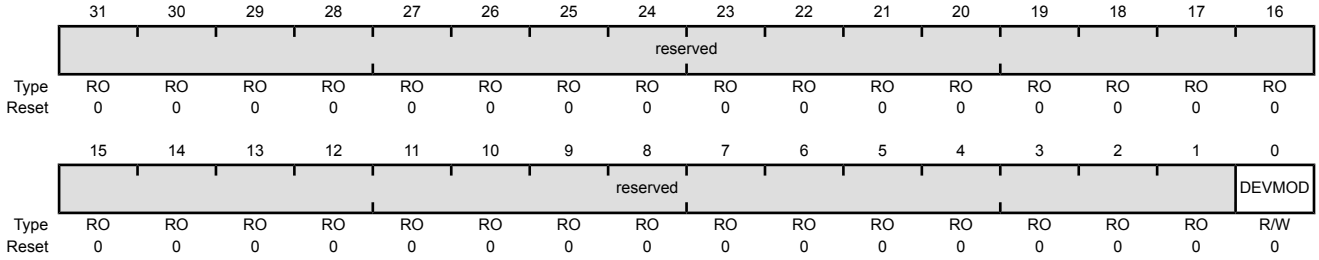
**Host**

**USBGPCS** provides the state of the internal ID signal.

USB General-Purpose Control and Status (USBGPCS)

**Device**

Base 0x4005.0000
Offset 0x41C
Type R/W, reset 0x0000.0000

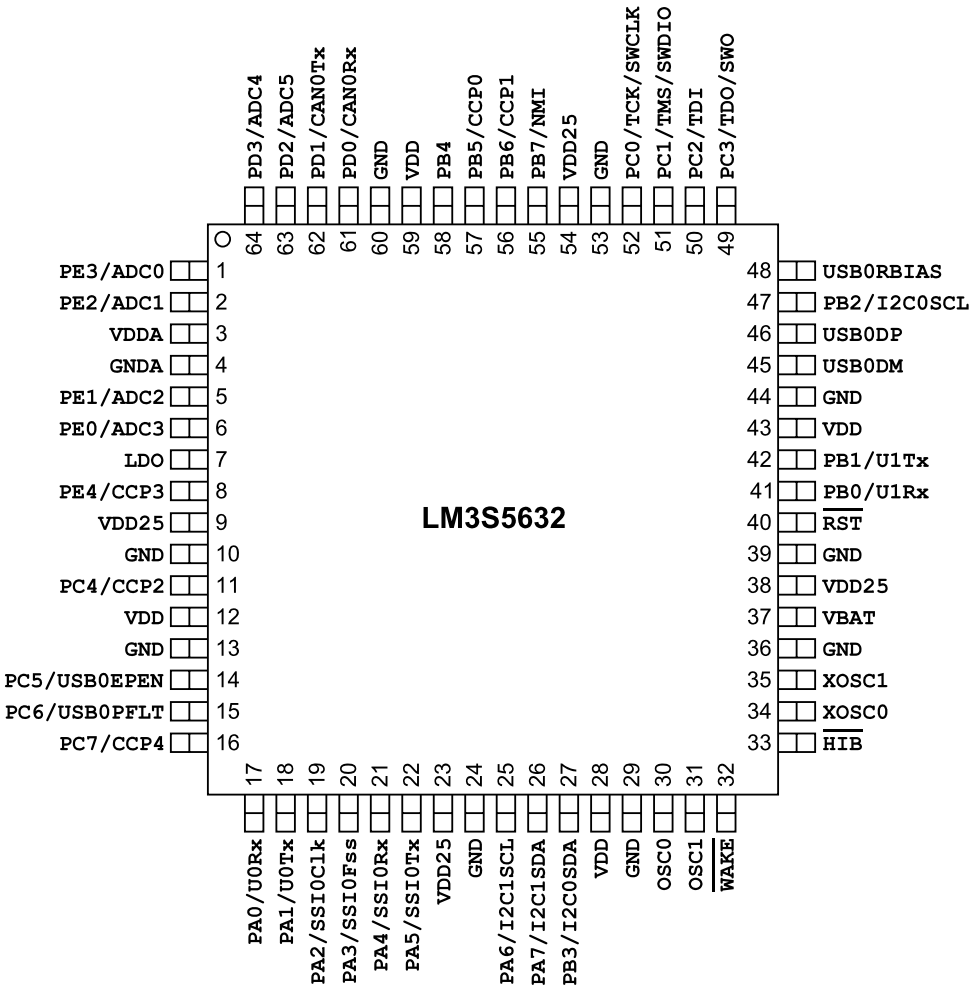



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DEVMOD | R/W | 0 | Device Mode<br>This bit is used to control the state of the internal ID signal.<br>In Device mode this bit is ignored (assumed set).<br><br>Value Description<br>0 Host mode<br>1 Device mode |

# 18 Pin Diagram

The LM3S5632 microcontroller pin diagram is shown below.

**Figure 18-1. 64-Pin LQFP Package Pin Diagram**



# 19 Signal Tables

**Important:** All multiplexed pins are GPIOs by default, with the exception of the four JTAG pins (PC[3:0]) which default to the JTAG functionality.

The following tables list the signals available for each pin. Functionality is enabled by software with the **GPIOAFSEL** register. All digital inputs are Schmitt triggered.

- Signals by Pin Number
- Signals by Signal Name
- Signals by Function, Except for GPIO
- GPIO Pins and Alternate Functions
- Connections for Unused Signals

## 19.1 Signals by Pin Number

**Table 19-1. Signals by Pin Number**

| Pin Number | Pin Name | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| 1 | PE3 | I/O | TTL | GPIO port E bit 3. |
| | ADC0 | I | Analog | Analog-to-digital converter input 0. |
| 2 | PE2 | I/O | TTL | GPIO port E bit 2. |
| | ADC1 | I | Analog | Analog-to-digital converter input 1. |
| 3 | VDDA | - | Power | The positive supply for the analog circuits (ADC, Analog Comparators, etc.). These are separated from VDD to minimize the electrical noise contained on VDD from affecting the analog functions. VDDA pins must be supplied with a voltage that meets the specification in “Recommended DC Operating Conditions” on page 766, regardless of system implementation. |
| 4 | GNDA | - | Power | The ground reference for the analog circuits (ADC, etc.). These are separated from GND to minimize the electrical noise contained on VDD from affecting the analog functions. |
| 5 | PE1 | I/O | TTL | GPIO port E bit 1. |
| | ADC2 | I | Analog | Analog-to-digital converter input 2. |
| 6 | PE0 | I/O | TTL | GPIO port E bit 0. |
| | ADC3 | I | Analog | Analog-to-digital converter input 3. |
| 7 | LDO | - | Power | Low drop-out regulator output voltage. This pin requires an external capacitor between the pin and GND of 1 µF or greater. The LDO pin must also be connected to the VDD25 pins at the board level in addition to the decoupling capacitor(s). |
| 8 | PE4 | I/O | TTL | GPIO port E bit 4. |
| | CCP3 | I/O | TTL | Capture/Compare/PWM 3. |
| 9 | VDD25 | - | Power | Positive supply for most of the logic function, including the processor core and most peripherals. |
| 10 | GND | - | Power | Ground reference for logic and I/O pins. |
| 11 | PC4 | I/O | TTL | GPIO port C bit 4. |
| | CCP2 | I/O | TTL | Capture/Compare/PWM 2. |
| 12 | VDD | - | Power | Positive supply for I/O and some logic. |
| 13 | GND | - | Power | Ground reference for logic and I/O pins. |

**Table 19-1. Signals by Pin Number *(continued)***

| Pin Number | Pin Name | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| 14 | PC5 | I/O | TTL | GPIO port C bit 5. |
| | USB0EPEN | O | TTL | Optionally used in Host mode to control an external power source to supply power to the USB bus. |
| 15 | PC6 | I/O | TTL | GPIO port C bit 6. |
| | USB0PFLT | I | TTL | Optionally used in Host mode by an external power source to indicate an error state by that power source. |
| 16 | PC7 | I/O | TTL | GPIO port C bit 7. |
| | CCP4 | I/O | TTL | Capture/Compare/PWM 4. |
| 17 | PA0 | I/O | TTL | GPIO port A bit 0. |
| | U0Rx | I | TTL | UART module 0 receive. When in IrDA mode, this signal has IrDA modulation. |
| 18 | PA1 | I/O | TTL | GPIO port A bit 1. |
| | U0Tx | O | TTL | UART module 0 transmit. When in IrDA mode, this signal has IrDA modulation. |
| 19 | PA2 | I/O | TTL | GPIO port A bit 2. |
| | SSI0Clk | I/O | TTL | SSI module 0 clock. |
| 20 | PA3 | I/O | TTL | GPIO port A bit 3. |
| | SSI0Fss | I/O | TTL | SSI module 0 frame signal. |
| 21 | PA4 | I/O | TTL | GPIO port A bit 4. |
| | SSI0Rx | I | TTL | SSI module 0 receive. |
| 22 | PA5 | I/O | TTL | GPIO port A bit 5. |
| | SSI0Tx | O | TTL | SSI module 0 transmit. |
| 23 | VDD25 | - | Power | Positive supply for most of the logic function, including the processor core and most peripherals. |
| 24 | GND | - | Power | Ground reference for logic and I/O pins. |
| 25 | PA6 | I/O | TTL | GPIO port A bit 6. |
| | I2C1SCL | I/O | OD | $I^2C$ module 1 clock. |
| 26 | PA7 | I/O | TTL | GPIO port A bit 7. |
| | I2C1SDA | I/O | OD | $I^2C$ module 1 data. |
| 27 | PB3 | I/O | TTL | GPIO port B bit 3. |
| | I2C0SDA | I/O | OD | $I^2C$ module 0 data. |
| 28 | VDD | - | Power | Positive supply for I/O and some logic. |
| 29 | GND | - | Power | Ground reference for logic and I/O pins. |
| 30 | OSC0 | I | Analog | Main oscillator crystal input or an external clock reference input. |
| 31 | OSC1 | O | Analog | Main oscillator crystal output. Leave unconnected when using a single-ended clock source. |
| 32 | WAKE | I | TTL | An external input that brings the processor out of Hibernate mode when asserted. |
| 33 | HIB | O | OD | An output that indicates the processor is in Hibernate mode. |
| 34 | XOSC0 | I | Analog | Hibernation module oscillator crystal input or an external clock reference input. Note that this is either a crystal or a 32.768-kHz oscillator for the Hibernation module RTC. |
| 35 | XOSC1 | O | Analog | Hibernation module oscillator crystal output. Leave unconnected when using a single-ended clock source. |

**Table 19-1. Signals by Pin Number *(continued)***

| Pin Number | Pin Name | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| 36 | GND | - | Power | Ground reference for logic and I/O pins. |
| 37 | VBAT | - | Power | Power source for the Hibernation module. It is normally connected to the positive terminal of a battery and serves as the battery backup/Hibernation module power-source supply. |
| 38 | VDD25 | - | Power | Positive supply for most of the logic function, including the processor core and most peripherals. |
| 39 | GND | - | Power | Ground reference for logic and I/O pins. |
| 40 | $\overline{\text{RST}}$ | I | TTL | System reset input. |
| 41 | PB0 | I/O | TTL | GPIO port B bit 0. |
| | U1Rx | I | TTL | UART module 1 receive. When in IrDA mode, this signal has IrDA modulation. |
| 42 | PB1 | I/O | TTL | GPIO port B bit 1. |
| | U1Tx | O | TTL | UART module 1 transmit. When in IrDA mode, this signal has IrDA modulation. |
| 43 | VDD | - | Power | Positive supply for I/O and some logic. |
| 44 | GND | - | Power | Ground reference for logic and I/O pins. |
| 45 | USB0DM | I/O | Analog | Bidirectional differential data pin (D- per USB specification) for USB0. |
| 46 | USB0DP | I/O | Analog | Bidirectional differential data pin (D+ per USB specification) for USB0. |
| 47 | PB2 | I/O | TTL | GPIO port B bit 2. |
| | I2C0SCL | I/O | OD | $I^2C$ module 0 clock. |
| 48 | USB0RBIAS | O | Analog | 9.1-kΩ resistor (1% precision) used internally for USB analog circuitry. |
| 49 | PC3 | I/O | TTL | GPIO port C bit 3. |
| | SWO | O | TTL | JTAG TDO and SWO. |
| | TDO | O | TTL | JTAG TDO and SWO. |
| 50 | PC2 | I/O | TTL | GPIO port C bit 2. |
| | TDI | I | TTL | JTAG TDI. |
| 51 | PC1 | I/O | TTL | GPIO port C bit 1. |
| | SWDIO | I/O | TTL | JTAG TMS and SWDIO. |
| | TMS | I/O | TTL | JTAG TMS and SWDIO. |
| 52 | PC0 | I/O | TTL | GPIO port C bit 0. |
| | SWCLK | I | TTL | JTAG/SWD CLK. |
| | TCK | I | TTL | JTAG/SWD CLK. |
| 53 | GND | - | Power | Ground reference for logic and I/O pins. |
| 54 | VDD25 | - | Power | Positive supply for most of the logic function, including the processor core and most peripherals. |
| 55 | PB7 | I/O | TTL | GPIO port B bit 7. |
| | NMI | I | TTL | Non-maskable interrupt. |
| 56 | PB6 | I/O | TTL | GPIO port B bit 6. |
| | CCP1 | I/O | TTL | Capture/Compare/PWM 1. |
| 57 | PB5 | I/O | TTL | GPIO port B bit 5. |
| | CCP0 | I/O | TTL | Capture/Compare/PWM 0. |

**Table 19-1. Signals by Pin Number *(continued)***

| Pin Number | Pin Name | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| 58 | PB4 | I/O | TTL | GPIO port B bit 4. |
| 59 | VDD | - | Power | Positive supply for I/O and some logic. |
| 60 | GND | - | Power | Ground reference for logic and I/O pins. |
| 61 | PD0 | I/O | TTL | GPIO port D bit 0. |
| | CAN0Rx | I | Analog | CAN module 0 receive. |
| 62 | PD1 | I/O | TTL | GPIO port D bit 1. |
| | CAN0Tx | O | Analog | CAN module 0 transmit. |
| 63 | PD2 | I/O | TTL | GPIO port D bit 2. |
| | ADC5 | I | Analog | Analog-to-digital converter input 5. |
| 64 | PD3 | I/O | TTL | GPIO port D bit 3. |
| | ADC4 | I | Analog | Analog-to-digital converter input 4. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 19.2 Signals by Signal Name

**Table 19-2. Signals by Signal Name**

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| ADC0 | 1 | I | Analog | Analog-to-digital converter input 0. |
| ADC1 | 2 | I | Analog | Analog-to-digital converter input 1. |
| ADC2 | 5 | I | Analog | Analog-to-digital converter input 2. |
| ADC3 | 6 | I | Analog | Analog-to-digital converter input 3. |
| ADC4 | 64 | I | Analog | Analog-to-digital converter input 4. |
| ADC5 | 63 | I | Analog | Analog-to-digital converter input 5. |
| CAN0Rx | 61 | I | Analog | CAN module 0 receive. |
| CAN0Tx | 62 | O | Analog | CAN module 0 transmit. |
| CCP0 | 57 | I/O | TTL | Capture/Compare/PWM 0. |
| CCP1 | 56 | I/O | TTL | Capture/Compare/PWM 1. |
| CCP2 | 11 | I/O | TTL | Capture/Compare/PWM 2. |
| CCP3 | 8 | I/O | TTL | Capture/Compare/PWM 3. |
| CCP4 | 16 | I/O | TTL | Capture/Compare/PWM 4. |
| GND | 10<br>13<br>24<br>29<br>36<br>39<br>44<br>53<br>60 | - | Power | Ground reference for logic and I/O pins. |
| GNDA | 4 | - | Power | The ground reference for the analog circuits (ADC, etc.). These are separated from GND to minimize the electrical noise contained on VDD from affecting the analog functions. |
| $\overline{\text{HIB}}$ | 33 | O | OD | An output that indicates the processor is in Hibernate mode. |
| I2C0SCL | 47 | I/O | OD | $I^2C$ module 0 clock. |

**Table 19-2. Signals by Signal Name *(continued)***

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| I2C0SDA | 27 | I/O | OD | $I^2C$ module 0 data. |
| I2C1SCL | 25 | I/O | OD | $I^2C$ module 1 clock. |
| I2C1SDA | 26 | I/O | OD | $I^2C$ module 1 data. |
| LDO | 7 | - | Power | Low drop-out regulator output voltage. This pin requires an external capacitor between the pin and GND of 1 µF or greater. The LDO pin must also be connected to the VDD25 pins at the board level in addition to the decoupling capacitor(s). |
| NMI | 55 | I | TTL | Non-maskable interrupt. |
| OSC0 | 30 | I | Analog | Main oscillator crystal input or an external clock reference input. |
| OSC1 | 31 | O | Analog | Main oscillator crystal output. Leave unconnected when using a single-ended clock source. |
| PA0 | 17 | I/O | TTL | GPIO port A bit 0. |
| PA1 | 18 | I/O | TTL | GPIO port A bit 1. |
| PA2 | 19 | I/O | TTL | GPIO port A bit 2. |
| PA3 | 20 | I/O | TTL | GPIO port A bit 3. |
| PA4 | 21 | I/O | TTL | GPIO port A bit 4. |
| PA5 | 22 | I/O | TTL | GPIO port A bit 5. |
| PA6 | 25 | I/O | TTL | GPIO port A bit 6. |
| PA7 | 26 | I/O | TTL | GPIO port A bit 7. |
| PB0 | 41 | I/O | TTL | GPIO port B bit 0. |
| PB1 | 42 | I/O | TTL | GPIO port B bit 1. |
| PB2 | 47 | I/O | TTL | GPIO port B bit 2. |
| PB3 | 27 | I/O | TTL | GPIO port B bit 3. |
| PB4 | 58 | I/O | TTL | GPIO port B bit 4. |
| PB5 | 57 | I/O | TTL | GPIO port B bit 5. |
| PB6 | 56 | I/O | TTL | GPIO port B bit 6. |
| PB7 | 55 | I/O | TTL | GPIO port B bit 7. |
| PC0 | 52 | I/O | TTL | GPIO port C bit 0. |
| PC1 | 51 | I/O | TTL | GPIO port C bit 1. |
| PC2 | 50 | I/O | TTL | GPIO port C bit 2. |
| PC3 | 49 | I/O | TTL | GPIO port C bit 3. |
| PC4 | 11 | I/O | TTL | GPIO port C bit 4. |
| PC5 | 14 | I/O | TTL | GPIO port C bit 5. |
| PC6 | 15 | I/O | TTL | GPIO port C bit 6. |
| PC7 | 16 | I/O | TTL | GPIO port C bit 7. |
| PD0 | 61 | I/O | TTL | GPIO port D bit 0. |
| PD1 | 62 | I/O | TTL | GPIO port D bit 1. |
| PD2 | 63 | I/O | TTL | GPIO port D bit 2. |
| PD3 | 64 | I/O | TTL | GPIO port D bit 3. |
| PE0 | 6 | I/O | TTL | GPIO port E bit 0. |
| PE1 | 5 | I/O | TTL | GPIO port E bit 1. |

**Table 19-2. Signals by Signal Name *(continued)***

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| PE2 | 2 | I/O | TTL | GPIO port E bit 2. |
| PE3 | 1 | I/O | TTL | GPIO port E bit 3. |
| PE4 | 8 | I/O | TTL | GPIO port E bit 4. |
| RST | 40 | I | TTL | System reset input. |
| SSI0Clk | 19 | I/O | TTL | SSI module 0 clock. |
| SSI0Fss | 20 | I/O | TTL | SSI module 0 frame signal. |
| SSI0Rx | 21 | I | TTL | SSI module 0 receive. |
| SSI0Tx | 22 | O | TTL | SSI module 0 transmit. |
| SWCLK | 52 | I | TTL | JTAG/SWD CLK. |
| SWDIO | 51 | I/O | TTL | JTAG TMS and SWDIO. |
| SWO | 49 | O | TTL | JTAG TDO and SWO. |
| TCK | 52 | I | TTL | JTAG/SWD CLK. |
| TDI | 50 | I | TTL | JTAG TDI. |
| TDO | 49 | O | TTL | JTAG TDO and SWO. |
| TMS | 51 | I/O | TTL | JTAG TMS and SWDIO. |
| U0Rx | 17 | I | TTL | UART module 0 receive. When in IrDA mode, this signal has IrDA modulation. |
| U0Tx | 18 | O | TTL | UART module 0 transmit. When in IrDA mode, this signal has IrDA modulation. |
| U1Rx | 41 | I | TTL | UART module 1 receive. When in IrDA mode, this signal has IrDA modulation. |
| U1Tx | 42 | O | TTL | UART module 1 transmit. When in IrDA mode, this signal has IrDA modulation. |
| USB0DM | 45 | I/O | Analog | Bidirectional differential data pin (D- per USB specification) for USB0. |
| USB0DP | 46 | I/O | Analog | Bidirectional differential data pin (D+ per USB specification) for USB0. |
| USB0EPEN | 14 | O | TTL | Optionally used in Host mode to control an external power source to supply power to the USB bus. |
| USB0PFLT | 15 | I | TTL | Optionally used in Host mode by an external power source to indicate an error state by that power source. |
| USB0RBIAS | 48 | O | Analog | 9.1-kΩ resistor (1% precision) used internally for USB analog circuitry. |
| VBAT | 37 | - | Power | Power source for the Hibernation module. It is normally connected to the positive terminal of a battery and serves as the battery backup/Hibernation module power-source supply. |
| VDD | 12<br>28<br>43<br>59 | - | Power | Positive supply for I/O and some logic. |
| VDD25 | 9<br>23<br>38<br>54 | - | Power | Positive supply for most of the logic function, including the processor core and most peripherals. |

**Table 19-2. Signals by Signal Name *(continued)***

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| VDDA | 3 | - | Power | The positive supply for the analog circuits (ADC, Analog Comparators, etc.). These are separated from VDD to minimize the electrical noise contained on VDD from affecting the analog functions. VDDA pins must be supplied with a voltage that meets the specification in "Recommended DC Operating Conditions" on page 766, regardless of system implementation. |
| WAKE | 32 | I | TTL | An external input that brings the processor out of Hibernate mode when asserted. |
| XOSC0 | 34 | I | Analog | Hibernation module oscillator crystal input or an external clock reference input. Note that this is either a crystal or a 32.768-kHz oscillator for the Hibernation module RTC. |
| XOSC1 | 35 | O | Analog | Hibernation module oscillator crystal output. Leave unconnected when using a single-ended clock source. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

## 19.3 Signals by Function, Except for GPIO

**Table 19-3. Signals by Function, Except for GPIO**

| Function | Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| ADC | ADC0 | 1 | I | Analog | Analog-to-digital converter input 0. |
| | ADC1 | 2 | I | Analog | Analog-to-digital converter input 1. |
| | ADC2 | 5 | I | Analog | Analog-to-digital converter input 2. |
| | ADC3 | 6 | I | Analog | Analog-to-digital converter input 3. |
| | ADC4 | 64 | I | Analog | Analog-to-digital converter input 4. |
| | ADC5 | 63 | I | Analog | Analog-to-digital converter input 5. |
| Controller Area Network | CAN0Rx | 61 | I | Analog | CAN module 0 receive. |
| | CAN0Tx | 62 | O | Analog | CAN module 0 transmit. |
| General-Purpose Timers | CCP0 | 57 | I/O | TTL | Capture/Compare/PWM 0. |
| | CCP1 | 56 | I/O | TTL | Capture/Compare/PWM 1. |
| | CCP2 | 11 | I/O | TTL | Capture/Compare/PWM 2. |
| | CCP3 | 8 | I/O | TTL | Capture/Compare/PWM 3. |
| | CCP4 | 16 | I/O | TTL | Capture/Compare/PWM 4. |
| Hibernate | HIB | 33 | O | OD | An output that indicates the processor is in Hibernate mode. |
| | VBAT | 37 | - | Power | Power source for the Hibernation module. It is normally connected to the positive terminal of a battery and serves as the battery backup/Hibernation module power-source supply. |
| | WAKE | 32 | I | TTL | An external input that brings the processor out of Hibernate mode when asserted. |
| | XOSC0 | 34 | I | Analog | Hibernation module oscillator crystal input or an external clock reference input. Note that this is either a crystal or a 32.768-kHz oscillator for the Hibernation module RTC. |
| | XOSC1 | 35 | O | Analog | Hibernation module oscillator crystal output. Leave unconnected when using a single-ended clock source. |

