

LUMINARY MICRO™

# LM3S316 Microcontroller

## DATA SHEET

DS-LM3S316-02

Copyright © 2006 Luminary Micro, Inc.

# Legal Disclaimers and Trademark Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH LUMINARY MICRO PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN LUMINARY MICRO'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, LUMINARY MICRO ASSUMES NO LIABILITY WHATSOEVER, AND LUMINARY MICRO DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF LUMINARY MICRO'S PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. LUMINARY MICRO'S PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE-SUSTAINING APPLICATIONS.

Luminary Micro may make changes to specifications and product descriptions at any time, without notice. Contact your local Luminary Micro sales office or your distributor to obtain the latest specifications before placing your product order.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Luminary Micro reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Copyright © 2006 Luminary Micro, Inc. All rights reserved. Stellaris and the Luminary Micro logo are trademarks of Luminary Micro, Inc. or its subsidiaries in the United States and other countries. 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.

Luminary Micro, Inc.
2499 South Capital of Texas Hwy, Suite A-100
Austin, TX 78746
Main: +1-512-279-8800
Fax: +1-512-279-8879
http://www.luminarymicro.com

# Table of Contents

**Legal Disclaimers and Trademark Information** ..... **2**
**Revision History** ..... **18**
**About This Document** ..... **19**
Audience ..... 19
About This Manual ..... 19
Related Documents ..... 19
Documentation Conventions ..... 19
**1. Architectural Overview** ..... **22**
1.1 Product Features ..... 22
1.2 Target Applications ..... 26
1.3 High-Level Block Diagram ..... 27
1.4 Functional Overview ..... 28
1.4.1 ARM Cortex™-M3 ..... 28
1.4.2 Motor Control Peripherals ..... 28
1.4.3 Analog Peripherals ..... 29
1.4.4 Serial Communications Peripherals ..... 29
1.4.5 System Peripherals ..... 30
1.4.6 Memory Peripherals ..... 31
1.4.7 Additional Features ..... 32
1.4.8 Hardware Details ..... 32
1.5 System Block Diagram ..... 33
**2. ARM Cortex-M3 Processor Core** ..... **34**
2.1 Block Diagram ..... 35
2.2 Functional Description ..... 35
2.2.1 Serial Wire and JTAG Debug ..... 35
2.2.2 Embedded Trace Macrocell (ETM) ..... 36
2.2.3 Trace Port Interface Unit (TPIU) ..... 36
2.2.4 ROM Table ..... 36
2.2.5 Memory Protection Unit (MPU) ..... 36
2.2.6 Nested Vectored Interrupt Controller (NVIC) ..... 36
**3. Memory Map** ..... **37**
**4. Interrupts** ..... **39**
**5. JTAG Interface** ..... **42**
5.1 Block Diagram ..... 43
5.2 Functional Description ..... 43
5.2.1 JTAG Interface Pins ..... 44
5.2.2 JTAG TAP Controller ..... 45
5.2.3 Shift Registers ..... 46
5.2.4 Operational Considerations ..... 46
5.3 Initialization and Configuration ..... 47
5.4 Register Descriptions ..... 48
5.4.1 Instruction Register (IR) ..... 48
5.4.2 Data Registers ..... 50
**6. System Control** ..... **52**
6.1 Functional Description ..... 52
6.1.1 Device Identification ..... 52

6.1.2 Reset Control ..... 52
6.1.3 Power Control ..... 55
6.1.4 Clock Control ..... 55
6.1.5 System Control ..... 57
6.2 Initialization and Configuration ..... 58
6.3 Register Map ..... 58
6.4 Register Descriptions ..... 59

**7. Internal Memory ..... 94**
7.1 Block Diagram ..... 94
7.2 Functional Description ..... 94
7.2.1 SRAM Memory ..... 94
7.2.2 Flash Memory ..... 95
7.3 Initialization and Configuration ..... 96
7.3.1 Changing Flash Protection Bits ..... 96
7.3.2 Flash Programming ..... 97
7.4 Register Map ..... 97
7.5 Register Descriptions ..... 98

**8. General-Purpose Input/Outputs (GPIOs) ..... 108**
8.1 Block Diagram ..... 109
8.2 Functional Description ..... 109
8.2.1 Data Register Operation ..... 110
8.2.2 Data Direction ..... 111
8.2.3 Interrupt Operation ..... 111
8.2.4 Mode Control ..... 112
8.2.5 Pad Configuration ..... 112
8.2.6 Identification ..... 112
8.3 Initialization and Configuration ..... 112
8.4 Register Map ..... 114
8.5 Register Descriptions ..... 115

**9. General-Purpose Timers ..... 146**
9.1 Block Diagram ..... 147
9.2 Functional Description ..... 147
9.2.1 GPTM Reset Conditions ..... 147
9.2.2 32-Bit Timer Operating Modes ..... 147
9.2.3 16-Bit Timer Operating Modes ..... 149
9.3 Initialization and Configuration ..... 153
9.3.1 32-Bit One-Shot/Periodic Timer Mode ..... 153
9.3.2 32-Bit Real-Time Clock (RTC) Mode ..... 154
9.3.3 16-Bit One-Shot/Periodic Timer Mode ..... 154
9.3.4 16-Bit Input Edge Count Mode ..... 154
9.3.5 16-Bit Input Edge Timing Mode ..... 155
9.3.6 16-Bit PWM Mode ..... 155
9.4 Register Map ..... 156
9.5 Register Descriptions ..... 157

**10. Watchdog Timer ..... 178**
10.1 Block Diagram ..... 178
10.2 Functional Description ..... 179
10.3 Initialization and Configuration ..... 179
10.4 Register Map ..... 179

10.5 Register Descriptions ..... 180
**11. Analog-to-Digital Converter (ADC) ..... 201**
11.1 Block Diagram ..... 202
11.2 Functional Description ..... 202
11.2.1 Sample Sequencers ..... 202
11.2.2 Module Control ..... 203
11.2.3 Hardware Sample Averaging Circuit ..... 204
11.2.4 Analog-to-Digital Converter ..... 204
11.2.5 Test Modes ..... 204
11.2.6 Internal Temperature Sensor ..... 204
11.3 Initialization and Configuration ..... 204
11.3.1 Module Initialization ..... 205
11.3.2 Sample Sequencer Configuration ..... 205
11.4 Register Map ..... 205
11.5 Register Descriptions ..... 206
**12. Universal Asynchronous Receivers/Transmitters (UARTs) ..... 231**
12.1 Block Diagram ..... 232
12.2 Functional Description ..... 232
12.2.1 Transmit/Receive Logic ..... 232
12.2.2 Baud-Rate Generation ..... 233
12.2.3 Data Transmission ..... 234
12.2.4 FIFO Operation ..... 234
12.2.5 Interrupts ..... 234
12.2.6 Loopback Operation ..... 235
12.3 Initialization and Configuration ..... 235
12.4 Register Map ..... 236
12.5 Register Descriptions ..... 237
**13. Synchronous Serial Interface (SSI) ..... 267**
13.1 Block Diagram ..... 267
13.2 Functional Description ..... 268
13.2.1 Bit Rate Generation ..... 268
13.2.2 FIFO Operation ..... 268
13.2.3 Interrupts ..... 268
13.2.4 Frame Formats ..... 269
13.3 Initialization and Configuration ..... 276
13.4 Register Map ..... 277
13.5 Register Descriptions ..... 278
**14. Inter-Integrated Circuit (I2C) Interface ..... 302**
14.1 Block Diagram ..... 302
14.2 Functional Description ..... 302
14.2.1 $I^2C$ Bus Functional Overview ..... 303
14.2.2 Available Speed Modes ..... 310
14.3 Initialization and Configuration ..... 311
14.4 Register Map ..... 311
14.5 Register Descriptions (I2C Master) ..... 312
14.6 Register Descriptions (I2C Slave) ..... 326
**15. Analog Comparator ..... 334**
15.1 Block Diagram ..... 334
15.2 Functional Description ..... 334

15.2.1 Internal Reference Programming ..... 335
15.3 Initialization and Configuration ..... 336
15.4 Register Map ..... 337
15.5 Register Descriptions ..... 337
**16. Pulse Width Modulator (PWM) ..... 345**
16.1 Block Diagram ..... 345
16.2 Functional Description ..... 345
16.2.1 PWM Timer ..... 345
16.2.2 PWM Comparators ..... 346
16.2.3 PWM Signal Generator ..... 347
16.2.4 Dead-Band Generator ..... 348
16.2.5 Interrupt/ADC-Trigger Selector ..... 348
16.2.6 Synchronization Methods ..... 348
16.2.7 Fault Conditions ..... 349
16.2.8 Output Control Block ..... 349
16.3 Initialization and Configuration ..... 349
16.4 Register Map ..... 350
16.5 Register Descriptions ..... 351
**17. Pin Diagram ..... 377**
**18. Signal Tables ..... 378**
**19. Operating Characteristics ..... 388**
**20. Electrical Characteristics ..... 389**
20.1 DC Characteristics ..... 389
20.1.1 Maximum Ratings ..... 389
20.1.2 Recommended DC Operating Conditions ..... 389
20.1.3 On-Chip Low Drop-Out (LDO) Regulator Characteristics ..... 390
20.1.4 Power Specifications ..... 391
20.1.5 Flash Memory Characteristics ..... 391
20.2 AC Characteristics ..... 392
20.2.1 Load Conditions ..... 392
20.2.2 Clocks ..... 392
20.2.3 Temperature Sensor ..... 393
20.2.4 Analog-to-Digital Converter ..... 393
20.2.5 Analog Comparator ..... 394
20.2.6 $I^2C$ ..... 394
20.2.7 Synchronous Serial Interface (SSI) ..... 396
20.2.8 JTAG and Boundary Scan ..... 398
20.2.9 General-Purpose I/O ..... 400
20.2.10 Reset ..... 400
**21. Package Information ..... 403**
**Appendix A. Serial Flash Loader ..... 404**
A.1 Interfaces ..... 404
A.1.1 UART ..... 404
A.1.2 SSI ..... 404
A.2 Packet Handling ..... 404
A.2.1 Packet Format ..... 405
A.2.2 Sending Packets ..... 405
A.2.3 Receiving Packets ..... 405
A.3 Commands ..... 405

A.3.1 COMMAND_PING (0x20) ..... 406
A.3.2 COMMAND_GET_STATUS (0x23) ..... 406
A.3.3 COMMAND_DOWNLOAD (0x21) ..... 406
A.3.4 COMMAND_SEND_DATA (0x24) ..... 406
A.3.5 COMMAND_RUN (0x22) ..... 407
A.3.6 COMMAND_RESET (0x25) ..... 407
**Ordering and Contact Information ..... 408**
Ordering Information ..... 408
Development Kit ..... 408
Company Information ..... 408
Support Information ..... 409

# List of Figures


Figure 1-1. Stellaris High-Level Block Diagram ..... 27
Figure 1-2. LM3S316 Controller System-Level Block Diagram ..... 33
Figure 2-1. CPU Block Diagram ..... 35
Figure 2-2. TPIU Block Diagram ..... 36
Figure 5-1. JTAG Module Block Diagram ..... 43
Figure 5-2. Test Access Port State Machine ..... 46
Figure 5-3. IDCODE Register Format ..... 50
Figure 5-4. BYPASS Register Format ..... 50
Figure 5-5. Boundary Scan Register Format ..... 51
Figure 6-1. External Circuitry to Extend Reset ..... 53
Figure 6-2. Main Clock Tree ..... 56
Figure 7-1. Flash Block Diagram ..... 94
Figure 8-1. GPIO Module Block Diagram ..... 109
Figure 8-2. GPIO Port Block Diagram ..... 110
Figure 8-3. GPIODATA Write Example ..... 111
Figure 8-4. GPIODATA Read Example ..... 111
Figure 9-1. GPTM Module Block Diagram ..... 147
Figure 9-2. 16-Bit Input Edge Count Mode Example ..... 151
Figure 9-3. 16-Bit Input Edge Time Mode Example ..... 152
Figure 9-4. 16-Bit PWM Mode Example ..... 153
Figure 10-1. WDT Module Block Diagram ..... 178
Figure 11-1. ADC Module Block Diagram ..... 202
Figure 11-2. Internal Temperature Sensor Characteristic ..... 204
Figure 12-1. UART Module Block Diagram ..... 232
Figure 12-2. UART Character Frame ..... 233
Figure 13-1. SSI Module Block Diagram ..... 267
Figure 13-2. TI Synchronous Serial Frame Format (Single Transfer) ..... 269
Figure 13-3. TI Synchronous Serial Frame Format (Continuous Transfer) ..... 270
Figure 13-4. Freescale SPI Format (Single Transfer) with SPO=0 and SPH=0 ..... 271
Figure 13-5. Freescale SPI Format (Continuous Transfer) with SPO=0 and SPH=0 ..... 271
Figure 13-6. Freescale SPI Frame Format with SPO=0 and SPH=1 ..... 272
Figure 13-7. Freescale SPI Frame Format (Single Transfer) with SPO=1 and SPH=0 ..... 272
Figure 13-8. Freescale SPI Frame Format (Continuous Transfer) with SPO=1 and SPH=0 ..... 273
Figure 13-9. Freescale SPI Frame Format with SPO=1 and SPH=1 ..... 273
Figure 13-10. MICROWIRE Frame Format (Single Frame) ..... 274
Figure 13-11. MICROWIRE Frame Format (Continuous Transfer) ..... 275
Figure 13-12. MICROWIRE Frame Format, SSIFss Input Setup and Hold Requirements ..... 276
Figure 14-1. $I^2C$ Block Diagram ..... 302
Figure 14-2. $I^2C$ Bus Configuration ..... 303
Figure 14-3. Data Validity During Bit Transfer on the $I^2C$ Bus ..... 303
Figure 14-4. START and STOP Conditions ..... 303
Figure 14-5. Complete Data Transfer with a 7-Bit Address ..... 304
Figure 14-6. R/S Bit in First Byte ..... 305
Figure 14-7. Master Single SEND ..... 305
Figure 14-8. Master Single RECEIVE ..... 306
Figure 14-9. Master Burst SEND ..... 307

Figure 14-10. Master Burst RECEIVE ..... 308
Figure 14-11. Master Burst RECEIVE after Burst SEND ..... 309
Figure 14-12. Master Burst SEND after Burst RECEIVE ..... 309
Figure 14-13. Slave Command Sequence ..... 310
Figure 15-1. Analog Comparator Module Block Diagram ..... 334
Figure 15-2. Structure of Comparator Unit ..... 335
Figure 15-3. Comparator Internal Reference Structure ..... 336
Figure 16-1. PWM Module Block Diagram ..... 345
Figure 16-2. PWM Count-Down Mode ..... 346
Figure 16-3. PWM Count-Up/Down Mode ..... 347
Figure 16-4. PWM Generation Example In Count-Up/Down Mode ..... 347
Figure 16-5. PWM Dead-Band Generator ..... 348
Figure 17-1. Pin Connection Diagram ..... 377
Figure 20-1. Load Conditions ..... 392
Figure 20-2. $I^2C$ Timing ..... 395
Figure 20-3. SSI Timing for TI Frame Format (FRF=01), Single Transfer Timing Measurement ..... 396
Figure 20-4. SSI Timing for MICROWIRE Frame Format (FRF=10), Single Transfer ..... 397
Figure 20-5. SSI Timing for SPI Frame Format (FRF=00), with SPH=1 ..... 397
Figure 20-6. JTAG Test Clock Input Timing ..... 399
Figure 20-7. JTAG Test Access Port (TAP) Timing ..... 399
Figure 20-8. JTAG $\overline{\text{TRST}}$ Timing ..... 399
Figure 20-9. External Reset Timing ($\overline{\text{RST}}$) ..... 401
Figure 20-10. Power-On Reset Timing ..... 401
Figure 20-11. Brown-Out Reset Timing ..... 401
Figure 20-12. Software Reset Timing ..... 401
Figure 20-13. Watchdog Reset Timing ..... 402
Figure 20-14. LDO Reset Timing ..... 402
Figure 21-1. 48-Pin LQFP Package ..... 403

# List of Tables


Table 0-1. Documentation Conventions ..... 19
Table 3-1. Memory Map ..... 37
Table 4-1. Exception Types ..... 39
Table 4-2. Interrupts ..... 40
Table 5-1. JTAG Port Pins Reset State ..... 44
Table 5-2. JTAG Instruction Register Commands ..... 48
Table 6-1. System Control Register Map ..... 58
Table 6-2. VADJ to VOUT ..... 71
Table 6-3. PLL Mode Control ..... 83
Table 6-4. Default Crystal Field Values and PLL Programming ..... 83
Table 7-1. Flash Protection Policy Combinations ..... 96
Table 7-2. Flash Register Map ..... 97
Table 8-1. GPIO Pad Configuration Examples ..... 113
Table 8-2. GPIO Interrupt Configuration Example ..... 113
Table 8-3. GPIO Register Map ..... 114
Table 9-1. 16-Bit Timer With Prescaler Configurations ..... 150
Table 9-2. GPTM Register Map ..... 156
Table 10-1. WDT Register Map ..... 179
Table 11-1. Samples and FIFO Depth of Sequencers ..... 202
Table 11-2. ADC Register Map ..... 205
Table 12-1. UART Register Map ..... 236
Table 13-1. SSI Register Map ..... 277
Table 14-1. Examples of $I^2C$ Master Timer Period versus Speed Mode ..... 311
Table 14-2. $I^2C$ Register Map ..... 312
Table 14-3. Write Field Decoding for I2CMCS[3:0] Field ..... 316
Table 15-1. Comparator 0 Operating Modes ..... 335
Table 15-2. Internal Reference Voltage and ACREFCTL Field Values ..... 336
Table 15-3. Analog Comparator Register Map ..... 337
Table 16-1. PWM Register Map ..... 350
Table 16-2. PWM Generator Action Encodings ..... 372
Table 18-1. Signals by Pin Number ..... 378
Table 18-2. Signals by Signal Name ..... 381
Table 18-3. Signals by Function, Except for GPIO ..... 384
Table 18-4. GPIO Pins and Alternate Functions ..... 386
Table 19-1. Temperature Characteristics ..... 388
Table 19-2. Thermal Characteristics ..... 388
Table 20-1. Maximum Ratings ..... 389
Table 20-2. Recommended DC Operating Conditions ..... 389
Table 20-3. LDO Regulator Characteristics ..... 390
Table 20-4. Power Specifications ..... 391
Table 20-5. Flash Memory Characteristics ..... 391
Table 20-6. Phase Locked Loop (PLL) Characteristics ..... 392
Table 20-7. Clock Characteristics ..... 392
Table 20-8. Temperature Sensor Characteristics ..... 393
Table 20-9. ADC Characteristics ..... 393
Table 20-10. Analog Comparator Characteristics ..... 394
Table 20-11. Analog Comparator Voltage Reference Characteristics ..... 394
Table 20-12. $I^2C$ Characteristics ..... 394

Table 20-13. SSI Characteristics ..... 396
Table 20-14. JTAG Characteristics ..... 398
Table 20-15. GPIO Characteristics ..... 400
Table 20-16. Reset Characteristics ..... 400

# List of Registers

**System Control ..... 52**
Register 1: Device Identification 0 (DID0), offset 0x000 ..... 60
Register 2: Device Identification 1 (DID1), offset 0x004 ..... 61
Register 3: Device Capabilities 0 (DC0), offset 0x008 ..... 63
Register 4: Device Capabilities 1 (DC1), offset 0x010 ..... 64
Register 5: Device Capabilities 2 (DC2), offset 0x014 ..... 66
Register 6: Device Capabilities 3 (DC3), offset 0x018 ..... 67
Register 7: Device Capabilities 4 (DC4), offset 0x01C ..... 69
Register 8: Power-On and Brown-Out Reset Control (PBORCTL), offset 0x030 ..... 70
Register 9: LDO Power Control (LDOPCTL), offset 0x034 ..... 71
Register 10: Software Reset Control 0 (SRCR0), offset 0x040 ..... 72
Register 11: Software Reset Control 1 (SRCR1), offset 0x044 ..... 73
Register 12: Software Reset Control 2 (SRCR2), offset 0x048 ..... 74
Register 13: Raw Interrupt Status (RIS), offset 0x050 ..... 75
Register 14: Interrupt Mask Control (IMC), offset 0x054 ..... 76
Register 15: Masked Interrupt Status and Clear (MISC), offset 0x058 ..... 78
Register 16: Reset Cause (RESC), offset 0x05C ..... 79
Register 17: Run-Mode Clock Configuration (RCC), offset 0x060 ..... 80
Register 18: XTAL to PLL Translation (PLLCFG), offset 0x064 ..... 85
Register 19: Run-Mode Clock Gating Control 0 (RCGC0), offset 0x100 ..... 86
Register 20: Sleep-Mode Clock Gating Control 0 (SCGC0), offset 0x110 ..... 86
Register 21: Deep-Sleep-Mode Clock Gating Control 0 (DCGC0), offset 0x120 ..... 86
Register 22: Run-Mode Clock Gating Control 1 (RCGC1), offset 0x104 ..... 88
Register 23: Sleep-Mode Clock Gating Control 1 (SCGC1), offset 0x114 ..... 88
Register 24: Deep-Sleep-Mode Clock Gating Control 1 (DCGC1), offset 0x124 ..... 88
Register 25: Run-Mode Clock Gating Control 2 (RCGC2), offset 0x108 ..... 90
Register 26: Sleep-Mode Clock Gating Control 2 (SCGC2), offset 0x118 ..... 90
Register 27: Deep-Sleep-Mode Clock Gating Control 2 (DCGC2), offset 0x128 ..... 90
Register 28: Deep-Sleep Clock Configuration (DSLPCLKCFG), offset 0x144 ..... 91
Register 29: Clock Verification Clear (CLKVCLR), offset 0x150 ..... 92
Register 30: Allow Unregulated LDO to Reset the Part (LDOARST), offset 0x160 ..... 93

**Internal Memory ..... 94**
Register 1: Flash Memory Protection Read Enable (FMPRE), offset 0x130 ..... 99
Register 2: Flash Memory Protection Program Enable (FMPPE), offset 0x134 ..... 99
Register 3: USec Reload (USECRL), offset 0x140 ..... 100
Register 4: Flash Memory Address (FMA), offset 0x000 ..... 101
Register 5: Flash Memory Data (FMD), offset 0x004 ..... 102
Register 6: Flash Memory Control (FMC), offset 0x008 ..... 103
Register 7: Flash Controller Raw Interrupt Status (FCRIS), offset 0x00C ..... 105
Register 8: Flash Controller Interrupt Mask (FCIM), offset 0x010 ..... 106
Register 9: Flash Controller Masked Interrupt Status and Clear (FCMISC), offset 0x014 ..... 107

**General-Purpose Input/Outputs (GPIOs) ..... 108**
Register 1: GPIO Data (GPIODATA), offset 0x000 ..... 116
Register 2: GPIO Direction (GPIODIR), offset 0x400 ..... 117
Register 3: GPIO Interrupt Sense (GPIOIS), offset 0x404 ..... 118
Register 4: GPIO Interrupt Both Edges (GPIOIBE), offset 0x408 ..... 119

Register 5: GPIO Interrupt Event (GPIOIEV), offset 0x40C ..... 120
Register 6: GPIO Interrupt Mask (GPIOIM), offset 0x410 ..... 121
Register 7: GPIO Raw Interrupt Status (GPIORIS), offset 0x414 ..... 122
Register 8: GPIO Masked Interrupt Status (GPIOMIS), offset 0x418 ..... 123
Register 9: GPIO Interrupt Clear (GPIOICR), offset 0x41C ..... 124
Register 10: GPIO Alternate Function Select (GPIOAFSEL), offset 0x420 ..... 125
Register 11: GPIO 2-mA Drive Select (GPIODR2R), offset 0x500 ..... 126
Register 12: GPIO 4-mA Drive Select (GPIODR4R), offset 0x504 ..... 127
Register 13: GPIO 8-mA Drive Select (GPIODR8R), offset 0x508 ..... 128
Register 14: GPIO Open Drain Select (GPIOODR), offset 0x50C ..... 129
Register 15: GPIO Pull-Up Select (GPIOPUR), offset 0x510 ..... 130
Register 16: GPIO Pull-Down Select (GPIOPDR), offset 0x514 ..... 131
Register 17: GPIO Slew Rate Control Select (GPIOSLR), offset 0x518 ..... 132
Register 18: GPIO Digital Input Enable (GPIODEN), offset 0x51C ..... 133
Register 19: GPIO Peripheral Identification 4 (GPIOPeriphID4), offset 0xFD0 ..... 134
Register 20: GPIO Peripheral Identification 5 (GPIOPeriphID5), offset 0xFD4 ..... 135
Register 21: GPIO Peripheral Identification 6 (GPIOPeriphID6), offset 0xFD8 ..... 136
Register 22: GPIO Peripheral Identification 7 (GPIOPeriphID7), offset 0xFDC ..... 137
Register 23: GPIO Peripheral Identification 0 (GPIOPeriphID0), offset 0xFE0 ..... 138
Register 24: GPIO Peripheral Identification 1(GPIOPeriphID1), offset 0xFE4 ..... 139
Register 25: GPIO Peripheral Identification 2 (GPIOPeriphID2), offset 0xFE8 ..... 140
Register 26: GPIO Peripheral Identification 3 (GPIOPeriphID3), offset 0xFEC ..... 141
Register 27: GPIO PrimeCell Identification 0 (GPIOPCellID0), offset 0xFF0 ..... 142
Register 28: GPIO PrimeCell Identification 1 (GPIOPCellID1), offset 0xFF4 ..... 143
Register 29: GPIO PrimeCell Identification 2 (GPIOPCellID2), offset 0xFF8 ..... 144
Register 30: GPIO PrimeCell Identification 3 (GPIOPCellID3), offset 0xFFC ..... 145

**General-Purpose Timers ..... 146**

Register 1: GPTM Configuration (GPTMCFG), offset 0x000 ..... 158
Register 2: GPTM TimerA Mode (GPTMTAMR), offset 0x004 ..... 159
Register 3: GPTM TimerB Mode (GPTMTBMR), offset 0x008 ..... 160
Register 4: GPTM Control (GPTMCTL), offset 0x00C ..... 161
Register 5: GPTM Interrupt Mask (GPTMIMR), offset 0x018 ..... 163
Register 6: GPTM Raw Interrupt Status (GPTMRIS), offset 0x01C ..... 165
Register 7: GPTM Masked Interrupt Status (GPTMMIS), offset 0x020 ..... 166
Register 8: GPTM Interrupt Clear (GPTMICR), offset 0x024 ..... 167
Register 9: GPTM TimerA Interval Load (GPTMTAILR), offset 0x028 ..... 168
Register 10: GPTM TimerB Interval Load (GPTMTBILR), offset 0x02C ..... 169
Register 11: GPTM TimerA Match (GPTMTAMATCHR), offset 0x030 ..... 170
Register 12: GPTM TimerB Match (GPTMTBMATCHR), offset 0x034 ..... 171
Register 13: GPTM TimerA Prescale (GPTMTAPR), offset 0x038 ..... 172
Register 14: GPTM TimerB Prescale (GPTMTBPR), offset 0x03C ..... 173
Register 15: GPTM TimerA Prescale Match (GPTMTAPMR), offset 0x040 ..... 174
Register 16: GPTM TimerB Prescale Match (GPTMTBPMR), offset 0x044 ..... 175
Register 17: GPTM TimerA (GPTMTAR), offset 0x048 ..... 176
Register 18: GPTM TimerB (GPTMTBR), offset 0x04C ..... 177