**Table 19-3. Signals by Function, Except for GPIO *(continued)***

| Function | Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| I2C | I2C0SCL | 47 | I/O | OD | $I^2C$ module 0 clock. |
| | I2C0SDA | 27 | I/O | OD | $I^2C$ module 0 data. |
| | I2C1SCL | 25 | I/O | OD | $I^2C$ module 1 clock. |
| | I2C1SDA | 26 | I/O | OD | $I^2C$ module 1 data. |
| JTAG/SWD/SWO | SWCLK | 52 | I | TTL | JTAG/SWD CLK. |
| | SWDIO | 51 | I/O | TTL | JTAG TMS and SWDIO. |
| | SWO | 49 | O | TTL | JTAG TDO and SWO. |
| | TCK | 52 | I | TTL | JTAG/SWD CLK. |
| | TDI | 50 | I | TTL | JTAG TDI. |
| | TDO | 49 | O | TTL | JTAG TDO and SWO. |
| | TMS | 51 | I/O | TTL | JTAG TMS and SWDIO. |
| Power | GND | 10<br>13<br>24<br>29<br>36<br>39<br>44<br>53<br>60 | - | Power | Ground reference for logic and I/O pins. |
| | GNDA | 4 | - | Power | The ground reference for the analog circuits (ADC, etc.). These are separated from GND to minimize the electrical noise contained on VDD from affecting the analog functions. |
| | LDO | 7 | - | Power | Low drop-out regulator output voltage. This pin requires an external capacitor between the pin and GND of 1 µF or greater. The LDO pin must also be connected to the VDD25 pins at the board level in addition to the decoupling capacitor(s). |
| | VDD | 12<br>28<br>43<br>59 | - | Power | Positive supply for I/O and some logic. |
| | VDD25 | 9<br>23<br>38<br>54 | - | Power | Positive supply for most of the logic function, including the processor core and most peripherals. |
| | VDDA | 3 | - | Power | The positive supply for the analog circuits (ADC, Analog Comparators, etc.). These are separated from VDD to minimize the electrical noise contained on VDD from affecting the analog functions. VDDA pins must be supplied with a voltage that meets the specification in “Recommended DC Operating Conditions” on page 766, regardless of system implementation. |
| SSI | SSI0Clk | 19 | I/O | TTL | SSI module 0 clock. |
| | SSI0Fss | 20 | I/O | TTL | SSI module 0 frame signal. |
| | SSI0Rx | 21 | I | TTL | SSI module 0 receive. |
| | SSI0Tx | 22 | O | TTL | SSI module 0 transmit. |

**Table 19-3. Signals by Function, Except for GPIO *(continued)***

| Function | Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| System Control & Clocks | NMI | 55 | I | TTL | Non-maskable interrupt. |
| | OSC0 | 30 | I | Analog | Main oscillator crystal input or an external clock reference input. |
| | OSC1 | 31 | O | Analog | Main oscillator crystal output. Leave unconnected when using a single-ended clock source. |
| | RST | 40 | I | TTL | System reset input. |
| UART | U0Rx | 17 | I | TTL | UART module 0 receive. When in IrDA mode, this signal has IrDA modulation. |
| | U0Tx | 18 | O | TTL | UART module 0 transmit. When in IrDA mode, this signal has IrDA modulation. |
| | U1Rx | 41 | I | TTL | UART module 1 receive. When in IrDA mode, this signal has IrDA modulation. |
| | U1Tx | 42 | O | TTL | UART module 1 transmit. When in IrDA mode, this signal has IrDA modulation. |
| USB | USB0DM | 45 | I/O | Analog | Bidirectional differential data pin (D- per USB specification) for USB0. |
| | USB0DP | 46 | I/O | Analog | Bidirectional differential data pin (D+ per USB specification) for USB0. |
| | USB0EPEN | 14 | O | TTL | Optionally used in Host mode to control an external power source to supply power to the USB bus. |
| | USB0PFLT | 15 | I | TTL | Optionally used in Host mode by an external power source to indicate an error state by that power source. |
| | USB0RBIAS | 48 | O | Analog | 9.1-kΩ resistor (1% precision) used internally for USB analog circuitry. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

# 19.4 GPIO Pins and Alternate Functions

**Table 19-4. GPIO Pins and Alternate Functions**

| IO | Pin Number | Multiplexed Function | Multiplexed Function |
|---|---|---|---|
| PA0 | 17 | U0Rx | |
| PA1 | 18 | U0Tx | |
| PA2 | 19 | SSI0Clk | |
| PA3 | 20 | SSI0Fss | |
| PA4 | 21 | SSI0Rx | |
| PA5 | 22 | SSI0Tx | |
| PA6 | 25 | I2C1SCL | |
| PA7 | 26 | I2C1SDA | |
| PB0 | 41 | U1Rx | |
| PB1 | 42 | U1Tx | |
| PB2 | 47 | I2C0SCL | |
| PB3 | 27 | I2C0SDA | |
| PB4 | 58 | | |
| PB5 | 57 | CCP0 | |

**Table 19-4. GPIO Pins and Alternate Functions *(continued)***

| IO | Pin Number | Multiplexed Function | Multiplexed Function |
|---|---|---|---|
| PB6 | 56 | CCP1 | |
| PB7 | 55 | NMI | |
| PC0 | 52 | TCK | SWCLK |
| PC1 | 51 | TMS | SWDIO |
| PC2 | 50 | TDI | |
| PC3 | 49 | TDO | SWO |
| PC4 | 11 | CCP2 | |
| PC5 | 14 | USB0EPEN | |
| PC6 | 15 | USB0PFLT | |
| PC7 | 16 | CCP4 | |
| PD0 | 61 | CAN0Rx | |
| PD1 | 62 | CAN0Tx | |
| PD2 | 63 | ADC5 | |
| PD3 | 64 | ADC4 | |
| PE0 | 6 | ADC3 | |
| PE1 | 5 | ADC2 | |
| PE2 | 2 | ADC1 | |
| PE3 | 1 | ADC0 | |
| PE4 | 8 | CCP3 | |

## 19.5 Connections for Unused Signals

Table 19-5 on page 763 shows how to handle signals for functions that are not used in a particular system implementation for devices that are in a 64-pin LQFP package. Two options are shown in the table: an acceptable practice and a preferred practice for reduced power consumption and improved EMC characteristics. If a module is not used in a system, and its inputs are grounded, it is important that the clock to the module is never enabled by setting the corresponding bit in the **RCGCx** register.

**Table 19-5. Connections for Unused Signals (64-pin LQFP)**

| Function | Signal Name | Pin Number | Acceptable Practice | Preferred Practice |
|---|---|---|---|---|
| GPIO | All unused GPIOs | - | NC | GND |
| Hibernate | $\overline{\text{HIB}}$ | 33 | NC | NC |
| | VBAT | 37 | NC | GND |
| | $\overline{\text{WAKE}}$ | 32 | NC | GND |
| | XOSC0 | 34 | NC | GND |
| | XOSC1 | 35 | NC | NC |
| No Connects | NC | - | NC | NC |
| System Control | OSC0 | 30 | NC | GND |
| | OSC1 | 31 | NC | NC |
| | $\overline{\text{RST}}$ | 40 | Pull up as shown in Figure 5-1 on page 173 | Connect through a capacitor to GND as close to pin as possible |

**Table 19-5. Connections for Unused Signals (64-pin LQFP) *(continued)***

| Function | Signal Name | Pin Number | Acceptable Practice | Preferred Practice |
|---|---|---|---|---|
| USB | USB0RBIAS | 48 | Connect to GND through 10-kΩ resistor. | Connect to GND through 10-kΩ resistor. |
| | USB0DM | 45 | NC | GND |
| | USB0DP | 46 | NC | GND |

# 20 Operating Characteristics

### Table 20-1. Temperature Characteristics

| Characteristic | Symbol | Value | Unit |
|---|---|---|---|
| Industrial operating temperature range | $T_A$ | -40 to +85 | °C |
| Unpowered storage temperature range | $T_S$ | -65 to +150 | °C |

### Table 20-2. Thermal Characteristics

| Characteristic | Symbol | Value | Unit |
|---|---|---|---|
| Thermal resistance (junction to ambient)[a] | $\Theta_{JA}$ | 37 | °C/W |
| Junction temperature[b] | $T_J$ | $T_A + (P \bullet \Theta_{JA})$ | °C |

a. Junction to ambient thermal resistance $\theta_{JA}$ numbers are determined by a package simulator.

b. Power dissipation is a function of temperature.

### Table 20-3. ESD Absolute Maximum Ratings[a]

| Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|
| $V_{ESDHBM}$ | - | - | 2.0 | kV |
| $V_{ESDCDM}$ | - | - | 1.0 | kV |
| $V_{ESDMM}$ | - | - | 100 | V |

a. All Stellaris parts are ESD tested following the JEDEC standard.

# 21 Electrical Characteristics

## 21.1 DC Characteristics

### 21.1.1 Maximum Ratings

The maximum ratings are the limits to which the device can be subjected without permanently damaging the device.

**Note:** The device is not guaranteed to operate properly at the maximum ratings.

**Table 21-1. Maximum Ratings**

| Characteristic[a] | Symbol | Value | | Unit |
|---|---|---|---|---|
| | | Min | Max | |
| I/O supply voltage ($V_{DD}$) | $V_{DD}$ | 0 | 4 | V |
| Core supply voltage ($V_{DD25}$) | $V_{DD25}$ | 0 | 3 | V |
| Analog supply voltage ($V_{DDA}$) | $V_{DDA}$ | 0 | 4 | V |
| Battery supply voltage ($V_{BAT}$) | $V_{BAT}$ | 0 | 4 | V |
| Input voltage | $V_{IN}$ | -0.3 | 5.5 | V |
| Input voltage for a GPIO configured as an analog input | | -0.3 | $V_{DD}$ + 0.3 | V |
| Maximum current per output pins | I | - | 25 | mA |
| Maximum input voltage on a non-power pin when the microcontroller is unpowered | $V_{NON}$ | - | 300 | mV |

a. Voltages are measured with respect to GND.

**Important:** This device contains circuitry to protect the inputs against damage due to high-static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are connected to an appropriate logic voltage level (for example, either GND or $V_{DD}$).

### 21.1.2 Recommended DC Operating Conditions

For special high-current applications, the GPIO output buffers may be used with the following restrictions. With the GPIO pins configured as 8-mA output drivers, a total of four GPIO outputs may be used to sink current loads up to 18 mA each. At 18-mA sink current loading, the $V_{OL}$ value is specified as 1.2 V. The high-current GPIO package pins must be selected such that there are only a maximum of two per side of the physical package with the total number of high-current GPIO outputs not exceeding four for the entire package.

**Table 21-2. Recommended DC Operating Conditions**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{DD}$ | I/O supply voltage | 3.0 | 3.3 | 3.6 | V |
| $V_{DD25}$ | Core supply voltage | 2.25 | 2.5 | 2.75 | V |
| $V_{DDA}$ | Analog supply voltage | 3.0 | 3.3 | 3.6 | V |
| $V_{BAT}$ | Battery supply voltage | 2.3 | 3.0 | 3.6 | V |
| $V_{IH}$ | High-level input voltage | 2.0 | - | 5.0 | V |
| $V_{IL}$ | Low-level input voltage | -0.3 | - | 1.3 | V |

**Table 21-2. Recommended DC Operating Conditions *(continued)***

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{OH}$[a] | High-level output voltage | 2.4 | - | - | V |
| $V_{OL}$[a] | Low-level output voltage | - | - | 0.4 | V |
| $I_{OH}$ | High-level source current, $V_{OH}$=2.4 V | | | | |
| | 2-mA Drive | 2.0 | - | - | mA |
| | 4-mA Drive | 4.0 | - | - | mA |
| | 8-mA Drive | 8.0 | - | - | mA |
| $I_{OL}$ | Low-level sink current, $V_{OL}$=0.4 V | | | | |
| | 2-mA Drive | 2.0 | - | - | mA |
| | 4-mA Drive | 4.0 | - | - | mA |
| | 8-mA Drive | 8.0 | - | - | mA |

a. $V_{OL}$ and $V_{OH}$ shift to 1.2 V when using high-current GPIOs.

## 21.1.3 On-Chip Low Drop-Out (LDO) Regulator Characteristics

**Table 21-3. LDO Regulator Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{LDOOUT}$ | Programmable internal (logic) power supply output value | 2.25 | 2.5 | 2.75 | V |
| | Output voltage accuracy | - | 2% | - | % |
| $t_{PON}$ | Power-on time | - | - | 100 | µs |
| $t_{ON}$ | Time on | - | - | 200 | µs |
| $t_{OFF}$ | Time off | - | - | 100 | µs |
| $V_{STEP}$ | Step programming incremental voltage | - | 50 | - | mV |
| $C_{LDO}$ | External filter capacitor size for internal power supply | 1.0 | - | 3.0 | µF |

## 21.1.4 GPIO Module Characteristics

**Table 21-4. GPIO Module DC Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $R_{GPIOPU}$ | GPIO internal pull-up resistor | 50 | - | 110 | kΩ |
| $R_{GPIOPD}$ | GPIO internal pull-down resistor | 55 | - | 180 | kΩ |
| $I_{LKG}$ | GPIO input leakage current[a] | - | - | 2 | µA |

a. The leakage current is measured with GND or $V_{DD}$ applied to the corresponding pin(s). The leakage of digital port pins is measured individually. The port pin is configured as an input and the pullup/pulldown resistor is disabled.

## 21.1.5 Power Specifications

The power measurements specified in the tables that follow are run on the core processor using SRAM with the following specifications (except as noted):

- $V_{DD}$ = 3.3 V
- $V_{DD25}$ = 2.50 V

- $V_{BAT}$ = 3.0 V
- $V_{DDA}$ = 3.3 V
- Temperature = 25°C
- Clock Source (MOSC) =3.579545 MHz Crystal Oscillator
- Main oscillator (MOSC) = enabled
- Internal oscillator (IOSC) = disabled

**Table 21-5. Detailed Power Specifications**

| Parameter | Parameter Name | Conditions | 3.3 V $V_{DD}$, $V_{DDA}$ | | 2.5 V $V_{DD25}$ | | 3.0 V $V_{BAT}$ | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Nom | Max | Nom | Max | Nom | Max | |
| $I_{DD\_RUN}$ | Run mode 1 (Flash loop) | $V_{DD25}$ = 2.50 V<br>Code= while(1){} executed out of Flash<br>Peripherals = All ON<br>System Clock = 50 MHz (with PLL) | 5.8 | pending[a] | 129 | pending[a] | 0 | pending[a] | mA |
| | Run mode 2 (Flash loop) | $V_{DD25}$ = 2.50 V<br>Code= while(1){} executed out of Flash<br>Peripherals = All OFF<br>System Clock = 50 MHz (with PLL) | 3.0 | pending[a] | 56 | pending[a] | 0 | pending[a] | mA |
| | Run mode 1 (SRAM loop) | $V_{DD25}$ = 2.50 V<br>Code= while(1){} executed in SRAM<br>Peripherals = All ON<br>System Clock = 50 MHz (with PLL) | 5.8 | pending[a] | 127 | pending[a] | 0 | pending[a] | mA |
| | Run mode 2 (SRAM loop) | $V_{DD25}$ = 2.50 V<br>Code= while(1){} executed in SRAM<br>Peripherals = All OFF<br>System Clock = 50 MHz (with PLL) | 3.0 | pending[a] | 49 | pending[a] | 0 | pending[a] | mA |
| $I_{DD\_SLEEP}$ | Sleep mode | $V_{DD25}$ = 2.50 V<br>Peripherals = All OFF<br>System Clock = 50 MHz (with PLL) | 3.0 | pending[a] | 22 | pending[a] | 0 | pending[a] | mA |
| $I_{DD\_DEEPSLEEP}$ | Deep-Sleep mode | LDO = 2.25 V<br>Peripherals = All OFF<br>System Clock = IOSC30KHZ/64 | 0.19 | pending[a] | 0.10 | pending[a] | 0 | pending[a] | mA |

**Table 21-5. Detailed Power Specifications *(continued)***

| Parameter | Parameter Name | Conditions | 3.3 V $V_{DD}$, $V_{DDA}$ | | 2.5 V $V_{DD25}$ | | 3.0 V $V_{BAT}$ | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | Nom | Max | Nom | Max | Nom | Max | |
| $I_{DD\_HIBERNATE}$ | Hibernate mode | $V_{BAT}$ = 3.0 V<br>$V_{DD}$ = 0 V<br>$V_{DD25}$ = 0 V<br>$V_{DDA}$ = 0 V<br>Peripherals = All OFF<br>System Clock = OFF<br>Hibernate Module = 32 kHz | 0 | 0 | 0 | 0 | 16 | pending[a] | µA |

a. Pending characterization completion.

## 21.1.6 Flash Memory Characteristics

**Table 21-6. Flash Memory Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $PE_{CYC}$ | Number of guaranteed program/erase cycles before failure[a] | 10,000 | 100,000 | - | cycles |
| $T_{RET}$ | Data retention at average operating temperature of 85˚C | 10 | - | - | years |
| $T_{PROG}$ | Word program time | 20 | - | - | µs |
| $T_{ERASE}$ | Page erase time | 20 | - | - | ms |
| $T_{ME}$ | Mass erase time | - | - | 250 | ms |

a. A program/erase cycle is defined as switching the bits from 1-> 0 -> 1.

## 21.1.7 Hibernation

**Table 21-7. Hibernation Module DC Characteristics**

| Parameter | Parameter Name | Value | Unit |
|---|---|---|---|
| $V_{LOWBAT}$ | Low battery detect voltage | 2.35 | V |
| $R_{WAKEPU}$ | WAKE internal pull-up resistor | 200 | kΩ |

## 21.1.8 USB

The Stellaris® USB controller DC electrical specifications are compliant with the “Universal Serial Bus Specification Rev. 2.0” (full-speed and low-speed support). Some components of the USB system are integrated within the LM3S5632 microcontroller and specific to the Stellaris microcontroller design. These components are specified in Table 21-8 on page 769.

**Table 21-8. USB Controller DC Characteristics**

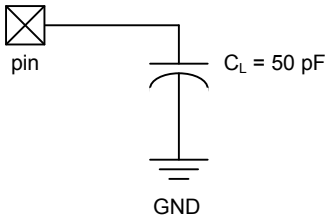
| Parameter | Parameter Name | Value | Unit |
|---|---|---|---|
| $R_{BIAS}$ | Value of the pull-down resistor on the USB0RBIAS pin | 9.1K ± 1 % | Ω |

# 21.2 AC Characteristics

## 21.2.1 Load Conditions

Unless otherwise specified, the following conditions are true for all timing measurements. Timing measurements are for 4-mA drive strength.

**Figure 21-1. Load Conditions**




## 21.2.2 Clocks

**Table 21-9. Phase Locked Loop (PLL) Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $f_{ref\_crystal}$ | Crystal reference[a] | 3.579545 | - | 16.384 | MHz |
| $f_{ref\_ext}$ | External clock reference[a] | 3.579545 | - | 16.384 | MHz |
| $f_{pll}$ | PLL frequency[b] | - | 400 | - | MHz |
| $T_{READY}$ | PLL lock time | - | - | 0.5 | ms |

a. The exact value is determined by the crystal value programmed into the XTAL field of the **Run-Mode Clock Configuration (RCC)** register.

b. PLL frequency is automatically calculated by the hardware based on the XTAL field of the **RCC** register.

Table 21-10 on page 770 shows the actual frequency of the PLL based on the crystal frequency used (defined by the XTAL field in the **RCC** register).

**Table 21-10. Actual PLL Frequency**

| XTAL | Crystal Frequency (MHz) | PLL Frequency (MHz) | Error |
|---|---|---|---|
| 0x04 | 3.5795 | 400.904 | 0.0023% |
| 0x05 | 3.6864 | 398.1312 | 0.0047% |
| 0x06 | 4.0 | 400 | - |
| 0x07 | 4.096 | 401.408 | 0.0035% |
| 0x08 | 4.9152 | 398.1312 | 0.0047% |
| 0x09 | 5.0 | 400 | - |
| 0x0A | 5.12 | 399.36 | 0.0016% |
| 0x0B | 6.0 | 400 | - |
| 0x0C | 6.144 | 399.36 | 0.0016% |
| 0x0D | 7.3728 | 398.1312 | 0.0047% |
| 0x0E | 8.0 | 400 | 0.0047% |
| 0x0F | 8.192 | 398.6773333 | 0.0033% |
| 0x10 | 10.0 | 400 | - |
| 0x11 | 12.0 | 400 | - |

**Table 21-10. Actual PLL Frequency *(continued)***

| XTAL | Crystal Frequency (MHz) | PLL Frequency (MHz) | Error |
|---|---|---|---|
| 0x12 | 12.288 | 401.408 | 0.0035% |
| 0x13 | 13.56 | 397.76 | 0.0056% |
| 0x14 | 14.318 | 400.90904 | 0.0023% |
| 0x15 | 16.0 | 400 | - |
| 0x16 | 16.384 | 404.1386667 | 0.010% |

**Table 21-11. Clock Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $f_{IOSC}$ | Internal 12 MHz oscillator frequency | 8.4 | 12 | 15.6 | MHz |
| $f_{IOSC30KHZ}$ | Internal 30 KHz oscillator frequency | 15 | 30 | 45 | KHz |
| $f_{XOSC}$ | Hibernation module oscillator frequency | - | 4.194304 | - | MHz |
| $f_{XOSC\_XTAL}$ | Crystal reference for hibernation oscillator | - | 4.194304 | - | MHz |
| $f_{XOSC\_EXT}$ | External clock reference for hibernation module | - | 32.768 | - | KHz |
| $f_{MOSC}$ | Main oscillator frequency | 1 | - | 16.384 | MHz |
| $t_{MOSC\_per}$ | Main oscillator period | 61 | - | 1000 | ns |
| $f_{ref\_crystal\_bypass}$ | Crystal reference using the main oscillator (PLL in BYPASS mode)[a] | 1 | - | 16.384 | MHz |
| $f_{ref\_ext\_bypass}$ | External clock reference (PLL in BYPASS mode)[a] | 0 | - | 50 | MHz |
| $f_{system\_clock}$ | System clock | 0 | - | 50 | MHz |

a. The ADC must be clocked from the PLL or directly from a 16-MHz clock source to operate properly.

**Table 21-12. Crystal Characteristics**

| Parameter Name | Value | | | | | | Units |
|---|---|---|---|---|---|---|---|
| Frequency | 16 | 12 | 8 | 6 | 4 | 3.5 | MHz |
| Frequency tolerance[a] | ±50 | ±50 | ±50 | ±50 | ±50 | ±50 | ppm |
| Oscillation mode | Parallel | Parallel | Parallel | Parallel | Parallel | Parallel | - |
| Motional capacitance (typ) | 13.9 | 18.5 | 27.8 | 37.0 | 55.6 | 63.5 | pF |
| Motional inductance (typ) | 7.15 | 9.5 | 14.3 | 19.1 | 28.6 | 32.7 | mH |
| Equivalent series resistance (max) | 80 | 100 | 120 | 160 | 200 | 220 | Ω |
| Shunt capacitance (max) | 10 | 10 | 10 | 10 | 10 | 10 | pF |
| Load capacitance (typ) | 16 | 16 | 16 | 16 | 16 | 16 | pF |
| Drive level (typ) | 100 | 100 | 100 | 100 | 100 | 100 | µW |

a. This tolerance provides a guard band for temperature stability and aging drift.

### 21.2.2.1 System Clock Specifications with ADC Operation

**Table 21-13. System Clock Characteristics with ADC Operation**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $f_{sysadc}$ | System clock frequency when the ADC module is operating (when PLL is bypassed) | 16 | - | - | MHz |

#### 21.2.2.2 System Clock Specification with USB Operation

**Table 21-14. System Clock Characteristics with USB Operation**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $F_{sysusb}$ | System clock frequency when the USB module is operating (note that MOSC must be supplied with a clock source) | 30 | - | - | MHz |

### 21.2.3 JTAG and Boundary Scan

**Table 21-15. JTAG Characteristics**

| Parameter No. | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| J1 | $f_{TCK}$ | TCK operational clock frequency | 0 | - | 10 | MHz |
| J2 | $t_{TCK}$ | TCK operational clock period | 100 | - | - | ns |
| J3 | $t_{TCK\_LOW}$ | TCK clock Low time | - | $t_{TCK}/2$ | - | ns |
| J4 | $t_{TCK\_HIGH}$ | TCK clock High time | - | $t_{TCK}/2$ | - | ns |
| J5 | $t_{TCK\_R}$ | TCK rise time | 0 | - | 10 | ns |
| J6 | $t_{TCK\_F}$ | TCK fall time | 0 | - | 10 | ns |
| J7 | $t_{TMS\_SU}$ | TMS setup time to TCK rise | 20 | - | - | ns |
| J8 | $t_{TMS\_HLD}$ | TMS hold time from TCK rise | 20 | - | - | ns |
| J9 | $t_{TDI\_SU}$ | TDI setup time to TCK rise | 25 | - | - | ns |
| J10 | $t_{TDI\_HLD}$ | TDI hold time from TCK rise | 25 | - | - | ns |
| J11 $t_{TDO\_ZDV}$ | TCK fall to Data Valid from High-Z | 2-mA drive | - | 23 | 35 | ns |
| | | 4-mA drive | | 15 | 26 | ns |
| | | 8-mA drive | | 14 | 25 | ns |
| | | 8-mA drive with slew rate control | | 18 | 29 | ns |
| J12 $t_{TDO\_DV}$ | TCK fall to Data Valid from Data Valid | 2-mA drive | - | 21 | 35 | ns |
| | | 4-mA drive | | 14 | 25 | ns |
| | | 8-mA drive | | 13 | 24 | ns |
| | | 8-mA drive with slew rate control | | 18 | 28 | ns |
| J13 $t_{TDO\_DVZ}$ | TCK fall to High-Z from Data Valid | 2-mA drive | - | 9 | 11 | ns |
| | | 4-mA drive | | 7 | 9 | ns |
| | | 8-mA drive | | 6 | 8 | ns |
| | | 8-mA drive with slew rate control | | 7 | 9 | ns |

**Figure 21-2. JTAG Test Clock Input Timing**

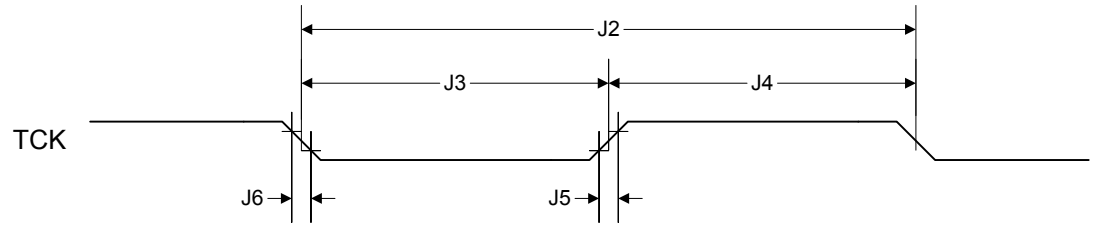
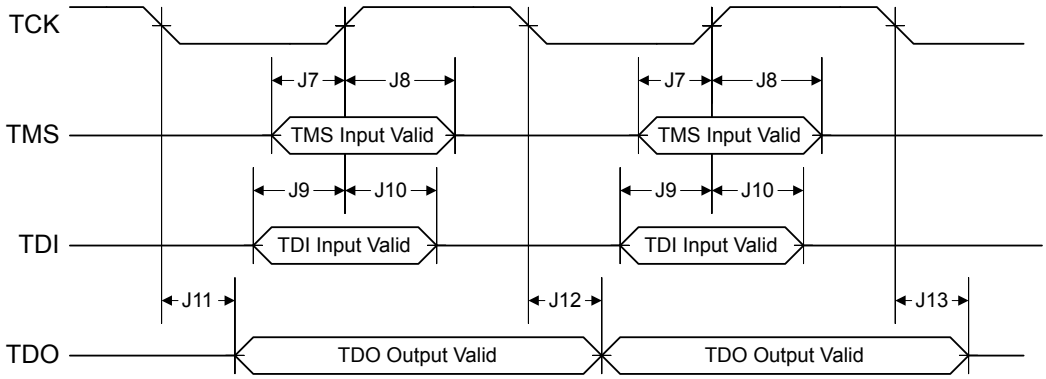


**Figure 21-3. JTAG Test Access Port (TAP) Timing**




## 21.2.4 Reset

**Table 21-16. Reset Characteristics**

| Parameter No. | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| R1 | $V_{TH}$ | Reset threshold | - | 2.0 | - | V |
| R2 | $V_{BTH}$ | Brown-Out threshold | 2.85 | 2.9 | 2.95 | V |
| R3 | $T_{POR}$ | Power-On Reset timeout | - | 10 | - | ms |
| R4 | $T_{BOR}$ | Brown-Out timeout | - | 500 | - | µs |
| R5 | $T_{IRPOR}$ | Internal reset timeout after POR | 6 | - | 11 | ms |
| R6 | $T_{IRBOR}$ | Internal reset timeout after BOR[a] | 0 | - | 1 | µs |
| R7 | $T_{IRHWR}$ | Internal reset timeout after hardware reset ($\overline{RST}$ pin) | 0 | - | 1 | ms |
| R8 | $T_{IRSWR}$ | Internal reset timeout after software-initiated system reset [a] | 2.5 | - | 20 | µs |
| R9 | $T_{IRWDR}$ | Internal reset timeout after watchdog reset[a] | 2.5 | - | 20 | µs |
| R10 | $T_{VDDRISE}$ | Supply voltage ($V_{DD}$) rise time (0V-3.3V), power on reset | - | - | 100 | ms |
| | | Supply voltage ($V_{DD}$) rise time (0V-3.3V), waking from hibernation | - | - | 250 | µs |
| R11 | $T_{MIN}$ | Minimum $\overline{RST}$ pulse width | 2 | - | - | µs |

a. 20 * $t_{MOSC\_per}$

**Figure 21-4. External Reset Timing ($\overline{RST}$)**

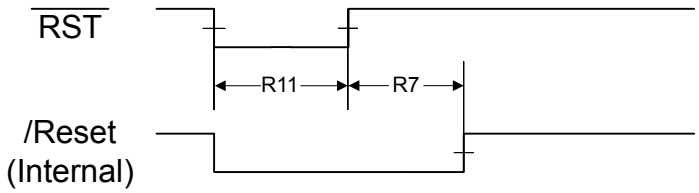


**Figure 21-5. Power-On Reset Timing**

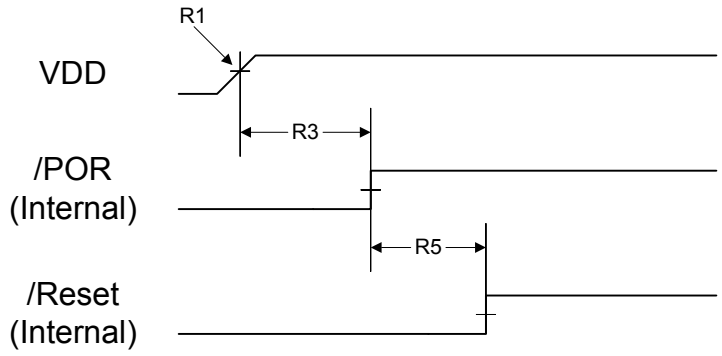



**Figure 21-6. Brown-Out Reset Timing**

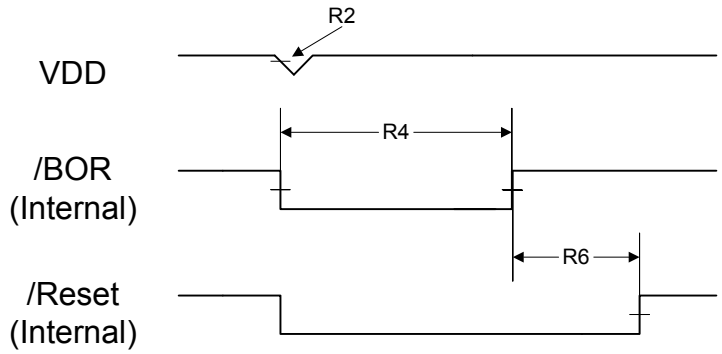



**Figure 21-7. Software Reset Timing**

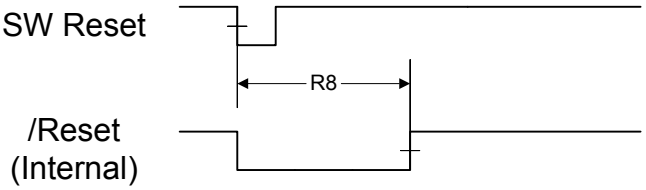
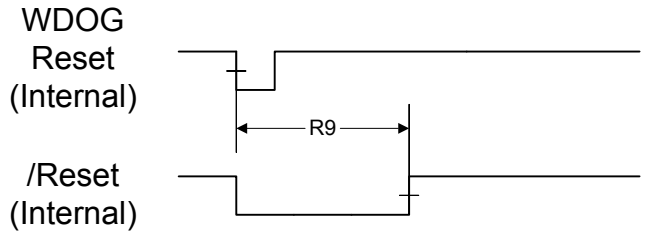



**Figure 21-8. Watchdog Reset Timing**



### 21.2.5 Sleep Modes

**Table 21-17. Sleep Modes AC Characteristics**[a]

| Parameter No | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| D1 | $t_{WAKE\_S}$ | Time to wake from interrupt in sleep or deep-sleep mode, not using the PLL | - | - | 7 | system clocks |
| D2 | $t_{WAKE\_PLL\_S}$ | Time to wake from interrupt in sleep or deep-sleep mode when using the PLL | - | - | $T_{READY}$ | ms |

a. Values in this table assume the IOSC is the clock source during sleep or deep-sleep mode.

### 21.2.6 Hibernation Module

The Hibernation Module requires special system implementation considerations since it is intended to power-down all other sections of its host device. The system power-supply distribution and interfaces to the device must be driven to 0 $V_{DC}$ or powered down with the same external voltage regulator controlled by $\overline{HIB}$.

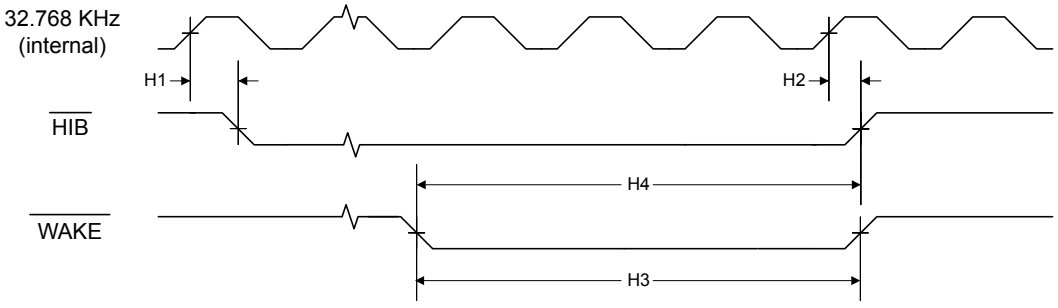
The external voltage regulators controlled by $\overline{HIB}$ must have a settling time of 250 µs or less.

**Table 21-18. Hibernation Module AC Characteristics**

| Parameter No | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| H1 | $t_{HIB\_LOW}$ | Internal 32.768 KHz clock reference rising edge to /HIB asserted | - | 200 | - | µs |
| H2 | $t_{HIB\_HIGH}$ | Internal 32.768 KHz clock reference rising edge to /HIB deasserted | - | 30 | - | µs |
| H3 | $t_{WAKE\_ASSERT}$ | /WAKE assertion time | 62 | - | - | µs |
| H4 | $t_{WAKETOHIB}$ | /WAKE assert to /HIB desassert | 62 | - | 124 | µs |
| H5 | $t_{XOSC\_SETTLE}$ | XOSC settling time[a] | 20 | - | - | ms |
| H6 | $t_{HIB\_REG\_ACCESS}$ | Access time to or from a non-volatile register in HIB module to complete | 92 | - | - | µs |
| H7 | $t_{HIB\_TO\_VDD}$ | $\overline{HIB}$ deassert to VDD and VDD25 at minimum operational level | - | - | 250 | µs |

a. This parameter is highly sensitive to PCB layout and trace lengths, which may make this parameter time longer. Care must be taken in PCB design to minimize trace lengths and RLC (resistance, inductance, capacitance).

**Figure 21-9. Hibernation Module Timing**




### 21.2.7 General-Purpose I/O (GPIO)

**Note:** All GPIOs are 5 V-tolerant.

**Table 21-19. GPIO Characteristics**

| Parameter | Parameter Name | Condition | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| $t_{GPIOR}$ | GPIO Rise Time (from 20% to 80% of $V_{DD}$) | 2-mA drive | - | 17 | 26 | ns |
| | | 4-mA drive | | 9 | 13 | ns |
| | | 8-mA drive | | 6 | 9 | ns |
| | | 8-mA drive with slew rate control | | 10 | 12 | ns |
| $t_{GPIOF}$ | GPIO Fall Time (from 80% to 20% of $V_{DD}$) | 2-mA drive | - | 17 | 25 | ns |
| | | 4-mA drive | | 8 | 12 | ns |
| | | 8-mA drive | | 6 | 10 | ns |
| | | 8-mA drive with slew rate control | | 11 | 13 | ns |

## 21.2.8 Analog-to-Digital Converter

**Table 21-20. ADC Characteristics**[a]

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{ADCIN}$ | Maximum single-ended, full-scale analog input voltage | - | - | 3.0 | V |
| | Minimum single-ended, full-scale analog input voltage | 0.0 | - | - | V |
| | Maximum differential, full-scale analog input voltage | - | - | 1.5 | V |
| | Minimum differential, full-scale analog input voltage | 0.0 | - | - | V |
| N | Resolution | | 10 | | bits |
| $f_{ADC}$ | ADC internal clock frequency[b] | 14 | 16 | 18 | MHz |
| $t_{ADCCONV}$ | Conversion time[c] | | 2 | | µs |
| $f_{ADCCONV}$ | Conversion rate[c] | | 500 | | k samples/s |
| $t_{LT}$ | Latency from trigger to start of conversion | - | 2 | - | system clocks |
| $I_L$ | ADC input leakage | - | - | ±3.0 | µA |
| $R_{ADC}$ | ADC equivalent resistance | - | - | 10 | kΩ |
| $C_{ADC}$ | ADC equivalent capacitance | 0.9 | 1.0 | 1.1 | pF |
| $E_L$ | Integral nonlinearity error | - | - | ±3 | LSB |
| $E_D$ | Differential nonlinearity error | - | - | ±2 | LSB |
| $E_O$ | Offset error | - | - | +6[d] | LSB |
| $E_G$ | Full-scale gain error | - | - | ±3 | LSB |
| $E_{TS}$ | Temperature sensor accuracy | - | - | ±5 | °C |

a. The ADC reference voltage is 3.0 V. This reference voltage is internally generated from the 3.3 VDDA supply by a band gap circuit.

b. The ADC must be clocked from the PLL or directly from an external clock source to operate properly.

c. The conversion time and rate scale from the specified number if the ADC internal clock frequency is any value other than 16 MHz.

d. The offset error listed above is the conversion result with 0 V applied to the ADC input.

**Figure 21-10. ADC Input Equivalency Diagram**

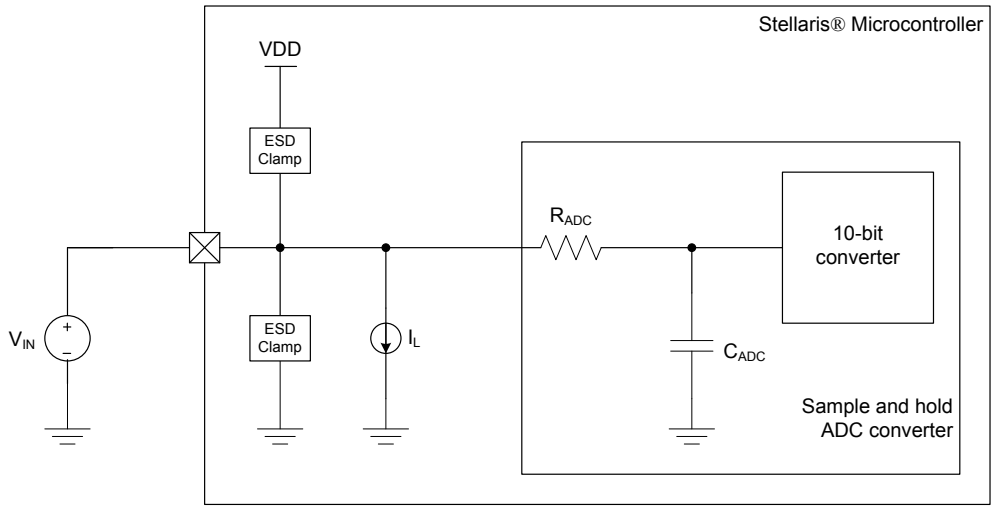



**Table 21-21. ADC Module Internal Reference Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{REFI}$ | Internal voltage reference for ADC | - | 3.0 | - | V |
| $E_{IR}$ | Internal voltage reference error | - | - | ±2.5 | % |

## 21.2.9 Synchronous Serial Interface (SSI)

**Table 21-22. SSI Characteristics**

| Parameter No. | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| S1 | $t_{clk\_per}$ | SSIClk cycle time | 2 | - | 65024 | system clocks |
| S2 | $t_{clk\_high}$ | SSIClk high time | - | 0.5 | - | t clk_per |
| S3 | $t_{clk\_low}$ | SSIClk low time | - | 0.5 | - | t clk_per |
| S4 | $t_{clkrf}$ | SSIClk rise/fall time[a] | - | 6 | 10 | ns |
| S5 | $t_{DMd}$ | Data from master valid delay time | 0 | - | 1 | system clocks |
| S6 | $t_{DMs}$ | Data from master setup time | 1 | - | - | system clocks |
| S7 | $t_{DMh}$ | Data from master hold time | 2 | - | - | system clocks |
| S8 | $t_{DSs}$ | Data from slave setup time | 1 | - | - | system clocks |
| S9 | $t_{DSh}$ | Data from slave hold time | 2 | - | - | system clocks |

a. Note that the delays shown are using 8-mA drive strength.

**Figure 21-11. SSI Timing for TI Frame Format (FRF=01), Single Transfer Timing Measurement**

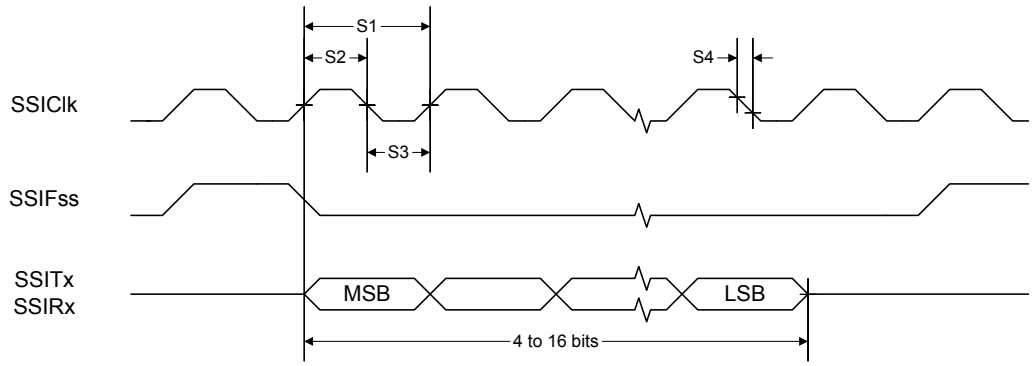



**Figure 21-12. SSI Timing for MICROWIRE Frame Format (FRF=10), Single Transfer**

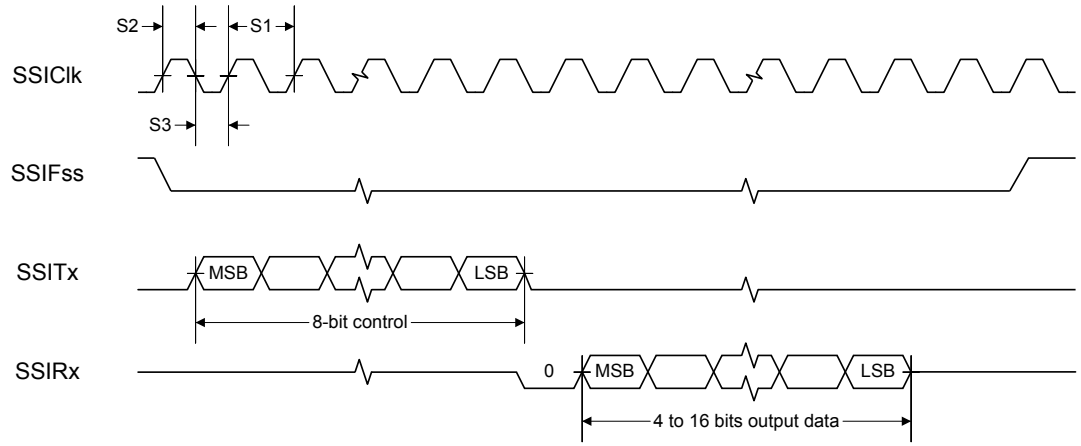
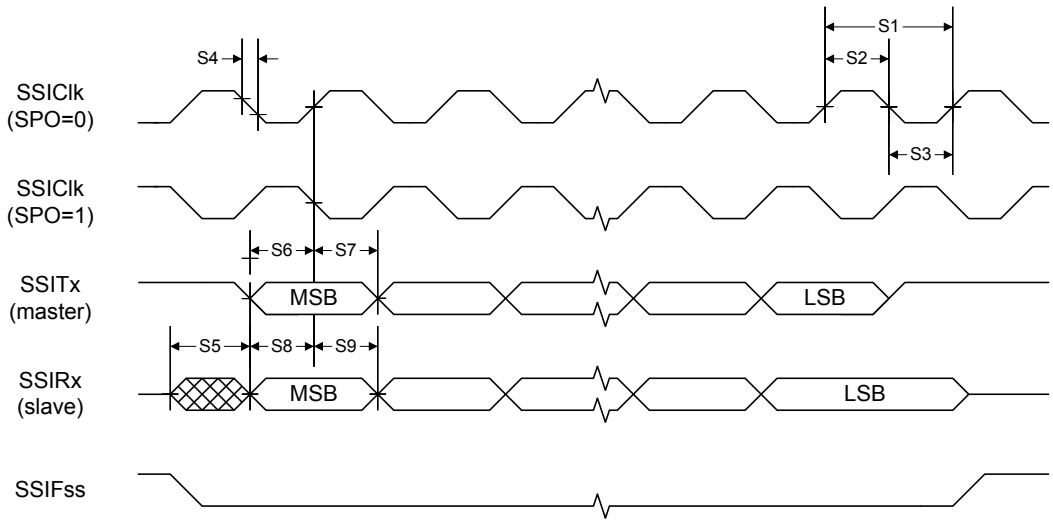


**Figure 21-13. SSI Timing for SPI Frame Format (FRF=00), with SPH=1**




## 21.2.10 Inter-Integrated Circuit ($I^2C$) Interface

**Table 21-23. $I^2C$ Characteristics**

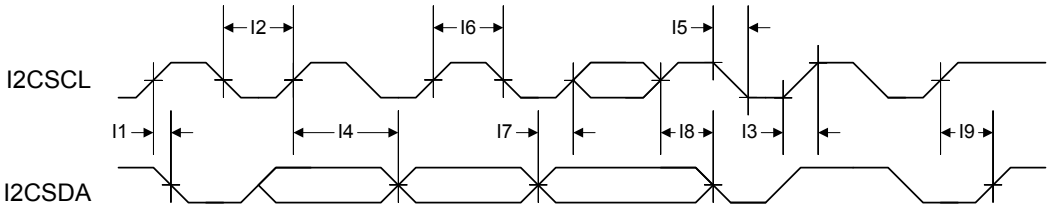
| Parameter No. | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| I1[a] | $t_{SCH}$ | Start condition hold time | 36 | - | - | system clocks |
| I2[a] | $t_{LP}$ | Clock Low period | 36 | - | - | system clocks |
| I3[b] | $t_{SRT}$ | I2CSCL/I2CSDA rise time ($V_{IL}$ =0.5 V to $V_{IH}$ =2.4 V) | - | - | (see note b) | ns |
| I4[a] | $t_{DH}$ | Data hold time | 2 | - | - | system clocks |
| I5[c] | $t_{SFT}$ | I2CSCL/I2CSDA fall time ($V_{IH}$ =2.4 V to $V_{IL}$ =0.5 V) | - | 9 | 10 | ns |
| I6[a] | $t_{HT}$ | Clock High time | 24 | - | - | system clocks |
| I7[a] | $t_{DS}$ | Data setup time | 18 | - | - | system clocks |
| I8[a] | $t_{SCSR}$ | Start condition setup time (for repeated start condition only) | 36 | - | - | system clocks |
| I9[a] | $t_{SCS}$ | Stop condition setup time | 24 | - | - | system clocks |

a. Values depend on the value programmed into the TPR bit in the **$I^2C$ Master Timer Period (I2CMTPR)** register; a TPR programmed for the maximum I2CSCL frequency (TPR=0x2) results in a minimum output timing as shown in the table above. The $I^2C$ interface is designed to scale the actual data transition time to move it to the middle of the I2CSCL Low period. The actual position is affected by the value programmed into the TPR; however, the numbers given in the above values are minimum values.

b. Because I2CSCL and I2CSDA are open-drain-type outputs, which the controller can only actively drive Low, the time I2CSCL or I2CSDA takes to reach a high level depends on external signal capacitance and pull-up resistor values.

c. Specified at a nominal 50 pF load.

**Figure 21-14. I²C Timing**




## 21.2.11 Universal Serial Bus (USB) Controller

The Stellaris USB controller AC electrical specifications are compliant with the “Universal Serial Bus Specification Rev. 2.0” (full-speed and low-speed support).

# A Boot Loader

## A.1 Boot Loader

The Stellaris® Boot Loader is executed from the ROM when flash is empty and is used to download code to the flash memory of a device without the use of a debug interface. The boot loader uses a simple packet interface to provide synchronous communication with the device. The boot loader runs off the internal oscillator and does not enable the PLL, so its speed is determined by the speed of the internal oscillator. The following serial interfaces can be used:

- UART0
- SSI0
- $I^2C0$

For simplicity, both the data format and communication protocol are identical for all serial interfaces.

See the *Stellaris Boot Loader User's Guide* for information on the boot loader software.

## A.2 Interfaces

Once communication with the boot loader is established via one of the serial interfaces, that interface is used until the boot loader is reset or new code takes over. For example, once you start communicating using the SSI port, communications with the boot loader via the UART are disabled until the device is reset.

### A.2.1 UART

The Universal Asynchronous Receivers/Transmitters (UART) communication uses a fixed serial format of 8 bits of data, no parity, and 1 stop bit. The baud rate used for communication is automatically detected by the boot loader and can be any valid baud rate supported by the host and the device. The auto detection sequence requires that the baud rate should be no more than 1/32 the internal oscillator frequency of the board that is running the boot loader (which is at least 8.4 MHz, providing support for up to 262,500 baud). This is actually the same as the hardware limitation for the maximum baud rate for any UART on a Stellaris device which is calculated as follows:

```
Max Baud Rate = System Clock Frequency / 16
```

In order to determine the baud rate, the boot loader needs to determine the relationship between the internal oscillator and the baud rate. This is enough information for the boot loader to configure its UART to the same baud rate as the host. This automatic baud-rate detection allows the host to use any valid baud rate that it wants to communicate with the device.