**Watchdog Timer ..... 178**

Register 1: Watchdog Load (WDTLOAD), offset 0x000 ..... 181
Register 2: Watchdog Value (WDTVALUE), offset 0x004 ..... 182

Register 3: Watchdog Control (WDTCTL), offset 0x008 ..... 183
Register 4: Watchdog Interrupt Clear (WDTICR), offset 0x00C ..... 184
Register 5: Watchdog Raw Interrupt Status (WDTRIS), offset 0x010 ..... 185
Register 6: Watchdog Masked Interrupt Status (WDTMIS), offset 0x014 ..... 186
Register 7: Watchdog Lock (WDTLOCK), offset 0xC00 ..... 187
Register 8: Watchdog Test (WDTTEST), offset 0x418 ..... 188
Register 9: Watchdog Peripheral Identification 4 (WDTPeriphID4), offset 0xFD0 ..... 189
Register 10: Watchdog Peripheral Identification 5 (WDTPeriphID5), offset 0xFD4 ..... 190
Register 11: Watchdog Peripheral Identification 6 (WDTPeriphID6), offset 0xFD8 ..... 191
Register 12: Watchdog Peripheral Identification 7 (WDTPeriphID7), offset 0xFDC ..... 192
Register 13: Watchdog Peripheral Identification 0 (WDTPeriphID0), offset 0xFE0 ..... 193
Register 14: Watchdog Peripheral Identification 1 (WDTPeriphID1), offset 0xFE4 ..... 194
Register 15: Watchdog Peripheral Identification 2 (WDTPeriphID2), offset 0xFE8 ..... 195
Register 16: Watchdog Peripheral Identification 3 (WDTPeriphID3), offset 0xFEC ..... 196
Register 17: Watchdog PrimeCell Identification 0 (WDTPCellID0), offset 0xFF0 ..... 197
Register 18: Watchdog PrimeCell Identification 1 (WDTPCellID1), offset 0xFF4 ..... 198
Register 19: Watchdog PrimeCell Identification 2 (WDTPCellID2), offset 0xFF8 ..... 199
Register 20: Watchdog PrimeCell Identification 3 (WDTPCellID3 ), offset 0xFFC ..... 200

**Analog-to-Digital Converter (ADC) ..... 201**

Register 1: ADC Active Sample Sequencer (ADCACTSS), offset 0x000 ..... 207
Register 2: ADC Raw Interrupt Status (ADCRIS), offset 0x004 ..... 208
Register 3: ADC Interrupt Mask (ADCIM), offset 0x008 ..... 209
Register 4: ADC Interrupt Status and Clear (ADCISC), offset 0x00C ..... 210
Register 5: ADC Overflow Status (ADCOSTAT), offset 0x010 ..... 211
Register 6: ADC Event Multiplexer Select (ADCEMUX), offset 0x014 ..... 212
Register 7: ADC Underflow Status (ADCUSTAT), offset 0x018 ..... 213
Register 8: ADC Sample Sequencer Priority (ADCSSPRI), offset 0x020 ..... 214
Register 9: ADC Processor Sample Sequence Initiate (ADCPSSI), offset 0x028 ..... 215
Register 10: ADC Sample Averaging Control (ADCSAC), offset 0x030 ..... 216
Register 11: ADC Sample Sequence Input Multiplexer Select 0 (ADCSSMUX0), offset 0x040 ..... 217
Register 12: ADC Sample Sequence Control 0 (ADCSSCTL0), offset 0x044 ..... 219
Register 13: ADC Sample Sequence Result FIFO 0 (ADCSSFIFO0), offset 0x048 ..... 221
Register 14: ADC Sample Sequence FIFO 0 Status (ADCSSFSTAT0), offset 0x04C ..... 222
Register 15: ADC Sample Sequence Input Multiplexer Select 1 (ADCSSMUX1), offset 0x060 ..... 223
Register 16: ADC Sample Sequence Control 1 (ADCSSCTL1), offset 0x064 ..... 224
Register 17: ADC Sample Sequence Result FIFO 1 (ADCSSFIFO1), offset 0x068 ..... 224
Register 18: ADC Sample Sequence FIFO 1 Status (ADCSSFSTAT1), offset 0x06C ..... 224
Register 19: ADC Sample Sequence Input Multiplexer Select 2 (ADCSSMUX2), offset 0x080 ..... 225
Register 20: ADC Sample Sequence Control 2 (ADCSSCTL2), offset 0x084 ..... 226
Register 21: ADC Sample Sequence Result FIFO 2 (ADCSSFIFO2), offset 0x088 ..... 226
Register 22: ADC Sample Sequence FIFO 2 Status (ADCSSFSTAT2), offset 0x08C ..... 226
Register 23: ADC Sample Sequence Input Multiplexer Select 3 (ADCSSMUX3), offset 0x0A0 ..... 227
Register 24: ADC Sample Sequence Control 3 (ADCSSCTL3), offset 0x064 ..... 228
Register 25: ADC Sample Sequence Result FIFO 3 (ADCSSFIFO3), offset 0x0A8 ..... 228
Register 26: ADC Sample Sequence FIFO 3 Status (ADCSSFSTAT3), offset 0x0AC ..... 228
Register 27: ADC Test Mode Loopback (ADCTMLB), offset 0x100 ..... 229

**Universal Asynchronous Receivers/Transmitters (UARTs) ..... 231**

Register 1: UART Data (UARTDR), offset 0x000 ..... 238

Register 2: UART Receive Status/Error Clear (UARTRSR/UARTECR), offset 0x004 ..... 240
Register 3: UART Flag (UARTFR), offset 0x018 ..... 242
Register 4: UART Integer Baud-Rate Divisor (UARTIBRD), offset 0x024 ..... 244
Register 5: UART Fractional Baud-Rate Divisor (UARTFBRD), offset 0x028 ..... 245
Register 6: UART Line Control (UARTLCRH), offset 0x02C ..... 246
Register 7: UART Control (UARTCTL), offset 0x030 ..... 248
Register 8: UART Interrupt FIFO Level Select (UARTIFLS), offset 0x034 ..... 249
Register 9: UART Interrupt Mask (UARTIM), offset 0x038 ..... 250
Register 10: UART Raw Interrupt Status (UARTRIS), offset 0x03C ..... 252
Register 11: UART Masked Interrupt Status (UARTMIS), offset 0x040 ..... 253
Register 12: UART Interrupt Clear (UARTICR), offset 0x044 ..... 254
Register 13: UART Peripheral Identification 4 (UARTPeriphID4), offset 0xFD0 ..... 255
Register 14: UART Peripheral Identification 5 (UARTPeriphID5), offset 0xFD4 ..... 256
Register 15: UART Peripheral Identification 6 (UARTPeriphID6), offset 0xFD8 ..... 257
Register 16: UART Peripheral Identification 7 (UARTPeriphID7), offset 0xFDC ..... 258
Register 17: UART Peripheral Identification 0 (UARTPeriphID0), offset 0xFE0 ..... 259
Register 18: UART Peripheral Identification 1 (UARTPeriphID1), offset 0xFE4 ..... 260
Register 19: UART Peripheral Identification 2 (UARTPeriphID2), offset 0xFE8 ..... 261
Register 20: UART Peripheral Identification 3 (UARTPeriphID3), offset 0xFEC ..... 262
Register 21: UART PrimeCell Identification 0 (UARTPCellID0), offset 0xFF0 ..... 263
Register 22: UART PrimeCell Identification 1 (UARTPCellID1), offset 0xFF4 ..... 264
Register 23: UART PrimeCell Identification 2 (UARTPCellID2), offset 0xFF8 ..... 265
Register 24: UART PrimeCell Identification 3 (UARTPCellID3), offset 0xFFC ..... 266

**Synchronous Serial Interface (SSI) ..... 267**

Register 1: SSI Control 0 (SSICR0), offset 0x000 ..... 279
Register 2: SSI Control 1 (SSICR1), offset 0x004 ..... 281
Register 3: SSI Data (SSIDR), offset 0x008 ..... 283
Register 4: SSI Status (SSISR), offset 0x00C ..... 284
Register 5: SSI Clock Prescale (SSICPSR), offset 0x010 ..... 285
Register 6: SSI Interrupt Mask (SSIIM), offset 0x014 ..... 286
Register 7: SSI Raw Interrupt Status (SSIRIS), offset 0x018 ..... 287
Register 8: SSI Masked Interrupt Status (SSIMIS), offset 0x01C ..... 288
Register 9: SSI Interrupt Clear (SSIICR), offset 0x020 ..... 289
Register 10: SSI Peripheral Identification 4 (SSIPeriphID4), offset 0xFD0 ..... 290
Register 11: SSI Peripheral Identification 5 (SSIPeriphID5), offset 0xFD4 ..... 291
Register 12: SSI Peripheral Identification 6 (SSIPeriphID6), offset 0xFD8 ..... 292
Register 13: SSI Peripheral Identification 7 (SSIPeriphID7), offset 0xFDC ..... 293
Register 14: SSI Peripheral Identification 0 (SSIPeriphID0), offset 0xFE0 ..... 294
Register 15: SSI Peripheral Identification 1 (SSIPeriphID1), offset 0xFE4 ..... 295
Register 16: SSI Peripheral Identification 2 (SSIPeriphID2), offset 0xFE8 ..... 296
Register 17: SSI Peripheral Identification 3 (SSIPeriphID3), offset 0xFEC ..... 297
Register 18: SSI PrimeCell Identification 0 (SSIPCellID0), offset 0xFF0 ..... 298
Register 19: SSI PrimeCell Identification 1 (SSIPCellID1), offset 0xFF4 ..... 299
Register 20: SSI PrimeCell Identification 2 (SSIPCellID2), offset 0xFF8 ..... 300
Register 21: SSI PrimeCell Identification 3 (SSIPCellID3), offset 0xFFC ..... 301

**Inter-Integrated Circuit (I2C) Interface ..... 302**

Register 1: I2C Master Slave Address (I2CMSA), offset 0x000 ..... 313
Register 2: I2C Master Control/Status (I2CMCS), offset 0x004 ..... 314

Register 3: I2C Master Data (I2CMDR), offset 0x008 ..... 319
Register 4: I²C Master Timer Period (I2CMTPR), offset 0x00C ..... 320
Register 5: I²C Master Interrupt Mask (I2CMIMR), offset 0x010 ..... 321
Register 6: I²C Master Raw Interrupt Status (I2CMRIS), offset 0x014 ..... 322
Register 7: I²C Master Masked Interrupt Status (I2CMMIS), offset 0x018 ..... 323
Register 8: I²C Master Interrupt Clear (I2CMICR), offset 0x01C ..... 324
Register 9: I²C Master Configuration (I2CMCR), offset 0x020 ..... 325
Register 10: I²C Slave Own Address (I2CSOAR), offset 0x000 ..... 326
Register 11: I²C Slave Control/Status (I2CSCSR), offset 0x004 ..... 327
Register 12: I²C Slave Data (I2CSDR), offset 0x008 ..... 329
Register 13: I²C Slave Interrupt Mask (I2CSIMR), offset 0x00C ..... 330
Register 14: I²C Slave Raw Interrupt Status (I2CSRIS), offset 0x010 ..... 331
Register 15: I²C Slave Masked Interrupt Status (I2CSMIS), offset 0x014 ..... 332
Register 16: I²C Slave Interrupt Clear (I2CSICR), offset 0x018 ..... 333

**Analog Comparator ..... 334**

Register 1: Analog Comparator Masked Interrupt Status (ACMIS), offset 0x00 ..... 338
Register 2: Analog Comparator Raw Interrupt Status (ACRIS), offset 0x04 ..... 339
Register 3: Analog Comparator Interrupt Enable (ACINTEN), offset 0x08 ..... 340
Register 4: Analog Comparator Reference Voltage Control (ACREFCTL), offset 0x10 ..... 341
Register 5: Analog Comparator Status 0 (ACSTAT0), offset 0x20 ..... 342
Register 6: Analog Comparator Control 0 (ACCTL0), offset 0x24 ..... 343

**Pulse Width Modulator (PWM) ..... 345**

Register 1: PWM Master Control (PWMCTL), offset 0x000 ..... 352
Register 2: PWM Time Base Sync (PWMSYNC), offset 0x004 ..... 353
Register 3: PWM Output Enable (PWMENABLE), offset 0x008 ..... 354
Register 4: PWM Output Inversion (PWMINVERT), offset 0x00C ..... 355
Register 5: PWM Output Fault (PWMFAULT), offset 0x010 ..... 356
Register 6: PWM Interrupt Enable (PWMINTEN), offset 0x014 ..... 357
Register 7: PWM Raw Interrupt Status (PWMRIS), offset 0x018 ..... 358
Register 8: PWM Interrupt Status and Clear (PWMISC), offset 0x01C ..... 359
Register 9: PWM Status (PWMSTATUS), offset 0x020 ..... 360
Register 10: PWM0 Control (PWM0CTL), offset 0x040 ..... 361
Register 11: PWM1 Control (PWM1CTL), offset 0x080 ..... 361
Register 12: PWM0 Interrupt/Trigger Enable (PWM0INTEN), offset 0x044 ..... 363
Register 13: PWM1 Interrupt/Trigger Enable (PWM1INTEN), offset 0x084 ..... 363
Register 14: PWM0 Raw Interrupt Status (PWM0RIS), offset 0x048 ..... 365
Register 15: PWM1 Raw Interrupt Status (PWM1RIS), offset 0x088 ..... 365
Register 16: PWM0 Interrupt Status and Clear (PWM0ISC), offset 0x04C ..... 366
Register 17: PWM1 Interrupt Status and Clear (PWM1ISC), offset 0x08C ..... 366
Register 18: PWM0 Load (PWM0LOAD), offset 0x050 ..... 367
Register 19: PWM1 Load (PWM1LOAD), offset 0x090 ..... 367
Register 20: PWM0 Counter (PWM0COUNT), offset 0x054 ..... 368
Register 21: PWM1 Counter (PWM1COUNT), offset 0x094 ..... 368
Register 22: PWM0 Compare A (PWM0CMPA), offset 0x058 ..... 369
Register 23: PWM1 Compare A (PWM1CMPA), offset 0x098 ..... 369
Register 24: PWM0 Compare B (PWM0CMPB), offset 0x05C ..... 370
Register 25: PWM1 Compare B (PWM1CMPB), offset 0x09C ..... 370
Register 26: PWM0 Generator A Control (PWM0GENA), offset 0x060 ..... 371
Register 27: PWM1 Generator A Control (PWM1GENA), offset 0x0A0 ..... 371

Register 28: PWM0 Generator B Control (PWM0GENB), offset 0x064 ..... 373
Register 29: PWM1 Generator B Control (PWM1GENB), offset 0x0A4 ..... 373
Register 30: PWM0 Dead-Band Control (PWM0DBCTL), offset 0x068 ..... 374
Register 31: PWM1 Dead-Band Control (PWM1DBCTL), offset 0x0A8 ..... 374
Register 32: PWM0 Dead-Band Rising-Edge Delay (PWM0DBRISE), offset 0x06C ..... 375
Register 33: PWM1 Dead-Band Rising-Edge Delay (PWM1DBRISE), offset 0x0AC ..... 375
Register 34: PWM0 Dead-Band Falling-Edge-Delay (PWM0DBFALL), offset 0x070 ..... 376
Register 35: PWM1 Dead-Band Falling-Edge-Delay (PWM1DBFALL), offset 0x0B0 ..... 376

# Revision History

This table provides a summary of the document revisions.

| Date | Revision | Description |
|---|---|---|
| May 2006 | 00 | Initial public release of LM3S301, LM3S310, LM3S315, and LM3S316 data sheets. |
| July 2006 | 01 | Second release of LM3S301, LM3S310, LM3S315, and LM3S316 data sheets. Includes the following changes:<br>• Added initialization and configuration content into PWM, I2C, Comparators, and JTAG chapters.<br>• Modified MAXADCSPD bit description to show how to set speed lower.<br>• Clarified that peripheral clock must be set before enabling peripherals in “Initialization and Configuration” sections. |
| October 2006 | 02 | Third release of LM3S301, LM3S310, LM3S315, and LM3S316 data sheets. Includes the following changes:<br>• Added information on hardware averaging to the ADC chapter.<br>• Updated the clocking examples in the I2C chapter.<br>• Added Serial Flash Loader usage information.<br>• Added “5-V-tolerant” description for GPIOs to feature list, GPIO chapter, and Electrical chapter.<br>• Added maximum values for 20 MHz and 25 MHz parts to Table 9-1, “16-Bit Timer With Prescaler Configurations” in the Timers chapter.<br>• Made the following changes in the System Control chapter:<br>- Updated field descriptions in the **Run-Mode Clock Configuration (RCC)** register and the **Device Identification 1 (DID1)** register.<br>- Updated the internal oscillator clock speed.<br>- Added the **Deep-Sleep Clock Configuration (DSLPCFG)** register.<br>- Added bus fault information to the clock gating registers. |

# About This Document

This data sheet provides reference information for the LM3S316 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 documents are referenced by the data sheet, and available on the documentation CD or from the Luminary Micro web site at www.luminarymicro.com:

- *ARM® Cortex™-M3 Technical Reference Manual*
- *CoreSight™ Design Kit Technical Reference Manual*
- *ARM® v7-M Architecture Application Level Reference Manual*

The following related documents are also referenced:

- *IEEE Standard 1149.1-Test Access Port and Boundary-Scan Architecture*

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

## Documentation Conventions

This document uses the conventions shown in Table 0-1.

**Table 0-1. 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 3-1, "Memory Map," on page 37. |

**Table 0-1. Documentation Conventions**

| Notation | Meaning |
|---|---|
| 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. Reserved bits return an indeterminate value, and should never be changed. Only write a reserved bit with its current value. |
| *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. |
| RO | Software can read this field. Always write the chip reset value. |
| R/W | Software can read or write this field. |
| 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. |
| 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 0-1. Documentation Conventions**

| 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. Binary numbers are indicated with a b suffix, for example, 1011b. Decimal numbers are written without a prefix or suffix. |

# 1 Architectural Overview

The Luminary Micro 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 LM3S316 controller in the Stellaris family offers the advantages of ARM's widely available development tools, System-on-Chip (SoC) infrastructure IP applications, and a large user community. Additionally, the controller uses ARM's Thumb®-compatible Thumb-2 instruction set to reduce memory requirements and, thereby, cost.

Luminary Micro offers a complete solution to get to market quickly, with a customer development board, white papers and application notes, and a strong support, sales, and distributor network.

## 1.1 Product Features

The LM3S316 microcontroller includes the following product features:

- 32-Bit RISC Performance
  - 32-bit ARM® Cortex™-M3 v7M architecture optimized for small-footprint embedded applications
  - Thumb®-compatible Thumb-2-only instruction set processor core for high code density
  - 25-MHz operation
  - Hardware-division and single-cycle-multiplication
  - Integrated Nested Vectored Interrupt Controller (NVIC) providing deterministic interrupt handling
  - 26 interrupts with eight priority levels
  - Memory protection unit (MPU) provides a privileged mode for protected operating system functionality
  - Unaligned data access, enabling data to be efficiently packed into memory
  - Atomic bit manipulation (bit-banding) delivers maximum memory utilization and streamlined peripheral control
- Internal Memory
  - 16 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
  - 4 KB single-cycle SRAM
- General-Purpose Timers
  - Three timers, each of which can be configured: as a single 32-bit timer , as two 16-bit timers, or to initiate an ADC event
  - 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 in periodic and one-shot mode when the controller asserts the CPU Halt flag during debug
    - ADC event trigger
  - 16-bit Timer modes:
    - General-purpose timer function with an 8-bit prescaler
    - 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
- Synchronous Serial Interface (SSI)
  - Master or slave operation
  - 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
- UART
  - Two fully programmable 16C550-type UARTs
  - Separate 16x8 transmit (TX) and 16x12 receive (RX) FIFOs to reduce CPU interrupt service loading
  - Programmable baud-rate generator with fractional divider
  - 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

- False-start-bit detection
- Line-break generation and detection

- ADC
  - Single- and differential-input configurations
  - Four 10-bit channels (inputs) when used as single-ended inputs
  - Sample rate of 250 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
  - Each sequence triggered by software or internal event (timers, analog comparators, PWM or GPIO)
- Analog Comparator
  - Configurable for output to: drive an output pin, generate an interrupt, or initiate an ADC sample sequence
  - Compare external pin input to external pin input or to internal programmable voltage reference
- $I^2C$
  - Master and slave receive and transmit operation with transmission speed up to 100 Kbps in Standard mode and 400 Kbps in Fast mode
  - Interrupt generation
  - Master with arbitration and clock synchronization, multimaster support, and 7-bit addressing mode
- PWM
  - Two PWM generator blocks, *each* with one 16-bit counter, two comparators, a PWM generator, and a dead-band generator
  - One 16-bit counter
    - Runs in Down or Up/Down mode
    - Output frequency controlled by a 16-bit load value
    - Load value updates can be synchronized
    - Produces output signals at zero and load value
  - Two comparators
    - Comparator value updates can be synchronized
    - Produces output signals on match
  - PWM generator
    - Output PWM signal is constructed based on actions taken as a result of the counter and comparator output signals
    - Produces two independent PWM signals
  - Dead-band generator
    - Produces two PWM signals with programmable dead-band delays suitable for driving a half-H bridge

    - Can be bypassed, leaving input PWM signals unmodified
  - Flexible output control block with PWM output enable of each PWM signal
    - PWM output enable of each PWM signal
    - Optional output inversion of each PWM signal (polarity control)
    - Optional fault handling for each PWM signal
    - Synchronization of timers in the PWM generator blocks
    - Synchronization of timer/comparator updates across the PWM generator blocks
    - Interrupt status summary of the PWM generator blocks
  - Can initiate an ADC sample sequence
- GPIOs
  - 3 to 32 GPIOs, depending on configuration
  - 5-V-tolerant input/outputs
  - Programmable interrupt generation as either edge-triggered or level-sensitive
  - Bit masking in both read and write operations through address lines
  - Can initiate an ADC sample sequence
  - Programmable control for GPIO pad configuration:
    - Weak pull-up or pull-down resistors
    - 2-mA, 4-mA, and 8-mA pad drive
    - Slew rate control for the 8-mA drive
    - Open drain enables
    - Digital input enables
- Power
  - On-chip Low Drop-Out (LDO) voltage regulator, with programmable output user-adjustable from 2.25 V to 2.75 V
  - Low-power options on controller: Sleep and Deep-sleep modes
  - Low-power options for peripherals: software controls shutdown of individual peripherals
  - User-enabled LDO unregulated voltage detection and automatic reset
  - 3.3-V supply brownout detection and reporting via interrupt or reset
  - On-chip temperature sensor
- 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
- Additional Features

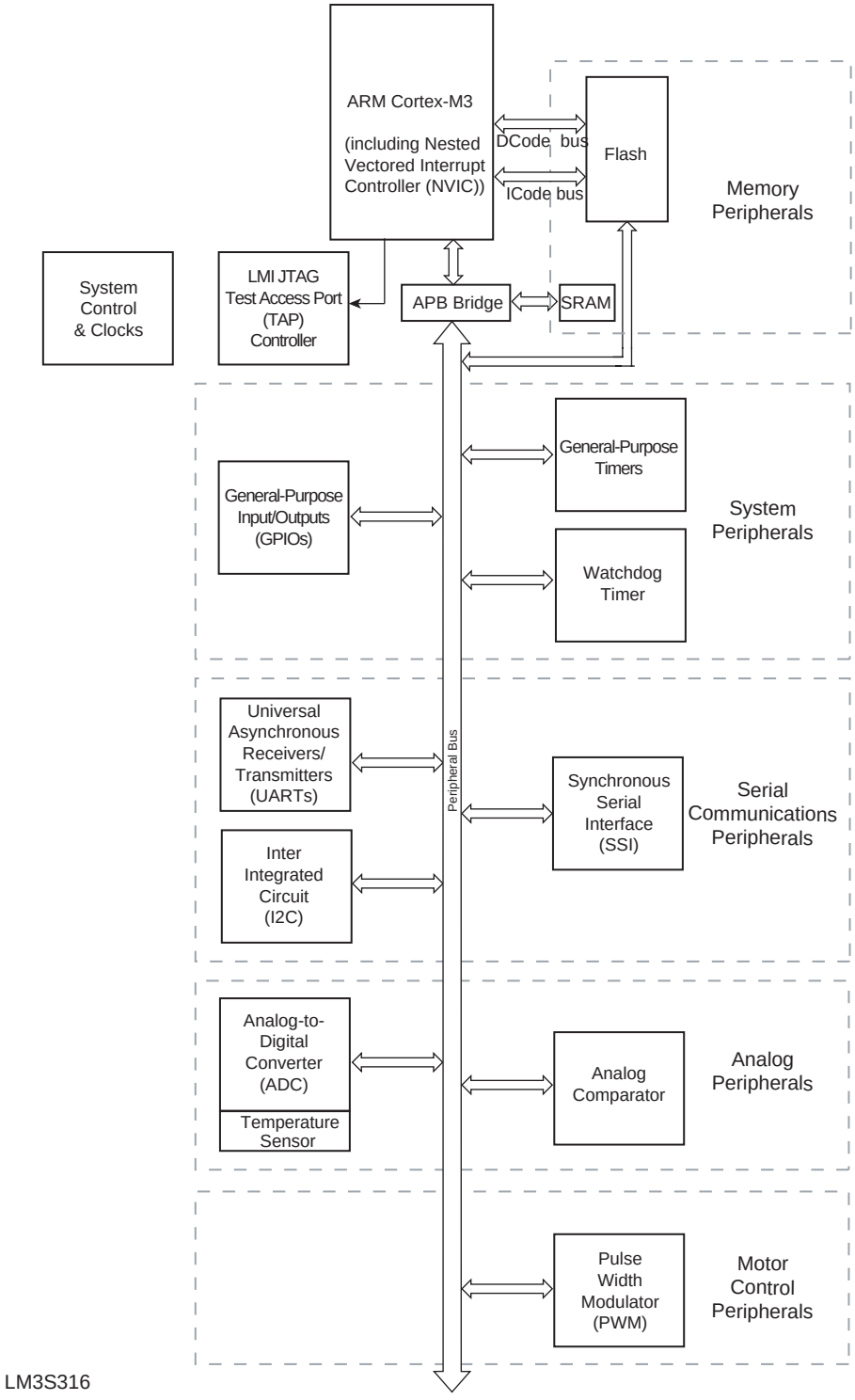
  - Six reset sources
  - Programmable clock source control
  - Clock gating to individual peripherals for power savings
  - IEEE 1149.1-1990 compliant Test Access Port (TAP) controller
  - Debug access via JTAG and Serial Wire interfaces
  - Full JTAG boundary scan
- Industrial-range 48-pin RoHS-compliant LQFP package

## 1.2 Target Applications

- Factory automation and control
- Industrial control power devices
- Building and home automation
- Stepper motors

## 1.3 High-Level Block Diagram

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



# 1.4 Functional Overview

The following sections provide an overview of the features of the LM3S316 microcontroller. The chapter 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 408.

## 1.4.1 ARM Cortex™-M3

### 1.4.1.1 Processor Core (Section 2 on page 34)

All members of the Stellaris product family, including the LM3S316 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.

Section 2, “ARM Cortex-M3 Processor Core,” on page 34 provides an overview of the ARM core; the core is detailed in the *ARM® Cortex™-M3 Technical Reference Manual*.