The method used to perform this automatic synchronization relies on the host sending the boot loader two bytes that are both 0x55. This generates a series of pulses to the boot loader that it can use to calculate the ratios needed to program the UART to match the host's baud rate. After the host sends the pattern, it attempts to read back one byte of data from the UART. The boot loader returns the value of 0xCC to indicate successful detection of the baud rate. If this byte is not received after at least twice the time required to transfer the two bytes, the host can resend another pattern of 0x55, 0x55, and wait for the 0xCC byte again until the boot loader acknowledges that it has received a synchronization pattern correctly. For example, the time to wait for data back from the boot loader should be calculated as at least 2*(20(bits/sync)/baud rate (bits/sec)). For a baud rate of 115200, this time is 2*(20/115200) or 0.35 ms.

### A.2.2 SSI

The Synchronous Serial Interface (SSI) port also uses a fixed serial format for communications, with the framing defined as Motorola format with SPH set to 1 and SPO set to 1. See “Frame Formats” on page 539 in the SSI chapter for more information on formats for this transfer protocol. Like the UART, this interface has hardware requirements that limit the maximum speed that the SSI clock can run. This allows the SSI clock to be at most 1/12 the the internal oscillator frequency of the board running the boot loader (which is at least 8.4 MHz, providing support for up to 700 KHz).. Since the host device is the master, the SSI on the boot loader device does not need to determine the clock as it is provided directly by the host.

### A.2.3 $I^2C$

The Inter-Integrated Circuit ($I^2C$) port operates in slave mode with a slave address of 0x42. The $I^2C$ port will work at both 100 Khz and 400 KHz $I^2C$ clock frequency. Since the host device is the master, the $I^2C$ on the boot loader device does not need to determine the clock as it is provided directly by the host.

## A.3 Packet Handling

All communications, with the exception of the UART auto-baud, are done via defined packets that are acknowledged (ACK) or not acknowledged (NAK) by the devices. The packets use the same format for receiving and sending packets, including the method used to acknowledge successful or unsuccessful reception of a packet.

### A.3.1 Packet Format

All packets sent and received from the device use the following byte-packed format.

```
struct
{
 unsigned char ucSize;
 unsigned char ucCheckSum;
 unsigned char Data[];
};
```

| | |
|---|---|
| ucSize | The first byte received holds the total size of the transfer including the size and checksum bytes. |
| ucChecksum | This holds a simple checksum of the bytes in the data buffer only. The algorithm is Data[0]+Data[1]+…+ Data[`ucSize`-3]. |
| Data | This is the raw data intended for the device, which is formatted in some form of command interface. There should be `ucSize`–2 bytes of data provided in this buffer to or from the device. |

### A.3.2 Sending Packets

The actual bytes of the packet can be sent individually or all at once; the only limitation is that commands that cause flash memory access should limit the download sizes to prevent losing bytes during flash programming. This limitation is discussed further in the section that describes the boot loader command, COMMAND_SEND_DATA (see “COMMAND_SEND_DATA (0x24)” on page 784).

Once the packet has been formatted correctly by the host, it should be sent out over the UART or SSI interface. Then the host should poll the UART or SSI interface for the first non-zero data returned from the device. The first non-zero byte will either be an ACK (0xCC) or a NAK (0x33) byte from

the device indicating the packet was received successfully (ACK) or unsuccessfully (NAK). This does not indicate that the actual contents of the command issued in the data portion of the packet were valid, just that the packet was received correctly.

### A.3.3 Receiving Packets

The boot loader sends a packet of data in the same format that it receives a packet. The boot loader may transfer leading zero data before the first actual byte of data is sent out. The first non-zero byte is the size of the packet followed by a checksum byte, and finally followed by the data itself. There is no break in the data after the first non-zero byte is sent from the boot loader. Once the device communicating with the boot loader receives all the bytes, it must either ACK or NAK the packet to indicate that the transmission was successful. The appropriate response after sending a NAK to the boot loader is to resend the command that failed and request the data again. If needed, the host may send leading zeros before sending down the ACK/NAK signal to the boot loader, as the boot loader only accepts the first non-zero data as a valid response. This zero padding is needed by the SSI interface in order to receive data to or from the boot loader.

## A.4 Commands

The next section defines the list of commands that can be sent to the boot loader. The first byte of the data should always be one of the defined commands, followed by data or parameters as determined by the command that is sent.

### A.4.1 COMMAND_PING (0X20)

This command simply accepts the command and sets the global status to success. The format of the packet is as follows:

```
Byte[0] = 0x03;
Byte[1] = checksum(Byte[2]);
Byte[2] = COMMAND_PING;
```

The ping command has 3 bytes and the value for `COMMAND_PING` is 0x20 and the checksum of one byte is that same byte, making Byte[1] also 0x20. Since the ping command has no real return status, the receipt of an ACK can be interpreted as a successful ping to the boot loader.

### A.4.2 COMMAND_DOWNLOAD (0x21)

This command is sent to the boot loader to indicate where to store data and how many bytes will be sent by the `COMMAND_SEND_DATA` commands that follow. The command consists of two 32-bit values that are both transferred MSB first. The first 32-bit value is the address to start programming data into, while the second is the 32-bit size of the data that will be sent. This command also triggers an erase of the full area to be programmed so this command takes longer than other commands. This results in a longer time to receive the ACK/NAK back from the board. This command should be followed by a `COMMAND_GET_STATUS` to ensure that the Program Address and Program size are valid for the device running the boot loader.

The format of the packet to send this command is a follows:

```
Byte[0] = 11
Byte[1] = checksum(Bytes[2:10])
Byte[2] = COMMAND_DOWNLOAD
Byte[3] = Program Address [31:24]
Byte[4] = Program Address [23:16]
Byte[5] = Program Address [15:8]
```

```
Byte[6] = Program Address [7:0]
Byte[7] = Program Size [31:24]
Byte[8] = Program Size [23:16]
Byte[9] = Program Size [15:8]
Byte[10] = Program Size [7:0]
```

### A.4.3 COMMAND_RUN (0x22)

This command is used to tell the boot loader to execute from the address passed as the parameter in this command. This command consists of a single 32-bit value that is interpreted as the address to execute. The 32-bit value is transmitted MSB first and the boot loader responds with an ACK signal back to the host device before actually executing the code at the given address. This allows the host to know that the command was received successfully and the code is now running.

```
Byte[0] = 7
Byte[1] = checksum(Bytes[2:6])
Byte[2] = COMMAND_RUN
Byte[3] = Execute Address[31:24]
Byte[4] = Execute Address[23:16]
Byte[5] = Execute Address[15:8]
Byte[6] = Execute Address[7:0]
```

### A.4.4 COMMAND_GET_STATUS (0x23)

This command returns the status of the last command that was issued. Typically, this command should be sent after every command to ensure that the previous command was successful or to properly respond to a failure. The command requires one byte in the data of the packet and should be followed by reading a packet with one byte of data that contains a status code. The last step is to ACK or NAK the received data so the boot loader knows that the data has been read.

```
Byte[0] = 0x03
Byte[1] = checksum(Byte[2])
Byte[2] = COMMAND_GET_STATUS
```

### A.4.5 COMMAND_SEND_DATA (0x24)

This command should only follow a COMMAND_DOWNLOAD command or another COMMAND_SEND_DATA command if more data is needed. Consecutive send data commands automatically increment address and continue programming from the previous location. For packets which do not contain the final portion of the downloaded data, a multiple of four bytes should always be transferred. The command terminates programming once the number of bytes indicated by the COMMAND_DOWNLOAD command has been received. Each time this function is called it should be followed by a COMMAND_GET_STATUS to ensure that the data was successfully programmed into the flash. If the boot loader sends a NAK to this command, the boot loader does not increment the current address to allow retransmission of the previous data. The following example shows a COMMAND_SEND_DATA packet with 8 bytes of packet data:

```
Byte[0] = 11
Byte[1] = checksum(Bytes[2:10])
Byte[2] = COMMAND_SEND_DATA
Byte[3] = Data[0]
Byte[4] = Data[1]
Byte[5] = Data[2]
Byte[6] = Data[3]
```

```
Byte[7] = Data[4]
Byte[8] = Data[5]
Byte[9] = Data[6]
Byte[10] = Data[7]
```

### A.4.6 COMMAND_RESET (0x25)

This command is used to tell the boot loader device to reset. Unlike the COMMAND_RUN command, this allows the initial stack pointer to be read by the hardware and set up for the new code. It can also be used to reset the boot loader if a critical error occurs and the host device wants to restart communication with the boot loader.

```
Byte[0] = 3
Byte[1] = checksum(Byte[2])
Byte[2] = COMMAND_RESET
```

The boot loader responds with an ACK signal back to the host device before actually executing the software reset to the device running the boot loader. This allows the host to know that the command was received successfully and the part will be reset.

# B ROM DriverLib Functions

## B.1 DriverLib Functions Included in the Integrated ROM

The Stellaris® Peripheral Driver Library (DriverLib) APIs that are available in the integrated ROM of the Stellaris family of devices are listed below. The detailed description of each function is available in the *Stellaris® ROM User's Guide*.

ROM_ADCHardwareOversampleConfigure
    // Configures the hardware oversampling factor of the ADC.

ROM_ADCIntClear
    // Clears sample sequence interrupt source.

ROM_ADCIntDisable
    // Disables a sample sequence interrupt.

ROM_ADCIntEnable
    // Enables a sample sequence interrupt.

ROM_ADCIntStatus
    // Gets the current interrupt status.

ROM_ADCProcessorTrigger
    // Causes a processor trigger for a sample sequence.

ROM_ADCSequenceConfigure
    // Configures the trigger source and priority of a sample sequence.

ROM_ADCSequenceDataGet
    // Gets the captured data for a sample sequence.

ROM_ADCSequenceDisable
    // Disables a sample sequence.

ROM_ADCSequenceEnable
    // Enables a sample sequence.

ROM_ADCSequenceOverflow
    // Determines if a sample sequence overflow occurred.

ROM_ADCSequenceOverflowClear
    // Clears the overflow condition on a sample sequence.

ROM_ADCSequenceStepConfigure
    // Configure a step of the sample sequencer.

ROM_ADCSequenceUnderflow
    // Determines if a sample sequence underflow occurred.

ROM_ADCSequenceUnderflowClear
    // Clears the underflow condition on a sample sequence.

ROM_FlashErase
    // Erases a block of flash.

ROM_FlashIntClear
    // Clears flash controller interrupt sources.

ROM_FlashIntDisable
    // Disables individual flash controller interrupt sources.

ROM_FlashIntEnable
    // Enables individual flash controller interrupt sources.

ROM_FlashIntGetStatus
    // Gets the current interrupt status.

ROM_FlashProgram
    // Programs flash.

ROM_FlashProtectGet
    // Gets the protection setting for a block of flash.

ROM_FlashProtectSave
    // Saves the flash protection settings.

ROM_FlashProtectSet
    // Sets the protection setting for a block of flash.

ROM_FlashUsecGet
    // Gets the number of processor clocks per micro-second.

ROM_FlashUsecSet
    // Sets the number of processor clocks per micro-second.

ROM_FlashUserGet
    // Gets the user registers.

ROM_FlashUserSave
    // Saves the user registers.

ROM_FlashUserSet
    // Sets the user registers.

ROM_GPIODirModeGet
    // Gets the direction and mode of a pin.

ROM_GPIODirModeSet
    // Sets the direction and mode of the specified pin(s).

ROM_GPIOIntTypeGet
    // Gets the interrupt type for a pin.

ROM_GPIOIntTypeSet
    // Sets the interrupt type for the specified pin(s).

ROM_GPIOPadConfigGet
    // Gets the pad configuration for a pin.

ROM_GPIOPadConfigSet
    // Sets the pad configuration for the specified pin(s).

ROM_GPIOPinIntClear
    // Clears the interrupt for the specified pin(s).

ROM_GPIOPinIntDisable
    // Disables interrupts for the specified pin(s).

ROM_GPIOPinIntEnable
    // Enables interrupts for the specified pin(s).

ROM_GPIOPinIntStatus
    // Gets interrupt status for the specified GPIO port.

ROM_GPIOPinRead
    // Reads the values present of the specified pin(s).

ROM_GPIOPinTypeCAN
    // Configures pin(s) for use as a CAN device.

ROM_GPIOPinTypeGPIOInput
    // Configures pin(s) for use as GPIO inputs.

ROM_GPIOPinTypeGPIOOutput
    // Configures pin(s) for use as GPIO outputs.

ROM_GPIOPinTypeGPIOOutputOD
    // Configures pin(s) for use as GPIO open drain outputs.

ROM_GPIOPinTypeI2C
    // Configures pin(s) for use by the I2C peripheral.

ROM_GPIOPinTypeSSI
    // Configures pin(s) for use by the SSI peripheral.

ROM_GPIOPinTypeTimer
    // Configures pin(s) for use by the Timer peripheral.

ROM_GPIOPinTypeUART
    // Configures pin(s) for use by the UART peripheral.

ROM_GPIOPinWrite
    // Writes a value to the specified pin(s).

ROM_I2CMasterBusBusy
    // Indicates whether or not the I2C bus is busy.

ROM_I2CMasterBusy
    // Indicates whether or not the I2C Master is busy.

ROM_I2CMasterControl
    // Controls the state of the I2C Master module.

ROM_I2CMasterDataGet
    // Receives a byte that has been sent to the I2C Master.

ROM_I2CMasterDataPut
    // Transmits a byte from the I2C Master.

ROM_I2CMasterDisable
    // Disables the I2C master block.

ROM_I2CMasterEnable
    // Enables the I2C Master block.

ROM_I2CMasterErr
    // Gets the error status of the I2C Master module.

ROM_I2CMasterInitExpClk
    // Initializes the I2C Master block.

ROM_I2CMasterIntClear
    // Clears I2C Master interrupt sources.

ROM_I2CMasterIntDisable
    // Disables the I2C Master interrupt.

ROM_I2CMasterIntEnable
    // Enables the I2C Master interrupt.

ROM_I2CMasterIntStatus
    // Gets the current I2C Master interrupt status.

ROM_I2CMasterSlaveAddrSet
    // Sets the address that the I2C Master will place on the bus.

ROM_I2CSlaveDataGet
    // Receives a byte that has been sent to the I2C Slave.

ROM_I2CSlaveDataPut
    // Transmits a byte from the I2C Slave.

ROM_I2CSlaveDisable
    // Disables the I2C slave block.

ROM_I2CSlaveEnable
    // Enables the I2C Slave block.

ROM_I2CSlaveInit
    // Initializes the I2C Slave block.

ROM_I2CSlaveIntClear
    // Clears I2C Slave interrupt sources.

ROM_I2CSlaveIntDisable
    // Disables the I2C Slave interrupt.

ROM_I2CSlaveIntEnable
    // Enables the I2C Slave interrupt.

ROM_I2CSlaveIntStatus
    // Gets the current I2C Slave interrupt status.

ROM_I2CSlaveStatus
    // Gets the I2C Slave module status.

ROM_IntDisable
    // Disables an interrupt.

ROM_IntEnable
    // Enables an interrupt.

ROM_IntPriorityGet
    // Gets the priority of an interrupt.

ROM_IntPriorityGroupingGet
    // Gets the priority grouping of the interrupt controller.

ROM_IntPriorityGroupingSet
    // Sets the priority grouping of the interrupt controller.

ROM_IntPrioritySet
    // Sets the priority of an interrupt.

ROM_SSIConfigSetExpClk
    // Configures the synchronous serial interface.

ROM_SSIDataGet
    // Gets a data element from the SSI receive FIFO.

ROM_SSIDataGetNonBlocking
    // Gets a data element from the SSI receive FIFO.

ROM_SSIDataPut
    // Puts a data element into the SSI transmit FIFO.

ROM_SSIDataPutNonBlocking
    // Puts a data element into the SSI transmit FIFO.

ROM_SSIDisable
    // Disables the synchronous serial interface.

ROM_SSIEnable
    // Enables the synchronous serial interface.

ROM_SSIIntClear
    // Clears SSI interrupt sources.

ROM_SSIIntDisable
    // Disables individual SSI interrupt sources.

ROM_SSIIntEnable
    // Enables individual SSI interrupt sources.

ROM_SSIIntStatus
    // Gets the current interrupt status.

ROM_SysCtlADCSpeedGet
    // Gets the sample rate of the ADC.

ROM_SysCtlADCSpeedSet
    // Sets the sample rate of the ADC.

ROM_SysCtlClockGet
    // Gets the processor clock rate.

ROM_SysCtlClockSet
    // Sets the clocking of the device.

ROM_SysCtlDeepSleep
    // Puts the processor into deep-sleep mode.

ROM_SysCtlFlashSizeGet
    // Gets the size of the flash.

ROM_SysCtlGPIOAHBDisable
    // Disables a GPIO peripheral for access from the AHB.

ROM_SysCtlGPIOAHBEnable
    // Enables a GPIO peripheral for access from the AHB.

ROM_SysCtlIntClear
    // Clears system control interrupt sources.

ROM_SysCtlIntDisable
    // Disables individual system control interrupt sources.

ROM_SysCtlIntEnable
    // Enables individual system control interrupt sources.

ROM_SysCtlIntStatus
    // Gets the current interrupt status.

ROM_SysCtlLDOGet
    // Gets the output voltage of the LDO.

ROM_SysCtlLDOSet
    // Sets the output voltage of the LDO.

ROM_SysCtlPeripheralClockGating
    // Controls peripheral clock gating in sleep and deep-sleep mode.

ROM_SysCtlPeripheralDeepSleepDisable
    // Disables a peripheral in deep-sleep mode.

ROM_SysCtlPeripheralDeepSleepEnable
    // Enables a peripheral in deep-sleep mode.

ROM_SysCtlPeripheralDisable
    // Disables a peripheral.

ROM_SysCtlPeripheralEnable
    // Enables a peripheral.

ROM_SysCtlPeripheralPresent
    // Determines if a peripheral is present.

ROM_SysCtlPeripheralReset
    // Performs a software reset of a peripheral.

ROM_SysCtlPeripheralSleepDisable
    // Disables a peripheral in sleep mode.

ROM_SysCtlPeripheralSleepEnable
    // Enables a peripheral in sleep mode.

ROM_SysCtlPinPresent
    // Determines if a pin is present.

ROM_SysCtlReset
    // Resets the device.

ROM_SysCtlResetCauseClear
    // Clears reset reasons.

ROM_SysCtlResetCauseGet
    // Gets the reason for a reset.

ROM_SysCtlSleep
    // Puts the processor into sleep mode.

ROM_SysCtlSRAMSizeGet
    // Gets the size of the SRAM.

ROM_SysTickDisable
    // Disables the SysTick counter.

ROM_SysTickEnable
    // Enables the SysTick counter.

ROM_SysTickIntDisable
    // Disables the SysTick interrupt.

ROM_SysTickIntEnable
    // Enables the SysTick interrupt.

ROM_SysTickPeriodGet
    // Gets the period of the SysTick counter.

ROM_SysTickPeriodSet
    // Sets the period of the SysTick counter.

ROM_SysTickValueGet
    // Gets the current value of the SysTick counter.

ROM_TimerConfigure
    // Configures the timer(s).

ROM_TimerControlEvent
    // Controls the event type.

ROM_TimerControlLevel
    // Controls the output level.

ROM_TimerControlStall
    // Controls the stall handling.

ROM_TimerControlTrigger
    // Enables or disables the trigger output.

ROM_TimerDisable
    // Disables the timer(s).

ROM_TimerEnable
    // Enables the timer(s).

ROM_TimerIntClear
    // Clears timer interrupt sources.

ROM_TimerIntDisable
    // Disables individual timer interrupt sources.

ROM_TimerIntEnable
    // Enables individual timer interrupt sources.

ROM_TimerIntStatus
    // Gets the current interrupt status.

ROM_TimerLoadGet
    // Gets the timer load value.

ROM_TimerLoadSet
    // Sets the timer load value.

ROM_TimerMatchGet
    // Gets the timer match value.

ROM_TimerMatchSet
    // Sets the timer match value.

ROM_TimerPrescaleGet
    // Get the timer prescale value.

ROM_TimerPrescaleSet
    // Set the timer prescale value.

ROM_TimerRTCDisable
    // Disable RTC counting.

ROM_TimerRTCEnable
    // Enable RTC counting.

ROM_TimerValueGet
    // Gets the current timer value.

ROM_UARTBreakCtl
    // Causes a BREAK to be sent.

ROM_UARTCharGet
    // Waits for a character from the specified port.

ROM_UARTCharGetNonBlocking
    // Receives a character from the specified port.

ROM_UARTCharPut
    // Waits to send a character from the specified port.

ROM_UARTCharPutNonBlocking
    // Sends a character to the specified port.

ROM_UARTCharsAvail
    // Determines if there are any characters in the receive FIFO.

ROM_UARTConfigGetExpClk
    // Gets the current configuration of a UART.

ROM_UARTConfigSetExpClk
    // Sets the configuration of a UART.

ROM_UARTDisable
    // Disables transmitting and receiving.

ROM_UARTDisableSIR
    // Disables SIR (IrDA) mode on the specified UART.

ROM_UARTEnable
    // Enables transmitting and receiving.

ROM_UARTEnableSIR
    // Enables SIR (IrDA) mode on specified UART.

ROM_UARTFIFOLevelGet
    // Gets the FIFO level at which interrupts are generated.

ROM_UARTFIFOLevelSet
    // Sets the FIFO level at which interrupts are generated.

ROM_UARTIntClear
    // Clears UART interrupt sources.

ROM_UARTIntDisable
    // Disables individual UART interrupt sources.

ROM_UARTIntEnable
    // Enables individual UART interrupt sources.

ROM_UARTIntStatus
    // Gets the current interrupt status.

ROM_UARTParityModeGet
    // Gets the type of parity currently being used.

ROM_UARTParityModeSet
    // Sets the type of parity.

ROM_UARTSpaceAvail
    // Determines if there is any space in the transmit FIFO.

ROM_UpdateI2C
    // Starts an update over the I2C0 interface.

ROM_UpdateSSI
    // Starts an update over the SSI0 interface.

ROM_UpdateUART
    // Starts an update over the UART0 interface.

ROM_WatchdogEnable
    // Enables the watchdog timer.

ROM_WatchdogIntClear
    // Clears the watchdog timer interrupt.

ROM_WatchdogIntEnable
    // Enables the watchdog timer interrupt.

ROM_WatchdogIntStatus
    // Gets the current watchdog timer interrupt status.

ROM_WatchdogLock
    // Enables the watchdog timer lock mechanism.

ROM_WatchdogLockState
    // Gets the state of the watchdog timer lock mechanism.

ROM_WatchdogReloadGet
    // Gets the watchdog timer reload value.

ROM_WatchdogReloadSet
    // Sets the watchdog timer reload value.

ROM_WatchdogResetDisable
    // Disables the watchdog timer reset.

ROM_WatchdogResetEnable
    // Enables the watchdog timer reset.

ROM_WatchdogRunning
    // Determines if the watchdog timer is enabled.

ROM_WatchdogStallDisable
    // Disables stalling of the watchdog timer during debug events.

ROM_WatchdogStallEnable
    // Enables stalling of the watchdog timer during debug events.

ROM_WatchdogUnlock
    // Disables the watchdog timer lock mechanism.

ROM_WatchdogValueGet
    // Gets the current watchdog timer value.

# C Register Quick Reference

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **The Cortex-M3 Processor** | | | | | | | | | | | | | | | |
| **R0, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R1, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R2, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R3, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R4, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R5, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R6, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R7, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R8, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R9, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R10, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R11, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R12, type R/W, , reset -** (see page 60) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **SP, type R/W, , reset -** (see page 61) | | | | | | | | | | | | | | | |
| SP | | | | | | | | | | | | | | | |
| SP | | | | | | | | | | | | | | | |
| **LR, type R/W, , reset 0xFFFF.FFFF** (see page 62) | | | | | | | | | | | | | | | |
| LINK | | | | | | | | | | | | | | | |
| LINK | | | | | | | | | | | | | | | |
| **PC, type R/W, , reset -** (see page 63) | | | | | | | | | | | | | | | |
| PC | | | | | | | | | | | | | | | |
| PC | | | | | | | | | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **PSR, type R/W, , reset 0x0100.0000** (see page 64) | | | | | | | | | | | | | | | |
| N | Z | C | V | Q | ICI / IT | | THUMB | | | | | | | | |
| ICI / IT | | | | | | | | | ISRNUM | | | | | | |
| **PRIMASK, type R/W, , reset 0x0000.0000** (see page 68) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | PRIMASK |
| **FAULTMASK, type R/W, , reset 0x0000.0000** (see page 69) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | FAULTMASK |
| **BASEPRI, type R/W, , reset 0x0000.0000** (see page 70) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | BASEPRI | | | | | | | |
| **CONTROL, type R/W, , reset 0x0000.0000** (see page 71) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | ASP | TMPL |

**Cortex-M3 Peripherals**

**System Timer (SysTick) Registers**

Base 0xE000.E000

| **STCTRL, type R/W, offset 0x010, reset 0x0000.0000** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | COUNT |
| | | | | | | | | | | | | | CLK_SRC | INTEN | ENABLE |
| **STRELOAD, type R/W, offset 0x014, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | RELOAD | | | | | | | |
| RELOAD | | | | | | | | | | | | | | | |
| **STCURRENT, type R/WC, offset 0x018, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | CURRENT | | | | | | | |
| CURRENT | | | | | | | | | | | | | | | |

**Cortex-M3 Peripherals**

**Nested Vectored Interrupt Controller (NVIC) Registers**

Base 0xE000.E000

| **EN0, type R/W, offset 0x100, reset 0x0000.0000** |
|---|
| INT |
| INT |
| **EN1, type R/W, offset 0x104, reset 0x0000.0000** |
| INT |
| **DIS0, type R/W, offset 0x180, reset 0x0000.0000** |
| INT |
| INT |
| **DIS1, type R/W, offset 0x184, reset 0x0000.0000** |
| INT |
| **PEND0, type R/W, offset 0x200, reset 0x0000.0000** |
| INT |
| INT |
| **PEND1, type R/W, offset 0x204, reset 0x0000.0000** |
| INT |
| **UNPEND0, type R/W, offset 0x280, reset 0x0000.0000** |
| INT |
| INT |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **UNPEND1, type R/W, offset 0x284, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| INT | | | | | | | | | | | | | | | |
| **ACTIVE0, type RO, offset 0x300, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INT | | | | | | | | | | | | | | | |
| INT | | | | | | | | | | | | | | | |
| **ACTIVE1, type RO, offset 0x304, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| INT | | | | | | | | | | | | | | | |
| **PRI0, type R/W, offset 0x400, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI1, type R/W, offset 0x404, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI2, type R/W, offset 0x408, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI3, type R/W, offset 0x40C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI4, type R/W, offset 0x410, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI5, type R/W, offset 0x414, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI6, type R/W, offset 0x418, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI7, type R/W, offset 0x41C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI8, type R/W, offset 0x420, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI9, type R/W, offset 0x424, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI10, type R/W, offset 0x428, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI11, type R/W, offset 0x42C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **SWTRIG, type WO, offset 0xF00, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | INTID | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Cortex-M3 Peripherals**<br>**System Control Block (SCB) Registers**<br>Base 0xE000.E000 | | | | | | | | | | | | | | | |
| **CPUID, type RO, offset 0xD00, reset 0x411F.C231** | | | | | | | | | | | | | | | |
| IMP | | | | | | | | VAR | | | | CON | | | |
| PARTNO | | | | | | | | | | | | REV | | | |
| **INTCTRL, type R/W, offset 0xD04, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| NMISET | | | PENDSV | UNPENDSV | PENDSTSET | PENDSTCLR | | ISRPRE | ISRPEND | | | | VECPEND | | |
| VECPEND | | | | RETBASE | | | | | VECACT | | | | | | |
| **VTABLE, type R/W, offset 0xD08, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | BASE | OFFSET | | | | | | | | | | | | |
| OFFSET | | | | | | | | | | | | | | | |
| **APINT, type R/W, offset 0xD0C, reset 0xFA05.0000** | | | | | | | | | | | | | | | |
| VECTKEY | | | | | | | | | | | | | | | |
| ENDIANESS | | | | | PRIGROUP | | | | | | | | SYSRESREQ | VECTCLRACT | VECTRESET |
| **SYSCTRL, type R/W, offset 0xD10, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | SEVONPEND | | SLEEPDEEP | SLEEPEXIT | |
| **CFGCTRL, type R/W, offset 0xD14, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | STKALIGN | BFHFNMIGN | | | | DIV0 | UNALIGNED | | MAINPEND | BASETHR |
| **SYSPRI1, type R/W, offset 0xD18, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | USAGE | | | | | | | |
| BUS | | | | | | | | MEM | | | | | | | |
| **SYSPRI2, type R/W, offset 0xD1C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| SVC | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| **SYSPRI3, type R/W, offset 0xD20, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| TICK | | | | | | | | PENDSV | | | | | | | |
| | | | | | | | | DEBUG | | | | | | | |
| **SYSHNDCTRL, type R/W, offset 0xD24, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | USAGE | BUS | MEM |
| SVC | BUSP | MEMP | USAGEP | TICK | PNDSV | | MON | SVCA | | | | USGA | | BUSA | MEMA |
| **FAULTSTAT, type R/W1C, offset 0xD28, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | DIV0 | UNALIGN | | | | | NOCP | INVPC | INVSTAT | UNDEF |
| BFARV | | | BSTKE | BUSTKE | IMPRE | PRECISE | IBUS | MMARV | | | MSTKE | MUSTKE | | DERR | IERR |
| **HFAULTSTAT, type R/W1C, offset 0xD2C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| DBG | FORCED | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | VECT | |
| **MMADDR, type R/W, offset 0xD34, reset -** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| **FAULTADDR, type R/W, offset 0xD38, reset -** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| **Cortex-M3 Peripherals**<br>**Memory Protection Unit (MPU) Registers**<br>Base 0xE000.E000 | | | | | | | | | | | | | | | |
| **MPUTYPE, type RO, offset 0xD90, reset 0x0000.0800** | | | | | | | | | | | | | | | |
| | | | | | | | | IREGION | | | | | | | |
| DREGION | | | | | | | | | | | | | | | SEPARATE |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **MPUCTRL, type R/W, offset 0xD94, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | PRIVDEFEN | HFNMIENA | ENABLE |
| **MPUNUMBER, type R/W, offset 0xD98, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | NUMBER | | |
| **MPUBASE, type R/W, offset 0xD9C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | VALID | | REGION | | |
| **MPUBASE1, type R/W, offset 0xDA4, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | VALID | | REGION | | |
| **MPUBASE2, type R/W, offset 0xDAC, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | VALID | | REGION | | |
| **MPUBASE3, type R/W, offset 0xDB4, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | VALID | | REGION | | |
| **MPUATTR, type R/W, offset 0xDA0, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | XN | | AP | | | | | TEX | | | S | C | B |
| SRD | | | | | | | | | | SIZE | | | | | ENABLE |
| **MPUATTR1, type R/W, offset 0xDA8, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | XN | | AP | | | | | TEX | | | S | C | B |
| SRD | | | | | | | | | | SIZE | | | | | ENABLE |
| **MPUATTR2, type R/W, offset 0xDB0, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | XN | | AP | | | | | TEX | | | S | C | B |
| SRD | | | | | | | | | | SIZE | | | | | ENABLE |
| **MPUATTR3, type R/W, offset 0xDB8, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | XN | | AP | | | | | TEX | | | S | C | B |
| SRD | | | | | | | | | | SIZE | | | | | ENABLE |
| **System Control**<br>Base 0x400F.E000 | | | | | | | | | | | | | | | |
| **DID0, type RO, offset 0x000, reset -** (see page 186) | | | | | | | | | | | | | | | |
| | VER | | | | | | | CLASS | | | | | | | |
| MAJOR | | | | | | | | MINOR | | | | | | | |
| **PBORCTL, type R/W, offset 0x030, reset 0x0000.7FFD** (see page 188) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | BORIOR | |
| **LDOPCTL, type R/W, offset 0x034, reset 0x0000.0000** (see page 189) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | VADJ | | | | | |
| **RIS, type RO, offset 0x050, reset 0x0000.0000** (see page 190) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | MOSCPUPRIS | USBPLLLRIS | PLLLRIS | | | | | BORRIS | |
| **IMC, type R/W, offset 0x054, reset 0x0000.0000** (see page 191) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | MOSCPUPIM | USBPLLLIM | PLLLIM | | | | | BORIM | |
| **MISC, type R/W1C, offset 0x058, reset 0x0000.0000** (see page 192) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | MOSCPUPMIS | USBPLLLMIS | PLLLMIS | | | | | BORMIS | |
| **RESC, type R/W, offset 0x05C, reset -** (see page 193) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | MOSCFAIL |
| | | | | | | | | | | | SW | WDT | BOR | POR | EXT |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **RCC, type R/W, offset 0x060, reset 0x0780.3AD1** (see page 194) | | | | | | | | | | | | | | | |
| | | | | ACG | SYSDIV | | | | USESYSDIV | | | | | | |
| | | PWRDN | | BYPASS | XTAL | | | | | OSCSRC | | | | IOSCDIS | MOSCDIS |
| **PLLCFG, type RO, offset 0x064, reset -** (see page 198) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | F | | | | | | | | | R | | | | |
| **GPIOHBCTL, type R/W, offset 0x06C, reset 0x0000.0000** (see page 199) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | PORTE | PORTD | PORTC | PORTB | PORTA |
| **RCC2, type R/W, offset 0x070, reset 0x0780.6810** (see page 201) | | | | | | | | | | | | | | | |
| USERCC2 | | | SYSDIV2 | | | | | | | | | | | | |
| | USBPWRDN | PWRDN2 | | BYPASS2 | | | | | OSCSRC2 | | | | | | |
| **MOSCCTL, type R/W, offset 0x07C, reset 0x0000.0000** (see page 203) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | CVAL |
| **DSLPCLKCFG, type R/W, offset 0x144, reset 0x0780.0000** (see page 204) | | | | | | | | | | | | | | | |
| | | | DSDIVORIDE | | | | | | | | | | | | |
| | | | | | | | | | DSOSCSRC | | | | | | |
| **DID1, type RO, offset 0x004, reset -** (see page 205) | | | | | | | | | | | | | | | |
| VER | | | | FAM | | | | PARTNO | | | | | | | |
| PINCOUNT | | | | | | | | TEMP | | | PKG | | ROHS | QUAL | |
| **DC0, type RO, offset 0x008, reset 0x007F.003F** (see page 207) | | | | | | | | | | | | | | | |
| SRAMSZ | | | | | | | | | | | | | | | |
| FLASHSZ | | | | | | | | | | | | | | | |
| **DC1, type RO, offset 0x010, reset 0x0101.32FF** (see page 208) | | | | | | | | | | | | | | | |
| | | | | | | | CAN0 | | | | | | | | ADC |
| MINSYSDIV | | | | | | MAXADCSPD | | MPU | HIB | TEMPSNS | PLL | WDT | SWO | SWD | JTAG |
| **DC2, type RO, offset 0x014, reset 0x0007.5013** (see page 210) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| | I2C1 | | I2C0 | | | | | | | | SSI0 | | | UART1 | UART0 |
| **DC3, type RO, offset 0x018, reset 0x9F3F.0000** (see page 211) | | | | | | | | | | | | | | | |
| 32KHZ | | | CCP4 | CCP3 | CCP2 | CCP1 | CCP0 | | | ADC5 | ADC4 | ADC3 | ADC2 | ADC1 | ADC0 |
| | | | | | | | | | | | | | | | |
| **DC4, type RO, offset 0x01C, reset 0x0000.301F** (see page 212) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | UDMA | ROM | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| **DC5, type RO, offset 0x020, reset 0x0000.0000** (see page 213) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| **DC6, type RO, offset 0x024, reset 0x0000.0002** (see page 214) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | USB0 | |
| **DC7, type RO, offset 0x028, reset 0x40C0.0F3F** (see page 215) | | | | | | | | | | | | | | | |
| | SW | | | | | | | UART1_TX | UART1_RX | | | | | | |
| | | | | SSI0_TX | SSI0_RX | UART0_TX | UART0_RX | | | USB_EP3_TX | USB_EP3_RX | USB_EP2_TX | USB_EP2_RX | USB_EP1_TX | USB_EP1_RX |
| **RCGC0, type R/W, offset 0x100, reset 0x00000040** (see page 217) | | | | | | | | | | | | | | | |
| | | | | | | | CAN0 | | | | | | | | ADC |
| | | | | | | MAXADCSPD | | | HIB | | | WDT | | | |
| **SCGC0, type R/W, offset 0x110, reset 0x00000040** (see page 219) | | | | | | | | | | | | | | | |
| | | | | | | | CAN0 | | | | | | | | ADC |
| | | | | | | MAXADCSPD | | | HIB | | | WDT | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **DCGC0, type R/W, offset 0x120, reset 0x00000040** (see page 221) | | | | | | | | | | | | | | | |
| | | | | | | | CAN0 | | | | | | | | ADC |
| | | | | | | | | | HIB | | | WDT | | | |
| **RCGC1, type R/W, offset 0x104, reset 0x00000000** (see page 223) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| | I2C1 | | I2C0 | | | | | | | | SSI0 | | | UART1 | UART0 |
| **SCGC1, type R/W, offset 0x114, reset 0x00000000** (see page 225) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| | I2C1 | | I2C0 | | | | | | | | SSI0 | | | UART1 | UART0 |
| **DCGC1, type R/W, offset 0x124, reset 0x00000000** (see page 227) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| | I2C1 | | I2C0 | | | | | | | | SSI0 | | | UART1 | UART0 |
| **RCGC2, type R/W, offset 0x108, reset 0x00000000** (see page 229) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | USB0 |
| | | UDMA | | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| **SCGC2, type R/W, offset 0x118, reset 0x00000000** (see page 231) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | USB0 |
| | | UDMA | | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| **DCGC2, type R/W, offset 0x128, reset 0x00000000** (see page 233) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | USB0 |
| | | UDMA | | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| **SRCR0, type R/W, offset 0x040, reset 0x00000000** (see page 235) | | | | | | | | | | | | | | | |
| | | | | | | | CAN0 | | | | | | | | ADC |
| | | | | | | | | | HIB | | | WDT | | | |
| **SRCR1, type R/W, offset 0x044, reset 0x00000000** (see page 236) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| | I2C1 | | I2C0 | | | | | | | | SSI0 | | | UART1 | UART0 |
| **SRCR2, type R/W, offset 0x048, reset 0x00000000** (see page 237) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | USB0 |
| | | UDMA | | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| **Hibernation Module**<br>Base 0x400F.C000 | | | | | | | | | | | | | | | |
| **HIBRTCC, type RO, offset 0x000, reset 0x0000.0000** (see page 247) | | | | | | | | | | | | | | | |
| RTCC | | | | | | | | | | | | | | | |
| RTCC | | | | | | | | | | | | | | | |
| **HIBRTCM0, type R/W, offset 0x004, reset 0xFFFF.FFFF** (see page 248) | | | | | | | | | | | | | | | |
| RTCM0 | | | | | | | | | | | | | | | |
| RTCM0 | | | | | | | | | | | | | | | |
| **HIBRTCM1, type R/W, offset 0x008, reset 0xFFFF.FFFF** (see page 249) | | | | | | | | | | | | | | | |
| RTCM1 | | | | | | | | | | | | | | | |
| RTCM1 | | | | | | | | | | | | | | | |
| **HIBRTCLD, type R/W, offset 0x00C, reset 0xFFFF.FFFF** (see page 250) | | | | | | | | | | | | | | | |
| RTCLD | | | | | | | | | | | | | | | |
| RTCLD | | | | | | | | | | | | | | | |
| **HIBCTL, type R/W, offset 0x010, reset 0x8000.0000** (see page 251) | | | | | | | | | | | | | | | |
| WRC | | | | | | | | | | | | | | | |
| | | | | | | | | VABORT | CLK32EN | LOWBATEN | PINWEN | RTCWEN | CLKSEL | HIBREQ | RTCEN |
| **HIBIM, type R/W, offset 0x014, reset 0x0000.0000** (see page 254) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EXTW | LOWBAT | RTCALT1 | RTCALT0 |
| **HIBRIS, type RO, offset 0x018, reset 0x0000.0000** (see page 255) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EXTW | LOWBAT | RTCALT1 | RTCALT0 |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **HIBMIS, type RO, offset 0x01C, reset 0x0000.0000** (see page 256) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EXTW | LOWBAT | RTCALT1 | RTCALT0 |
| **HIBIC, type R/W1C, offset 0x020, reset 0x0000.0000** (see page 257) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EXTW | LOWBAT | RTCALT1 | RTCALT0 |
| **HIBRTCT, type R/W, offset 0x024, reset 0x0000.7FFF** (see page 258) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| TRIM | | | | | | | | | | | | | | | |
| **HIBDATA, type R/W, offset 0x030-0x12C, reset -** (see page 259) | | | | | | | | | | | | | | | |
| RTD | | | | | | | | | | | | | | | |
| RTD | | | | | | | | | | | | | | | |
| **Internal Memory**<br>**ROM Registers (System Control Offset)**<br>Base 0x400F.E000 | | | | | | | | | | | | | | | |
| **RMCTL, type R/W1C, offset 0x0F0, reset -** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | BA |
| **Internal Memory**<br>**Flash Memory Control Registers (Flash Control Offset)**<br>Base 0x400F.D000 | | | | | | | | | | | | | | | |
| **FMA, type R/W, offset 0x000, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | OFFSET |
| OFFSET | | | | | | | | | | | | | | | |
| **FMD, type R/W, offset 0x004, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **FMC, type R/W, offset 0x008, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| WRKEY | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | COMT | MERASE | ERASE | WRITE |
| **FCRIS, type RO, offset 0x00C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | PRIS | ARIS |
| **FCIM, type R/W, offset 0x010, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | PMASK | AMASK |
| **FCMISC, type R/W1C, offset 0x014, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | PMISC | AMISC |
| **Internal Memory**<br>**Flash Memory Protection Registers (System Control Offset)**<br>Base 0x400F.E000 | | | | | | | | | | | | | | | |
| **USECRL, type R/W, offset 0x140, reset 0x31** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | USEC | | | | | | | |
| **FMPRE0, type R/W, offset 0x130 and 0x200, reset 0xFFFF.FFFF** | | | | | | | | | | | | | | | |
| READ_ENABLE | | | | | | | | | | | | | | | |
| READ_ENABLE | | | | | | | | | | | | | | | |
| **FMPPE0, type R/W, offset 0x134 and 0x400, reset 0xFFFF.FFFF** | | | | | | | | | | | | | | | |
| PROG_ENABLE | | | | | | | | | | | | | | | |
| PROG_ENABLE | | | | | | | | | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **USER_DBG, type R/W, offset 0x1D0, reset 0xFFFF.FFFE** | | | | | | | | | | | | | | | |
| NW | DATA | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | DBG1 | DBG0 |
| **USER_REG0, type R/W, offset 0x1E0, reset 0xFFFF.FFFF** | | | | | | | | | | | | | | | |
| NW | DATA | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **USER_REG1, type R/W, offset 0x1E4, reset 0xFFFF.FFFF** | | | | | | | | | | | | | | | |
| NW | DATA | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **USER_REG2, type R/W, offset 0x1E8, reset 0xFFFF.FFFF** | | | | | | | | | | | | | | | |
| NW | DATA | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **USER_REG3, type R/W, offset 0x1EC, reset 0xFFFF.FFFF** | | | | | | | | | | | | | | | |
| NW | DATA | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **FMPRE1, type R/W, offset 0x204, reset 0xFFFF.FFFF** | | | | | | | | | | | | | | | |
| READ_ENABLE | | | | | | | | | | | | | | | |
| READ_ENABLE | | | | | | | | | | | | | | | |
| **FMPRE2, type R/W, offset 0x208, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| READ_ENABLE | | | | | | | | | | | | | | | |
| READ_ENABLE | | | | | | | | | | | | | | | |
| **FMPRE3, type R/W, offset 0x20C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| READ_ENABLE | | | | | | | | | | | | | | | |
| READ_ENABLE | | | | | | | | | | | | | | | |
| **FMPPE1, type R/W, offset 0x404, reset 0xFFFF.FFFF** | | | | | | | | | | | | | | | |
| PROG_ENABLE | | | | | | | | | | | | | | | |
| PROG_ENABLE | | | | | | | | | | | | | | | |
| **FMPPE2, type R/W, offset 0x408, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| PROG_ENABLE | | | | | | | | | | | | | | | |
| PROG_ENABLE | | | | | | | | | | | | | | | |
| **FMPPE3, type R/W, offset 0x40C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| PROG_ENABLE | | | | | | | | | | | | | | | |
| PROG_ENABLE | | | | | | | | | | | | | | | |