### 1.4.1.2 Nested Vectored Interrupt Controller (NVIC)

The LM3S316 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 26 interrupts.

Section 4, “Interrupts,” on page 39 provides an overview of the NVIC controller and the interrupt map. Exceptions and interrupts are detailed in the *ARM® Cortex™-M3 Technical Reference Manual*.

## 1.4.2 Motor Control Peripherals

To enhance motor control, the LM3S316 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 LM3S316, PWM motion control functionality can be achieved through dedicated, flexible motion control hardware (the PWM pins) or through the motion control features of the general-purpose timers (using the CCP pins).

#### *PWM Pins (Section 16 on page 345)*

The LM3S316 PWM module consists of two PWM generator blocks and a control block. Each PWM generator block contains one timer (16-bit down or up/down counter), two comparators, a PWM signal generator, a dead-band generator, and an interrupt/ADC-trigger selector. The control block determines the polarity of the PWM signals, and which signals are passed through to the pins.

Each PWM generator block produces two PWM signals that can either be independent signals or a single pair of complementary signals with dead-band delays inserted. The output of the PWM generation blocks are managed by the output control block before being passed to the device pins.

***CCP Pins (“16-Bit PWM Mode” on page 155)***

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 LM3S316 controller offers an Analog-to-Digital Converter (ADC) and an analog comparator.

### 1.4.3.1 ADC (Section 11 on page 201)

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 four 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.

### 1.4.3.2 Analog Comparator (Section 15 on page 334)

An analog comparator is a peripheral that compares two analog voltages, and provides a logical output that signals the comparison result.

The LM3S316 controller provides one independent integrated analog comparators that can be configured to drive an output or generate an interrupt or ADC event.

A comparator can compare a test voltage against any one of these voltages:

- An individual external reference voltage
- A shared single external reference voltage
- A shared internal reference voltage

The comparator can provide its output to a device pin, acting as a replacement for an analog comparator on the board, or it can be used to signal the application via interrupts or triggers to the ADC to cause it to start capturing a sample sequence. The interrupt generation and ADC triggering logic is separate. This means, for example, that an interrupt can be generated on a rising edge and the ADC triggered on a falling edge.

## 1.4.4 Serial Communications Peripherals

The LM3S316 controller supports both asynchronous and synchronous serial communications with two fully programmable 16C550-type UARTs, SSI and $I^2C$ serial communications.

### 1.4.4.1 UART (Section 12 on page 231)

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 LM3S316 controller includes two fully programmable 16C550-type UARTs that support data transfer speeds up to 460.8 Kbps. (Although similar in functionality to a 16C550 UART, it is not register compatible.)

Separate 16x8 transmit (TX) and 16x12 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 (Section 13 on page 267)

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

The Stellaris SSI module 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 $I^2C$ (Section 14 on page 302)

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 $I^2C$ bus 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 Stellaris $I^2C$ module provides the ability to communicate to other IC devices over an $I^2C$ bus. The $I^2C$ bus supports devices that can both transmit and receive (write and read) data.

Devices on the $I^2C$ bus can be designated as either a master or a slave. The $I^2C$ 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 $I^2C$ modes are: Master Transmit, Master Receive, Slave Transmit, and Slave Receive.

The Stellaris $I^2C$ module can operate at two speeds: Standard (100 Kbps) and Fast (400 Kbps).

Both the $I^2C$ master and slave can generate interrupts. The $I^2C$ master generates interrupts when a transmit or receive operation completes (or aborts due to an error). The $I^2C$ slave generates interrupts when data has been sent or requested by a master.

### 1.4.5 System Peripherals

#### 1.4.5.1 Programmable GPIOs (Section 8 on page 108)

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

The Stellaris GPIO module is composed 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 3 to 32 programmable input/output pins. The number of GPIOs available depends on the peripherals being used (see Table 18-4 on page 386 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.

#### 1.4.5.2 Three Programmable Timers (Section 9 on page 146)

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 timer/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 one-shot timer, periodic timer, or Real-Time Clock (RTC). 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 (Section 10 on page 178)

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 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 Stellaris controllers offer both SRAM and Flash memory.

#### 1.4.6.1 SRAM (Section 7.2.1 on page 94)

The LM3S316 static random access memory (SRAM) controller supports 4 KB SRAM. The internal SRAM of the Stellaris devices is located at address 0x20000000 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 (Section 7.2.2 on page 95)

The LM3S316 Flash controller supports 16 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.

## 1.4.7 Additional Features

### 1.4.7.1 Memory Map (Section 3 on page 37)

A memory map lists the location of instructions and data in memory. The memory map for the LM3S316 controller can be found on page 37. Register addresses are given as a hexadecimal increment, relative to the module's base address as shown in the memory map.

The *ARM® Cortex™-M3 Technical Reference Manual* provides further information on the memory map.

### 1.4.7.2 JTAG TAP Controller (Section 5 on page 42)

The Joint Test Action Group (JTAG) port provides a standardized serial interface for controlling the Test Access Port (TAP) and associated test logic. The TAP, JTAG instruction register, and JTAG data registers can be used to test the interconnects of assembled printed circuit boards, obtain manufacturing information on the components, and observe and/or control the inputs and outputs of the controller during normal operation. The JTAG port provides a high degree of testability and chip-level access at a low cost.

The JTAG port is comprised of the standard five pins: $\overline{\text{TRST}}$, 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 LMI 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 LMI JTAG instructions select the LMI TDO outputs. The multiplexer is controlled by the LMI JTAG controller, which has comprehensive programming for the ARM, LMI, and unimplemented JTAG instructions.

### 1.4.7.3 System Control and Clocks (Section 6 on page 52)

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.

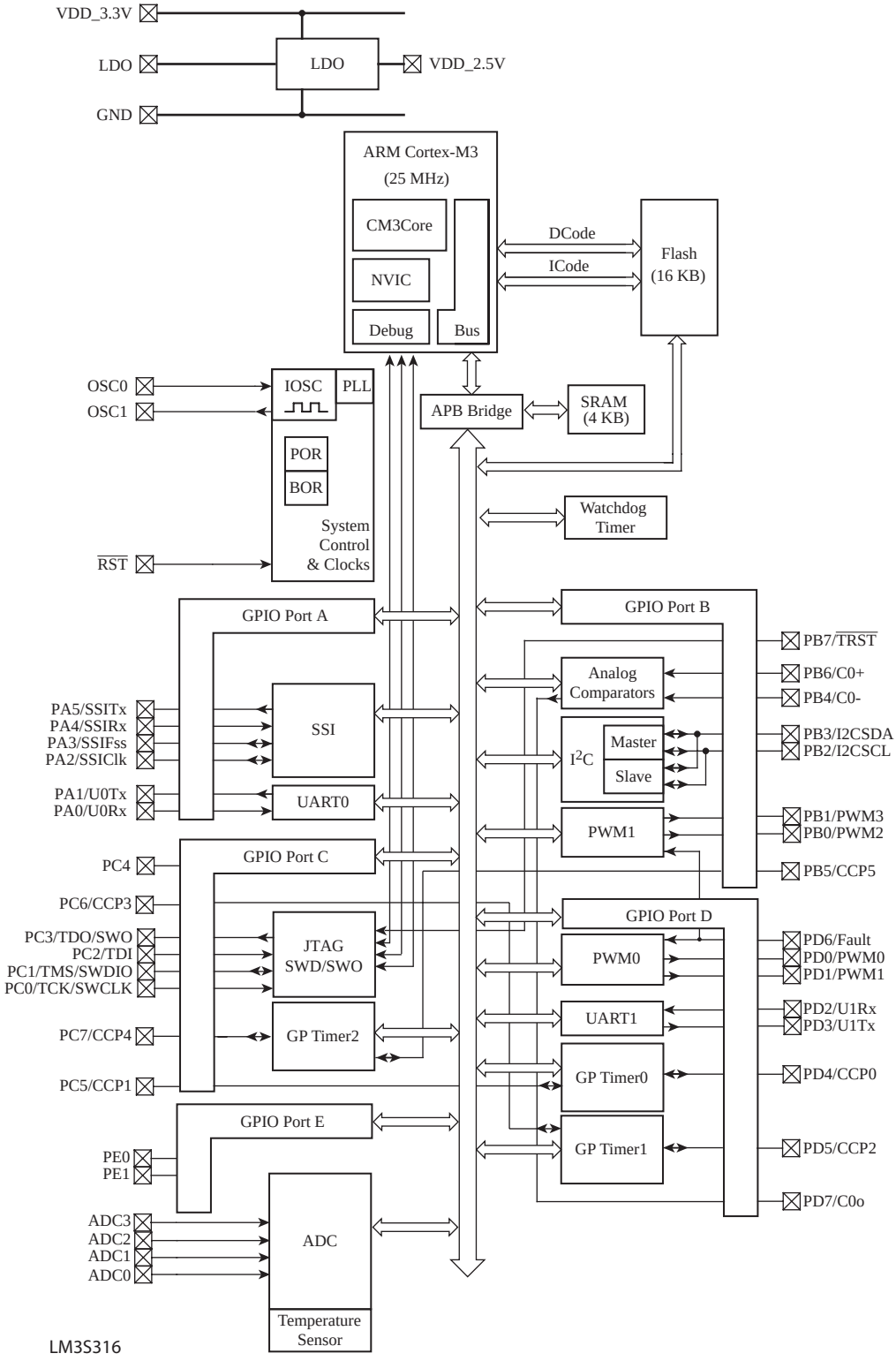
## 1.4.8 Hardware Details

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

- Section 17, "Pin Diagram," on page 377
- Section 18, "Signal Tables," on page 378
- Section 19, "Operating Characteristics," on page 388
- Section 20, "Electrical Characteristics," on page 389
- Section 21, "Package Information," on page 403

## 1.5 System Block Diagram

**Figure 1-2. LM3S316 Controller System-Level Block Diagram**



# 2 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. Features include:

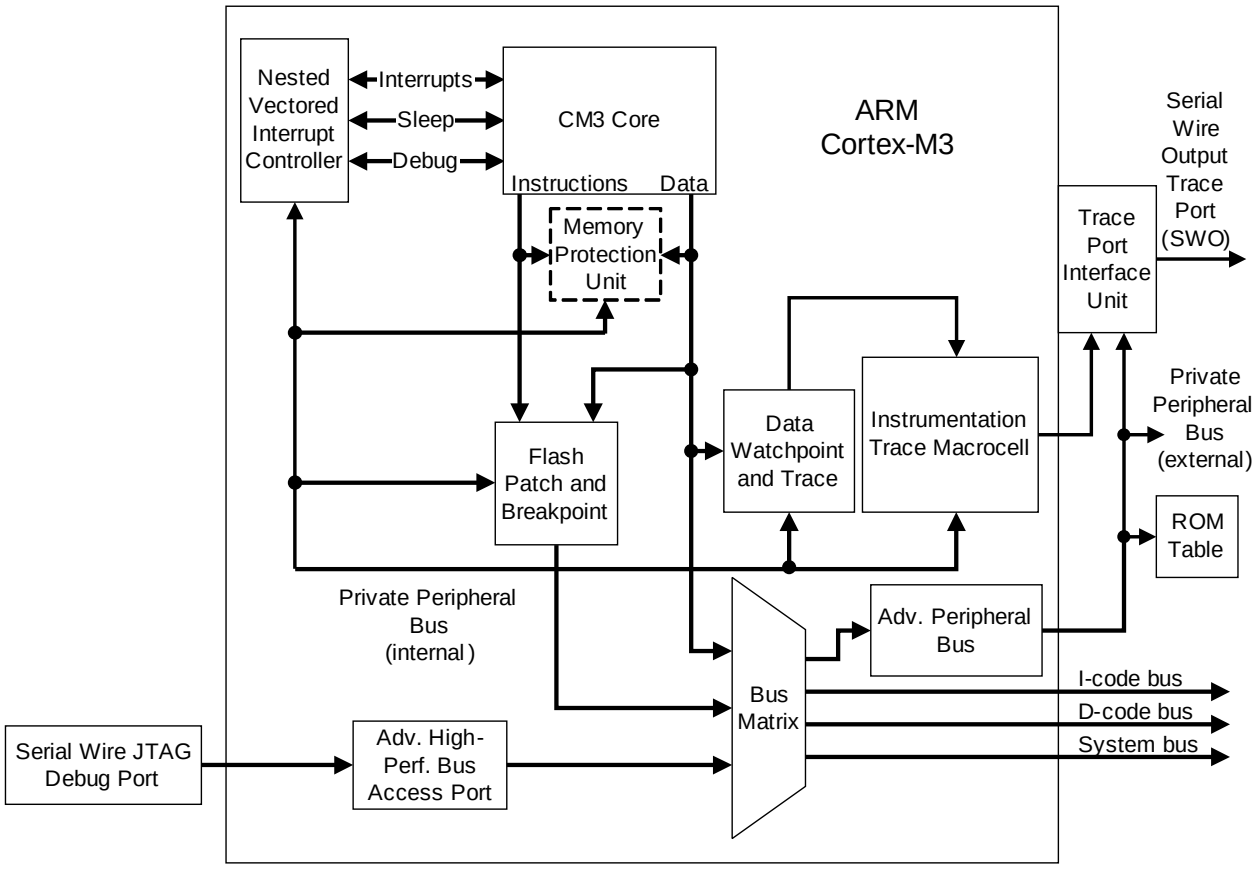
- Compact core.
- Thumb-2 instruction set, delivering the high-performance expected of an ARM core in the memory size usually associated with 8- and 16-bit devices; typically in the range of a few kilobytes of memory for microcontroller class applications.
- Exceptional interrupt handling, by implementing the register manipulations required for handling an interrupt in hardware.
- Memory protection unit (MPU) to provide a privileged mode of operation for complex applications.
- Full-featured debug solution with a:
  - 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

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, and building and home automation.

For more information on the ARM Cortex-M3 processor core, see the *ARM® Cortex™-M3 Technical Reference Manual*. For information on SWJ-DP, see the *CoreSight™ Design Kit Technical Reference Manual*.

## 2.1 Block Diagram

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




## 2.2 Functional Description

**Important:** The *ARM® Cortex™-M3 Technical Reference Manual* describes all the features of an ARM Cortex-M3 in detail. However, these features differ based on the implementation. This section describes the Stellaris implementation.

Luminary Micro has implemented the ARM Cortex-M3 core as shown in Figure 2-1. As noted in the *ARM® Cortex™-M3 Technical Reference Manual*, several Cortex-M3 components are flexible in their implementation: SW/JTAG-DP, ETM, TPIU, the ROM table, the MPU, and the Nested Vectored Interrupt Controller (NVIC). Each of these is addressed in the sections that follow.

### 2.2.1 Serial Wire and JTAG Debug

Luminary Micro has replaced the ARM SW-DP and JTAG-DP with the ARM CoreSight™-compliant Serial Wire JTAG Debug Port (SWJ-DP) interface. This means Chapter 12, “Debug Port,” of the *ARM® Cortex™-M3 Technical Reference Manual* does not apply to Stellaris devices.

The SWJ-DP interface combines the SWD and JTAG debug ports into one module. See the *CoreSight™ Design Kit Technical Reference Manual* for details on SWJ-DP.

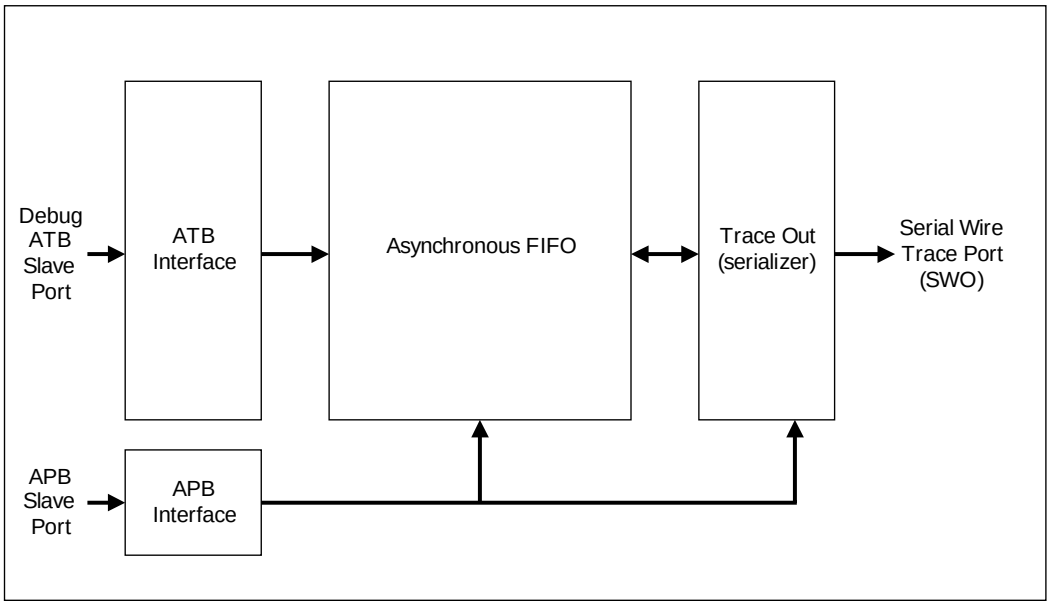
### 2.2.2 Embedded Trace Macrocell (ETM)

ETM was not implemented in the Stellaris devices. This means Chapters 15 and 16 of the *ARM® Cortex™-M3 Technical Reference Manual* can be ignored.

### 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. The Stellaris devices have implemented TPIU as shown in Figure 2-2. This is similar to the non-ETM version described in the *ARM® Cortex™-M3 Technical Reference Manual*, however, SWJ-DP only provides `SWV` output for the TPIU.

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




### 2.2.4 ROM Table

The default ROM table was implemented as described in the *ARM® Cortex™-M3 Technical Reference Manual*.

### 2.2.5 Memory Protection Unit (MPU)

The Memory Protection Unit (MPU) is included on the LM3S316 controller and 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.

### 2.2.6 Nested Vectored Interrupt Controller (NVIC)

#### 2.2.6.1 Interrupts

The *ARM® Cortex™-M3 Technical Reference Manual* describes the maximum number of interrupts and interrupt priorities. The LM3S316 microcontroller supports 26 interrupts with eight priority levels.

#### 2.2.6.2 SysTick Calibration Value Registers

The SysTick Calibration Value register is not implemented.

# 3 Memory Map

The memory map for the LM3S316 is provided in Table 3-1. In this manual, register addresses are given as a hexadecimal increment, relative to the module's base address as shown in the memory map. See also Chapter 4, "Memory Map" in the *ARM® Cortex™-M3 Technical Reference Manual*.

**Table 3-1. Memory Map (Sheet 1 of 2)**

| Start | End | Description | For details on registers, see ... |
|---|---|---|---|
| **Memory** | | | |
| 0x00000000 | 0x00003FFF | On-chip flash | page 98 |
| 0x00004000 | 0x1FFFFFFF | Reserved[a] | |
| 0x20000000 | 0x20000FFF | Bit-banded on-chip SRAM | - |
| 0x20001000 | 0x200FFFFF | Reserved[a] | - |
| 0x22000000 | 0x2201FFFF | Bit-band alias of 0x20000000 through 0x20000FFF | - |
| 0x22020000 | 0x23FFFFFF | Reserved[a] | - |
| **FiRM Peripherals** | | | |
| 0x40000000 | 0x40000FFF | Watchdog timer | page 180 |
| 0x40001000 | 0x40003FFF | Reserved for three additional watchdog timers (per FiRM specification)[a] | - |
| 0x40004000 | 0x40004FFF | GPIO Port A | page 115 |
| 0x40005000 | 0x40005FFF | GPIO Port B | page 115 |
| 0x40006000 | 0x40006FFF | GPIO Port C | page 115 |
| 0x40007000 | 0x40007FFF | GPIO Port D | |
| 0x40008000 | 0x40008FFF | SSI | page 278 |
| 0x40009000 | 0x4000BFFF | Reserved for three additional SSIs (per FiRM specification)[a] | - |
| 0x4000C000 | 0x4000CFFF | UART0 | page 237 |
| 0x4000D000 | 0x4000DFFF | UART1 | page 237 |
| 0x4000E000 | 0x4000FFFF | Reserved for two additional UARTs (per FiRM specification)[a] | - |
| 0x40010000 | 0x4001FFFF | Reserved for future FiRM peripherals[a] | - |
| **Peripherals** | | | |
| 0x40020000 | 0x400207FF | $I^2C$ Master | page 312 |
| 0x40020800 | 0x40020FFF | $I^2C$ Slave | page 326 |
| 0x40021000 | 0x40023FFF | Reserved[a] | - |

**Table 3-1. Memory Map (Sheet 2 of 2)**

| Start | End | Description | For details on registers, see ... |
|---|---|---|---|
| 0x40024000 | 0x40024FFF | GPIO Port E | page 115 |
| 0x40025000 | 0x40025FFF | Reserved[a] | - |
| 0x40028000 | 0x40028FFF | PWM | page 351 |
| 0x40029000 | 0x4002BFFF | Reserved[a] | - |
| 0x4002C000 | 0x4002FFFF | Reserved[a] | - |
| 0x40030000 | 0x40030FFF | Timer0 | page 157 |
| 0x40031000 | 0x40031FFF | Timer1 | page 157 |
| 0x40032000 | 0x40032FFF | Timer2 | page 157 |
| 0x40033000 | 0x40037FFF | Reserved[a] | - |
| 0x40038000 | 0x40038FFF | ADC | page 206 |
| 0x40039000 | 0x4003BFFF | Reserved[a] | - |
| 0x4003C000 | 0x4003CFFF | Analog comparator | page 337 |
| 0x4003D000 | 0x400FCFFF | Reserved[a] | - |
| 0x400FD000 | 0x400FDFFF | Flash control | page 98 |
| 0x400FE000 | 0x400FFFFF | System control | page 59 |
| 0x40100000 | 0x41FFFFFF | Reserved[a] | - |
| 0x42000000 | 0x43FFFFFF | Bit-band alias of 0x40000000 through 0x400FFFFF | - |
| 0x44000000 | 0xDFFFFFFF | Reserved[a] | - |
| **Private Peripheral Bus** | | | |
| 0xE0000000 | 0xE0000FFF | Instrumentation Trace Macrocell (ITM) | *ARM® Cortex™-M3 Technical Reference Manual* |
| 0xE0001000 | 0xE0001FFF | Data Watchpoint and Trace (DWT) | |
| 0xE0002000 | 0xE0002FFF | Flash Patch and Breakpoint (FPB) | |
| 0xE0003000 | 0xE000DFFF | Reserved[a] | |
| 0xE000E000 | 0xE000EFFF | Nested Vectored Interrupt Controller (NVIC) | |
| 0xE000F000 | 0xE003FFFF | Reserved[a] | |
| 0xE0040000 | 0xE0040FFF | Trace Port Interface Unit (TPIU) | |
| 0xE0041000 | 0xE0041FFF | Reserved[a] | - |
| 0xE0042000 | 0xE00FFFFF | Reserved[a] | - |
| 0xE0100000 | 0xFFFFFFFF | Reserved for vendor peripherals[a] | - |

a. All reserved space returns a bus fault when read or written.

# 4 Interrupts

The ARM Cortex-M3 processor and the Nested Vectored Interrupt Controller (NVIC) 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.

Table 4-1 lists all the exceptions. Software can set eight priority levels on seven of these exceptions (system handlers) as well as on 26 interrupts (listed in Table 4-2). Priorities on the system handlers are set with the NVIC System Handler Priority registers. Interrupts are enabled through the NVIC Interrupt Set Enable register and prioritized with the NVIC Interrupt Priority registers. You can also group priorities by splitting priority levels into pre-emption priorities and subpriorities. All the interrupt registers are described in Chapter 8, “Nested Vectored Interrupt Controller” in the *ARM® Cortex™-M3 Technical Reference Manual*.

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

If you assign the same priority level to two or more interrupts, their hardware priority (the lower the position number) determines the order in which the processor activates them. For example, if both GPIO Port A and GPIO Port B are priority level 1, then GPIO Port A has higher priority.

See Chapter 5, “Exceptions” and Chapter 8, “Nested Vectored Interrupt Controller” in the *ARM® Cortex™-M3 Technical Reference Manual* for more information on exceptions and interrupts.

**Table 4-1. Exception Types**

| Exception Type | Position | Priority[a] | Description |
|---|---|---|---|
| - | 0 | - | Stack top is loaded from first entry of vector table on reset. |
| Reset | 1 | -3 (highest) | Invoked on power up and warm reset. On first instruction, drops to lowest priority (and then is called the base level of activation). This is asynchronous. |
| Non-Maskable Interrupt (NMI) | 2 | -2 | Cannot be stopped or preempted by any exception but reset. This is asynchronous.<br>An NMI is only producible by software, using the NVIC **Interrupt Control State** register. |
| Hard Fault | 3 | -1 | All classes of Fault, when the fault cannot activate due to priority or the configurable fault handler has been disabled. This is synchronous. |
| Memory Management | 4 | settable | MPU mismatch, including access violation and no match. This is synchronous.<br>The priority of this exception can be changed. |
| Bus Fault | 5 | settable | Pre-fetch fault, memory access fault, and other address/memory related faults. This is synchronous when precise and asynchronous when imprecise.<br>You can enable or disable this fault. |

**Table 4-1. Exception Types (Continued)**

| Exception Type | Position | Priority[a] | Description |
|---|---|---|---|
| Usage Fault | 6 | settable | Usage fault, such as undefined instruction executed or illegal state transition attempt. This is synchronous. |
| - | 7-10 | - | Reserved. |
| SVCall | 11 | settable | System service call with SVC instruction. This is synchronous. |
| Debug Monitor | 12 | settable | Debug monitor (when not halting). This is synchronous, but only active when enabled. It does not activate if lower priority than the current activation. |
| - | 13 | - | Reserved. |
| PendSV | 14 | settable | Pendable request for system service. This is asynchronous and only pended by software. |
| SysTick | 15 | settable | System tick timer has fired. This is asynchronous. |
| Interrupts | 16 and above | settable | Asserted from outside the ARM Cortex-M3 core and fed through the NVIC (prioritized). These are all asynchronous. Table 4-2 lists the interrupts on the LM3S316 controller. |

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

**Table 4-2. Interrupts**

| Interrupt (Bit in Interrupt Registers) | Description |
|---|---|
| 0 | GPIO Port A |
| 1 | GPIO Port B |
| 2 | GPIO Port C |
| 3 | GPIO Port D |
| 4 | GPIO Port E |
| 5 | UART0 |
| 6 | UART1 |
| 7 | SSI |
| 8 | $I^2C$ |
| 9 | PWM Fault |
| 10 | PWM Generator 0 |
| 11 | PWM Generator 1 |
| 12 | Reserved |

**Table 4-2. Interrupts (Continued)**

| Interrupt (Bit in Interrupt Registers) | Description |
|---|---|
| 13 | Reserved |
| 14 | ADC Sequence 0 |
| 15 | ADC Sequence 1 |
| 16 | ADC Sequence 2 |
| 17 | ADC Sequence 3 |
| 18 | Watchdog timer |
| 19 | Timer0a |
| 20 | Timer0b |
| 21 | Timer1a |
| 22 | Timer1b |
| 23 | Timer2a |
| 24 | Timer2b |
| 25 | Analog Comparator 0 |
| 26 | Reserved |
| 27 | Reserved |
| 28 | System Control |
| 29 | Flash Control |
| 30 | Reserved |
| 31 | Reserved |

# 5 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 the standard five pins: $\overline{\text{TRST}}$, 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 LMI 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 LMI JTAG instructions select the LMI TDO outputs. The multiplexer is controlled by the LMI JTAG controller, which has comprehensive programming for the ARM, LMI, and unimplemented JTAG instructions.

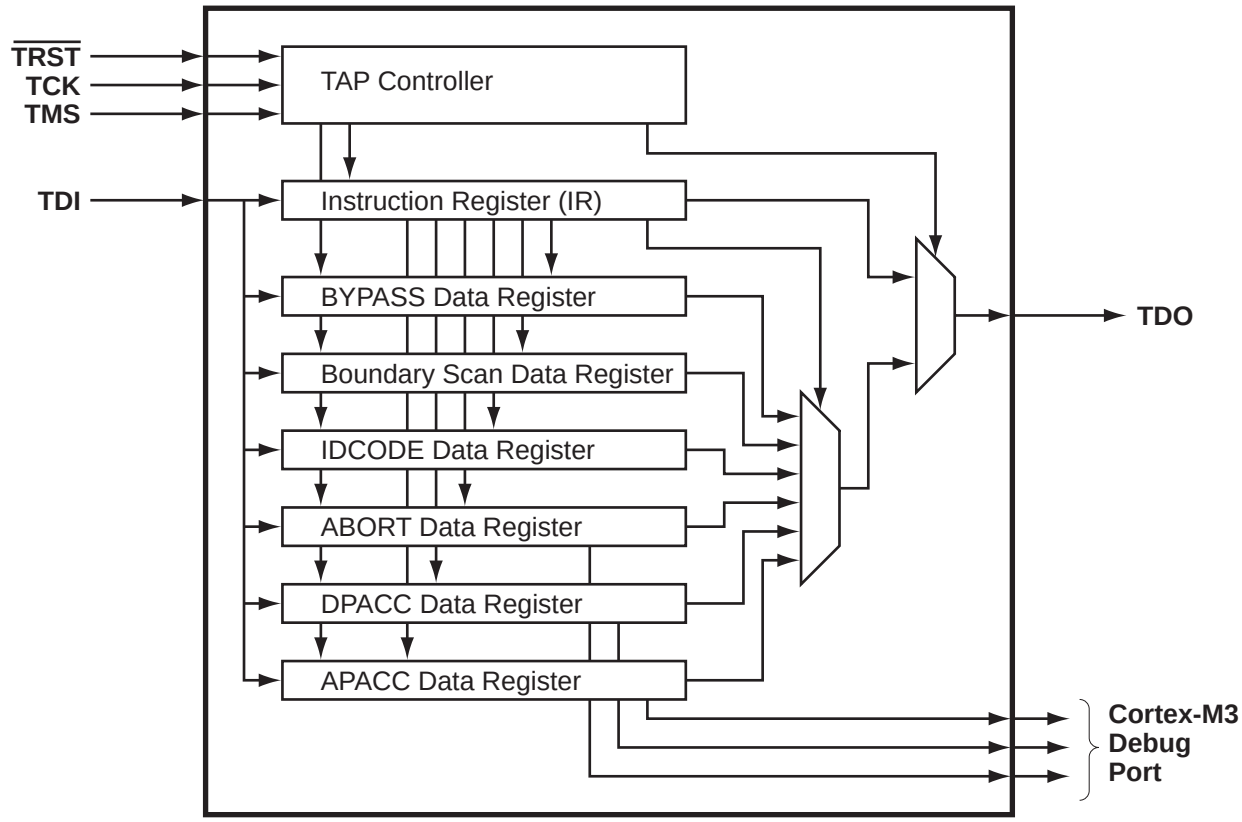
The JTAG module has the following features:

- IEEE 1149.1-1990 compatible Test Access Port (TAP) controller
- Four-bit Instruction Register (IR) chain for storing JTAG instructions
- IEEE standard instructions:
  - BYPASS instruction
  - IDCODE instruction
  - SAMPLE/PRELOAD instruction
  - EXTEST instruction
  - INTEST instruction
- ARM additional instructions:
  - APACC instruction
  - DPACC instruction
  - ABORT instruction
- Integrated ARM Serial Wire Debug (SWD)

See the *ARM® Cortex™-M3 Technical Reference Manual* for more information on the ARM JTAG controller.

## 5.1 Block Diagram

**Figure 5-1. JTAG Module Block Diagram**




## 5.2 Functional Description

A high-level conceptual drawing of the JTAG module is shown in Figure 5-1. 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 $\overline{\text{TRST}}$, TCK and TMS inputs. The current state of the TAP controller depends on the current value of $\overline{\text{TRST}}$ and 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 5-2 on page 48 for a list of implemented instructions).

See “JTAG and Boundary Scan” on page 398 for JTAG timing diagrams.

### 5.2.1 JTAG Interface Pins

The JTAG interface consists of five standard pins: $\overline{\text{TRST}}$, TCK, TMS, TDI, and TDO. These pins and their associated reset state are given in Table 5-1. Detailed information on each pin follows.

**Table 5-1. JTAG Port Pins Reset State**

| Pin Name | Data Direction | Internal Pull-Up | Internal Pull-Down | Drive Strength | Drive Value |
|---|---|---|---|---|---|
| $\overline{\text{TRST}}$ | Input | Enabled | Disabled | N/A | N/A |
| 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 |

#### 5.2.1.1 Test Reset Input ($\overline{\text{TRST}}$)

The $\overline{\text{TRST}}$ pin is an asynchronous active Low input signal for initializing and resetting the JTAG TAP controller and associated JTAG circuitry. When $\overline{\text{TRST}}$ is asserted, the TAP controller resets to the Test-Logic-Reset state and remains there while $\overline{\text{TRST}}$ is asserted. When the TAP controller enters the Test-Logic-Reset state, the JTAG Instruction Register (IR) resets to the default instruction, IDCODE.

By default, the internal pull-up resistor on the $\overline{\text{TRST}}$ pin is enabled after reset. Changes to the pull-up resistor settings on GPIO Port B should ensure that the internal pull-up resistor remains enabled on PB7/$\overline{\text{TRST}}$; otherwise JTAG communication could be lost.

#### 5.2.1.2 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.

#### 5.2.1.3 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 Instruction Register (IR) resets to the default instruction, IDCODE. Therefore, this sequence can be used as a reset mechanism, similar to asserting $\overline{\text{TRST}}$. The JTAG Test Access Port state machine can be seen in its entirety in Figure 5-2 on page 46.

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.

#### 5.2.1.4 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.

#### 5.2.1.5 Test Data Output (TDO)

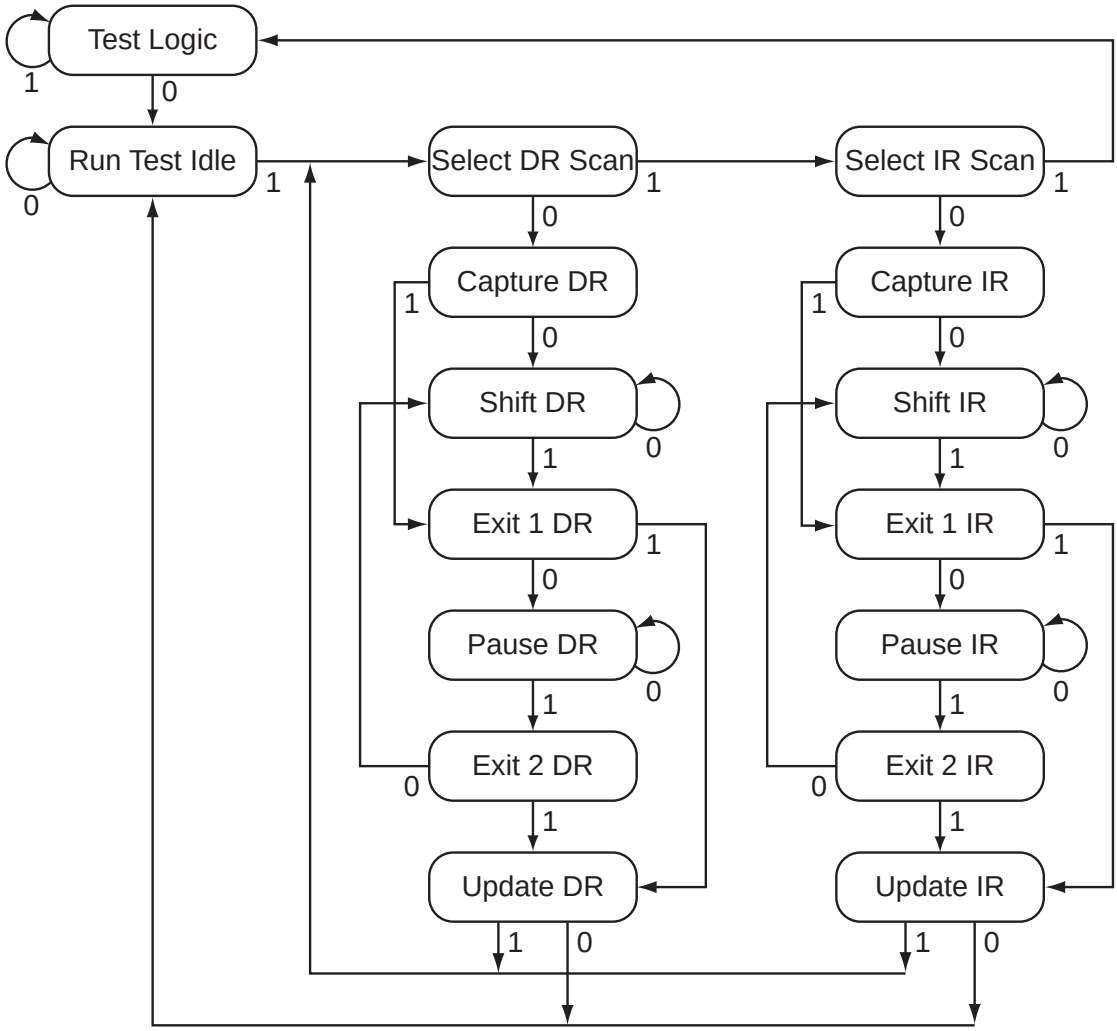
The TDO pin provides an output stream of serial information from the IR chain or the DR chains. The value of TDO depends on the current TAP state, the current instruction, and the data in the chain being accessed. In order to save power when the JTAG port is not being used, the TDO pin is placed in an 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.

### 5.2.2 JTAG TAP Controller

The JTAG TAP controller state machine is shown in Figure 5-2 on page 46. The TAP controller state machine is reset to the Test-Logic-Reset state on the assertion of a Power-On-Reset (POR) or the assertion of $\overline{\text{TRST}}$. 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 5-2. Test Access Port State Machine**




## 5.2.3 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 "Shift Registers" on page 46.

## 5.2.4 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 requires clarification.

### 5.2.4.1 GPIO Functionality

When the controller is reset with either a POR or $\overline{\text{RST}}$, the JTAG port pins default to their JTAG configurations. The default configuration includes enabling the pull-up resistors (setting **GPIOPUR**

to 1 for PB7 and PC[3:0]) and enabling the alternate hardware function (setting **GPIOAFSEL** to 1 for PB7 and PC[3:0]) on the JTAG pins.

It is possible for software to configure these pins as GPIOs after reset by writing 0s to PB7 and PC[3:0] in the **GPIOAFSEL** register. If the user does not require the JTAG port for debugging or board-level testing, this provides five more GPIOs for use in the design.

**Caution – If the JTAG pins are used as GPIOs in a design, PB7 and PC2 cannot have external pull-down resistors connected to both of them at the same time. If both pins are pulled Low during reset, the controller has unpredictable behavior. If this happens, remove one or both of the pull-down resistors, and apply RST or power-cycle the part**

**In addition, 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 does not have enough time to connect and halt the controller before the JTAG pin functionality switches. This locks the debugger out of the part. This can be avoided with a software routine that restores JTAG functionality using an external trigger.**

#### 5.2.4.2 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 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, Capture IR, Exit1 IR, Update IR, Run Test Idle, Select DR, Select IR, Capture IR, Exit1 IR, Update IR, Run Test Idle, Select DR, Select IR, and Test-Logic-Reset states.

Stepping through the JTAG TAP Instruction Register (IR) load sequences of the TAP state machine twice without shifting in a new instruction enables the SWD interface and disables the JTAG interface. For more information on this operation and the SWD interface, see the *ARM® Cortex™-M3 Technical Reference Manual* and the *ARM® CoreSight Technical Reference Manual*.

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 occuring during normal operation of the TAP controller, it should not affect normal performance of the JTAG interface.

## 5.3 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 five JTAG pins (PB7 and PC[3:0]) for their alternate function using the **GPIOAFSEL** register.

## 5.4 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.

### 5.4.1 Instruction Register (IR)

The JTAG TAP Instruction Register (IR) is a four-bit serial scan chain with a parallel load register connected between the JTAG TDI and TDO pins. 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 5-2. A detailed explanation of each instruction, along with its associated Data Register, follows.

**Table 5-2. 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. |

#### 5.4.1.1 EXTEST Instruction

The EXTEST instruction does not have an associated 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.

#### 5.4.1.2 INTEST Instruction

The INTEST instruction does not have an associated 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. It is important to note that although the RST input pin is on the Boundary Scan Data Register chain, it is only observable.

#### 5.4.1.3 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 50 for more information.

#### 5.4.1.4 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 51 for more information.

#### 5.4.1.5 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 51 for more information.

#### 5.4.1.6 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 51 for more information.

#### 5.4.1.7 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, TRST is asserted, or the Test-Logic-Reset state is entered. Please see "IDCODE Data Register" on page 50 for more information.

#### 5.4.1.8 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 50 for more information.

### 5.4.2 Data Registers

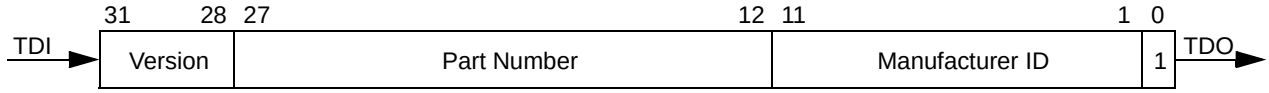
The JTAG module contains six Data Registers. These include: IDCODE, BYPASS, Boundary Scan, APACC, DPACC, and ABORT serial Data Register chains. Each of these Data Registers is discussed in the following sections.

#### 5.4.2.1 IDCODE Data Register

The format for the 32-bit IDCODE Data Register defined by the *IEEE Standard 1149.1* is shown in Figure 5-3. 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 0x1BA00477. This value indicates an ARM Cortex-M3, Version 1 processor. This allows the debuggers to automatically configure themselves to work correctly with the Cortex-M3 during debug.

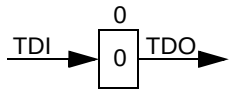
**Figure 5-3. IDCODE Register Format**




#### 5.4.2.2 BYPASS Data Register

The format for the 1-bit BYPASS Data Register defined by the *IEEE Standard 1149.1* is shown in Figure 5-4. 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 5-4. BYPASS Register Format**



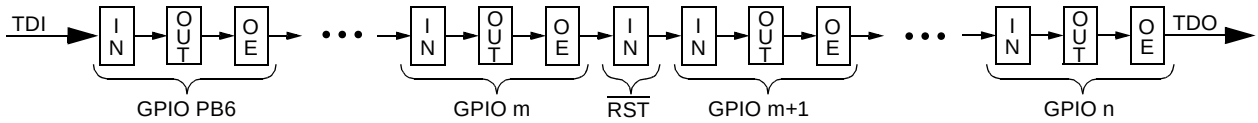

#### 5.4.2.3 Boundary Scan Data Register

The format of the Boundary Scan Data Register is shown in Figure 5-5. Each GPIO pin, in a counter-clockwise direction from 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. In addition to the GPIO pins, the controller reset pin, RST, is included in the chain. Because the reset pin is always an input, only the input signal is included in the Data Register chain.

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 5-5. Boundary Scan Register Format**




For detailed information on the order of the input, output, and output enable bits for each of the GPIO ports, please refer to the Stellaris Family Boundary Scan Description Language (BSDL) files, downloadable from www.luminarymicro.com.

#### 5.4.2.4 APACC Data Register

The format for the 35-bit APACC Data Register defined by ARM is described in the *ARM® Cortex™-M3 Technical Reference Manual*.

#### 5.4.2.5 DPACC Data Register

The format for the 35-bit DPACC Data Register defined by ARM is described in the *ARM® Cortex™-M3 Technical Reference Manual*.

#### 5.4.2.6 ABORT Data Register

The format for the 35-bit ABORT Data Register defined by ARM is described in the *ARM® Cortex™-M3 Technical Reference Manual*.

# 6 System Control

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.

## 6.1 Functional Description

The System Control module provides the following capabilities:

- Device identification, see page 52
- Local control, such as reset (see page 52), power (see page 55) and clock control (see page 55)
- System control (Run, Sleep, and Deep-Sleep modes), see page 57

### 6.1.1 Device Identification

Seven 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**-**DC4** registers starting on page 60.

### 6.1.2 Reset Control

This section discusses aspects of hardware functions during reset as well as system software requirements following the reset sequence.

#### 6.1.2.1 Reset Sources

The controller has six sources of reset:

1. External reset input pin ($\overline{\text{RST}}$) assertion, see page 52.
2. Power-on reset (POR), see page 53.
3. Internal brown-out (BOR) detector, see page 53.
4. Software-initiated reset (with the software reset registers), see page 54.
5. A watchdog timer reset condition violation, see page 54.
6. Internal low drop-out (LDO) regulator output, see page 55.

After a reset, the **Reset Cause (RESC)** register (see page 79) is set with the reset cause. The bits in this register are sticky and maintain their state across multiple reset sequences, except when an external reset is the cause, and then all the other bits in the **RESC** register are cleared.

**Note:** The main oscillator is used for external resets and power-on resets; the internal oscillator is used during the internal process by internal reset and clock verification circuitry.

#### 6.1.2.2 $\overline{\text{RST}}$ Pin Assertion

The external reset pin ($\overline{\text{RST}}$) resets the controller. This resets the core and all the peripherals except the JTAG TAP controller (see “JTAG Interface” on page 42). The external reset sequence is as follows:

1. The external reset pin ($\overline{\text{RST}}$) is asserted and then de-asserted.
2. After $\overline{\text{RST}}$ is de-assserted, the main crystal oscillator must be allowed to settle and there is an internal main oscillator counter that takes from 15-30 ms to account for this. During this time, internal reset to the rest of the controller is held active.

3. The internal reset is released and the controller fetches and loads the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution.

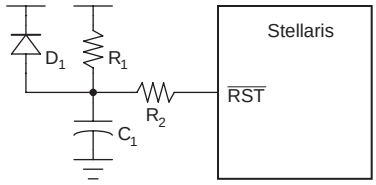
The external reset timing is shown in Figure 20-9 on page 401.

#### 6.1.2.3 Power-On Reset (POR)

The Power-On Reset (POR) circuitry detects a rise in power-supply voltage and generates an on-chip reset pulse. To use the on-chip circuitry, the $\overline{\text{RST}}$ input needs a pull-up resistor (1K to 10K Ω).

The device must be operating within the specified operating parameters at the point when the on-chip power-on reset pulse is complete. The specified operating parameters include supply voltage, frequency, temperature, and so on. If the operating conditions are not met at the point of POR end, the Stellaris controller does not operate correctly. In this case, the reset must be extended using external circuitry. The $\overline{\text{RST}}$ input may be used with the circuit as shown in Figure 6-1.

**Figure 6-1. External Circuitry to Extend Reset**




The $R_1$ and $C_1$ components define the power-on delay. The $R_2$ resistor mitigates any leakage from the $\overline{\text{RST}}$ input. The diode discharges $C_1$ rapidly when the power supply is turned off.

The Power-On Reset sequence is as follows:

1. The controller waits for the later of external reset ($\overline{\text{RST}}$) or internal POR to go inactive.
2. After the resets are inactive, the main crystal oscillator must be allowed to settle and there is an internal main oscillator counter that takes from 15-30 ms to account for this. During this time, internal reset to the rest of the controller is held active.
3. The internal reset is released and the controller fetches and loads 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 controller. The Power-On Reset timing is shown in Figure 20-10 on page 401.

#### 6.1.2.4 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 $V_{DD}$ drops below $V_{BTH}$. The circuit is provided to guard against improper operation of logic and peripherals that operate off $V_{DD}$ and not the LDO voltage. If a brown-out condition is detected, the system may generate a controller interrupt or a system reset. The BOR circuit has a digital filter that protects against noise-related detection. This feature may be optionally enabled.

Brown-out resets are controlled with the **Power-On and Brown-Out Reset Control (PBORCTL)** register (see page 70). The BORIOR bit in the **PBORCTL** register must be set for a brown-out to trigger a reset. The brown-out reset sequence is as follows:

1. When $V_{DD}$ drops below $V_{BTH}$, an internal BOR condition is set.
2. If the BORWT bit in the **PBORCTL** register is set, the BOR condition is resampled sometime later (specified by BORTIM) to determine if the original condition was caused by noise. If the BOR condition is not met the second time, then no action is taken.
3. If the BOR condition exists, an internal reset is asserted.
4. The internal reset is released and the controller fetches and loads the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution.
5. The internal $\overline{\text{BOR}}$ signal is released after 500 µs to prevent another BOR condition from being set before software has a chance to investigate the original cause.

The internal Brown-Out Reset timing is shown in Figure 20-11 on page 401.

#### 6.1.2.5 Software Reset

Each peripheral can be reset by software. There are three registers that control this function (see the **SRCRn** registers, starting on page 72). If the bit position corresponding to a peripheral is set, 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 57). Writing a bit lane with a value of 1 initiates a reset of the corresponding unit. 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 also. 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 in 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 released and the controller fetches and loads 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 20-12 on page 401.

#### 6.1.2.6 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 (see page 181), 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 fetches and loads the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution.

The watchdog reset timing is shown in Figure 20-13 on page 402.

#### 6.1.2.7 Low Drop-Out

A reset can be made when the internal low drop-out (LDO) regulator output goes unregulated. This is initially disabled and may be enabled by software. LDO is controlled with the **LDO Power Control (LDOPCTL)** register (see page 71). The LDO reset sequence is as follows:

1. LDO goes unregulated and the `LDOARST` bit in the **LDOARST** register is set.
2. An internal reset is asserted.
3. The internal reset is released and the controller fetches and loads the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution.

The LDO reset timing is shown in Figure 20-14 on page 402.

### 6.1.3 Power Control

The LDO regulator permits the adjustment of the on-chip output voltage ($V_{OUT}$). The output may be adjusted in 50 mV increments between the range of 2.25 V through 2.75 V. The adjustment is made through the `VADJ` field of the **LDO Power Control (LDOPCTL)** register (see page 71).

### 6.1.4 Clock Control

System control determines the clocking and control of clocks in this part.

#### 6.1.4.1 Fundamental Clock Sources

There are two fundamental clock sources for use in the device:

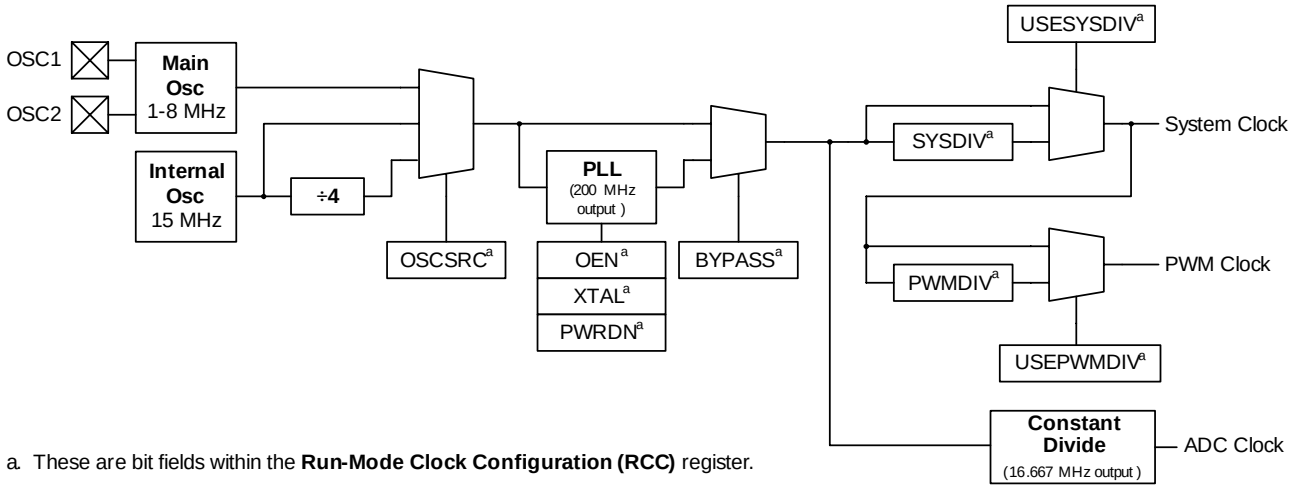
- The main oscillator, driven from either an external crystal or a single-ended source. As a crystal, the main oscillator source is specified to run from 1-8 MHz. However, when the crystal is being used as the PLL source, it must be from 3.579545–8.192 MHz to meet PLL requirements. As a single-ended source, the range is from DC to the specified speed of the device.
- The internal oscillator, which is an on-chip free running clock. The internal oscillator is specified to run at 12 MHz ± 50%. It can be used to clock the system, but the tolerance of frequency range must be met.

The internal system clock may be driven by either of the above two reference sources as well as the internal PLL, provided that the PLL input is connected to a clock source that meets its AC requirements.

Nearly all of the control for the clocks is provided by the **Run-Mode Clock Configuration (RCC)** register (see page 80).

Figure 6-2 shows the logic for the main clock tree. The peripheral blocks are driven by the System Clock signal and can be programmatically enabled/disabled. The ADC clock signal is automatically divided down to 14-18 MHz for proper ADC operation. The PWM clock signal is a synchronous divide by of the system clock to provide the PWM circuit with more range.

**Figure 6-2. Main Clock Tree**




a. These are bit fields within the **Run-Mode Clock Configuration (RCC)** register.

### 6.1.4.2 PLL Frequency Configuration

The user does not have direct control over the PLL frequency, but is required to match the external crystal used to an internal PLL-Crystal table. This table is used to create the best fit for PLL parameters to the crystal chosen. Not all crystals result in the PLL operating at exactly 200 MHz, though the frequency is within ±1%. The result of the lookup is kept in the **XTAL to PLL Translation (PLLCTL)** register (see page 85).

Table 6-4 on page 83 describes the available crystal choices and default programming of the **PLLCTL** register. The crystal number is written into the `XTAL` field of the **Run-Mode Clock Configuration (RCC)** register (see page 80). Any time the `XTAL` field changes, a read of the internal table is performed to get the correct value. Table 6-4 on page 83 describes the available crystal choices and default programming values.

### 6.1.4.3 PLL Modes

The PLL has 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** register fields as shown in Table 6-4 on page 83.

### 6.1.4.4 PLL Operation

If the PLL configuration is changed, the PLL output is not stable for a period of time (PLL $T_{READY}$=0.5 ms) and during this time, the PLL is not usable as a clock reference.

The PLL is changed by one of the following:

- Change to the `XTAL` value in the **RCC** register (see page 80)—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 a 8.192-MHz external oscillator clock). 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** register is switched to use the PLL.

#### 6.1.4.5 Clock Verification Timers

There are three identical clock verification circuits that can be enabled though software. The circuit checks the faster clock by a slower clock using timers:

- The main oscillator checks the PLL.
- The main oscillator checks the internal oscillator.
- The internal oscillator divided by 64 checks the main oscillator.