### Micro Direct Memory Access (µDMA)

### µDMA Channel Control Structure

Base n/a

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **DMASRCENDP, type R/W, offset 0x000, reset -** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| **DMADSTENDP, type R/W, offset 0x004, reset -** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| **DMACHCTL, type R/W, offset 0x008, reset -** | | | | | | | | | | | | | | | |
| DSTINC | | DSTSIZE | | SRCINC | | SRCSIZE | | | | | | | | ARBSIZE | |
| ARBSIZE | | | | XFERSIZE | | | | | | | | NXTUSEBURST | | XFERMODE | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Micro Direct Memory Access (µDMA)**<br>**µDMA Registers**<br>Base 0x400F.F000 | | | | | | | | | | | | | | | |
| **DMASTAT, type RO, offset 0x000, reset 0x001F.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | DMACHANS | | | | |
| | | | | | | | | STATE | | | | | | | MASTEN |
| **DMACFG, type WO, offset 0x004, reset -** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | MASTEN |
| **DMACTLBASE, type R/W, offset 0x008, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| **DMAALTBASE, type RO, offset 0x00C, reset 0x0000.0200** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| **DMAWAITSTAT, type RO, offset 0x010, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| WAITREQ[n] | | | | | | | | | | | | | | | |
| WAITREQ[n] | | | | | | | | | | | | | | | |
| **DMASWREQ, type WO, offset 0x014, reset -** | | | | | | | | | | | | | | | |
| SWREQ[n] | | | | | | | | | | | | | | | |
| SWREQ[n] | | | | | | | | | | | | | | | |
| **DMAUSEBURSTSET, type RO, offset 0x018, reset 0x0000.0000 (Reads)** | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| **DMAUSEBURSTSET, type WO, offset 0x018, reset 0x0000.0000 (Writes)** | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| **DMAUSEBURSTCLR, type WO, offset 0x01C, reset -** | | | | | | | | | | | | | | | |
| CLR[n] | | | | | | | | | | | | | | | |
| CLR[n] | | | | | | | | | | | | | | | |
| **DMAREQMASKSET, type RO, offset 0x020, reset 0x0000.0000 (Reads)** | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| **DMAREQMASKSET, type WO, offset 0x020, reset 0x0000.0000 (Writes)** | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| **DMAREQMASKCLR, type WO, offset 0x024, reset -** | | | | | | | | | | | | | | | |
| CLR[n] | | | | | | | | | | | | | | | |
| CLR[n] | | | | | | | | | | | | | | | |
| **DMAENASET, type RO, offset 0x028, reset 0x0000.0000 (Reads)** | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| **DMAENASET, type WO, offset 0x028, reset 0x0000.0000 (Writes)** | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| **DMAENACLR, type WO, offset 0x02C, reset -** | | | | | | | | | | | | | | | |
| CLR[n] | | | | | | | | | | | | | | | |
| CLR[n] | | | | | | | | | | | | | | | |
| **DMAALTSET, type RO, offset 0x030, reset 0x0000.0000 (Reads)** | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **DMAALTSET, type WO, offset 0x030, reset 0x0000.0000 (Writes)** | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| **DMAALTCLR, type WO, offset 0x034, reset -** | | | | | | | | | | | | | | | |
| CLR[n] | | | | | | | | | | | | | | | |
| CLR[n] | | | | | | | | | | | | | | | |
| **DMAPRIOSET, type RO, offset 0x038, reset 0x0000.0000 (Reads)** | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| **DMAPRIOSET, type WO, offset 0x038, reset 0x0000.0000 (Writes)** | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| SET[n] | | | | | | | | | | | | | | | |
| **DMAPRIOCLR, type WO, offset 0x03C, reset -** | | | | | | | | | | | | | | | |
| CLR[n] | | | | | | | | | | | | | | | |
| CLR[n] | | | | | | | | | | | | | | | |
| **DMAERRCLR, type RO, offset 0x04C, reset 0x0000.0000 (Reads)** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | ERRCLR |
| **DMAERRCLR, type WO, offset 0x04C, reset 0x0000.0000 (Writes)** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | ERRCLR |
| **DMAPeriphID0, type RO, offset 0xFE0, reset 0x0000.0030** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID0 | | | | | | | |
| **DMAPeriphID1, type RO, offset 0xFE4, reset 0x0000.00B2** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID1 | | | | | | | |
| **DMAPeriphID2, type RO, offset 0xFE8, reset 0x0000.000B** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID2 | | | | | | | |
| **DMAPeriphID3, type RO, offset 0xFEC, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID3 | | | | | | | |
| **DMAPeriphID4, type RO, offset 0xFD0, reset 0x0000.0004** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID4 | | | | | | | |
| **DMAPCellID0, type RO, offset 0xFF0, reset 0x0000.000D** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID0 | | | | | | | |
| **DMAPCellID1, type RO, offset 0xFF4, reset 0x0000.00F0** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID1 | | | | | | | |
| **DMAPCellID2, type RO, offset 0xFF8, reset 0x0000.0005** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID2 | | | | | | | |
| **DMAPCellID3, type RO, offset 0xFFC, reset 0x0000.00B1** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID3 | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **General-Purpose Input/Outputs (GPIOs)**<br>GPIO Port A (APB) base: 0x4000.4000<br>GPIO Port A (AHB) base: 0x4005.8000<br>GPIO Port B (APB) base: 0x4000.5000<br>GPIO Port B (AHB) base: 0x4005.9000<br>GPIO Port C (APB) base: 0x4000.6000<br>GPIO Port C (AHB) base: 0x4005.A000<br>GPIO Port D (APB) base: 0x4000.7000<br>GPIO Port D (AHB) base: 0x4005.B000<br>GPIO Port E (APB) base: 0x4002.4000<br>GPIO Port E (AHB) base: 0x4005.C000 | | | | | | | | | | | | | | | |
| **GPIODATA, type R/W, offset 0x000, reset 0x0000.0000** (see page 362) | | | | | | | | | | | | | | | |
| | | | | | | | | DATA | | | | | | | |
| **GPIODIR, type R/W, offset 0x400, reset 0x0000.0000** (see page 363) | | | | | | | | | | | | | | | |
| | | | | | | | | DIR | | | | | | | |
| **GPIOIS, type R/W, offset 0x404, reset 0x0000.0000** (see page 364) | | | | | | | | | | | | | | | |
| | | | | | | | | IS | | | | | | | |
| **GPIOIBE, type R/W, offset 0x408, reset 0x0000.0000** (see page 365) | | | | | | | | | | | | | | | |
| | | | | | | | | IBE | | | | | | | |
| **GPIOIEV, type R/W, offset 0x40C, reset 0x0000.0000** (see page 366) | | | | | | | | | | | | | | | |
| | | | | | | | | IEV | | | | | | | |
| **GPIOIM, type R/W, offset 0x410, reset 0x0000.0000** (see page 367) | | | | | | | | | | | | | | | |
| | | | | | | | | IME | | | | | | | |
| **GPIORIS, type RO, offset 0x414, reset 0x0000.0000** (see page 368) | | | | | | | | | | | | | | | |
| | | | | | | | | RIS | | | | | | | |
| **GPIOMIS, type RO, offset 0x418, reset 0x0000.0000** (see page 369) | | | | | | | | | | | | | | | |
| | | | | | | | | MIS | | | | | | | |
| **GPIOICR, type W1C, offset 0x41C, reset 0x0000.0000** (see page 370) | | | | | | | | | | | | | | | |
| | | | | | | | | IC | | | | | | | |
| **GPIOAFSEL, type R/W, offset 0x420, reset -** (see page 371) | | | | | | | | | | | | | | | |
| | | | | | | | | AFSEL | | | | | | | |
| **GPIODR2R, type R/W, offset 0x500, reset 0x0000.00FF** (see page 373) | | | | | | | | | | | | | | | |
| | | | | | | | | DRV2 | | | | | | | |
| **GPIODR4R, type R/W, offset 0x504, reset 0x0000.0000** (see page 374) | | | | | | | | | | | | | | | |
| | | | | | | | | DRV4 | | | | | | | |
| **GPIODR8R, type R/W, offset 0x508, reset 0x0000.0000** (see page 375) | | | | | | | | | | | | | | | |
| | | | | | | | | DRV8 | | | | | | | |
| **GPIOODR, type R/W, offset 0x50C, reset 0x0000.0000** (see page 376) | | | | | | | | | | | | | | | |
| | | | | | | | | ODE | | | | | | | |
| **GPIOPUR, type R/W, offset 0x510, reset -** (see page 377) | | | | | | | | | | | | | | | |
| | | | | | | | | PUE | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **GPIOPDR, type R/W, offset 0x514, reset 0x0000.0000** (see page 379) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PDE | | | | | | | |
| **GPIOSLR, type R/W, offset 0x518, reset 0x0000.0000** (see page 380) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | SRL | | | | | | | |
| **GPIODEN, type R/W, offset 0x51C, reset -** (see page 381) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | DEN | | | | | | | |
| **GPIOLOCK, type R/W, offset 0x520, reset 0x0000.0001** (see page 383) | | | | | | | | | | | | | | | |
| LOCK | | | | | | | | | | | | | | | |
| LOCK | | | | | | | | | | | | | | | |
| **GPIOCR, type -, offset 0x524, reset -** (see page 384) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CR | | | | | | | |
| **GPIOAMSEL, type R/W, offset 0x528, reset 0x0000.0000** (see page 386) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | GPIOAMSEL | | | |
| **GPIOPeriphID4, type RO, offset 0xFD0, reset 0x0000.0000** (see page 387) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID4 | | | | | | | |
| **GPIOPeriphID5, type RO, offset 0xFD4, reset 0x0000.0000** (see page 388) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID5 | | | | | | | |
| **GPIOPeriphID6, type RO, offset 0xFD8, reset 0x0000.0000** (see page 389) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID6 | | | | | | | |
| **GPIOPeriphID7, type RO, offset 0xFDC, reset 0x0000.0000** (see page 390) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID7 | | | | | | | |
| **GPIOPeriphID0, type RO, offset 0xFE0, reset 0x0000.0061** (see page 391) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID0 | | | | | | | |
| **GPIOPeriphID1, type RO, offset 0xFE4, reset 0x0000.0000** (see page 392) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID1 | | | | | | | |
| **GPIOPeriphID2, type RO, offset 0xFE8, reset 0x0000.0018** (see page 393) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID2 | | | | | | | |
| **GPIOPeriphID3, type RO, offset 0xFEC, reset 0x0000.0001** (see page 394) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID3 | | | | | | | |
| **GPIOPCellID0, type RO, offset 0xFF0, reset 0x0000.000D** (see page 395) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID0 | | | | | | | |
| **GPIOPCellID1, type RO, offset 0xFF4, reset 0x0000.00F0** (see page 396) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID1 | | | | | | | |
| **GPIOPCellID2, type RO, offset 0xFF8, reset 0x0000.0005** (see page 397) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID2 | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **GPIOPCellID3, type RO, offset 0xFFC, reset 0x0000.00B1** (see page 398) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID3 | | | | | | | |
| **General-Purpose Timers**<br>Timer0 base: 0x4003.0000<br>Timer1 base: 0x4003.1000<br>Timer2 base: 0x4003.2000 | | | | | | | | | | | | | | | |
| **GPTMCFG, type R/W, offset 0x000, reset 0x0000.0000** (see page 411) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | GPTMCFG | | |
| **GPTMTAMR, type R/W, offset 0x004, reset 0x0000.0000** (see page 412) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TAAMS | TACMR | TAMR | |
| **GPTMTBMR, type R/W, offset 0x008, reset 0x0000.0000** (see page 414) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TBAMS | TBCMR | TBMR | |
| **GPTMCTL, type R/W, offset 0x00C, reset 0x0000.0000** (see page 416) | | | | | | | | | | | | | | | |
| | TBPWML | TBOTE | | TBEVENT | | TBSTALL | TBEN | | TAPWML | TAOTE | RTCEN | TAEVENT | | TASTALL | TAEN |
| **GPTMIMR, type R/W, offset 0x018, reset 0x0000.0000** (see page 419) | | | | | | | | | | | | | | | |
| | | | | | CBEIM | CBMIM | TBTOIM | | | | | RTCIM | CAEIM | CAMIM | TATOIM |
| **GPTMRIS, type RO, offset 0x01C, reset 0x0000.0000** (see page 421) | | | | | | | | | | | | | | | |
| | | | | | CBERIS | CBMRIS | TBTORIS | | | | | RTCRIS | CAERIS | CAMRIS | TATORIS |
| **GPTMMIS, type RO, offset 0x020, reset 0x0000.0000** (see page 422) | | | | | | | | | | | | | | | |
| | | | | | CBEMIS | CBMMIS | TBTOMIS | | | | | RTCMIS | CAEMIS | CAMMIS | TATOMIS |
| **GPTMICR, type W1C, offset 0x024, reset 0x0000.0000** (see page 423) | | | | | | | | | | | | | | | |
| | | | | | CBECINT | CBMCINT | TBTOCINT | | | | | RTCCINT | CAECINT | CAMCINT | TATOCINT |
| **GPTMTAILR, type R/W, offset 0x028, reset 0xFFFF.FFFF** (see page 425) | | | | | | | | | | | | | | | |
| | | | | | | | | TAILRH | | | | | | | |
| | | | | | | | | TAILRL | | | | | | | |
| **GPTMTBILR, type R/W, offset 0x02C, reset 0x0000.FFFF** (see page 426) | | | | | | | | | | | | | | | |
| | | | | | | | | TBILRL | | | | | | | |
| **GPTMTAMATCHR, type R/W, offset 0x030, reset 0xFFFF.FFFF** (see page 427) | | | | | | | | | | | | | | | |
| | | | | | | | | TAMRH | | | | | | | |
| | | | | | | | | TAMRL | | | | | | | |
| **GPTMTBMATCHR, type R/W, offset 0x034, reset 0x0000.FFFF** (see page 428) | | | | | | | | | | | | | | | |
| | | | | | | | | TBMRL | | | | | | | |
| **GPTMTAPR, type R/W, offset 0x038, reset 0x0000.0000** (see page 429) | | | | | | | | | | | | | | | |
| | | | | | | | | TAPSR | | | | | | | |
| **GPTMTBPR, type R/W, offset 0x03C, reset 0x0000.0000** (see page 430) | | | | | | | | | | | | | | | |
| | | | | | | | | TBPSR | | | | | | | |
| **GPTMTAR, type RO, offset 0x048, reset 0xFFFF.FFFF** (see page 431) | | | | | | | | | | | | | | | |
| | | | | | | | | TARH | | | | | | | |
| | | | | | | | | TARL | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **GPTMTBR, type RO, offset 0x04C, reset 0x0000.FFFF** (see page 432) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| TBRL | | | | | | | | | | | | | | | |
| **Watchdog Timer**<br>Base 0x4000.0000 | | | | | | | | | | | | | | | |
| **WDTLOAD, type R/W, offset 0x000, reset 0xFFFF.FFFF** (see page 437) | | | | | | | | | | | | | | | |
| WDTLoad | | | | | | | | | | | | | | | |
| WDTLoad | | | | | | | | | | | | | | | |
| **WDTVALUE, type RO, offset 0x004, reset 0xFFFF.FFFF** (see page 438) | | | | | | | | | | | | | | | |
| WDTValue | | | | | | | | | | | | | | | |
| WDTValue | | | | | | | | | | | | | | | |
| **WDTCTL, type R/W, offset 0x008, reset 0x0000.0000** (see page 439) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | RESEN | INTEN |
| **WDTICR, type WO, offset 0x00C, reset -** (see page 440) | | | | | | | | | | | | | | | |
| WDTIntClr | | | | | | | | | | | | | | | |
| WDTIntClr | | | | | | | | | | | | | | | |
| **WDTRIS, type RO, offset 0x010, reset 0x0000.0000** (see page 441) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | WDTRIS |
| **WDTMIS, type RO, offset 0x014, reset 0x0000.0000** (see page 442) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | WDTMIS |
| **WDTTEST, type R/W, offset 0x418, reset 0x0000.0000** (see page 443) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | STALL | | | | | | | | |
| **WDTLOCK, type R/W, offset 0xC00, reset 0x0000.0000** (see page 444) | | | | | | | | | | | | | | | |
| WDTLock | | | | | | | | | | | | | | | |
| WDTLock | | | | | | | | | | | | | | | |
| **WDTPeriphID4, type RO, offset 0xFD0, reset 0x0000.0000** (see page 445) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID4 | | | | | | | |
| **WDTPeriphID5, type RO, offset 0xFD4, reset 0x0000.0000** (see page 446) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID5 | | | | | | | |
| **WDTPeriphID6, type RO, offset 0xFD8, reset 0x0000.0000** (see page 447) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID6 | | | | | | | |
| **WDTPeriphID7, type RO, offset 0xFDC, reset 0x0000.0000** (see page 448) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID7 | | | | | | | |
| **WDTPeriphID0, type RO, offset 0xFE0, reset 0x0000.0005** (see page 449) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID0 | | | | | | | |
| **WDTPeriphID1, type RO, offset 0xFE4, reset 0x0000.0018** (see page 450) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID1 | | | | | | | |
| **WDTPeriphID2, type RO, offset 0xFE8, reset 0x0000.0018** (see page 451) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID2 | | | | | | | |
| **WDTPeriphID3, type RO, offset 0xFEC, reset 0x0000.0001** (see page 452) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID3 | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **WDTPCellID0, type RO, offset 0xFF0, reset 0x0000.000D** (see page 453) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID0 | | | | | | | |
| **WDTPCellID1, type RO, offset 0xFF4, reset 0x0000.00F0** (see page 454) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID1 | | | | | | | |
| **WDTPCellID2, type RO, offset 0xFF8, reset 0x0000.0005** (see page 455) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID2 | | | | | | | |
| **WDTPCellID3, type RO, offset 0xFFC, reset 0x0000.00B1** (see page 456) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID3 | | | | | | | |
| **Analog-to-Digital Converter (ADC)**<br>Base 0x4003.8000 | | | | | | | | | | | | | | | |
| **ADCACTSS, type R/W, offset 0x000, reset 0x0000.0000** (see page 467) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | ASEN3 | ASEN2 | ASEN1 | ASEN0 |
| **ADCRIS, type RO, offset 0x004, reset 0x0000.0000** (see page 468) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | INR3 | INR2 | INR1 | INR0 |
| **ADCIM, type R/W, offset 0x008, reset 0x0000.0000** (see page 469) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | MASK3 | MASK2 | MASK1 | MASK0 |
| **ADCISC, type R/W1C, offset 0x00C, reset 0x0000.0000** (see page 470) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | IN3 | IN2 | IN1 | IN0 |
| **ADCOSTAT, type R/W1C, offset 0x010, reset 0x0000.0000** (see page 471) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | OV3 | OV2 | OV1 | OV0 |
| **ADCEMUX, type R/W, offset 0x014, reset 0x0000.0000** (see page 472) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| EM3 | | | | EM2 | | | | EM1 | | | | EM0 | | | |
| **ADCUSTAT, type R/W1C, offset 0x018, reset 0x0000.0000** (see page 475) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | UV3 | UV2 | UV1 | UV0 |
| **ADCSSPRI, type R/W, offset 0x020, reset 0x0000.3210** (see page 476) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | SS3 | | | | SS2 | | | | SS1 | | | | SS0 | |
| **ADCPSSI, type WO, offset 0x028, reset -** (see page 478) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | SS3 | SS2 | SS1 | SS0 |
| **ADCSAC, type R/W, offset 0x030, reset 0x0000.0000** (see page 479) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | AVG | | |
| **ADCSSMUX0, type R/W, offset 0x040, reset 0x0000.0000** (see page 480) | | | | | | | | | | | | | | | |
| | | MUX7 | | | | MUX6 | | | | MUX5 | | | | MUX4 | |
| | | MUX3 | | | | MUX2 | | | | MUX1 | | | | MUX0 | |
| **ADCSSCTL0, type R/W, offset 0x044, reset 0x0000.0000** (see page 482) | | | | | | | | | | | | | | | |
| TS7 | IE7 | END7 | D7 | TS6 | IE6 | END6 | D6 | TS5 | IE5 | END5 | D5 | TS4 | IE4 | END4 | D4 |
| TS3 | IE3 | END3 | D3 | TS2 | IE2 | END2 | D2 | TS1 | IE1 | END1 | D1 | TS0 | IE0 | END0 | D0 |
| **ADCSSFIFO0, type RO, offset 0x048, reset -** (see page 485) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | DATA | | | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **ADCSSFIFO1, type RO, offset 0x068, reset -** (see page 485) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | DATA | | | | | | | | | |
| **ADCSSFIFO2, type RO, offset 0x088, reset -** (see page 485) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | DATA | | | | | | | | | |
| **ADCSSFIFO3, type RO, offset 0x0A8, reset -** (see page 485) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | DATA | | | | | | | | | |
| **ADCSSFSTAT0, type RO, offset 0x04C, reset 0x0000.0100** (see page 486) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | FULL | | | | EMPTY | | | HPTR | | | | TPTR | |
| **ADCSSFSTAT1, type RO, offset 0x06C, reset 0x0000.0100** (see page 486) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | FULL | | | | EMPTY | | | HPTR | | | | TPTR | |
| **ADCSSFSTAT2, type RO, offset 0x08C, reset 0x0000.0100** (see page 486) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | FULL | | | | EMPTY | | | HPTR | | | | TPTR | |
| **ADCSSFSTAT3, type RO, offset 0x0AC, reset 0x0000.0100** (see page 486) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | FULL | | | | EMPTY | | | HPTR | | | | TPTR | |
| **ADCSSMUX1, type R/W, offset 0x060, reset 0x0000.0000** (see page 487) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | MUX3 | | | | MUX2 | | | | MUX1 | | | | MUX0 | | |
| **ADCSSMUX2, type R/W, offset 0x080, reset 0x0000.0000** (see page 487) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | MUX3 | | | | MUX2 | | | | MUX1 | | | | MUX0 | | |
| **ADCSSCTL1, type R/W, offset 0x064, reset 0x0000.0000** (see page 488) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| TS3 | IE3 | END3 | D3 | TS2 | IE2 | END2 | D2 | TS1 | IE1 | END1 | D1 | TS0 | IE0 | END0 | D0 |
| **ADCSSCTL2, type R/W, offset 0x084, reset 0x0000.0000** (see page 488) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| TS3 | IE3 | END3 | D3 | TS2 | IE2 | END2 | D2 | TS1 | IE1 | END1 | D1 | TS0 | IE0 | END0 | D0 |
| **ADCSSMUX3, type R/W, offset 0x0A0, reset 0x0000.0000** (see page 490) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | MUX0 | |
| **ADCSSCTL3, type R/W, offset 0x0A4, reset 0x0000.0002** (see page 491) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TS0 | IE0 | END0 | D0 |
| **Universal Asynchronous Receivers/Transmitters (UARTs)**<br>UART0 base: 0x4000.C000<br>UART1 base: 0x4000.D000 | | | | | | | | | | | | | | | |
| **UARTDR, type R/W, offset 0x000, reset 0x0000.0000** (see page 502) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | OE | BE | PE | FE | | | DATA | | | | | |
| **UARTRSR/UARTECR, type RO, offset 0x004, reset 0x0000.0000 (Reads)** (see page 504) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | OE | BE | PE | FE |
| **UARTRSR/UARTECR, type WO, offset 0x004, reset 0x0000.0000 (Writes)** (see page 504) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | DATA | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **UARTFR, type RO, offset 0x018, reset 0x0000.0090** (see page 506) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | TXFE | RXFF | TXFF | RXFE | BUSY | | | |
| **UARTILPR, type R/W, offset 0x020, reset 0x0000.0000** (see page 508) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | ILPDVSR | | | | | | | |
| **UARTIBRD, type R/W, offset 0x024, reset 0x0000.0000** (see page 509) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| DIVINT | | | | | | | | | | | | | | | |
| **UARTFBRD, type R/W, offset 0x028, reset 0x0000.0000** (see page 510) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | DIVFRAC | | | | | |
| **UARTLCRH, type R/W, offset 0x02C, reset 0x0000.0000** (see page 511) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | SPS | WLEN | | FEN | STP2 | EPS | PEN | BRK |
| **UARTCTL, type R/W, offset 0x030, reset 0x0000.0300** (see page 513) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | RXE | TXE | LBE | | | | | SIRLP | SIREN | UARTEN |
| **UARTIFLS, type R/W, offset 0x034, reset 0x0000.0012** (see page 515) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | RXIFLSEL | | | TXIFLSEL | | |
| **UARTIM, type R/W, offset 0x038, reset 0x0000.0000** (see page 517) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | OEIM | BEIM | PEIM | FEIM | RTIM | TXIM | RXIM | | | | |
| **UARTRIS, type RO, offset 0x03C, reset 0x0000.0000** (see page 519) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | OERIS | BERIS | PERIS | FERIS | RTRIS | TXRIS | RXRIS | | | | |
| **UARTMIS, type RO, offset 0x040, reset 0x0000.0000** (see page 520) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | OEMIS | BEMIS | PEMIS | FEMIS | RTMIS | TXMIS | RXMIS | | | | |
| **UARTICR, type W1C, offset 0x044, reset 0x0000.0000** (see page 521) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | OEIC | BEIC | PEIC | FEIC | RTIC | TXIC | RXIC | | | | |
| **UARTDMACTL, type R/W, offset 0x048, reset 0x0000.0000** (see page 523) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | DMAERR | TXDMAE | RXDMAE |
| **UARTPeriphID4, type RO, offset 0xFD0, reset 0x0000.0000** (see page 524) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID4 | | | | | | | |
| **UARTPeriphID5, type RO, offset 0xFD4, reset 0x0000.0000** (see page 525) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID5 | | | | | | | |
| **UARTPeriphID6, type RO, offset 0xFD8, reset 0x0000.0000** (see page 526) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID6 | | | | | | | |
| **UARTPeriphID7, type RO, offset 0xFDC, reset 0x0000.0000** (see page 527) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID7 | | | | | | | |
| **UARTPeriphID0, type RO, offset 0xFE0, reset 0x0000.0011** (see page 528) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID0 | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **UARTPeriphID1, type RO, offset 0xFE4, reset 0x0000.0000** (see page 529) | | | | | | | | | | | | | | | |
| | | | | | | | | PID1 | | | | | | | |
| **UARTPeriphID2, type RO, offset 0xFE8, reset 0x0000.0018** (see page 530) | | | | | | | | | | | | | | | |
| | | | | | | | | PID2 | | | | | | | |
| **UARTPeriphID3, type RO, offset 0xFEC, reset 0x0000.0001** (see page 531) | | | | | | | | | | | | | | | |
| | | | | | | | | PID3 | | | | | | | |
| **UARTPCellID0, type RO, offset 0xFF0, reset 0x0000.000D** (see page 532) | | | | | | | | | | | | | | | |
| | | | | | | | | CID0 | | | | | | | |
| **UARTPCellID1, type RO, offset 0xFF4, reset 0x0000.00F0** (see page 533) | | | | | | | | | | | | | | | |
| | | | | | | | | CID1 | | | | | | | |
| **UARTPCellID2, type RO, offset 0xFF8, reset 0x0000.0005** (see page 534) | | | | | | | | | | | | | | | |
| | | | | | | | | CID2 | | | | | | | |
| **UARTPCellID3, type RO, offset 0xFFC, reset 0x0000.00B1** (see page 535) | | | | | | | | | | | | | | | |
| | | | | | | | | CID3 | | | | | | | |
| **Synchronous Serial Interface (SSI)**<br>SSI0 base: 0x4000.8000 | | | | | | | | | | | | | | | |
| **SSICR0, type R/W, offset 0x000, reset 0x0000.0000** (see page 550) | | | | | | | | | | | | | | | |
| SCR | | | | | | | | SPH | SPO | FRF | | DSS | | | |
| **SSICR1, type R/W, offset 0x004, reset 0x0000.0000** (see page 552) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | SOD | MS | SSE | LBM |
| **SSIDR, type R/W, offset 0x008, reset 0x0000.0000** (see page 554) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **SSISR, type RO, offset 0x00C, reset 0x0000.0003** (see page 555) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | BSY | RFF | RNE | TNF | TFE |
| **SSICPSR, type R/W, offset 0x010, reset 0x0000.0000** (see page 557) | | | | | | | | | | | | | | | |
| | | | | | | | | CPSDVSR | | | | | | | |
| **SSIIM, type R/W, offset 0x014, reset 0x0000.0000** (see page 558) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TXIM | RXIM | RTIM | RORIM |
| **SSIRIS, type RO, offset 0x018, reset 0x0000.0008** (see page 560) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TXRIS | RXRIS | RTRIS | RORRIS |
| **SSIMIS, type RO, offset 0x01C, reset 0x0000.0000** (see page 561) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TXMIS | RXMIS | RTMIS | RORMIS |
| **SSIICR, type W1C, offset 0x020, reset 0x0000.0000** (see page 562) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | RTIC | RORIC |
| **SSIDMACTL, type R/W, offset 0x024, reset 0x0000.0000** (see page 563) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | TXDMAE | RXDMAE |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **SSIPeriphID4, type RO, offset 0xFD0, reset 0x0000.0000** (see page 564) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID4 | | | | | | | |
| **SSIPeriphID5, type RO, offset 0xFD4, reset 0x0000.0000** (see page 565) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID5 | | | | | | | |
| **SSIPeriphID6, type RO, offset 0xFD8, reset 0x0000.0000** (see page 566) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID6 | | | | | | | |
| **SSIPeriphID7, type RO, offset 0xFDC, reset 0x0000.0000** (see page 567) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID7 | | | | | | | |
| **SSIPeriphID0, type RO, offset 0xFE0, reset 0x0000.0022** (see page 568) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID0 | | | | | | | |
| **SSIPeriphID1, type RO, offset 0xFE4, reset 0x0000.0000** (see page 569) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID1 | | | | | | | |
| **SSIPeriphID2, type RO, offset 0xFE8, reset 0x0000.0018** (see page 570) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID2 | | | | | | | |
| **SSIPeriphID3, type RO, offset 0xFEC, reset 0x0000.0001** (see page 571) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID3 | | | | | | | |
| **SSIPCellID0, type RO, offset 0xFF0, reset 0x0000.000D** (see page 572) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID0 | | | | | | | |
| **SSIPCellID1, type RO, offset 0xFF4, reset 0x0000.00F0** (see page 573) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID1 | | | | | | | |
| **SSIPCellID2, type RO, offset 0xFF8, reset 0x0000.0005** (see page 574) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID2 | | | | | | | |
| **SSIPCellID3, type RO, offset 0xFFC, reset 0x0000.00B1** (see page 575) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID3 | | | | | | | |
| **Inter-Integrated Circuit ($I^2C$) Interface**<br>**$I^2C$ Master**<br>I2C 0 base: 0x4002.0000<br>I2C 1 base: 0x4002.1000 | | | | | | | | | | | | | | | |
| **I2CMSA, type R/W, offset 0x000, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | SA | | | | | | | R/S |
| **I2CMCS, type RO, offset 0x004, reset 0x0000.0000 (Reads)** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | BUSBSY | IDLE | ARBLST | DATACK | ADRACK | ERROR | BUSY |
| **I2CMCS, type WO, offset 0x004, reset 0x0000.0000 (Writes)** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | ACK | STOP | START | RUN |
| **I2CMDR, type R/W, offset 0x008, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | DATA | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **I2CMTPR, type R/W, offset 0x00C, reset 0x0000.0001** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | TPR | | | | | | |
| **I2CMIMR, type R/W, offset 0x010, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | IM |
| **I2CMRIS, type RO, offset 0x014, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | RIS |
| **I2CMMIS, type RO, offset 0x018, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | MIS |
| **I2CMICR, type WO, offset 0x01C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | IC |
| **I2CMCR, type R/W, offset 0x020, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | SFE | MFE | | | | LPBK |
| **Inter-Integrated Circuit (I²C) Interface**<br>**I²C Slave**<br>I2C 0 base: 0x4002.0000<br>I2C 1 base: 0x4002.1000 | | | | | | | | | | | | | | | |
| **I2CSOAR, type R/W, offset 0x800, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | OAR | | | |
| **I2CSCSR, type RO, offset 0x804, reset 0x0000.0000 (Reads)** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | FBR | TREQ | RREQ |
| **I2CSCSR, type WO, offset 0x804, reset 0x0000.0000 (Writes)** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DA |
| **I2CSDR, type R/W, offset 0x808, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | DATA | | | | | | | |
| **I2CSIMR, type R/W, offset 0x80C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DATAIM |
| **I2CSRIS, type RO, offset 0x810, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DATARIS |
| **I2CSMIS, type RO, offset 0x814, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DATAMIS |
| **I2CSICR, type WO, offset 0x818, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DATAIC |
| **Controller Area Network (CAN) Module**<br>CAN0 base: 0x4004.0000 | | | | | | | | | | | | | | | |
| **CANCTL, type R/W, offset 0x000, reset 0x0000.0001** (see page 634) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | TEST | CCE | DAR | | EIE | SIE | IE | INIT |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **CANSTS, type R/W, offset 0x004, reset 0x0000.0000** (see page 636) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | BOFF | EWARN | EPASS | RXOK | TXOK | LEC | | |
| **CANERR, type RO, offset 0x008, reset 0x0000.0000** (see page 638) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| RP | REC | | | | | | | TEC | | | | | | | |
| **CANBIT, type R/W, offset 0x00C, reset 0x0000.2301** (see page 639) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | TSEG2 | | | TSEG1 | | | | SJW | | BRP | | | | | |
| **CANINT, type RO, offset 0x010, reset 0x0000.0000** (see page 640) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| INTID | | | | | | | | | | | | | | | |
| **CANTST, type R/W, offset 0x014, reset 0x0000.0000** (see page 641) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | RX | TX | | LBACK | SILENT | BASIC | | |
| **CANBRPE, type R/W, offset 0x018, reset 0x0000.0000** (see page 643) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | BRPE | | | |
| **CANIF1CRQ, type R/W, offset 0x020, reset 0x0000.0001** (see page 644) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| BUSY | | | | | | | | | | MNUM | | | | | |
| **CANIF2CRQ, type R/W, offset 0x080, reset 0x0000.0001** (see page 644) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| BUSY | | | | | | | | | | MNUM | | | | | |
| **CANIF1CMSK, type R/W, offset 0x024, reset 0x0000.0000** (see page 645) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | WRNRD | MASK | ARB | CONTROL | CLRINTPND | NEWDAT / TXRQST | DATAA | DATAB |
| **CANIF2CMSK, type R/W, offset 0x084, reset 0x0000.0000** (see page 645) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | WRNRD | MASK | ARB | CONTROL | CLRINTPND | NEWDAT / TXRQST | DATAA | DATAB |
| **CANIF1MSK1, type R/W, offset 0x028, reset 0x0000.FFFF** (see page 647) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| MSK | | | | | | | | | | | | | | | |
| **CANIF2MSK1, type R/W, offset 0x088, reset 0x0000.FFFF** (see page 647) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| MSK | | | | | | | | | | | | | | | |
| **CANIF1MSK2, type R/W, offset 0x02C, reset 0x0000.FFFF** (see page 648) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| MXTD | MDIR | | MSK | | | | | | | | | | | | |
| **CANIF2MSK2, type R/W, offset 0x08C, reset 0x0000.FFFF** (see page 648) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| MXTD | MDIR | | MSK | | | | | | | | | | | | |
| **CANIF1ARB1, type R/W, offset 0x030, reset 0x0000.0000** (see page 649) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| ID | | | | | | | | | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **CANIF2ARB1, type R/W, offset 0x090, reset 0x0000.0000** (see page 649) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| ID | | | | | | | | | | | | | | | |
| **CANIF1ARB2, type R/W, offset 0x034, reset 0x0000.0000** (see page 650) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| MSGVAL | XTD | DIR | ID | | | | | | | | | | | | |
| **CANIF2ARB2, type R/W, offset 0x094, reset 0x0000.0000** (see page 650) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| MSGVAL | XTD | DIR | ID | | | | | | | | | | | | |
| **CANIF1MCTL, type R/W, offset 0x038, reset 0x0000.0000** (see page 652) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| NEWDAT | MSGLST | INTPND | UMASK | TXIE | RXIE | RMTEN | TXRQST | EOB | | | | DLC | | | |
| **CANIF2MCTL, type R/W, offset 0x098, reset 0x0000.0000** (see page 652) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| NEWDAT | MSGLST | INTPND | UMASK | TXIE | RXIE | RMTEN | TXRQST | EOB | | | | DLC | | | |
| **CANIF1DA1, type R/W, offset 0x03C, reset 0x0000.0000** (see page 654) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **CANIF1DA2, type R/W, offset 0x040, reset 0x0000.0000** (see page 654) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **CANIF1DB1, type R/W, offset 0x044, reset 0x0000.0000** (see page 654) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **CANIF1DB2, type R/W, offset 0x048, reset 0x0000.0000** (see page 654) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **CANIF2DA1, type R/W, offset 0x09C, reset 0x0000.0000** (see page 654) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **CANIF2DA2, type R/W, offset 0x0A0, reset 0x0000.0000** (see page 654) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **CANIF2DB1, type R/W, offset 0x0A4, reset 0x0000.0000** (see page 654) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **CANIF2DB2, type R/W, offset 0x0A8, reset 0x0000.0000** (see page 654) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **CANTXRQ1, type RO, offset 0x100, reset 0x0000.0000** (see page 655) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| TXRQST | | | | | | | | | | | | | | | |
| **CANTXRQ2, type RO, offset 0x104, reset 0x0000.0000** (see page 655) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| TXRQST | | | | | | | | | | | | | | | |
| **CANNWDA1, type RO, offset 0x120, reset 0x0000.0000** (see page 656) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| NEWDAT | | | | | | | | | | | | | | | |
| **CANNWDA2, type RO, offset 0x124, reset 0x0000.0000** (see page 656) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| NEWDAT | | | | | | | | | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **CANMSG1INT, type RO, offset 0x140, reset 0x0000.0000** (see page 657) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| INTPND | | | | | | | | | | | | | | | |
| **CANMSG2INT, type RO, offset 0x144, reset 0x0000.0000** (see page 657) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| INTPND | | | | | | | | | | | | | | | |
| **CANMSG1VAL, type RO, offset 0x160, reset 0x0000.0000** (see page 658) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| MSGVAL | | | | | | | | | | | | | | | |
| **CANMSG2VAL, type RO, offset 0x164, reset 0x0000.0000** (see page 658) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| MSGVAL | | | | | | | | | | | | | | | |
| **Universal Serial Bus (USB) Controller**<br>Base 0x4005.0000 | | | | | | | | | | | | | | | |
| **USBFADDR, type R/W, offset 0x000, reset 0x00** (see page 676) | | | | | | | | | | | | | | | |
| | | | | | | | | | FUNCADDR | | | | | | |
| **USBPOWER, type R/W, offset 0x001, reset 0x20 (Host Mode)** (see page 677) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | RESET | RESUME | SUSPEND | PWRDNPHY |
| **USBPOWER, type R/W, offset 0x001, reset 0x20 (Device Mode)** (see page 677) | | | | | | | | | | | | | | | |
| | | | | | | | | ISOUP | SOFTCONN | | | RESET | RESUME | SUSPEND | PWRDNPHY |
| **USBTXIS, type RO, offset 0x002, reset 0x0000** (see page 680) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EP3 | EP2 | EP1 | EP0 |
| **USBRXIS, type RO, offset 0x004, reset 0x0000** (see page 681) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EP3 | EP2 | EP1 | |
| **USBTXIE, type R/W, offset 0x006, reset 0x000F** (see page 682) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EP3 | EP2 | EP1 | EP0 |
| **USBRXIE, type R/W, offset 0x008, reset 0x000E** (see page 683) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EP3 | EP2 | EP1 | |
| **USBIS, type RO, offset 0x00A, reset 0x00 ( Host Mode)** (see page 684) | | | | | | | | | | | | | | | |
| | | | | | | | | | | DISCON | CONN | SOF | BABBLE | RESUME | |
| **USBIS, type RO, offset 0x00A, reset 0x00 (Device Mode)** (see page 684) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | SOF | RESET | RESUME | SUSPEND |
| **USBIE, type R/W, offset 0x00B, reset 0x06 (Host Mode)** (see page 687) | | | | | | | | | | | | | | | |
| | | | | | | | | | | DISCON | CONN | SOF | BABBLE | RESUME | |
| **USBIE, type R/W, offset 0x00B, reset 0x06 (Device Mode)** (see page 687) | | | | | | | | | | | | | | | |
| | | | | | | | | | | DISCON | | SOF | RESET | RESUME | SUSPEND |
| **USBFRAME, type RO, offset 0x00C, reset 0x0000** (see page 690) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | FRAME | | | | | | | | | | |
| **USBEPIDX, type R/W, offset 0x00E, reset 0x00** (see page 691) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EPIDX | | | |
| **USBTEST, type R/W, offset 0x00F, reset 0x00 (Host Mode)** (see page 692) | | | | | | | | | | | | | | | |
| | | | | | | | | FORCEH | FIFOACC | FORCEFS | | | | | |
| **USBTEST, type R/W, offset 0x00F, reset 0x00 (Device Mode)** (see page 692) | | | | | | | | | | | | | | | |
| | | | | | | | | | FIFOACC | FORCEFS | | | | | |
| **USBFIFO0, type R/W, offset 0x020, reset 0x0000.0000** (see page 694) | | | | | | | | | | | | | | | |
| EPDATA | | | | | | | | | | | | | | | |
| EPDATA | | | | | | | | | | | | | | | |
| **USBFIFO1, type R/W, offset 0x024, reset 0x0000.0000** (see page 694) | | | | | | | | | | | | | | | |
| EPDATA | | | | | | | | | | | | | | | |
| EPDATA | | | | | | | | | | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **USBFIFO2, type R/W, offset 0x028, reset 0x0000.0000** (see page 694) | | | | | | | | | | | | | | | |
| EPDATA | | | | | | | | | | | | | | | |
| EPDATA | | | | | | | | | | | | | | | |
| **USBFIFO3, type R/W, offset 0x02C, reset 0x0000.0000** (see page 694) | | | | | | | | | | | | | | | |
| EPDATA | | | | | | | | | | | | | | | |
| EPDATA | | | | | | | | | | | | | | | |
| **USBDEVCTL, type RO, offset 0x060, reset 0x80** (see page 695) | | | | | | | | | | | | | | | |
| | | | | | | | | | FSDEV | LSDEV | | | HOST | | |
| **USBTXFIFOSZ, type R/W, offset 0x062, reset 0x00** (see page 696) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | DPB | SIZE | | | |
| **USBRXFIFOSZ, type R/W, offset 0x063, reset 0x00** (see page 696) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | DPB | SIZE | | | |
| **USBTXFIFOADD, type R/W, offset 0x064, reset 0x0000** (see page 697) | | | | | | | | | | | | | | | |
| | | | | | | | ADDR | | | | | | | | |
| **USBRXFIFOADD, type R/W, offset 0x066, reset 0x0000** (see page 697) | | | | | | | | | | | | | | | |
| | | | | | | | ADDR | | | | | | | | |
| **USBCONTIM, type R/W, offset 0x07A, reset 0x5C** (see page 698) | | | | | | | | | | | | | | | |
| | | | | | | | | WTCON | | | | WTID | | | |
| **USBFSEOF, type R/W, offset 0x07D, reset 0x77** (see page 699) | | | | | | | | | | | | | | | |
| | | | | | | | | FSEOFG | | | | | | | |
| **USBLSEOF, type R/W, offset 0x07E, reset 0x72** (see page 700) | | | | | | | | | | | | | | | |
| | | | | | | | | LSEOFG | | | | | | | |
| **USBTXFUNCADDR0, type R/W, offset 0x080, reset 0x00** (see page 701) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBTXFUNCADDR1, type R/W, offset 0x088, reset 0x00** (see page 701) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBTXFUNCADDR2, type R/W, offset 0x090, reset 0x00** (see page 701) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBTXFUNCADDR3, type R/W, offset 0x098, reset 0x00** (see page 701) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBTXHUBADDR0, type R/W, offset 0x082, reset 0x00** (see page 702) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBTXHUBADDR1, type R/W, offset 0x08A, reset 0x00** (see page 702) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBTXHUBADDR2, type R/W, offset 0x092, reset 0x00** (see page 702) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBTXHUBADDR3, type R/W, offset 0x09A, reset 0x00** (see page 702) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBTXHUBPORT0, type R/W, offset 0x083, reset 0x00** (see page 703) | | | | | | | | | | | | | | | |
| | | | | | | | | | PORT | | | | | | |
| **USBTXHUBPORT1, type R/W, offset 0x08B, reset 0x00** (see page 703) | | | | | | | | | | | | | | | |
| | | | | | | | | | PORT | | | | | | |
| **USBTXHUBPORT2, type R/W, offset 0x093, reset 0x00** (see page 703) | | | | | | | | | | | | | | | |
| | | | | | | | | | PORT | | | | | | |
| **USBTXHUBPORT3, type R/W, offset 0x09B, reset 0x00** (see page 703) | | | | | | | | | | | | | | | |
| | | | | | | | | | PORT | | | | | | |
| **USBRXFUNCADDR1, type R/W, offset 0x08C, reset 0x00** (see page 704) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBRXFUNCADDR2, type R/W, offset 0x094, reset 0x00** (see page 704) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBRXFUNCADDR3, type R/W, offset 0x09C, reset 0x00** (see page 704) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **USBRXHUBADDR1, type R/W, offset 0x08E, reset 0x00** (see page 705) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBRXHUBADDR2, type R/W, offset 0x096, reset 0x00** (see page 705) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBRXHUBADDR3, type R/W, offset 0x09E, reset 0x00** (see page 705) | | | | | | | | | | | | | | | |
| | | | | | | | | | ADDR | | | | | | |
| **USBRXHUBPORT1, type R/W, offset 0x08F, reset 0x00** (see page 706) | | | | | | | | | | | | | | | |
| | | | | | | | | | PORT | | | | | | |
| **USBRXHUBPORT2, type R/W, offset 0x097, reset 0x00** (see page 706) | | | | | | | | | | | | | | | |
| | | | | | | | | | PORT | | | | | | |
| **USBRXHUBPORT3, type R/W, offset 0x09F, reset 0x00** (see page 706) | | | | | | | | | | | | | | | |
| | | | | | | | | | PORT | | | | | | |
| **USBTXMAXP1, type R/W, offset 0x110, reset 0x0000** (see page 707) | | | | | | | | | | | | | | | |
| | | | | | MAXLOAD | | | | | | | | | | |
| **USBTXMAXP2, type R/W, offset 0x120, reset 0x0000** (see page 707) | | | | | | | | | | | | | | | |
| | | | | | MAXLOAD | | | | | | | | | | |
| **USBTXMAXP3, type R/W, offset 0x130, reset 0x0000** (see page 707) | | | | | | | | | | | | | | | |
| | | | | | MAXLOAD | | | | | | | | | | |
| **USBCSRL0, type R/W, offset 0x102, reset 0x00 (Host Mode)** (see page 708) | | | | | | | | | | | | | | | |
| | | | | | | | | NAKTO | STATUS | REQPKT | ERROR | SETUP | STALLED | TXRDY | RXRDY |
| **USBCSRL0, type R/W, offset 0x102, reset 0x00 (Device Mode)** (see page 708) | | | | | | | | | | | | | | | |
| | | | | | | | | SETENDC | RXRDYC | STALL | SETEND | DATAEND | STALLED | TXRDY | RXRDY |
| **USBCSRH0, type R/W, offset 0x103, reset 0x00 (Host Mode)** (see page 712) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | DTWE | DT | FLUSH |
| **USBCSRH0, type R/W, offset 0x103, reset 0x00 (Device Mode)** (see page 712) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | FLUSH |
| **USBCOUNT0, type RO, offset 0x108, reset 0x00** (see page 714) | | | | | | | | | | | | | | | |
| | | | | | | | | | COUNT | | | | | | |
| **USBTYPE0, type R/W, offset 0x10A, reset 0x00** (see page 715) | | | | | | | | | | | | | | | |
| | | | | | | | | SPEED | | | | | | | |
| **USBNAKLMT, type R/W, offset 0x10B, reset 0x00** (see page 716) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | NAKLMT | | | | |
| **USBTXCSRL1, type R/W, offset 0x112, reset 0x00 (Host Mode)** (see page 717) | | | | | | | | | | | | | | | |
| | | | | | | | | NAKTO | CLRDT | STALLED | SETUP | FLUSH | ERROR | FIFONE | TXRDY |
| **USBTXCSRL2, type R/W, offset 0x122, reset 0x00 (Host Mode)** (see page 717) | | | | | | | | | | | | | | | |
| | | | | | | | | NAKTO | CLRDT | STALLED | SETUP | FLUSH | ERROR | FIFONE | TXRDY |
| **USBTXCSRL3, type R/W, offset 0x132, reset 0x00 (Host Mode)** (see page 717) | | | | | | | | | | | | | | | |
| | | | | | | | | NAKTO | CLRDT | STALLED | SETUP | FLUSH | ERROR | FIFONE | TXRDY |
| **USBTXCSRL1, type R/W, offset 0x112, reset 0x00 (Device Mode)** (see page 717) | | | | | | | | | | | | | | | |
| | | | | | | | | | CLRDT | STALLED | STALL | FLUSH | UNDRN | FIFONE | TXRDY |
| **USBTXCSRL2, type R/W, offset 0x122, reset 0x00 (Device Mode)** (see page 717) | | | | | | | | | | | | | | | |
| | | | | | | | | | CLRDT | STALLED | STALL | FLUSH | UNDRN | FIFONE | TXRDY |
| **USBTXCSRL3, type R/W, offset 0x132, reset 0x00 (Device Mode)** (see page 717) | | | | | | | | | | | | | | | |
| | | | | | | | | | CLRDT | STALLED | STALL | FLUSH | UNDRN | FIFONE | TXRDY |
| **USBTXCSRH1, type R/W, offset 0x113, reset 0x00 (Host Mode)** (see page 721) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOSET | | MODE | DMAEN | FDT | DMAMOD | DTWE | DT |
| **USBTXCSRH2, type R/W, offset 0x123, reset 0x00 (Host Mode)** (see page 721) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOSET | | MODE | DMAEN | FDT | DMAMOD | DTWE | DT |
| **USBTXCSRH3, type R/W, offset 0x133, reset 0x00 (Host Mode)** (see page 721) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOSET | | MODE | DMAEN | FDT | DMAMOD | DTWE | DT |
| **USBTXCSRH1, type R/W, offset 0x113, reset 0x00 (Device Mode)** (see page 721) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOSET | ISO | MODE | DMAEN | FDT | DMAMOD | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **USBTXCSRH2, type R/W, offset 0x123, reset 0x00 (Device Mode)** (see page 721) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOSET | ISO | MODE | DMAEN | FDT | DMAMOD | | |
| **USBTXCSRH3, type R/W, offset 0x133, reset 0x00 (Device Mode)** (see page 721) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOSET | ISO | MODE | DMAEN | FDT | DMAMOD | | |
| **USBRXMAXP1, type R/W, offset 0x114, reset 0x0000** (see page 725) | | | | | | | | | | | | | | | |
| | | | | | MAXLOAD | | | | | | | | | | |
| **USBRXMAXP2, type R/W, offset 0x124, reset 0x0000** (see page 725) | | | | | | | | | | | | | | | |
| | | | | | MAXLOAD | | | | | | | | | | |
| **USBRXMAXP3, type R/W, offset 0x134, reset 0x0000** (see page 725) | | | | | | | | | | | | | | | |
| | | | | | MAXLOAD | | | | | | | | | | |
| **USBRXCSRL1, type R/W, offset 0x116, reset 0x00 (Host Mode)** (see page 726) | | | | | | | | | | | | | | | |
| | | | | | | | | CLRDT | STALLED | REQPKT | FLUSH | DATAERR / NAKTO | ERROR | FULL | RXRDY |
| **USBRXCSRL2, type R/W, offset 0x126, reset 0x00 (Host Mode)** (see page 726) | | | | | | | | | | | | | | | |
| | | | | | | | | CLRDT | STALLED | REQPKT | FLUSH | DATAERR / NAKTO | ERROR | FULL | RXRDY |
| **USBRXCSRL3, type R/W, offset 0x136, reset 0x00 (Host Mode)** (see page 726) | | | | | | | | | | | | | | | |
| | | | | | | | | CLRDT | STALLED | REQPKT | FLUSH | DATAERR / NAKTO | ERROR | FULL | RXRDY |
| **USBRXCSRL1, type R/W, offset 0x116, reset 0x00 (Device Mode)** (see page 726) | | | | | | | | | | | | | | | |
| | | | | | | | | CLRDT | STALLED | STALL | FLUSH | DATAERR | OVER | FULL | RXRDY |
| **USBRXCSRL2, type R/W, offset 0x126, reset 0x00 (Device Mode)** (see page 726) | | | | | | | | | | | | | | | |
| | | | | | | | | CLRDT | STALLED | STALL | FLUSH | DATAERR | OVER | FULL | RXRDY |
| **USBRXCSRL3, type R/W, offset 0x136, reset 0x00 (Device Mode)** (see page 726) | | | | | | | | | | | | | | | |
| | | | | | | | | CLRDT | STALLED | STALL | FLUSH | DATAERR | OVER | FULL | RXRDY |
| **USBRXCSRH1, type R/W, offset 0x117, reset 0x00 (Host Mode)** (see page 731) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOCL | AUTORQ | DMAEN | PIDERR | DMAMOD | DTWE | DT | |
| **USBRXCSRH2, type R/W, offset 0x127, reset 0x00 (Host Mode)** (see page 731) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOCL | AUTORQ | DMAEN | PIDERR | DMAMOD | DTWE | DT | |
| **USBRXCSRH3, type R/W, offset 0x137, reset 0x00 (Host Mode)** (see page 731) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOCL | AUTORQ | DMAEN | PIDERR | DMAMOD | DTWE | DT | |
| **USBRXCSRH1, type R/W, offset 0x117, reset 0x00 (Device Mode)** (see page 731) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOCL | ISO | DMAEN | DISNYET / PIDERR | DMAMOD | | | |
| **USBRXCSRH2, type R/W, offset 0x127, reset 0x00 (Device Mode)** (see page 731) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOCL | ISO | DMAEN | DISNYET / PIDERR | DMAMOD | | | |