If the verification timer function is enabled and a failure is detected, the main clock tree is immediately switched to a working clock and an interrupt is generated to the controller. Software can then determine the course of action to take. The actual failure indication and clock switching does not clear without a write to the **CLKVCLR** register, an external reset, or a POR reset. The clock verification timers are controlled by the `PLLVER`, `IOSCVER`, and `MOSCVER` bits in the **RCC** register (see page 80).

### 6.1.5 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. The **DC1**, **DC2** and **DC4** registers act as a write mask for the **RCGCn**, **SCGCn**, and **DCGCn** registers.

In Run mode, the controller is actively executing code. In Sleep mode, the clocking of the device is unchanged but the controller no longer executes code (and is no longer clocked). In Deep-Sleep mode, the clocking of the device may change (depending on the Run mode clock configuration) and the controller no longer executes code (and is no longer clocked). An interrupt returns the device to Run mode from one of the sleep modes; the sleep modes are entered on request from the code. Each mode is described in more detail in this section.

#### 6.1.5.1 Run Mode

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.

#### 6.1.5.2 Sleep Mode

In Sleep mode, the Cortex-M3 processor core and the memory subsystem are not clocked. Peripherals are clocked that are enabled in the **SCGCn** register when Auto Clock Gating is enabled (see **RCC** register on page 80) 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.

#### 6.1.5.3 Deep-Sleep Mode

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 **RCC** register) or the **RCGCn** register when the 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 (see page 91). 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 powers the PLL down and overrides the `SYSDIV` field of the active **RCC** register to be /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 were stopped during the Deep-Sleep duration.

## 6.2 Initialization and Configuration

The PLL is configured using direct register writes to the **Run-Mode Clock Configuration (RCC)** register. 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 (using the main oscillator or internal oscillator) 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 and OE bits in **RCC**. Setting the XTAL field automatically pulls valid PLL configuration data for the appropriate crystal, and clearing the PWRDN and OE bits powers and enables the PLL and its output.
3. Select the desired system divider (SYSDIV) 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. If the PLL doesn’t lock, the configuration is invalid.
5. Enable use of the PLL by clearing the BYPASS bit in **RCC**.

**Important:** If the BYPASS bit is cleared before the PLL locks, it is possible to render the device unusable.

## 6.3 Register Map

Table 6-1 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 0x400FE000.

**Table 6-1. System Control Register Map (Sheet 1 of 2)**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| **Device Identification and Capabilities** | | | | | |
| 0x000 | DID0 | - | RO | Device identification 0 | 60 |
| 0x004 | DID1 | - | RO | Device identification 1 | 61 |
| 0x008 | DC0 | 0x000F0007 | RO | Device capabilities 0 | 63 |
| 0x010 | DC1 | 0x00000007 | RO | Device capabilities 1 | 64 |
| 0x014 | DC2 | 0x01071013 | RO | Device capabilities 2 | 66 |
| 0x018 | DC3 | 0x3F0F01CF | RO | Device Capabilities 3 | 67 |
| 0x01C | DC4 | 0x0000001F | RO | Device Capabilities 4 | 69 |
| **Local Control** | | | | | |
| 0x030 | PBORCTL | 0x00007FFD | R/W | Power-On and Brown-Out Reset Control | 70 |
| 0x034 | LDOPCTL | 0x00000000 | R/W | LDO Power Control | 71 |
| 0x040 | SRCR0 | 0x00000000 | R/W | Software Reset Control 0 | 72 |

**Table 6-1. System Control Register Map (Sheet 2 of 2)**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x044 | SRCR1 | 0x00000000 | R/W | Software Reset Control 1 | 73 |
| 0x048 | SRCR2 | 0x00000000 | R/W | Software Reset Control 2 | 74 |
| 0x050 | RIS | 0x00000000 | RO | Raw Interrupt Status | 75 |
| 0x054 | IMC | 0x00000000 | R/W | Interrupt Mask Control | 76 |
| 0x058 | MISC | 0x00000000 | R/W1C | Masked Interrupt Status and Clear | 78 |
| 0x05C | RESC | - | R/W | Reset Cause | 79 |
| 0x060 | RCC | 0x078E3AC0 | R/W | Run-Mode Clock Configuration | 80 |
| 0x064 | PLLCFG | - | RO | XTAL to PLL translation | 85 |
| **System Control** | | | | | |
| 0x100 | RCGC0 | 0x00000001 | R/W | Run-Mode Clock Gating Control 0 | 86 |
| 0x104 | RCGC1 | 0x00000000 | R/W | Run-Mode Clock Gating Control 1 | 88 |
| 0x108 | RCGC2 | 0x00000000 | R/W | Run-Mode Clock Gating Control 2 | 90 |
| 0x110 | SCGC0 | 0x00000001 | R/W | Sleep-Mode Clock Gating Control 0 | 86 |
| 0x114 | SCGC1 | 0x00000000 | R/W | Sleep-Mode Clock Gating Control 1 | 88 |
| 0x118 | SCGC2 | 0x00000000 | R/W | Sleep-Mode Clock Gating Control 2 | 90 |
| 0x120 | DCGC0 | 0x00000001 | R/W | Deep-Sleep-Mode Clock Gating Control 0 | 86 |
| 0x124 | DCGC1 | 0x00000000 | R/W | Deep-Sleep-Mode Clock Gating Control 1 | 88 |
| 0x128 | DCGC2 | 0x00000000 | R/W | Deep-Sleep-Mode Clock Gating Control 2 | 90 |
| 0X144 | DSLPCLKCFG | 0x07800000 | R/W | Deep-Sleep Clock Configuration | 91 |
| 0x150 | CLKVCLR | 0x00000000 | R/W | Clock verification clear | 92 |
| 0x160 | LDOARST | 0x00000000 | R/W | Allow unregulated LDO to reset the part | 93 |

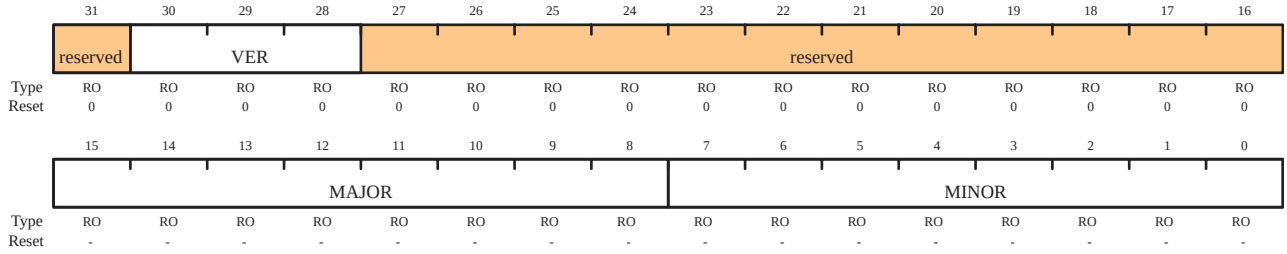
## 6.4 Register Descriptions

The remainder of this section lists and describes the System Control registers, in numerical order by address offset.

### Register 1: Device Identification 0 (DID0), offset 0x000

This register identifies the version of the device.

Device Identification 0 (DID0)
Offset 0x000



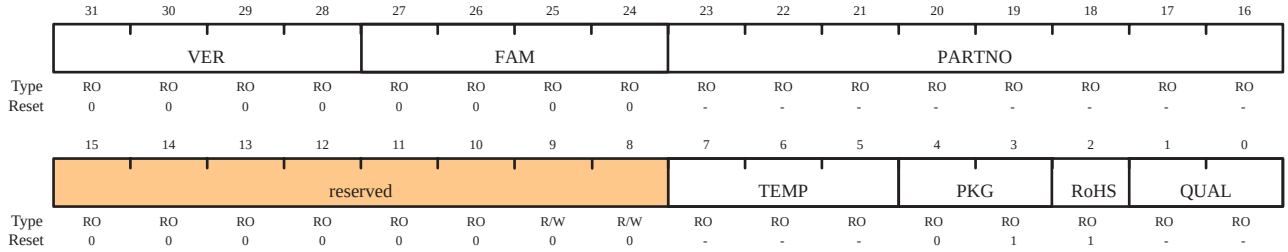

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 30:28 | VER | RO | 0 | This field defines the version of the **DID0** register format:<br>0=Register version for the Stellaris microcontrollers |
| 27:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:8 | MAJOR | RO | - | This field specifies the major revision number of the device. 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>0: Revision A (initial device)<br>1: Revision B (first revision)<br>and so on. |
| 7:0 | MINOR | RO | - | This field specifies the minor revision number of the device. This field is numeric and is encoded as follows:<br>0: No changes. Major revision was most recent update.<br>1: One interconnect change made since last major revision update.<br>2: Two interconnect changes made since last major revision update.<br>and so on. |

### Register 2: Device Identification 1 (DID1), offset 0x004

This register identifies the device family, part number, temperature range, and package type.

**Note:** The bit diagram indicates some values are device-specific. The table below indicates values for your part.

Device Identification 1 (DID1)
Offset 0x004




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:28 | VER | RO | 0x0 | This field defines the version of the **DID1** register format:<br>0=Register version for the Stellaris microcontrollers |
| 27:24 | FAM | RO | 0x0 | Family<br>This field provides the family identification of the device within the Luminary Micro product portfolio.<br>The 0x0 value indicates the Stellaris family of microcontrollers. |
| 23:16 | PARTNO | RO | 0x14 | Part Number<br>This field provides the part number of the device within the family.<br>The 0x14 value indicates the LM3S316 microcontroller. |
| 15:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:5 | TEMP | RO | see table | Temperature Range<br>This field specifies the temperature rating of the device. This field is encoded as follows:<br>TEMP: Description<br>000: Commercial temperature range (0°C to 70°C)<br>001: Industrial temperature range (-40°C to 85°C)<br>010-111: Reserved |
| 4:3 | PKG | RO | 0x1 | This field specifies the package type. A value of 1 indicates a 48-pin LQFP package. |

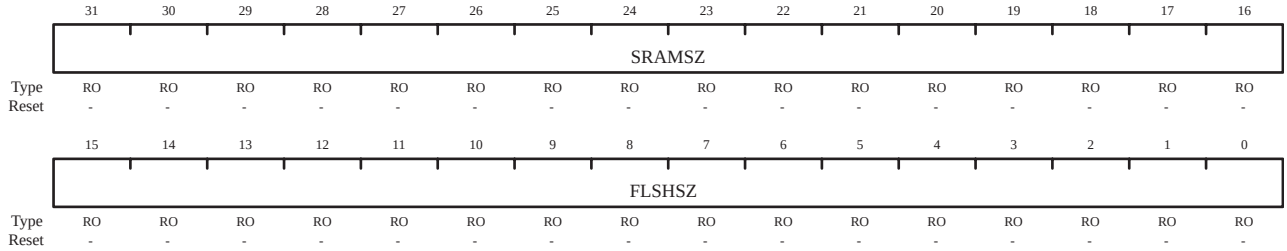
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | RoHS | RO | 1 | RoHS-Compliance<br>A 1 in this bit specifies the device is RoHS-compliant. |
| 1:0 | QUAL | RO | see table | This field specifies the qualification status of the device. This field is encoded as follows:<br>QUAL — Description<br>00 — Engineering Sample (unqualified)<br>01 — Pilot Production (unqualified)<br>10 — Fully Qualified<br>11 — Reserved |

### Register 3: Device Capabilities 0 (DC0), offset 0x008

This register is predefined by the part and can be used to verify features.

**Note:** The bit diagram indicates the values are device-specific. The table below indicates values for your specific part.

Device Capabilities Register 0 (DC0)
Offset 0x004

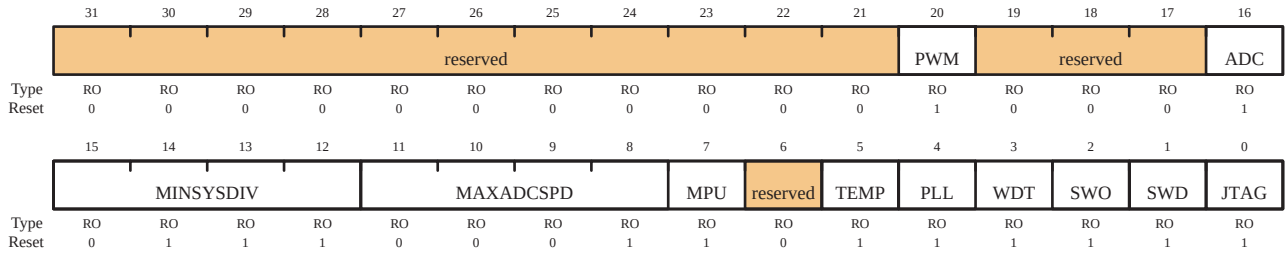



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | SRAMSZ | RO | 0x000F | Indicates the size of the on-chip SRAM. A value of 0x000F indicates 4 KB of SRAM. |
| 15:0 | FLSHSZ | RO | 0x0007 | Indicates the size of the on-chip flash memory. A value of 0x0007 indicates 16 KB of Flash. |

### Register 4: Device Capabilities 1 (DC1), offset 0x010

This register is predefined by the part and can be used to verify features.

Device Capabilities 1 (DC1)
Offset 0x010




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:21 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 20 | PWM[a] | RO | 1 | A 1 in this bit indicates the presence of the PWM module. |
| 19:17 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 16 | ADC[a] | RO | 1 | A 1 in this bit indicates the presence of the ADC module. |
| 15:12 | MINSYSDIV | RO | 0x07 | The reset value is hardware-dependent. A value of 0x7 specifies a 25-MHz clock with a PLL divider of 8. See the **RCC** register (page 80) for how to change the system clock divisor using the `SYSDIV` bit. |
| 11:8 | MAXADCSPD[a] | RO | 0x1 | This field indicates the maximum rate at which the ADC samples data. A value of 0x1 indicates 250K samples per second. |
| 7 | MPU | RO | 1 | This bit indicates whether the Memory Protection Unit (MPU) in the Cortex-M3 is available. A 0 in this bit indicates the MPU is not available; a 1 indicates the MPU is available.<br>See the *ARM® Cortex™-M3 Technical Reference Manual* for details on the MPU. |
| 6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 5 | TEMP | RO | 1 | This bit specifies the presence of an internal temperature sensor. |
| 4 | PLL | RO | 1 | A 1 in this bit indicates the presence of an implemented PLL in the device. |
| 3 | WDT[a] | RO | 1 | A 1 in this bit indicates a watchdog timer on the device. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | SWO[a] | RO | 1 | A 1 in this bit indicates the presence of the ARM Serial Wire Output (SWO) trace port capabilities. |
| 1 | SWD[a] | RO | 1 | A 1 in this bit indicates the presence of the ARM Serial Wire Debug (SWD) capabilities. |
| 0 | JTAG[a] | RO | 1 | A 1 in this bit indicates the presence of a JTAG port. |

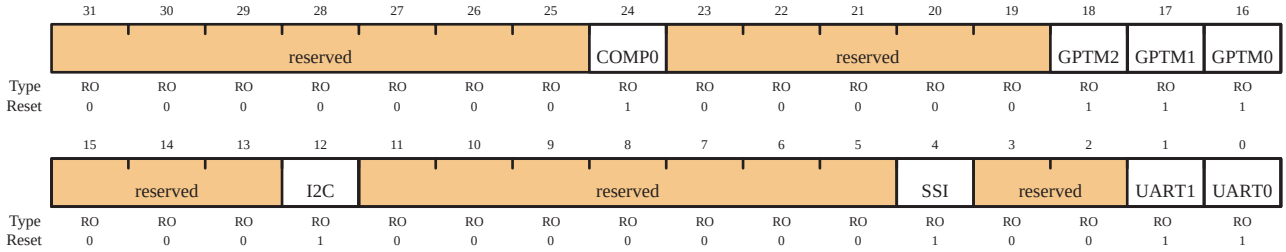
a. These bits mask the **Run-Mode Clock Gating Control 0 (RCGC0)** register (see page 113), **Sleep-Mode Clock Gating Control 0 (SCGC0)** register (see page 113), and **Deep-Sleep-Mode Clock Gating Control 0 (DCGC0)** register (see page 113). Bits that are not noted are passed as 0. ADCSP is clipped to the maximum value specified in DC1.

### Register 5: Device Capabilities 2 (DC2), offset 0x014

**Note:** The bit diagram indicates all possible features. The table below indicates values for your specific part.

This register is predefined by the part and can be used to verify features.

Device Capabilities 2 (DC2)
Offset 0x014




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:25 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 24 | COMP0 | RO | 1 | A 1 in this bit indicates the presence of analog comparator 0. |
| 23:19 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 18 | GPTM2 | RO | 1 | A 1 in this bit indicates the presence of General-Purpose Timer module 2. |
| 17 | GPTM1 | RO | 1 | A 1 in this bit indicates the presence of General-Purpose Timer module 1. |
| 16 | GPTM0 | RO | 1 | A 1 in this bit indicates the presence of General-Purpose Timer module 0. |
| 15:13 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 12 | I2C | RO | 1 | A 1 in this bit indicates the presence of the $I^2C$ module. |
| 11:5 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 4 | SSI | RO | 1 | A 1 in this bit indicates the presence of the SSI module. |
| 3:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | UART1 | RO | 1 | A 1 in this bit indicates the presence of the UART1 module. |
| 0 | UART0 | RO | 1 | A 1 in this bit indicates the presence of the UART0 module. |

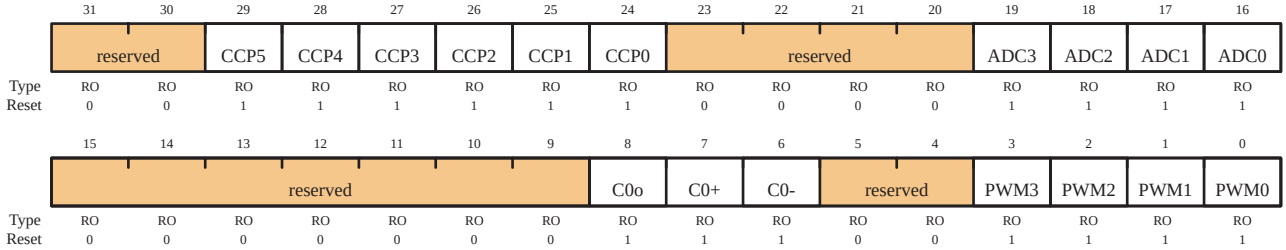
### Register 6: Device Capabilities 3 (DC3), offset 0x018

**Note:** The bit diagram indicates all possible features. The table below indicates values for your specific part.

This register is predefined by the part and can be used to verify features.

Device Capabilities 3 (DC3)

Offset 0x018



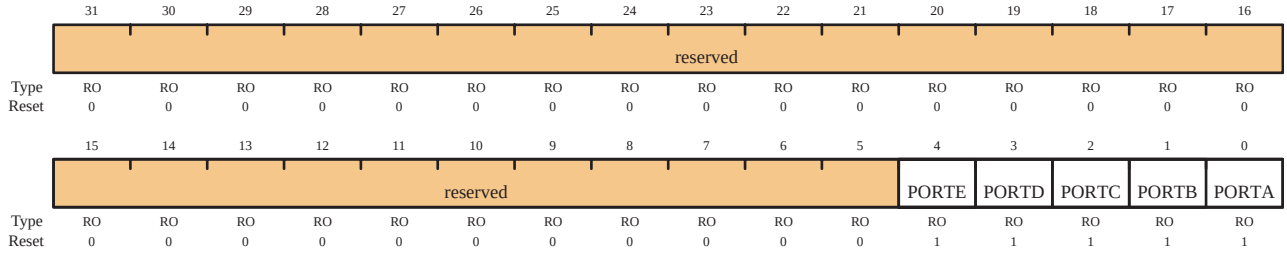

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 29 | CCP5 | RO | 1 | A 1 in this bit indicates the presence of the Capture/ Compare/PWM pin 5. |
| 28 | CCP4 | RO | 1 | A 1 in this bit indicates the presence of the Capture/ Compare/PWM pin 4. |
| 27 | CCP3 | RO | 1 | A 1 in this bit indicates the presence of the Capture/ Compare/PWM pin 3. |
| 26 | CCP2 | RO | 1 | A 1 in this bit indicates the presence of the Capture/ Compare/PWM pin 2. |
| 25 | CCP1 | RO | 1 | A 1 in this bit indicates the presence of the Capture/ Compare/PWM pin 1. |
| 24 | CCP0 | RO | 1 | A 1 in this bit indicates the presence of the Capture/ Compare/PWM pin 0. |
| 23:20 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 19 | ADC3 | RO | 1 | A 1 in this bit indicates the presence of the ADC3 pin. |
| 18 | ADC2 | RO | 1 | A 1 in this bit indicates the presence of the ADC2 pin. |
| 17 | ADC1 | RO | 1 | A 1 in this bit indicates the presence of the ADC1 pin. |
| 16 | ADC0 | RO | 1 | A 1 in this bit indicates the presence of the ADC0 pin. |
| 15:9 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 8 | C0o | RO | 1 | A 1 in this bit indicates the presence of the C0o pin. |
| 7 | C0+ | RO | 1 | A 1 in this bit indicates the presence of the C0+ pin. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | C0- | RO | 1 | A 1 in this bit indicates the presence of the C0- pin. |
| 5:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | PWM3 | RO | 1 | A 1 in this bit indicates the presence of the PWM3 pin. |
| 2 | PWM2 | RO | 1 | A 1 in this bit indicates the presence of the PWM2 pin. |
| 1 | PWM1 | RO | 1 | A 1 in this bit indicates the presence of the PWM1 pin. |
| 0 | PWM0 | RO | 1 | A 1 in this bit indicates the presence of the PWM0 pin. |

### Register 7: Device Capabilities 4 (DC4), offset 0x01C

This register is predefined by the part and can be used to verify features.

Device Capabilities 4 (DC4)
Offset 0x01C

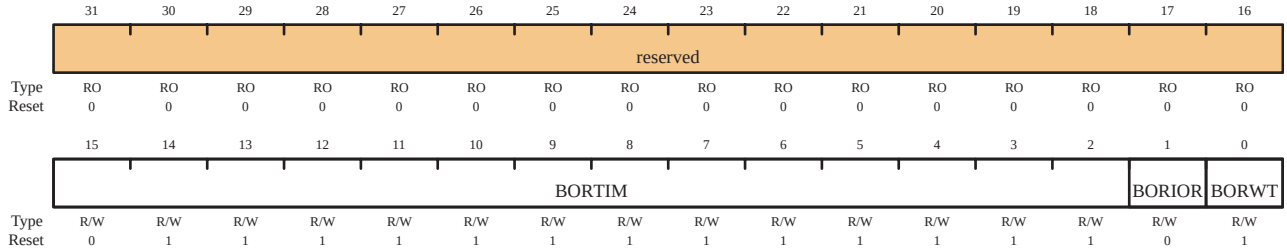



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 4 | PORTE | RO | 1 | A 1 in this bit indicates the presence of GPIO Port E. |
| 3 | PORTD | RO | 1 | A 1 in this bit indicates the presence of GPIO Port D. |
| 2 | PORTC | RO | 1 | A 1 in this bit indicates the presence of GPIO Port C. |
| 1 | PORTB | RO | 1 | A 1 in this bit indicates the presence of GPIO Port B. |
| 0 | PORTA | RO | 1 | A 1 in this bit indicates the presence of GPIO Port A. |

### Register 8: Power-On and Brown-Out Reset Control (PBORCTL), offset 0x030

This register is responsible for controlling reset conditions after initial power-on reset.

Power-On and Brown-Out Reset Control (PBORCTL)
Offset 0x030

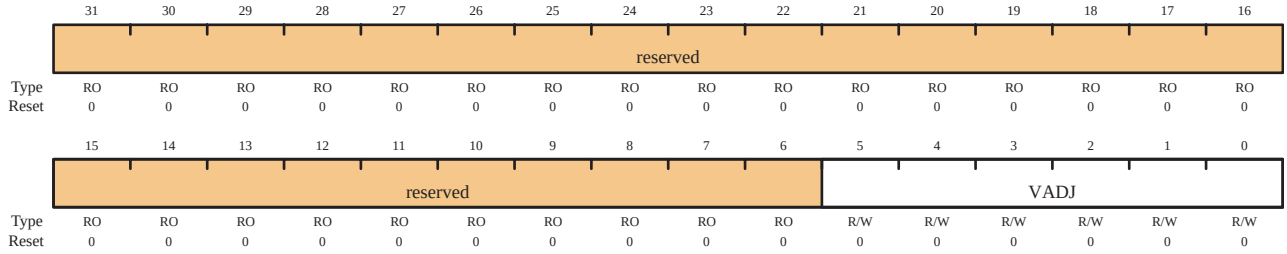



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:2 | BORTIM | R/W | 0x1FFF | This field specifies the number of internal oscillator clocks delayed before the BOR output is resampled if the BORWT bit is set.<br><br>The width of this field is derived by the $t_{BOR}$ width of 500 µs and the internal oscillator (IOSC) frequency of 15 MHz ± 50%. At +50%, the counter value has to exceed 10,000. |
| 1 | BORIOR | R/W | 0 | BOR Interrupt or Reset<br><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 | BORWT | R/W | 1 | BOR Wait and Check for Noise<br><br>This bit specifies the response to a brown-out signal assertion. If BORWT is set to 1, the controller waits BORTIM IOSC periods before resampling the BOR output, and if asserted, it signals a BOR condition interrupt or reset. If the BOR resample is deasserted, the cause of the initial assertion was likely noise and the interrupt or reset is suppressed. If BORWT is 0, BOR assertions do not resample the output and any condition is reported immediately if enabled. |

### Register 9: LDO Power Control (LDOPCTL), offset 0x034

The VADJ field in this register adjusts the on-chip output voltage ($V_{OUT}$).

LDO Power Control (LDOPCTL)
Offset 0x034




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 5:0 | VADJ | R/W | 0x0 | This field sets the on-chip output voltage. The programming values for the VADJ field are provided in Table 6-2. |

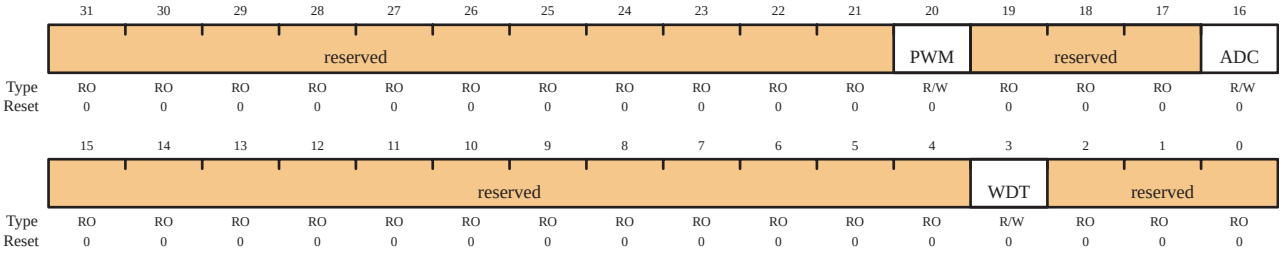
**Table 6-2. VADJ to $V_{OUT}$**

| VADJ Value | $V_{OUT}$ (V) | VADJ Value | $V_{OUT}$ (V) | VADJ Value | $V_{OUT}$ (V) |
|---|---|---|---|---|---|
| 0x1B | 2.75 | 0x1F | 2.55 | 0x03 | 2.35 |
| 0x1C | 2.70 | 0x00 | 2.50 | 0x04 | 2.30 |
| 0x1D | 2.65 | 0x01 | 2.45 | 0x05 | 2.25 |
| 0x1E | 2.60 | 0x02 | 2.40 | 0x06-0x3F | Reserved |

### Register 10: Software Reset Control 0 (SRCR0), offset 0x040

Writes to this register are masked by the bits in the **Device Capabilities 1 (DC1)** register (see page 64).