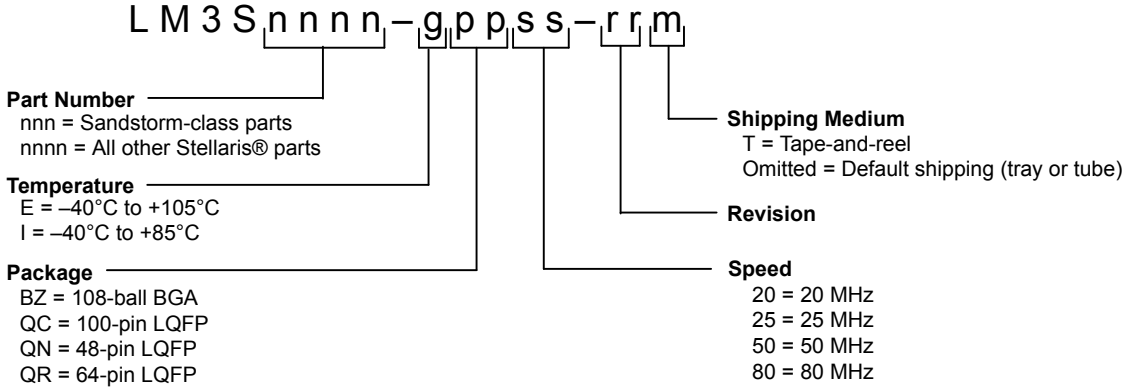
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **USBRXCSRH3, type R/W, offset 0x137, reset 0x00 (Device Mode)** (see page 731) | | | | | | | | | | | | | | | |
| | | | | | | | | AUTOCL | ISO | DMAEN | DISNYET / PIDERR | DMAMOD | | | |
| **USBRXCOUNT1, type RO, offset 0x118, reset 0x0000** (see page 735) | | | | | | | | | | | | | | | |
| | | | COUNT | | | | | | | | | | | | |
| **USBRXCOUNT2, type RO, offset 0x128, reset 0x0000** (see page 735) | | | | | | | | | | | | | | | |
| | | | COUNT | | | | | | | | | | | | |
| **USBRXCOUNT3, type RO, offset 0x138, reset 0x0000** (see page 735) | | | | | | | | | | | | | | | |
| | | | COUNT | | | | | | | | | | | | |
| **USBTXTYPE1, type R/W, offset 0x11A, reset 0x00** (see page 736) | | | | | | | | | | | | | | | |
| | | | | | | | | SPEED | | PROTO | | TEP | | | |
| **USBTXTYPE2, type R/W, offset 0x12A, reset 0x00** (see page 736) | | | | | | | | | | | | | | | |
| | | | | | | | | SPEED | | PROTO | | TEP | | | |
| **USBTXTYPE3, type R/W, offset 0x13A, reset 0x00** (see page 736) | | | | | | | | | | | | | | | |
| | | | | | | | | SPEED | | PROTO | | TEP | | | |
| **USBTXINTERVAL1, type R/W, offset 0x11B, reset 0x00** (see page 737) | | | | | | | | | | | | | | | |
| | | | | | | | | TXPOLL / NAKLMT | | | | | | | |
| **USBTXINTERVAL2, type R/W, offset 0x12B, reset 0x00** (see page 737) | | | | | | | | | | | | | | | |
| | | | | | | | | TXPOLL / NAKLMT | | | | | | | |
| **USBTXINTERVAL3, type R/W, offset 0x13B, reset 0x00** (see page 737) | | | | | | | | | | | | | | | |
| | | | | | | | | TXPOLL / NAKLMT | | | | | | | |
| **USBRXTYPE1, type R/W, offset 0x11C, reset 0x00** (see page 738) | | | | | | | | | | | | | | | |
| | | | | | | | | SPEED | | PROTO | | TEP | | | |
| **USBRXTYPE2, type R/W, offset 0x12C, reset 0x00** (see page 738) | | | | | | | | | | | | | | | |
| | | | | | | | | SPEED | | PROTO | | TEP | | | |
| **USBRXTYPE3, type R/W, offset 0x13C, reset 0x00** (see page 738) | | | | | | | | | | | | | | | |
| | | | | | | | | SPEED | | PROTO | | TEP | | | |
| **USBRXINTERVAL1, type R/W, offset 0x11D, reset 0x00** (see page 739) | | | | | | | | | | | | | | | |
| | | | | | | | | TXPOLL / NAKLMT | | | | | | | |
| **USBRXINTERVAL2, type R/W, offset 0x12D, reset 0x00** (see page 739) | | | | | | | | | | | | | | | |
| | | | | | | | | TXPOLL / NAKLMT | | | | | | | |
| **USBRXINTERVAL3, type R/W, offset 0x13D, reset 0x00** (see page 739) | | | | | | | | | | | | | | | |
| | | | | | | | | TXPOLL / NAKLMT | | | | | | | |
| **USBRQPKTCOUNT1, type R/W, offset 0x304, reset 0x0000** (see page 740) | | | | | | | | | | | | | | | |
| | | | | | | | COUNT | | | | | | | | |
| **USBRQPKTCOUNT2, type R/W, offset 0x308, reset 0x0000** (see page 740) | | | | | | | | | | | | | | | |
| | | | | | | | COUNT | | | | | | | | |
| **USBRQPKTCOUNT3, type R/W, offset 0x30C, reset 0x0000** (see page 740) | | | | | | | | | | | | | | | |
| | | | | | | | COUNT | | | | | | | | |
| **USBRXDPKTBUFDIS, type R/W, offset 0x340, reset 0x0000** (see page 741) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EP3 | EP2 | EP1 | |
| **USBTXDPKTBUFDIS, type R/W, offset 0x342, reset 0x0000** (see page 742) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | EP3 | EP2 | EP1 | |
| **USBEPC, type R/W, offset 0x400, reset 0x0000.0000** (see page 743) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | PFLTACT | | | PFLTAEN | PFLTSEN | PFLTEN | | EPENDE | EPEN | |
| **USBEPCRIS, type RO, offset 0x404, reset 0x0000.0000** (see page 746) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | PF |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **USBEPCIM, type R/W, offset 0x408, reset 0x0000.0000** (see page 747) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | PF |
| **USBEPCISC, type R/W1C, offset 0x40C, reset 0x0000.0000** (see page 748) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | PF |
| **USBDRRIS, type RO, offset 0x410, reset 0x0000.0000** (see page 749) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | RESUME |
| **USBDRIM, type R/W, offset 0x414, reset 0x0000.0000** (see page 750) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | RESUME |
| **USBDRISC, type R/W1C, offset 0x418, reset 0x0000.0000** (see page 751) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | RESUME |
| **USBGPCS, type R/W, offset 0x41C, reset 0x0000.0000** (see page 752) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DEVMOD |