Software Reset Control 0 (SRCR0)
Offset 0x040

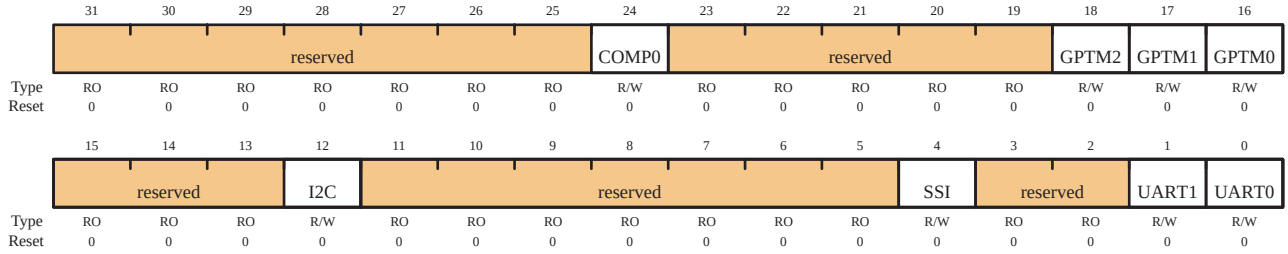



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:21 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 20 | PWM | R/W | 0 | Reset control for the PWM units. |
| 19:17 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 16 | ADC | R/W | 0 | Reset control for the ADC units. |
| 15:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | WDT | R/W | 0 | Reset control for the Watchdog unit. |
| 2:0 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |

### Register 11: Software Reset Control 1 (SRCR1), offset 0x044

Writes to this register are masked by the bits in the **Device Capabilities 2 (DC2)** register (see page 66).

Software Reset Control 1 (SRCR1)
Offset 0x044

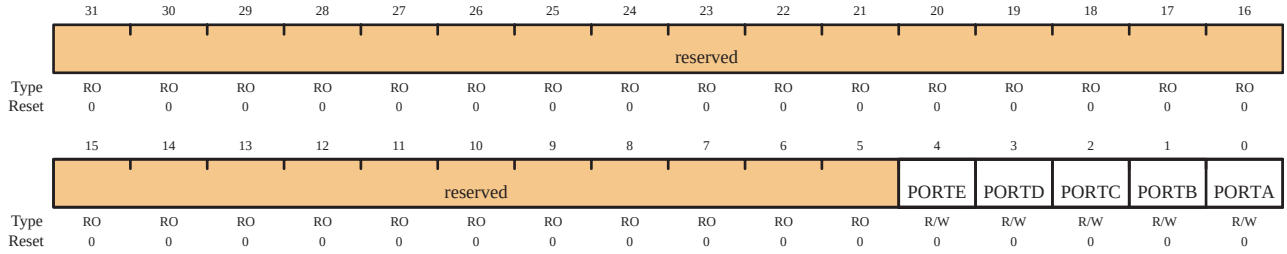



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:25 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 24 | COMP0 | R/W | 0 | Reset control for analog comparator 0. |
| 23:19 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 18 | GPTM2 | R/W | 0 | Reset control for General-Purpose Timer module 2. |
| 17 | GPTM1 | R/W | 0 | Reset control for General-Purpose Timer module 1. |
| 16 | GPTM0 | R/W | 0 | Reset control for General-Purpose Timer module 0. |
| 15:13 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 12 | I2C | R/W | 0 | Reset control for the $I^2C$ units. |
| 11:5 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 4 | SSI | R/W | 0 | Reset control for the SSI units. |
| 3:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | UART1 | R/W | 0 | Reset control for the UART1 module. |
| 0 | UART0 | R/W | 0 | Reset control for the UART0 module. |

### Register 12: Software Reset Control 2 (SRCR2), offset 0x048

Writes to this register are masked by the bits in the **Device Capabilities 4 (DC4)** register (see page 69).

Software Reset Control (SRCR2)
Offset 0x048

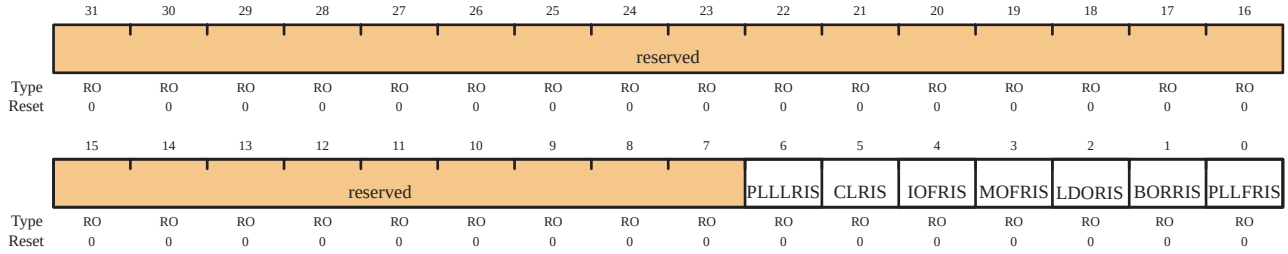



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 4 | PORTE | R/W | 0 | Reset control for GPIO Port E. |
| 3 | PORTD | R/W | 0 | Reset control for GPIO Port D. |
| 2 | PORTC | R/W | 0 | Reset control for GPIO Port C. |
| 1 | PORTB | R/W | 0 | Reset control for GPIO Port B. |
| 0 | PORTA | R/W | 0 | Reset control for GPIO Port A. |

### Register 13: Raw Interrupt Status (RIS), offset 0x050

Central location for system control raw interrupts. These are set and cleared by hardware.

Raw Interrupt Status (RIS)
Offset 0x050

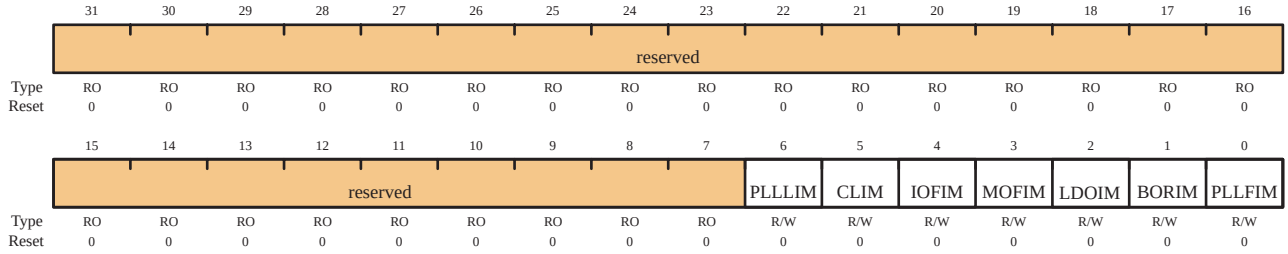



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 6 | PLLLRIS | RO | 0 | PLL Lock Raw Interrupt Status<br>This bit is set when the PLL $T_{READY}$ Timer asserts. |
| 5 | CLRIS | RO | 0 | Current Limit Raw Interrupt Status<br>This bit is set if the LDO's CLE output asserts. |
| 4 | IOFRIS | RO | 0 | Internal Oscillator Fault Raw Interrupt Status<br>This bit is set if an internal oscillator fault is detected. |
| 3 | MOFRIS | RO | 0 | Main Oscillator Fault Raw Interrupt Status<br>This bit is set if a main oscillator fault is detected. |
| 2 | LDORIS | RO | 0 | LDO Power Unregulated Raw Interrupt Status<br>This bit is set if a LDO voltage is unregulated. |
| 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 was detected. 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 | PLLFRIS | RO | 0 | PLL Fault Raw Interrupt Status<br>This bit is set if a PLL fault is detected (stops oscillating). |

### Register 14: Interrupt Mask Control (IMC), offset 0x054

Central location for system control interrupt masks.

Interrupt Mask Control (IMC)
Offset 0x054



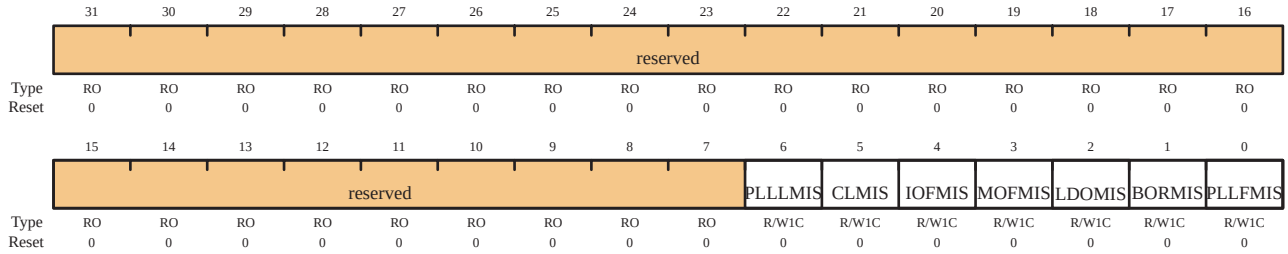

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 6 | PLLLIM | R/W | 0 | PLL Lock Interrupt Mask<br>This bit specifies whether a current limit detection is promoted to a controller interrupt. If set, an interrupt is generated if PLLLRIS in **RIS** is set; otherwise, an interrupt is not generated. |
| 5 | CLIM | R/W | 0 | Current Limit Interrupt Mask<br>This bit specifies whether a current limit detection is promoted to a controller interrupt. If set, an interrupt is generated if CLRIS is set; otherwise, an interrupt is not generated. |
| 4 | IOFIM | R/W | 0 | Internal Oscillator Fault Interrupt Mask<br>This bit specifies whether an internal oscillator fault detection is promoted to a controller interrupt. If set, an interrupt is generated if IOFRIS is set; otherwise, an interrupt is not generated. |
| 3 | MOFIM | R/W | 0 | Main Oscillator Fault Interrupt Mask<br>This bit specifies whether a main oscillator fault detection is promoted to a controller interrupt. If set, an interrupt is generated if MOFRIS is set; otherwise, an interrupt is not generated. |
| 2 | LDOIM | R/W | 0 | LDO Power Unregulated Interrupt Mask<br>This bit specifies whether an LDO unregulated power situation is promoted to a controller interrupt. If set, an interrupt is generated if LDORIS is set; otherwise, an interrupt is not generated. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 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 | PLLFIM | R/W | 0 | PLL Fault Interrupt Mask<br>This bit specifies whether a PLL fault detection is promoted to a controller interrupt. If set, an interrupt is generated if PLLFRIS is set; otherwise, an interrupt is not generated. |

### Register 15: Masked Interrupt Status and Clear (MISC), offset 0x058

Central location for system control result of RIS AND IMC to generate an interrupt to the controller. All of the bits are R/W1C and this action also clears the corresponding raw interrupt bit in the **RIS** register (see page 75).

Masked Interrupt Status and Clear (MISC)
Offset 0x058

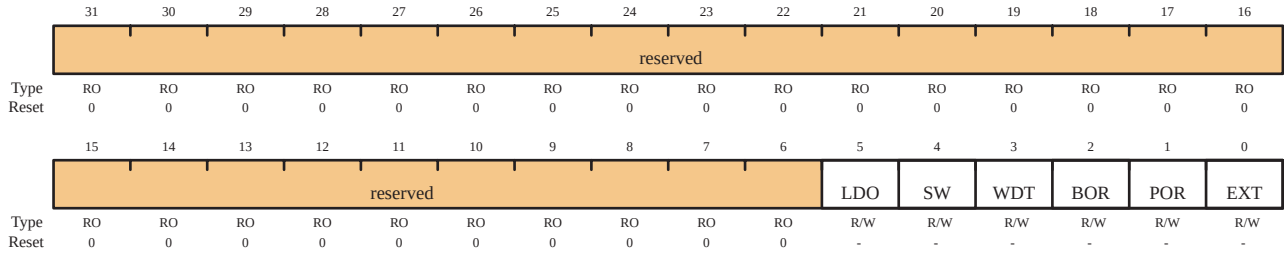



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 6 | PLLLMIS | R/W1C | 0 | PLL Lock Masked Interrupt Status<br>This bit is set when the PLL $T_{READY}$ timer asserts. The interrupt is cleared by writing a 1 to this bit. |
| 5 | CLMIS | R/W1C | 0 | Current Limit Masked Interrupt Status<br>This bit is set if the LDO's CLE output asserts. The interrupt is cleared by writing a 1 to this bit. |
| 4 | IOFMIS | R/W1C | 0 | Internal Oscillator Fault Masked Interrupt Status<br>This bit is set if an internal oscillator fault is detected. The interrupt is cleared by writing a 1 to this bit. |
| 3 | MOFMIS | R/W1C | 0 | Main Oscillator Fault Masked Interrupt Status<br>This bit is set if a main oscillator fault is detected. The interrupt is cleared by writing a 1 to this bit. |
| 2 | LDOMIS | R/W1C | 0 | LDO Power Unregulated Masked Interrupt Status<br>This bit is set if LDO power is unregulated. The interrupt is cleared by writing a 1 to this bit. |
| 1 | BORMIS | R/W1C | 0 | Brown-Out Reset Masked Interrupt Status<br>This bit is the masked interrupt status for any brown-out conditions. If set, a brown-out condition was detected. An interrupt is reported if the `BORIM` bit in the **IMC** register is set and the `BORIOR` bit in the **PBORCTL** register is cleared. The interrupt is cleared by writing a 1 to this bit. |
| 0 | PLLFMIS | R/W1C | 0 | PLL Fault Masked Interrupt Status<br>This bit is set if a PLL fault is detected (stops oscillating). The interrupt is cleared by writing a 1 to this bit. |

### Register 16: Reset Cause (RESC), offset 0x05C

This field specifies the cause of the reset event to software. The reset value is determined by the cause of the reset. When an external reset is the cause (EXT is set), all other reset bits are cleared. However, if the reset is due to any other cause, the remaining bits are sticky, allowing software to see all causes.

Reset Cause (RESC)
Offset 0x05C

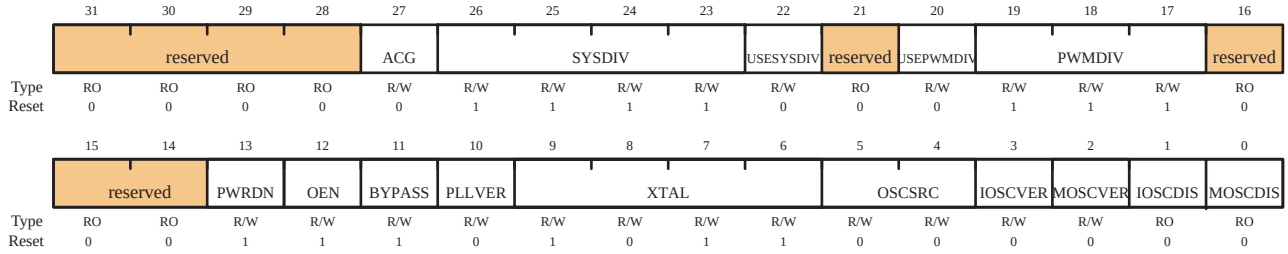



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 5 | LDO | R/W | - | When set to 1, LDO power OK lost is the cause of the reset event. |
| 4 | SW | R/W | - | When set to 1, a software reset is the cause of the reset event. |
| 3 | WDT | R/W | - | When set to 1, a watchdog reset is the cause of the reset event. |
| 2 | BOR | R/W | - | When set to 1, a brown-out reset is the cause of the reset event. |
| 1 | POR | R/W | - | When set to 1, a power-on reset is the cause of the reset event. |
| 0 | EXT | R/W | - | When set to 1, an external reset ($\overline{\text{RST}}$ assertion) is the cause of the reset event. |

### Register 17: Run-Mode Clock Configuration (RCC), offset 0x060

This register is defined to provide source control and frequency speed.

Run-Mode Clock Configuration (RCC)
Offset 0x060




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:28 | Reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 27 | ACG | R/W | 0 | Auto Clock Gating<br><br>This bit specifies whether the system uses the **Sleep-Mode Clock Gating Control (SCGCn)** registers (see page 86) and **Deep-Sleep-Mode Clock Gating Control (DCGCn)** registers (see page 86) 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 (see page 86) are used when the controller enters a sleep mode.<br><br>The **RCGCn** registers are always used to control the clocks in Run mode.<br><br>This allows peripherals to consume less power when the controller is in a sleep mode and the peripheral is unused. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 26:23 | SYSDIV | R/W | 0xF | System Clock Divisor<br>Specifies which divisor is used to generate the system clock from the PLL output (200 MHz).<br>When reading the **Run-Mode Clock Configuration (RCC)** register (see page 80), the `SYSDIV` value is `MINSYSDIV` if a lower divider was requested and the PLL is being used. This lower value is allowed to divide a non-PLL source. |
| 22 | USESYSDIV | R/W | 0 | 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. |
| 21 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 20 | USEPWMDIV | R/W | 0 | Use the PWM clock divider as the source for the PWM clock. |

| Binary Value | Divisor (BYPASS=1) | Frequency (BYPASS=0) |
|---|---|---|
| 0000 | reserved | reserved |
| 0001 | /2 | reserved |
| 0010 | /3 | reserved |
| 0011 | /4 | reserved |
| 0100 | /5 | reserved |
| 0101 | /6 | reserved |
| 0110 | /7 | reserved |
| 0111 | /8 | 25 MHz |
| 1000 | /9 | 22.22 MHz |
| 1001 | /10 | 20 MHz |
| 1010 | /11 | 18.18 MHz |
| 1011 | /12 | 16.67 MHz |
| 1100 | /13 | 15.38 MHz |
| 1101 | /14 | 14.29 MHz |
| 1110 | /15 | 13.33 MHz |
| 1111 | /16 | 12.5 MHz (default) |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 19:17 | PWMDIV | R/W | 0x7 | PWM Unit Clock Divisor<br><br>This field specifies the binary divisor used to predivide the system clock down for use as the timing reference for the PWM module. This clock is only power 2 divide and rising edge is synchronous without phase shift from PCLK/HCLK.<br><br>Value Divisor<br>000 /2<br>001 /4<br>010 /8<br>011 /16<br>100 /32<br>101 /64<br>110 /64<br>111 /64 (default) |
| 16:14 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 13 | PWRDN | R/W | 1 | PLL Power Down<br><br>This bit connects to the PLL PWRDN input. The reset value of 1 powers down the PLL. See Table 6-4 on page 83 for PLL mode control. |
| 12 | OEN | R/W | 1 | PLL Output Enable<br><br>This bit specifies whether the PLL output driver is enabled. If cleared, the driver transmits the PLL clock to the output. Otherwise, the PLL clock does not oscillate outside the PLL module.<br><br>**Note:** Both PWRDN and OEN must be cleared to run the PLL. |
| 11 | BYPASS | R/W | 1 | PLL Bypass<br><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><br>**Note:** The ADC module cannot be used when the PLL is in Bypass mode (BYPASS set to 1). |
| 10 | PLLVER | R/W | 0 | PLL Verification<br><br>This bit controls the PLL verification timer function. If set, the verification timer is enabled and an interrupt is generated if the PLL becomes inoperative. Otherwise, the verification timer is not enabled. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 9:6 | XTAL | R/W | 0xB | This field specifies the crystal value attached to the main oscillator. The encoding for this field is provided in Table 6-4 on page 83. |
| | | | Oscillator-Related Bits | |
| 5:4 | OSCSRC | R/W | 0x0 | Picks among the four input sources for the OSC. The values are:<br>Value — Input Source<br>00 — Main oscillator (default)<br>01 — Internal oscillator<br>10 — Internal oscillator / 4 (this is necessary if used as input to PLL)<br>11 — reserved |
| 3 | IOSCVER | R/W | 0 | This bit controls the internal oscillator verification timer function. If set, the verification timer is enabled and an interrupt is generated if the timer becomes inoperative. Otherwise, the verification timer is not enabled. |
| 2 | MOSCVER | R/W | 0 | This bit controls the main oscillator verification timer function. If set, the verification timer is enabled and an interrupt is generated if the timer becomes inoperative. Otherwise, the verification timer is not enabled. |
| 1 | IOSCDIS | R/W | 0 | Internal Oscillator Disable<br>0: Internal oscillator is enabled.<br>1: Internal oscillator is disabled. |
| 0 | MOSCDIS | R/W | 0 | Main Oscillator Disable<br>0: Main oscillator is enabled.<br>1: Main oscillator is disabled. |

**Table 6-3. PLL Mode Control**

| PWRDN | OEN | Mode |
|---|---|---|
| 1 | X | Power down |
| 0 | 0 | Normal |

**Table 6-4. Default Crystal Field Values and PLL Programming**

| Crystal Number (XTAL Binary Value) | Crystal Frequency (MHz) |
|---|---|
| 0000-0011 | reserved |
| 0100 | 3.579545 MHz |
| 0101 | 3.6864 MHz |

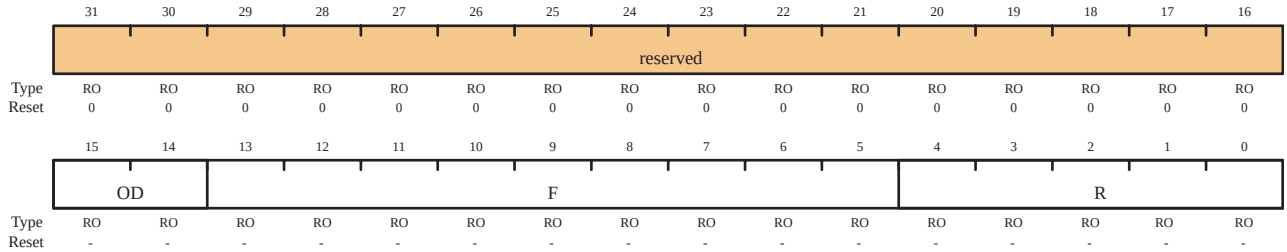
**Table 6-4. Default Crystal Field Values and PLL Programming (Continued)**

| Crystal Number (XTAL Binary Value) | Crystal Frequency (MHz) |
|---|---|
| 0110 | 4 MHz |
| 0111 | 4.096 MHz |
| 1000 | 4.9152 MHz |
| 1001 | 5 MHz |
| 1010 | 5.12 MHz |
| 1011 | 6 MHz (reset value) |
| 1100 | 6.144 MHz |
| 1101 | 7.3728 MHz |
| 1110 | 8 MHz |
| 1111 | 8.192 MHz |

### Register 18: 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 80).

XTAL to PLL Translation (PLLCFG)
Offset 0x064




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:14 | OD | RO | - | This field specifies the value supplied to the PLL's OD input. |
| 13:5 | F | RO | - | This field specifies the value supplied to the PLL's F input. |
| 4:0 | R | RO | - | This field specifies the value supplied to the PLL's R input. |

### Register 19: Run-Mode Clock Gating Control 0 (RCGC0), offset 0x100

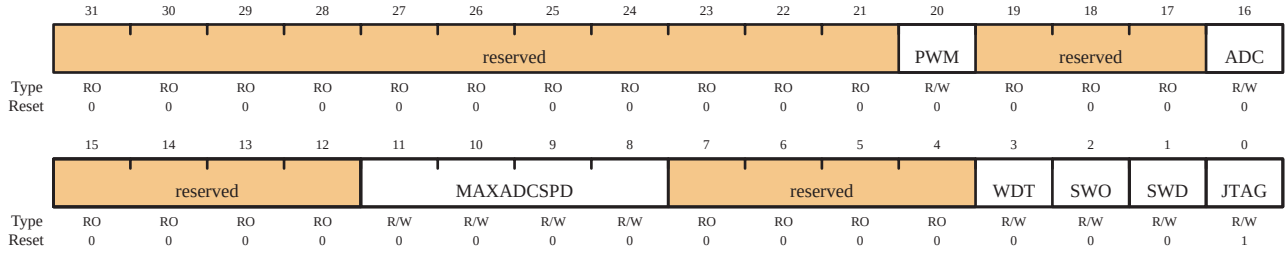
### Register 20: Sleep-Mode Clock Gating Control 0 (SCGC0), offset 0x110

### Register 21: Deep-Sleep-Mode Clock Gating Control 0 (DCGC0), offset 0x120

These registers control 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 (see page 80) specifies that the system uses sleep modes.

Run-Mode, Sleep-Mode and Deep-Sleep-Mode Clock Gating Control 0 (RCGC0, SCG0, and DCGC0)
Offset 0x100, 0x110, 0x120




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:21 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 20 | PWM | R/W | 0 | This bit controls the clock gating for the PWM module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 19:17 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 16 | ADC | R/W | 1 | This bit controls the clock gating for the ADC module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 15:12 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 11:8 | MAXADCSPD | R/W | 0x1 | This field sets the rate at which the ADC samples data. A value of 0x1 indicates the maximum rate of 250K samples per second. *You cannot set the rate higher than the maximum rate.*<br>You can set the sample rate by setting the MAXADCSPD bit as follows:<br>Value — Sample Rate<br>0x0 — 125K samples/second<br>0x1 — 250K samples/second |
| 7:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | WDT | R/W | 0 | 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.[a] |
| 2 | SWO | R/W | 0 | This bit controls the clock gating for the SWO module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 1 | SWD | R/W | 0 | This bit controls the clock gating for the SWD module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 0 | JTAG | R/W | 1 | This bit controls the clock gating for the JTAG module. The reset state for this bit is 1. At reset, the unit receives a clock and functions. Setting this bit to 0 leaves the unit unclocked and disabled.[a] |

a. If the unit is unclocked, reads or writes to the unit will generate a bus fault.

### Register 22: Run-Mode Clock Gating Control 1 (RCGC1), offset 0x104

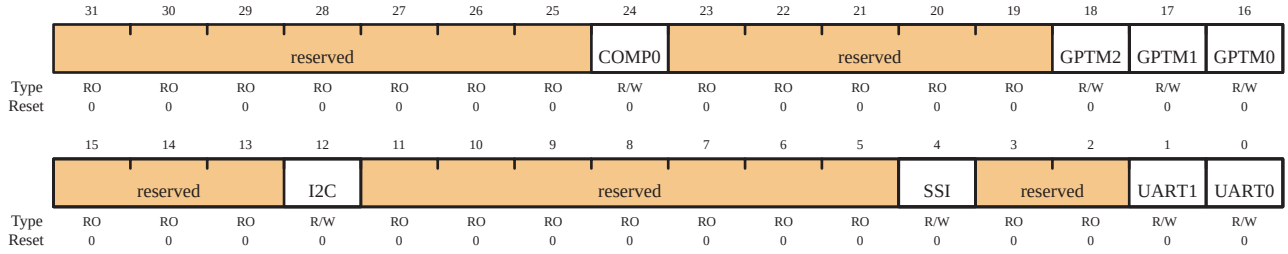
### Register 23: Sleep-Mode Clock Gating Control 1 (SCGC1), offset 0x114

### Register 24: Deep-Sleep-Mode Clock Gating Control 1 (DCGC1), offset 0x124

These registers control 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 (see page 80) specifies that the system uses sleep modes.

Run-Mode, Sleep-Mode, and Deep-Sleep-Mode Clock Gating Control 1 (RCGC1, SCGC1, and DCGC1)
Offset 0x104, 0x114, and 0x124




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:25 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 24 | COMP0 | R/W | 0 | This bit controls the clock gating for the Comparator 0 module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 23:19 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 18 | GPTM2 | R/W | 0 | This bit controls the clock gating for the General Purpose Timer 2 module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 17 | GPTM1 | R/W | 0 | This bit controls the clock gating for the General Purpose Timer 1 module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 16 | GPTM0 | R/W | 0 | This bit controls the clock gating for the General Purpose Timer 0 module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 15:13 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 12 | I2C | R/W | 0 | This bit controls the clock gating for the $I^2C$ module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 11:5 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 4 | SSI | R/W | 0 | This bit controls the clock gating for the SSI module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 3:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | UART1 | R/W | 0 | This bit controls the clock gating for the UART1 module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 0 | UART0 | R/W | 0 | This bit controls the clock gating for the UART0 module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |

a. If the unit is unclocked, reads or writes to the unit will generate a bus fault.

### Register 25: Run-Mode Clock Gating Control 2 (RCGC2), offset 0x108

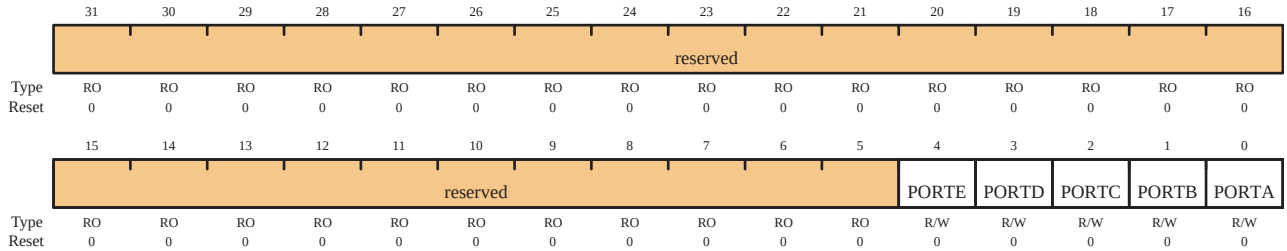
### Register 26: Sleep-Mode Clock Gating Control 2 (SCGC2), offset 0x118

### Register 27: Deep-Sleep-Mode Clock Gating Control 2 (DCGC2), offset 0x128

These registers control 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 (see page 80) specifies that the system uses sleep modes.

Run-Mode, Sleep-Mode, and Deep-Sleep-Mode Clock Gating Control 2 (RCGC2, SCGC2, and DCGC2)
Offset 0x108, 0x118, and 0x128



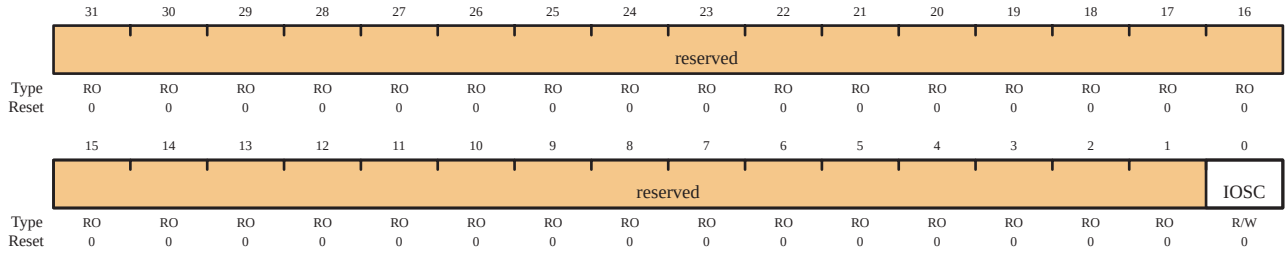

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 4 | PORTE | R/W | 0 | This bit controls the clock gating for the GPIO Port E module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 3 | PORTD | R/W | 0 | This bit controls the clock gating for the GPIO Port D module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 2 | PORTC | R/W | 0 | This bit controls the clock gating for the GPIO Port C module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 1 | PORTB | R/W | 0 | This bit controls the clock gating for the GPIO Port B module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |
| 0 | PORTA | R/W | 0 | This bit controls the clock gating for the GPIO Port A module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled.[a] |

a. If the unit is unclocked, reads or writes to the unit will generate a bus fault.

### Register 28: Deep-Sleep Clock Configuration (DSLPCLKCFG), offset 0x144

This register is used to automatically switch from the main oscillator to the internal oscillator when entering Deep-Sleep mode. The system clock source is the main oscillator by default. When this register is set, the internal oscillator is powered up and the main oscillator is powered down. 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.

Deep-Sleep Clock Configuration (DSLPCLKCFG)
Offset 0x144



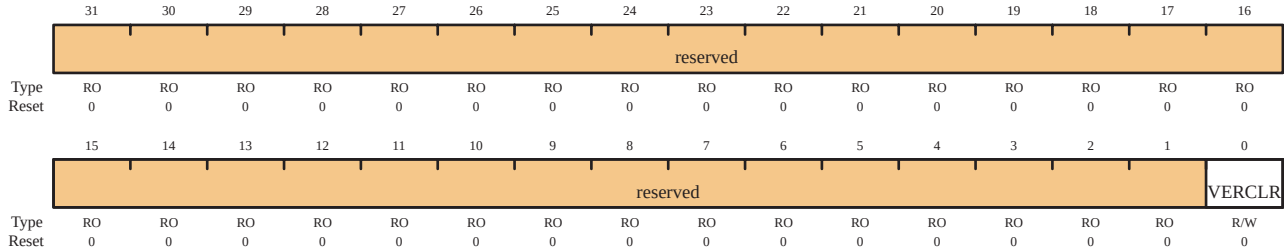

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | Reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | IOSC | R/W | 0 | This field allows an override of the main oscillator when Deep-Sleep mode is running. When set, this field forces the internal oscillator to be the clock source during Deep-Sleep mode. Otherwise, the main oscillator remains as the default system clock source. |

### Register 29: Clock Verification Clear (CLKVCLR), offset 0x150

This register is provided as a means of clearing the clock verification circuits by software. Since the clock verification circuits force a known good clock to control the process, the controller is allowed the opportunity to solve the problem and clear the verification fault. This register clears all clock verification faults. To clear a clock verification fault, the VERCLR bit must be set and then cleared by software. This bit is not self-clearing.

Clock Verification Clear (CLKVCLR)

Offset 0x150

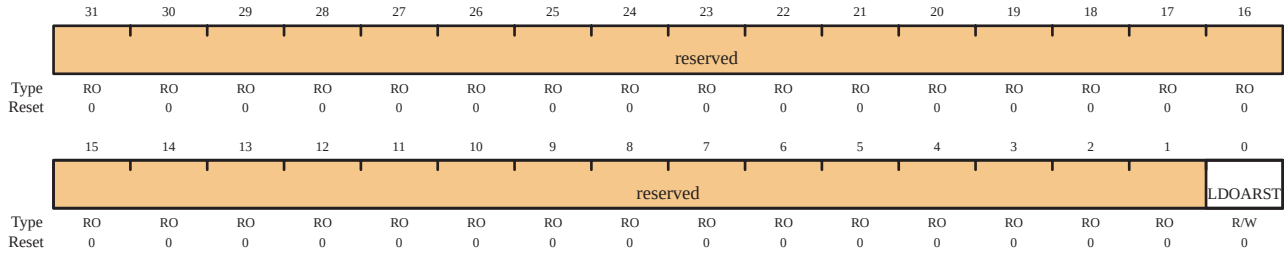



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | Reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | VERCLR | R/W | 0 | Clear clock verification faults. |

### Register 30: Allow Unregulated LDO to Reset the Part (LDOARST), offset 0x160

This register is provided as a means of allowing the LDO to reset the part if the voltage goes unregulated. Use this register to choose whether to automatically reset the part if the LDO goes unregulated, based on the design tolerance for LDO fluctuation.

Allow Unregulated LDO to Reset the Part (LDOARST)
Offset 0x160



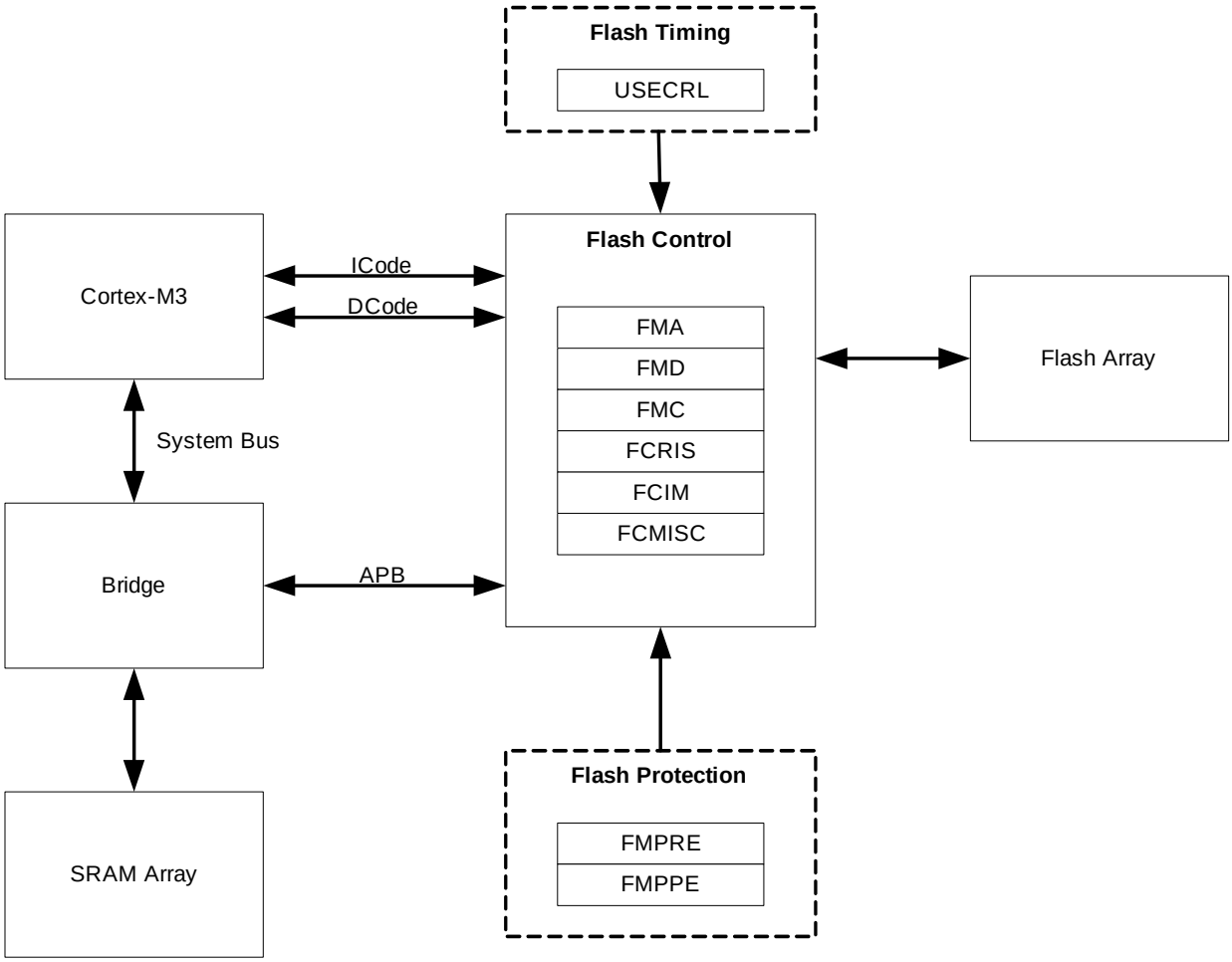

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | Reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | LDOARST | R/W | 0 | Set to 1 to allow unregulated LDO output to reset the part. |

# 7 Internal Memory

The LM3S316 microcontroller comes with 4 KB of bit-banded SRAM and 16 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. Flash Block Diagram**




## 7.2 Functional Description

This section describes the functionality of both memories.

### 7.2.1 SRAM Memory

The internal SRAM of the Stellaris devices is located at address 0x20000000 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.

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 0x20001000 is to be modified, the bit-band alias is calculated as:

```
0x22000000 + (0x1000 * 32) + (3 * 4) = 0x2202000C
```

With the alias address calculated, an instruction performing a read/write to address 0x2202000C allows direct access to only bit 3 of the byte at address 0x20001000.

For details about bit-banding, please refer to Chapter 4, “Memory Map” in the *ARM® Cortex™-M3 Technical Reference Manual*.

## 7.2.2 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.

### 7.2.2.1 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 (see page 100).

On reset, **USECRL** is loaded with a value that configures the flash timing so that it works with the selected crystal value. If software changes the system operating frequency, the new operating frequency 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 must be written to the **USECRL** register.

### 7.2.2.2 Flash Memory Protection

The user is provided two forms of flash protection per 2-KB flash blocks in two 32-bit wide registers. The protection policy for each form is controlled by individual bits (per policy per block) in the **FMPPE** and **FMPRE** registers (see page 99).

- **Flash Memory Protection Program Enable (FMPPE)**: If set, the block may be programmed (written) or erased. If cleared, the block may not be changed.
- **Flash Memory Protection Read Enable (FMPRE)**: If set, the block may be executed or read by software or debuggers. If cleared, the block may only be executed. The contents of the memory block are prohibited from being accessed as data and traversing the DCode bus.

The policies may be combined as shown in Table 7-1.

**Table 7-1. Flash Protection Policy Combinations**

| FMPPE | FMPRE | 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. |

An access that attempts to program or erase a PE-protected block is prohibited. A controller interrupt may be optionally generated (by setting the AMASK bit in the **FIM** register) to alert software developers of poorly behaving software during the development and debug phases.

An access that attempts to read an RE-protected block is prohibited. Such accesses return data filled with all 0s. A controller interrupt may be optionally generated to alert software developers of poorly behaving software during the development and debug phases.

The factory settings for the **FMPRE** and **FMPPE** registers are a value of 1 for all implemented banks. This implements a policy of open access and programmability. The register bits may be changed by writing 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.

#### 7.2.2.3 Flash Memory Programming

Writing the flash memory requires that the code be executed out of SRAM to avoid corrupting or interrupting the bus timing. Flash pages can be erased on a page basis (1 KB in size), or by performing a mass erase of the entire flash.

All erase and program operations are performed using the **Flash Memory Address (FMA)**, **Flash Memory Data (FMD)** and **Flash Memory Control (FMC)** registers. See section 7.3 for examples.

## 7.3 Initialization and Configuration

This section shows examples for using the flash controller to perform various operations on the contents of the flash memory.

### 7.3.1 Changing Flash Protection Bits

As discussed in Section 7.2.2.2, changes to the protection bits must be committed before they take effect. The sequence to change and commit a bit in software is as follows:

1. The **Flash Memory Protection Read Enable (FMPRE)** and **Flash Memory Protection Program Enable (FMPPE)** registers are written, changing the intended bit(s). The action of these changes can be tested by software while in this state.
2. The **Flash Memory Address (FMA)** register (see page 101) bit 0 is set to 1 if the **FMPPE** register is to be committed; otherwise, a 0 commits the **FMPRE** register.
3. The **Flash Memory Control (FMC)** register (see page 103) is written with the COMT bit set. This initiates a write sequence and commits the changes.

### 7.3.2 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**.

***The flash is programmed using the following sequence:***

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 0xA4420001) to the **FMC** register.
4. Poll the **FMC** register until the `WRITE` bit is cleared.

***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 0xA4420002) to the **FMC** register.
3. Poll the **FMC** register until the `ERASE` bit is cleared.

***To perform a mass erase of the flash:***

1. Write the flash write key and the `MERASE` bit (a value of 0xA4420004) to the **FMC** register.
2. Poll the **FMC** register until the `MERASE` bit is cleared.

## 7.4 Register Map

Table 7-2 lists the Flash memory and control registers. The offset listed is a hexadecimal increment to the register's address, relative to the Flash control base address of 0x400FD000, except for **FMPRE** and **FMPPE**, which are relative to the System Control base address of 0x400FE000.

**Table 7-2. Flash Register Map**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x130[a] | FMPRE | 0x000000FF | R/W0 | Flash memory read protect | 99 |
| 0x134[a] | FMPPE | 0x000000FF | R/W0 | Flash memory program protect | 99 |
| 0X140[a] | USECRL | 0x00000018 | R/W | USec reload | 100 |
| 0x000 | FMA | 0x00000000 | R/W | Flash memory address | 101 |
| 0x004 | FMD | 0x00000000 | R/W | Flash memory data | 102 |
| 0x008 | FMC | 0x00000000 | R/W | Flash memory control | 103 |
| 0x00C | FCRIS | 0x00000000 | RO | Flash controller raw interrupt status | 105 |
| 0x010 | FCIM | 0x00000000 | R/W | Flash controller interrupt mask | 106 |
| 0x014 | FCMISC | 0x00000000 | R/W1C | Flash controller masked interrupt status and clear | 107 |

a. Relative to System Control base address of 0x400FE000.

## 7.5 Register Descriptions

The remainder of this section lists and describes the Flash Memory registers, in numerical order by address offset.

### Register 1: Flash Memory Protection Read Enable (FMPRE), offset 0x130

### Register 2: Flash Memory Protection Program Enable (FMPPE), offset 0x134

**Note:** Offset is relative to System Control base address of 0x400FE000

These registers store the read-only (**FMPRE**) and execute-only (**FMPPE**) protection bits for each 2 KB flash block. This register is loaded during the power-on reset sequence.

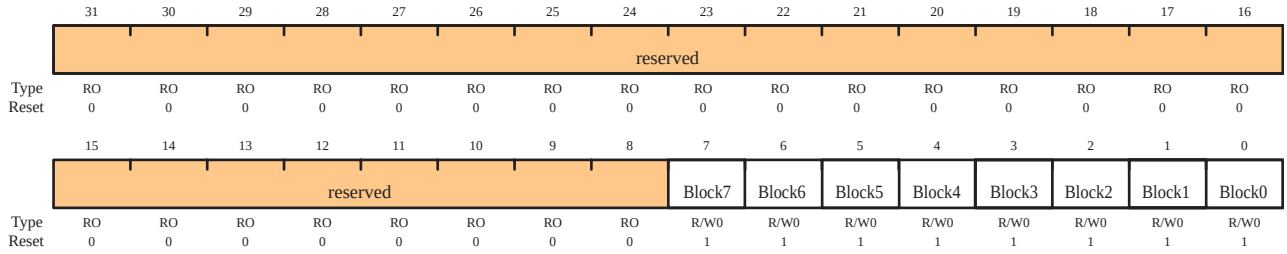
The factory settings for the **FMPRE** and **FMPPE** registers are a value of 1 for all implemented banks. This implements a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection 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.

For additional information, see “Flash Memory Protection” on page 95.

Flash Memory Protection Read Enable and Program Enable (FMPRE and FMPPE)

Offset 0x130 and 0x134




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | Block7-Block0 | R/W0 | 1 | Enable 2 KB flash blocks to be written or erased (**FMPPE** register), or executed or read (**FMPRE** register). The policies may be combined as shown in Table 7-1 on page 96. |

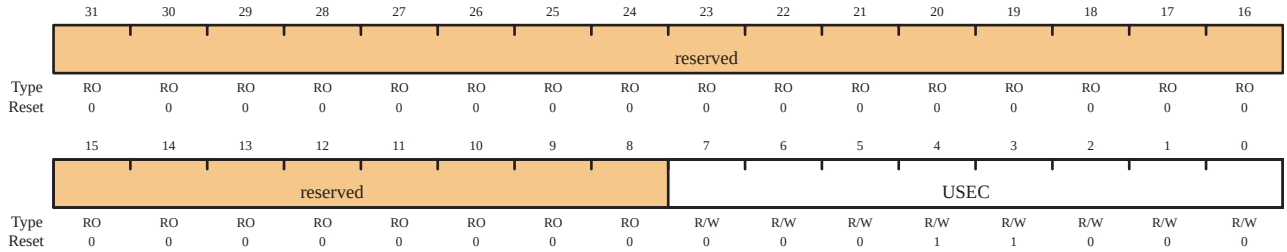
### Register 3: USec Reload (USECRL), offset 0x140

**Note:** Offset is relative to System Control base address of 0x400FE000

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)

Offset 0x140



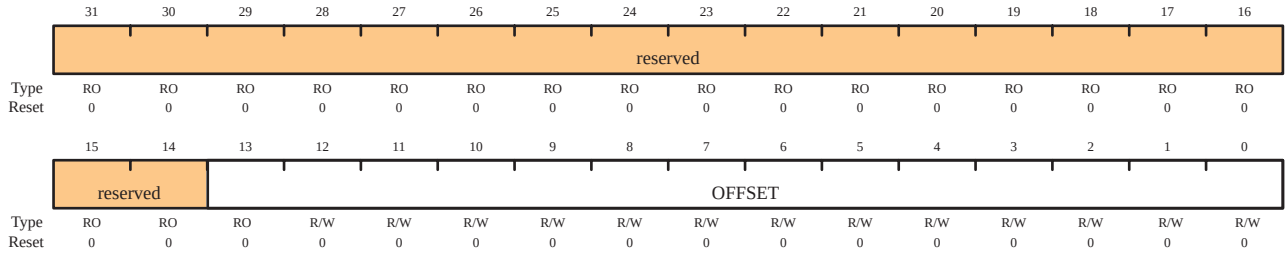

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | USEC | R/W | 0x16 | MHz -1 of the controller clock when the flash is being erased or programmed.<br>USEC should be set to 0x16 (24 MHz) whenever the flash is being erased or programmed. |

### Register 4: 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)

Offset 0x000



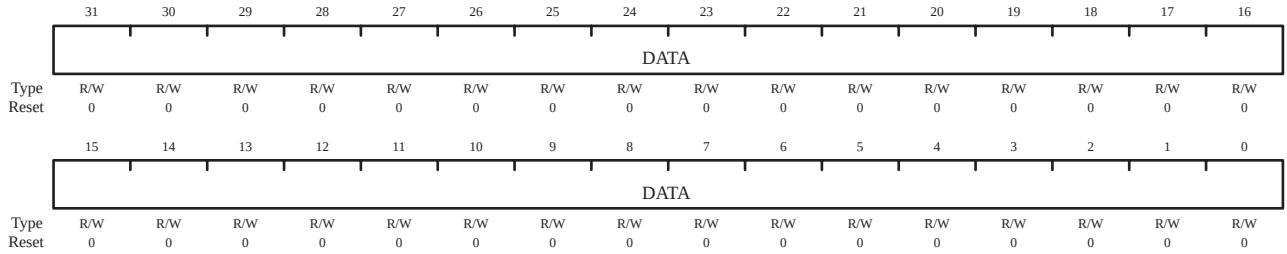

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:14 | reserved | RO | 0x0 | Reserved bits return an indeterminate value, and should never be changed. |
| 13:0 | OFFSET | R/W | 0x0 | Address offset in flash where operation is performed. |

### Register 5: 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)

Offset 0x004




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R/W | 0x0 | Data value for write operation. |

### Register 6: Flash Memory Control (FMC), offset 0x008

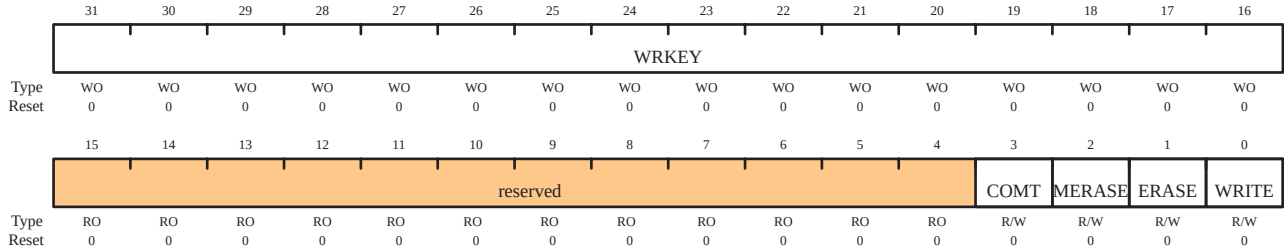
When this register is written, the flash controller initiates the appropriate access cycle for the location specified by the **Flash Memory Address (FMA)** register (see page 101). If the access is a write access, the data contained in the **Flash Memory Data (FMD)** register (see page 102) is written.

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)

Offset 0x008




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | WRKEY | WO | 0x0 | 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 | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | COMT | R/W | 0 | 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. |

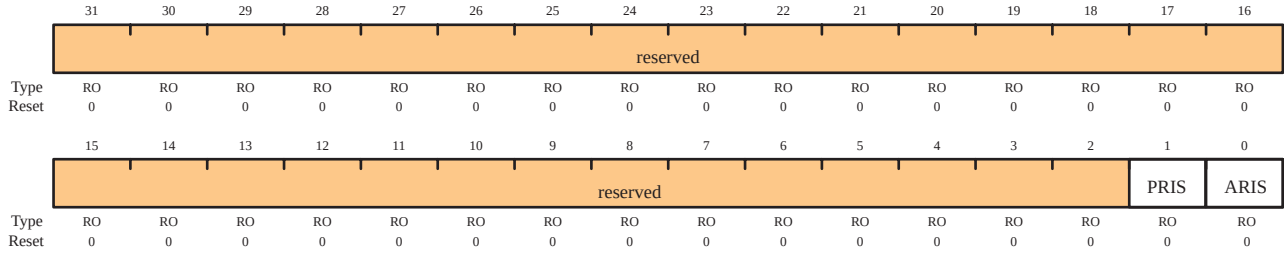
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | ERASE | R/W | 0 | Erase a page of flash memory<br><br>If this bit is set, the page of flash main memory as specified by the contents of **FMA** is erased. A write of 0 has no effect on the state of this bit.<br><br>If read, 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><br>This can take up to 25 ms. |
| 0 | WRITE | R/W | 0 | Write a word into flash memory<br><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><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><br>This can take up to 50 µs. |

### Register 7: 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)

Offset 0x00C

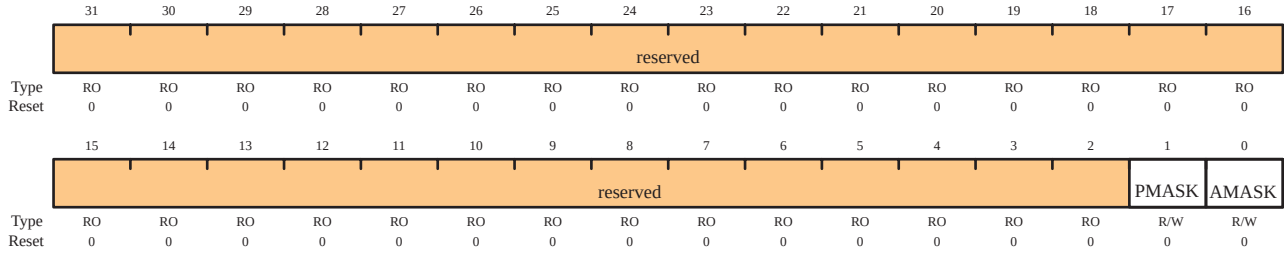



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | PRIS | RO | 0 | Programming Raw Interrupt Status<br>This bit indicates the current state of the programming cycle. If set, the programming cycle completed; if cleared, the programming cycle has not completed. Programming cycles are either write or erase actions generated through the **Flash Memory Control (FMC)** register bits (see page 103). |
| 0 | ARIS | RO | 0 | Access Raw Interrupt Status<br>This bit indicates if the flash was improperly accessed. If set, the program tried to access the flash counter to the policy as set in the **Flash Memory Protection Read Enable (FMPRE)** and **Flash Memory Protection Program Enable (FMPPE)** registers (see page 99). Otherwise, no access has tried to improperly access the flash. |

### Register 8: Flash Controller Interrupt Mask (FCIM), offset 0x010

This register controls whether the flash controller generates interrupts to the controller.