# D Ordering and Contact Information

## D.1 Ordering Information

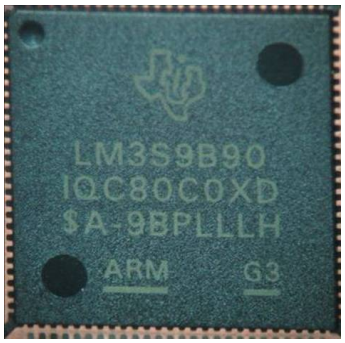
The figure below defines the full set of potential orderable part numbers for all the Stellaris® LM3S microcontrollers. See the Package Option Addendum for the valid orderable part numbers for the LM3S5632 microcontroller.




## D.2 Part Markings

The Stellaris microcontrollers are marked with an identifying number. This code contains the following information:

- The first line indicates the part number, for example, LM3S9B90.
- In the second line, the first eight characters indicate the temperature, package, speed, revision, and product status. For example in the figure below, IQC80C0X indicates an Industrial temperature (I), 100-pin LQFP package (QC), 80-MHz (80), revision C0 (C0) device. The letter immediately following the revision indicates product status. An X indicates experimental and requires a waiver; an S indicates the part is fully qualified and released to production.
- The remaining characters contain internal tracking numbers.




## D.3 Kits

The Stellaris Family provides the hardware and software tools that engineers need to begin development quickly.

- Reference Design Kits accelerate product development by providing ready-to-run hardware and comprehensive documentation including hardware design files
- Evaluation Kits provide a low-cost and effective means of evaluating Stellaris microcontrollers before purchase
- Development Kits provide you with all the tools you need to develop and prototype embedded applications right out of the box

See the website at www.ti.com/stellaris for the latest tools available, or ask your distributor.

## D.4 Support Information

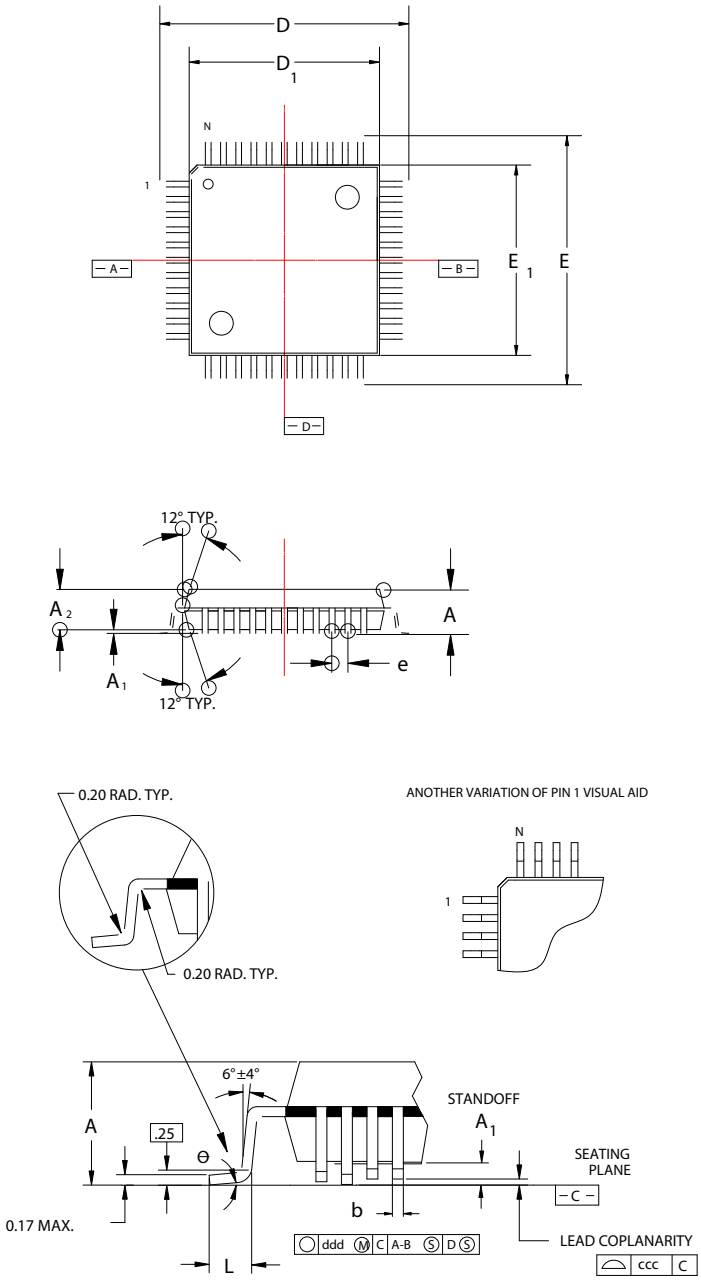
For support on Stellaris products, contact the TI Worldwide Product Information Center nearest you: http://www-k.ext.ti.com/sc/technical-support/product-information-centers.htm.

# E Package Information

## E.1 64-Pin LQFP Package

### E.1.1 Package Dimensions

**Figure E-1. Stellaris LM3S5632 64-Pin LQFP Package**



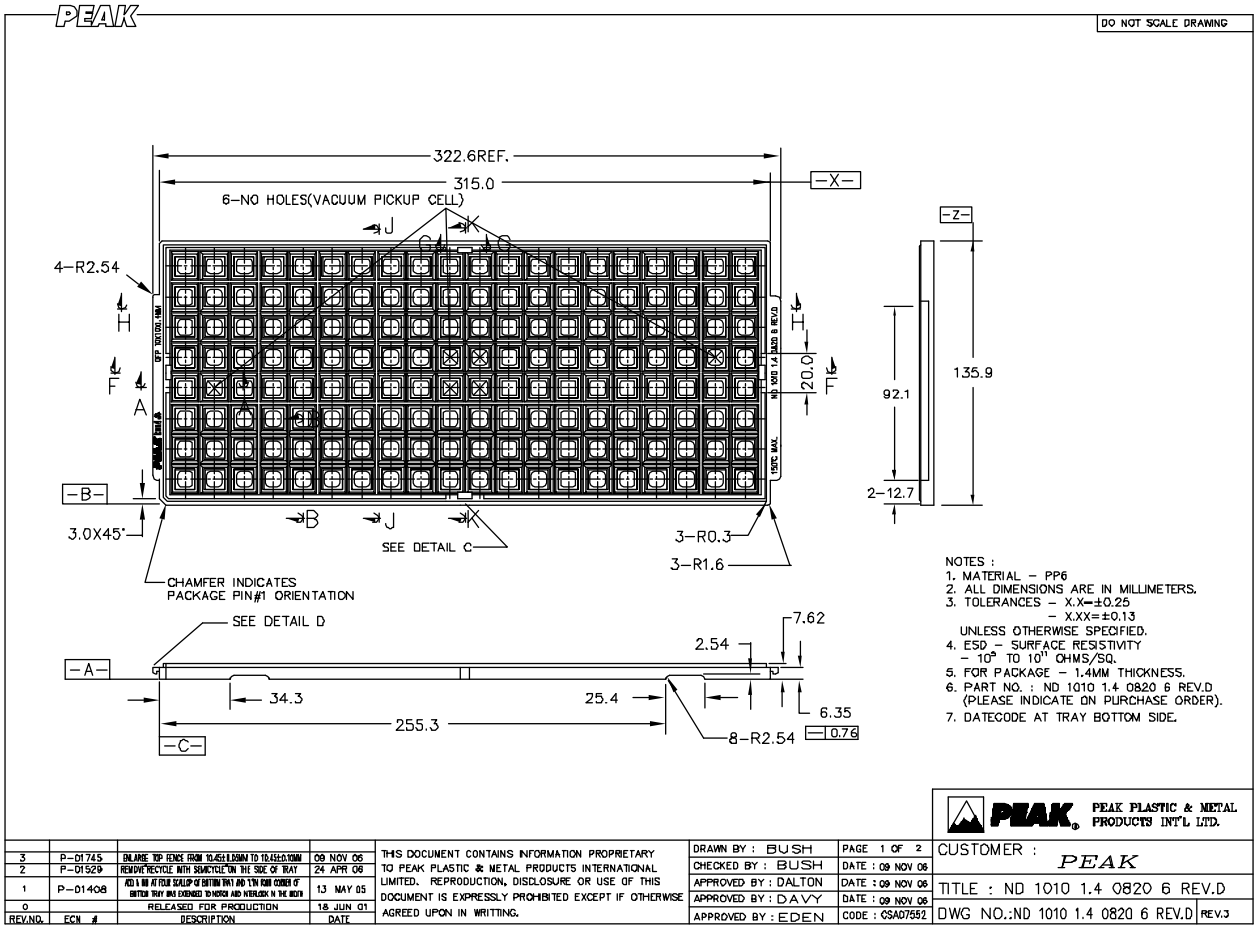
**Note:** The following notes apply to the package drawing.

1. All dimensions shown in mm.
2. Dimensions shown are nominal with tolerances indicated.
3. Foot length 'L' is measured at gage plane 0.25 mm above seating plane.
4. L/F: Eftec 64T Cu or equivalent, 0.127mm (0.005") thick.

| Body +2.00 mm Footprint, 1.4 mm package thickness | | |
|---|---|---|
| Symbols | Leads | 64L |
| A | Max. | 1.60 |
| $A_1$ | - | 0.05 Min./0.15 Max. |
| $A_2$ | ±0.05 | 1.40 |
| D | ±0.20 | 12.00 |
| $D_1$ | ±0.10 | 10.00 |
| E | ±0.20 | 12.00 |
| $E_1$ | ±0.10 | 10.00 |
| L | +0.15/-0.10 | 0.60 |
| e | Basic | 0.50 |
| b | ±0.05 | 0.22 |
| θ | - | 0˚-7˚ |
| ddd | Max. | 0.08 |
| ccc | Max. | 0.08 |
| JEDEC Reference Drawing | | MS-026 |
| Variation Designator | | BCD |

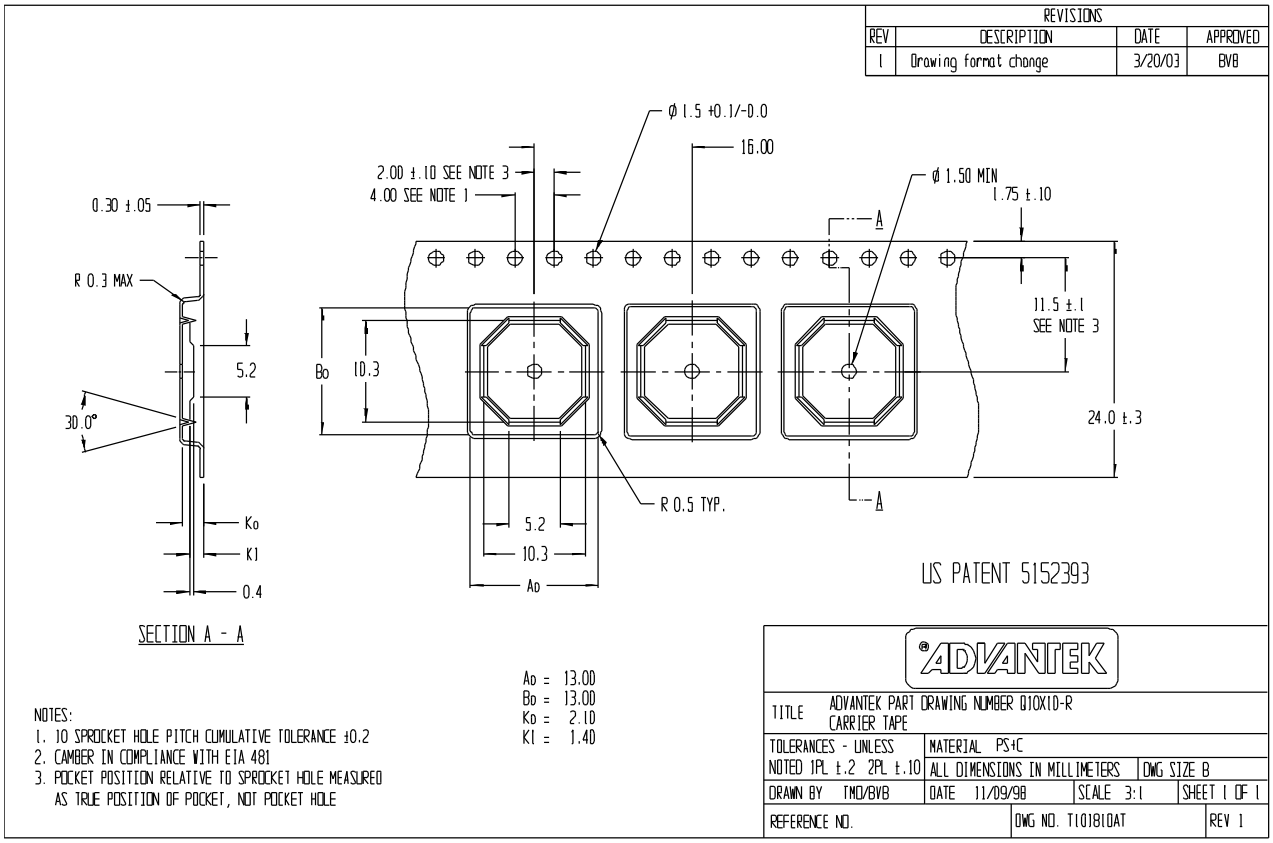
## E.1.2 Tray Dimensions

**Figure E-2. 64-Pin LQFP Tray Dimensions**



## E.1.3 Tape and Reel Dimensions

**Figure E-3. 64-Pin LQFP Tape and Reel Dimensions**



## PACKAGING INFORMATION

| Orderable part number | Status (1) | Material type (2) | Package \| Pins | Package qty \| Carrier | RoHS (3) | Lead finish/ Ball material (4) | MSL rating/ Peak reflow (5) | Op temp (°C) | Part marking (6) |
|---|---|---|---|---|---|---|---|---|---|
| LM3S5632-IQR50-A0 | NRND | Production | LQFP (PM) \| 64 | 160 \| null | Yes | NIPDAU | Level-3-260C-168 HR | -40 to 85 | LM3S5632<br>IQR50<br>PM |
| LM3S5632-IQR50-A0.A | NRND | Production | LQFP (PM) \| 64 | 160 \| null | Yes | NIPDAU | Level-3-260C-168 HR | -40 to 85 | LM3S5632<br>IQR50<br>PM |
| LM3S5632-IQR50-A0T | NRND | Production | LQFP (PM) \| 64 | 1000 \| LARGE T&R | Yes | NIPDAU | Level-3-260C-168 HR | -40 to 85 | LM3S5632<br>IQR50<br>PM |
| LM3S5632-IQR50-A0T.A | NRND | Production | LQFP (PM) \| 64 | 1000 \| LARGE T&R | Yes | NIPDAU | Level-3-260C-168 HR | -40 to 85 | LM3S5632<br>IQR50<br>PM |

(1) **Status:** For more details on status, see our product life cycle.

(2) **Material type:** When designated, preproduction parts are prototypes/experimental devices, and are not yet approved or released for full production. Testing and final process, including without limitation quality assurance, reliability performance testing, and/or process qualification, may not yet be complete, and this item is subject to further changes or possible discontinuation. If available for ordering, purchases will be subject to an additional waiver at checkout, and are intended for early internal evaluation purposes only. These items are sold without warranties of any kind.

(3) **RoHS values:** Yes, No, RoHS Exempt. See the TI RoHS Statement for additional information and value definition.

(4) **Lead finish/Ball material:** Parts may have multiple material finish options. Finish options are separated by a vertical ruled line. Lead finish/Ball material values may wrap to two lines if the finish value exceeds the maximum column width.

(5) **MSL rating/Peak reflow:** The moisture sensitivity level ratings and peak solder (reflow) temperatures. In the event that a part has multiple moisture sensitivity ratings, only the lowest level per JEDEC standards is shown. Refer to the shipping label for the actual reflow temperature that will be used to mount the part to the printed circuit board.

(6) **Part marking:** There may be an additional marking, which relates to the logo, the lot trace code information, or the environmental category of the part.

Multiple part markings will be inside parentheses. Only one part marking contained in parentheses and separated by a "~" will appear on a part. If a line is indented then it is a continuation of the previous line and the two combined represent the entire part marking for that device.

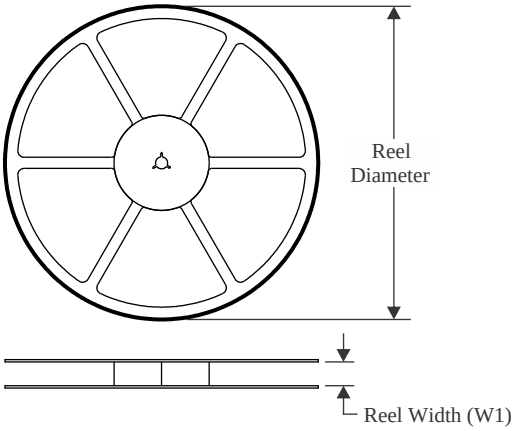
**Important Information and Disclaimer:**The information provided on this page represents TI's knowledge and belief as of the date that it is provided. TI bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. TI has taken and continues to take reasonable steps to provide representative

and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. TI and TI suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

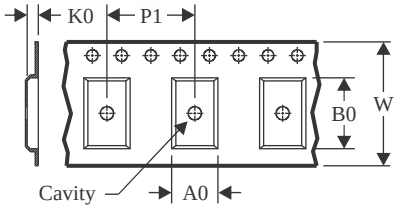
In no event shall TI's liability arising out of such information exceed the total purchase price of the TI part(s) at issue in this document sold by TI to Customer on an annual basis.

## TAPE AND REEL INFORMATION

**REEL DIMENSIONS**

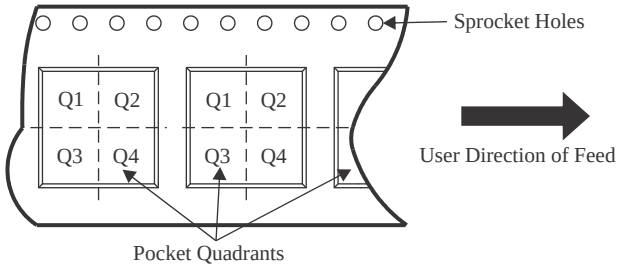



**TAPE DIMENSIONS**




| | |
|---|---|
| A0 | Dimension designed to accommodate the component width |
| B0 | Dimension designed to accommodate the component length |
| K0 | Dimension designed to accommodate the component thickness |
| W | Overall width of the carrier tape |
| P1 | Pitch between successive cavity centers |

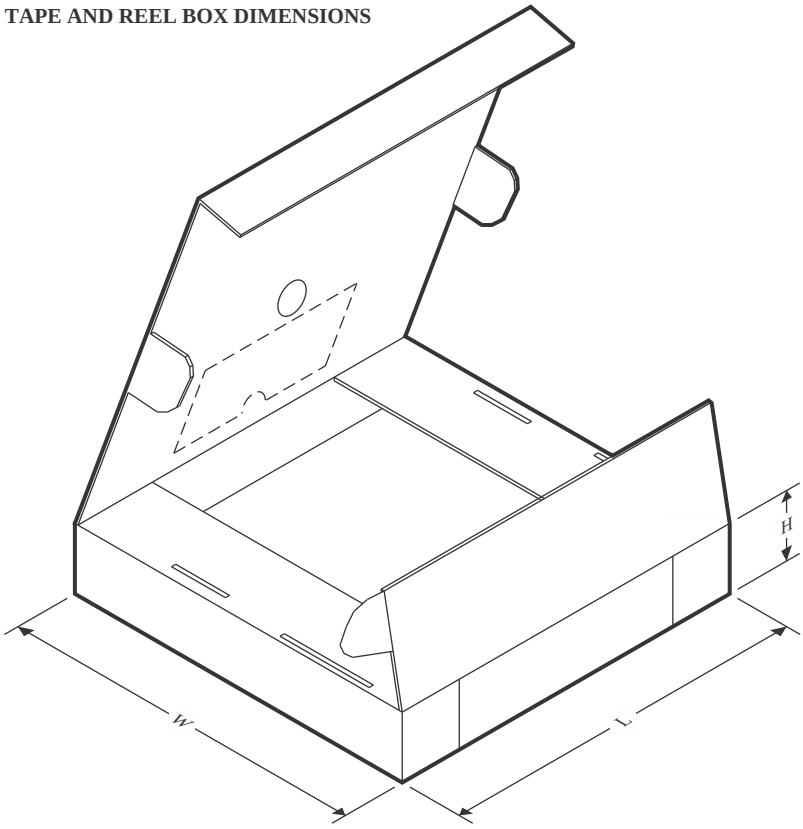
**QUADRANT ASSIGNMENTS FOR PIN 1 ORIENTATION IN TAPE**




*All dimensions are nominal

| Device | Package Type | Package Drawing | Pins | SPQ | Reel Diameter (mm) | Reel Width W1 (mm) | A0 (mm) | B0 (mm) | K0 (mm) | P1 (mm) | W (mm) | Pin1 Quadrant |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LM3S5632-IQR50-A0T | LQFP | PM | 64 | 1000 | 330.0 | 24.4 | 12.3 | 12.3 | 2.5 | 16.0 | 24.0 | Q2 |

TAPE AND REEL BOX DIMENSIONS




*All dimensions are nominal

| Device | Package Type | Package Drawing | Pins | SPQ | Length (mm) | Width (mm) | Height (mm) |
|---|---|---|---|---|---|---|---|
| LM3S5632-IQR50-A0T | LQFP | PM | 64 | 1000 | 367.0 | 367.0 | 45.0 |

# IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2025, Texas Instruments Incorporated