Flash Controller Interrupt Mask (FCIM)
Offset 0x010

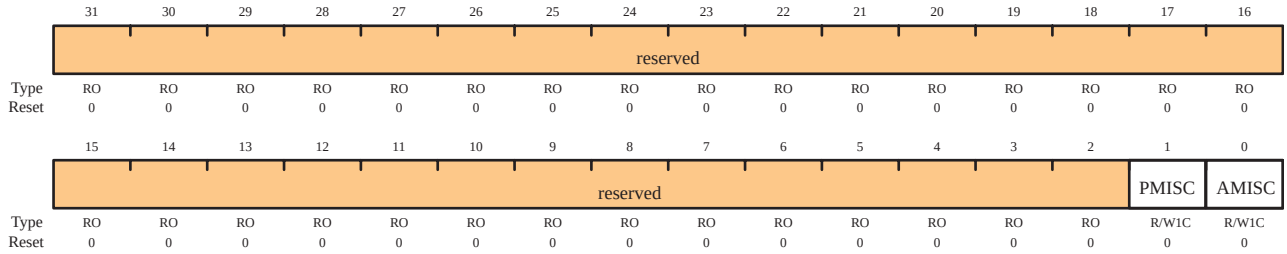



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | PMASK | R/W | 0 | Programming Interrupt Mask<br>This bit controls the reporting of the programming raw interrupt status to the controller. If set, a programming-generated interrupt is promoted to the controller. Otherwise, interrupts are recorded but suppressed from the controller. |
| 0 | AMASK | R/W | 0 | Access Interrupt Mask<br>This bit controls the reporting of the access raw interrupt status to the controller. If set, an access-generated interrupt is promoted to the controller. Otherwise, interrupts are recorded but suppressed from the controller. |

### Register 9: 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)
Offset 0x014




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | PMISC | R/W1C | 0 | Programming Masked Interrupt Status and Clear<br>This bit indicates whether an interrupt was signaled because a programming cycle completed and was not masked. This bit is cleared by writing a 1. The `PRIS` bit in the **FCRIS** register (see page 105) is also cleared when the `PMISC` bit is cleared. |
| 0 | AMISC | R/W1C | 0 | Access Masked Interrupt Status and Clear<br>This bit indicates whether an interrupt was signaled because an improper access was attempted and was not masked. This bit is cleared by writing a 1. The `ARIS` bit in the **FCRIS** register is also cleared when the `AMISC` bit is cleared. |

# 8 General-Purpose Input/Outputs (GPIOs)

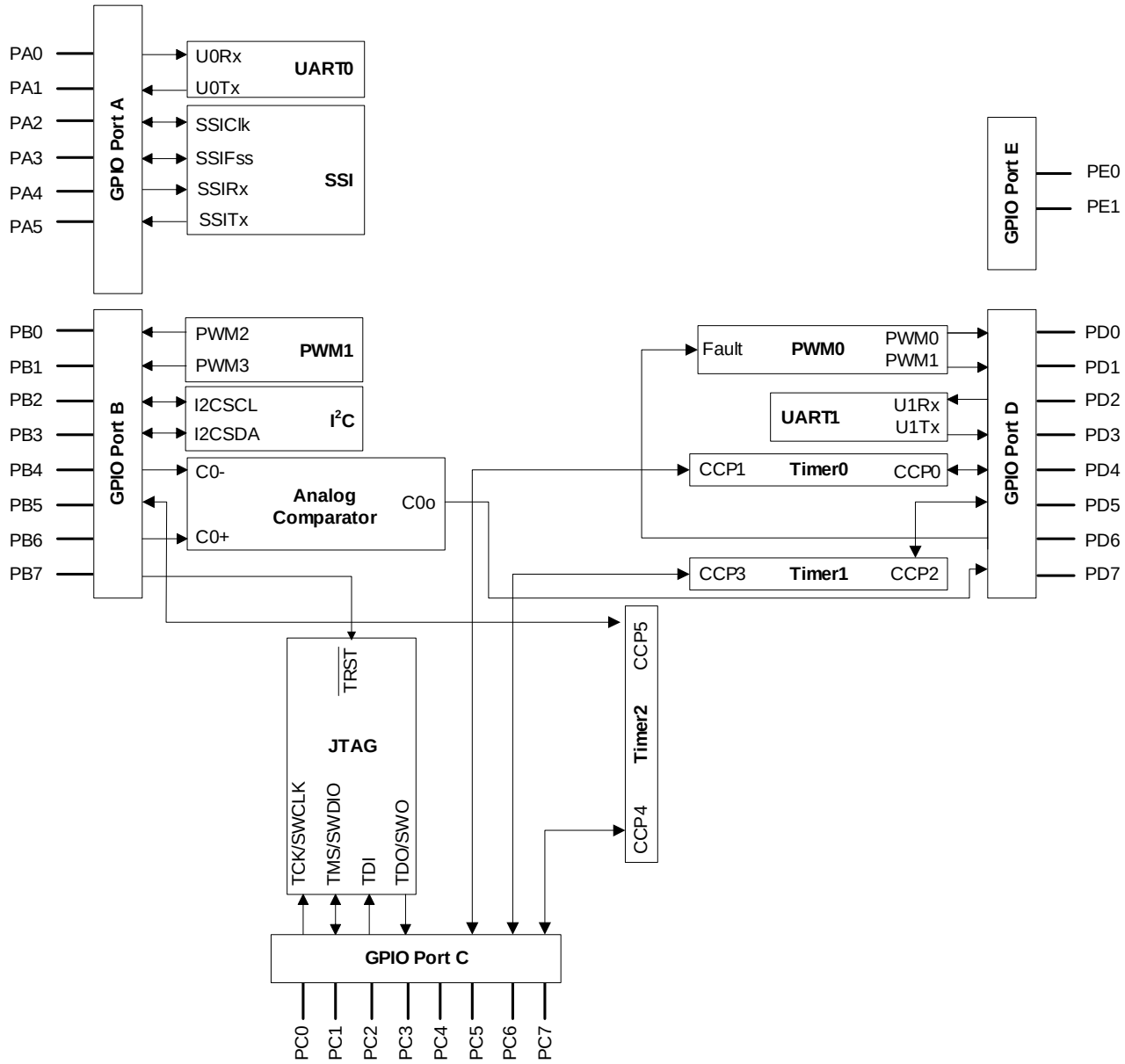
The GPIO module is composed of five physical GPIO blocks, each corresponding to an individual GPIO port (Port A, Port B, Port C, Port D, and Port E). The GPIO module is FiRM-compliant and supports 3 to 32 programmable input/output pins, depending on the peripherals being used.

The GPIO module has the following features:

- Programmable control for GPIO interrupts:
  - Interrupt generation masking
  - Edge-triggered on rising, falling, or both
  - Level-sensitive on High or Low values
- 5-V-tolerant input/outputs
- Bit masking in both read and write operations through address lines
- Programmable control for GPIO pad configuration:
  - Weak pull-up or pull-down resistors
  - 2-mA, 4-mA, and 8-mA pad drive
  - Slew rate control for the 8-mA drive
  - Open drain enables
  - Digital input enables

## 8.1 Block Diagram

**Figure 8-1. GPIO Module Block Diagram**

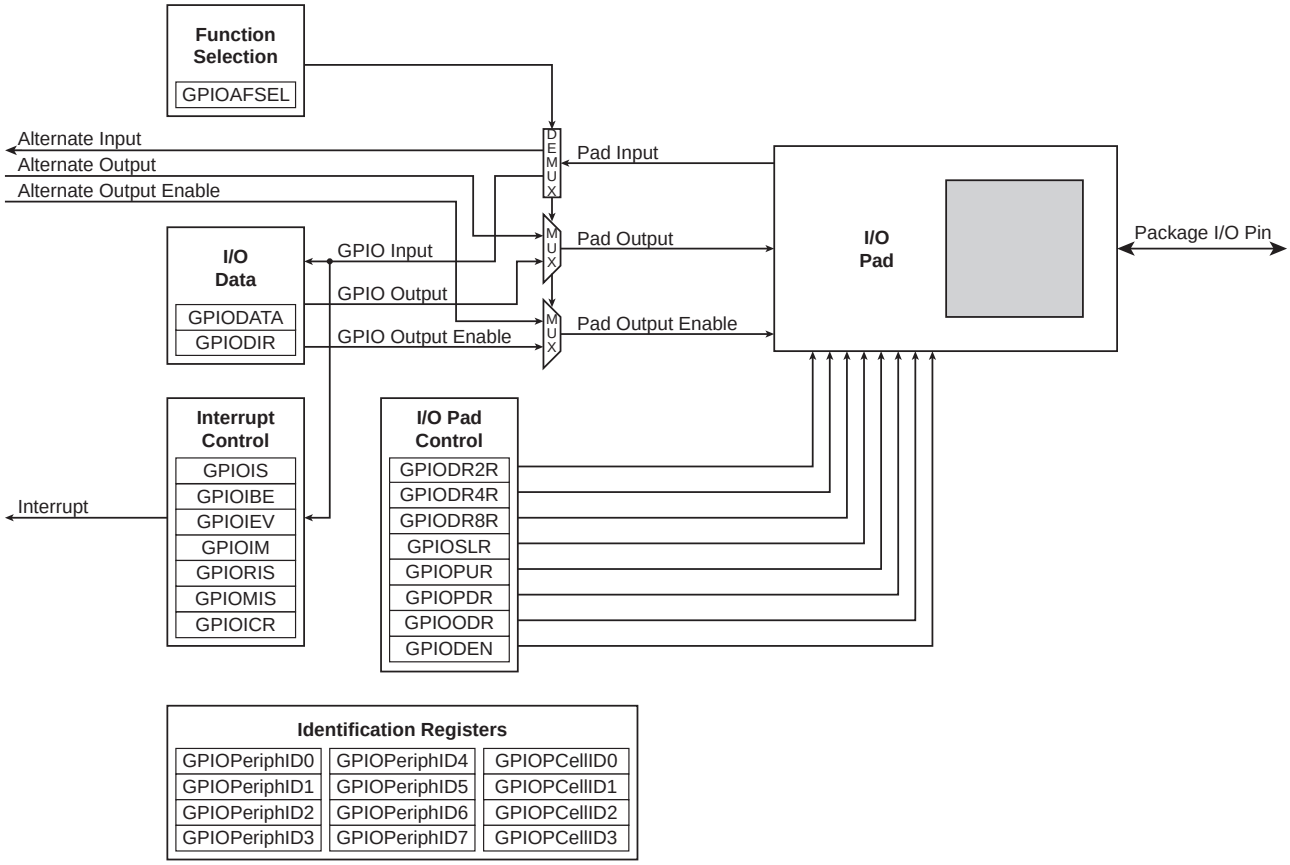



## 8.2 Functional Description

**Important:** All GPIO pins are inputs by default (**GPIODIR**=0 and **GPIOAFSEL**=0), with the exception of the five JTAG pins (PB7 and PC[3:0]. The JTAG pins default to their JTAG functionality (**GPIOAFSEL**=1). Asserting a Power-On-Reset (POR) or an external reset ($\overline{\text{RST}}$) 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 8-2). The LM3S316 microcontroller contains five ports and thus five of these physical GPIO blocks.

**Figure 8-2. GPIO Port Block Diagram**



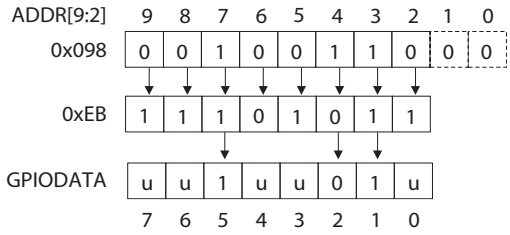

## 8.2.1 Data Register Operation

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 116) 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.

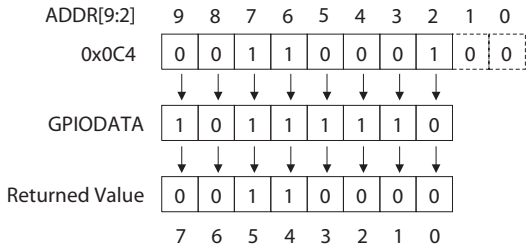
For example, writing a value of 0xEB to the address GPIODATA + 0x098 would yield as shown in Figure 8-3, where u is data unchanged by the write.

**Figure 8-3. 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 8-4.

**Figure 8-4. GPIODATA Read Example**




## 8.2.2 Data Direction

The **GPIO Direction (GPIODIR)** register (see page 117) is used to configure each individual pin as an input or output.

## 8.2.3 Interrupt Operation

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 118)
- **GPIO Interrupt Both Edges (GPIOIBE)** register (see page 119)
- **GPIO Interrupt Event (GPIOIEV)** register (see page 120)

Interrupts are enabled/disabled via the **GPIO Interrupt Mask (GPIOIM)** register (see page 121). 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 pages 122 and 123). 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 (**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 ARM Integrated Nested Vectored Interrupt Controller (NVIC) Interrupt Set Enable (SETNA) 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 B4, and wait for the ADC interrupt or the ADC interrupt needs to be disabled in the SETNA register and the PortB interrupt handler polls the ADC registers until the conversion is completed.

Interrupts are cleared by writing a 1 to the **GPIO Interrupt Clear (GPIOICR)** register (see page 124).

When programming interrupts, 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.

### 8.2.4 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 125), 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.

### 8.2.5 Pad Configuration

The pad configuration registers allow for GPIO pad configuration by software based on the application requirements. The pad configuration registers include the **GPIODR2R**, **GPIODR4R**, **GPIODR8R**, **GPIOODR**, **GPIOPUR**, **GPIOPDR**, **GPIOSLR**, and **GPIODEN** registers.

### 8.2.6 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.

## 8.3 Initialization and Configuration

To use the GPIO, the peripheral clock must be enabled by setting PORTA, PORTB, PORTC, PORTD, and PORTE in the **RCGC2** register.

On reset, all GPIO pins (except for the five JTAG pins) default to general-purpose input mode (**GPIODIR** and **GPIOAFSEL** both set to 0). Table 8-1 shows all possible configurations of the

GPIO pads and the control register settings required to achieve them. Table 8-2 shows how a rising edge interrupt would be configured for pin 2 of a GPIO port.

**Table 8-1. GPIO Pad Configuration Examples**

| Configuration | Register Bit Value[a] | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | GPIOAFSEL | GPIODIR | GPIOODR | GPIODEN | GPIOPUR | GPIOPDR | GPIODR2R | GPIODR4R | GPIODR8R | GPIOSLR |
| Digital Input (GPIO) | 0 | 0 | 0 | 1 | ? | ? | X | X | X | X |
| Digital Output (GPIO) | 0 | 1 | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Open Drain Input (GPIO) | 0 | 0 | 1 | 1 | X | X | X | X | X | X |
| 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 (PWM) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Digital Output (Timer PWM) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Digital Input/Output (SSI) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Digital Input/Output (UART) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Analog Input (Comparator) | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X |
| Digital Output (Comparator) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |

a. X=Ignored (don't care bit)
?=Can be either 0 or 1, depending on the configuration

**Table 8-2. 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)

## 8.4 Register Map

Table 8-2 lists the GPIO registers. The offset listed is a hexadecimal increment to the register's address, relative to that GPIO port's base address:

- GPIO Port A: 0x40004000
- GPIO Port B: 0x40005000
- GPIO Port C: 0x40006000
- GPIO Port D: 0x40007000
- GPIO Port E: 0x40024000

**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 (see Figure 8-1 on page 109). In those cases, writing to those unconnected bits has no effect and reading those unconnected bits returns no meaningful data.

**Table 8-3. GPIO Register Map**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x000 | GPIODATA | 0x00000000 | R/W | Data | 116 |
| 0x400 | GPIODIR | 0x00000000 | R/W | Data direction | 117 |
| 0x404 | GPIOIS | 0x00000000 | R/W | Interrupt sense | 118 |
| 0x408 | GPIOIBE | 0x00000000 | R/W | Interrupt both edges | 119 |
| 0x40C | GPIOIEV | 0x00000000 | R/W | Interrupt event | 120 |
| 0x410 | GPIOIM | 0x00000000 | R/W | Interrupt mask enable | 121 |
| 0x414 | GPIORIS | 0x00000000 | RO | Raw interrupt status | 122 |
| 0x418 | GPIOMIS | 0x00000000 | RO | Masked interrupt status | 123 |
| 0x41C | GPIOICR | 0x00000000 | W1C | Interrupt clear | 124 |
| 0x420 | GPIOAFSEL | see note[a] | R/W | Alternate function select | 125 |
| 0x500 | GPIODR2R | 0x000000FF | R/W | 2-mA drive select | 126 |
| 0x504 | GPIODR4R | 0x00000000 | R/W | 4-mA drive select | 127 |
| 0x508 | GPIODR8R | 0x00000000 | R/W | 8-mA drive select | 128 |
| 0x50C | GPIOODR | 0x00000000 | R/W | Open drain select | 129 |
| 0x510 | GPIOPUR | 0x000000FF | R/W | Pull-up select | 130 |
| 0x514 | GPIOPDR | 0x00000000 | R/W | Pull-down select | 131 |
| 0x518 | GPIOSLR | 0x00000000 | R/W | Slew rate control select | 132 |
| 0x51C | GPIODEN | 0x000000FF | R/W | Digital input enable | 133 |
| 0xFD0 | GPIOPeriphID4 | 0x00000000 | RO | Peripheral identification 4 | 134 |

**Table 8-3. GPIO Register Map (Continued)**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0xFD4 | GPIOPeriphID5 | 0x00000000 | RO | Peripheral identification 5 | 135 |
| 0xFD8 | GPIOPeriphID6 | 0x00000000 | RO | Peripheral identification 6 | 136 |
| 0xFDC | GPIOPeriphID7 | 0x00000000 | RO | Peripheral identification 7 | 137 |
| 0xFE0 | GPIOPeriphID0 | 0x00000061 | RO | Peripheral identification 0 | 138 |
| 0xFE4 | GPIOPeriphID1 | 0x00000000 | RO | Peripheral identification 1 | 139 |
| 0xFE8 | GPIOPeriphID2 | 0x00000018 | RO | Peripheral identification 2 | 140 |
| 0xFEC | GPIOPeriphID3 | 0x00000001 | RO | Peripheral identification 3 | 141 |
| 0xFF0 | GPIOPCellID0 | 0x0000000D | RO | GPIO PrimeCell identification 0 | 142 |
| 0xFF4 | GPIOPCellID1 | 0x000000F0 | RO | GPIO PrimeCell identification 1 | 143 |
| 0xFF8 | GPIOPCellID2 | 0x00000005 | RO | GPIO PrimeCell identification 2 | 144 |
| 0xFFC | GPIOPCellID3 | 0x000000B1 | RO | GPIO PrimeCell identification 3 | 145 |

a. The default reset value for the **GPIOAFSEL** register is 0x00000000 for all GPIO pins, with the exception of the five JTAG pins (PB7 and PC[3:0]. These five pins default to JTAG functionality. Because of this, the default reset value of **GPIOAFSEL** for GPIO Port B is 0x00000080 while the default reset value of **GPIOAFSEL** for Port C is 0x0000000F.

## 8.5 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 117).

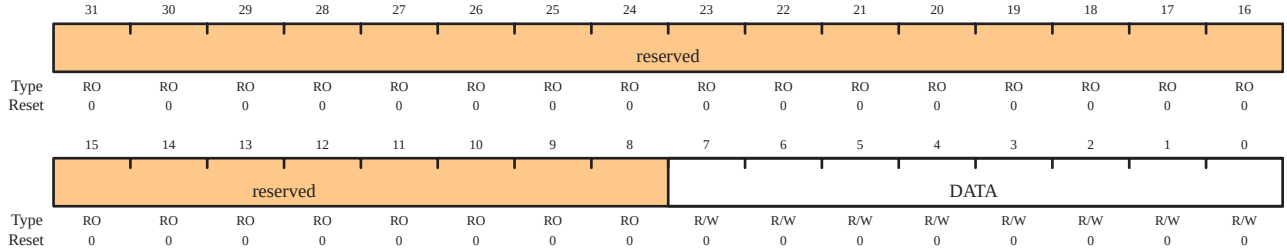
In order to write to **GPIODATA**, the corresponding bits in the mask, resulting from the address bus bits [9:2], must be High. Otherwise, the bit values remain unchanged by the write.

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)

Offset 0x000



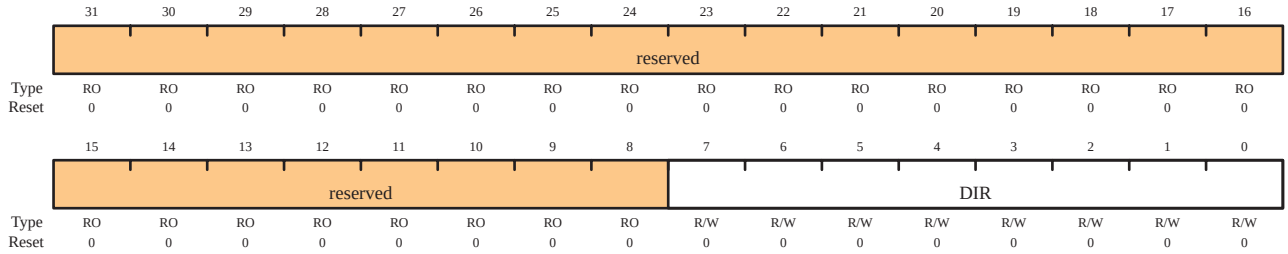

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | DATA | R/W | 0 | GPIO Data<br>This register is virtually mapped to 256 locations in the address space. To facilitate 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 110 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)

Offset 0x400



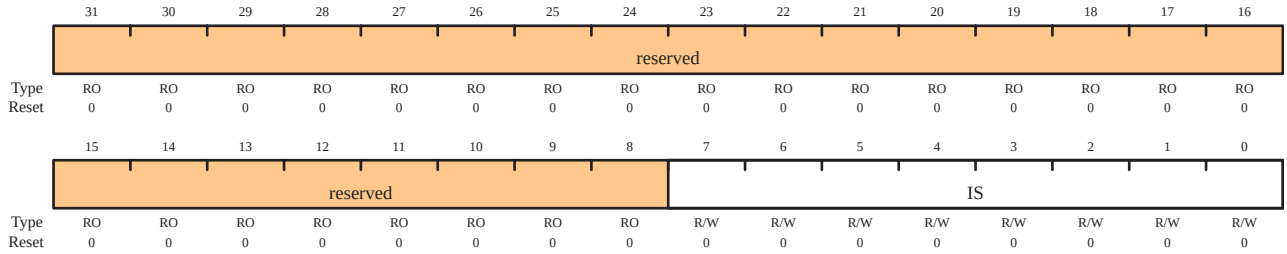

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | DIR | R/W | 0x00 | GPIO Data Direction<br>0: Pins are inputs.<br>1: Pins are outputs. |

### 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)

Offset 0x404



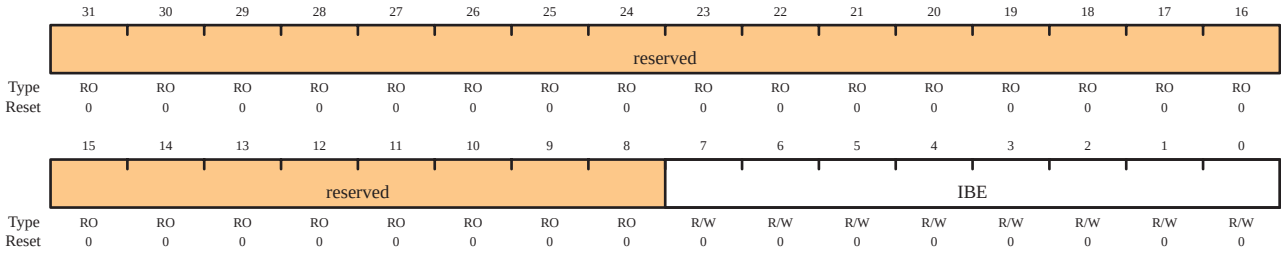

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | IS | R/W | 0x00 | GPIO Interrupt Sense<br>0: Edge on corresponding pin is detected (edge-sensitive).<br>1: Level on 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 118) 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 120). Clearing a bit configures the pin to be controlled by **GPIOIEV**. All bits are cleared by a reset.

GPIO Interrupt Both Edges (GPIOIBE)

Offset 0x408



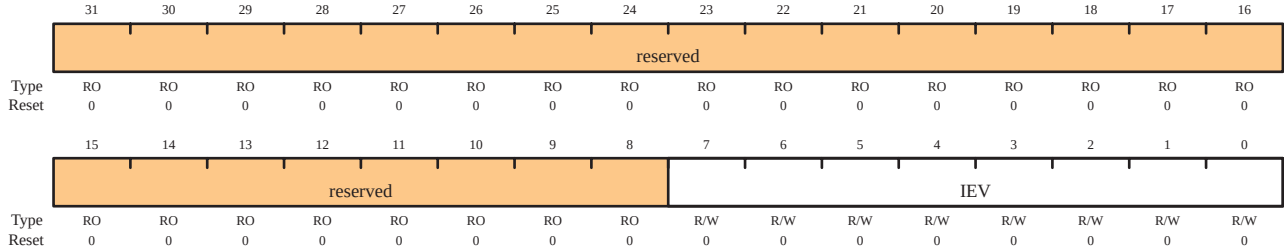

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | IBE | R/W | 0x00 | GPIO Interrupt Both Edges<br>0: Interrupt generation is controlled by the **GPIO Interrupt Event (GPIOIEV)** register (see page 142).<br>1: Both edges on the corresponding pin trigger an interrupt.<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 118). 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)

Offset 0x40C



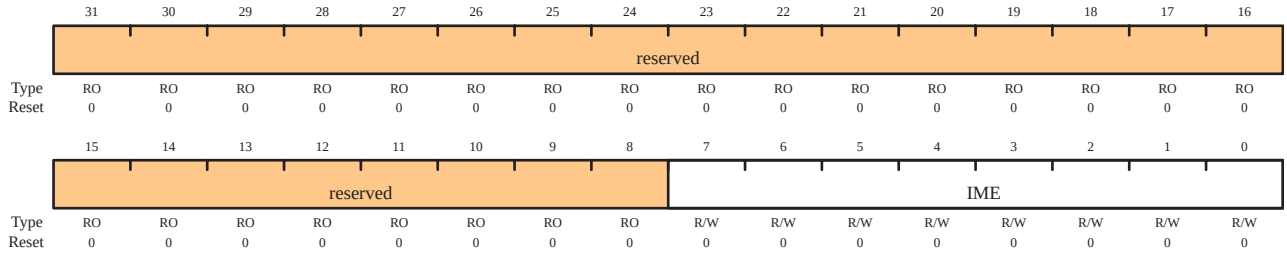

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | IEV | R/W | 0x00 | GPIO Interrupt Event<br>0: Falling edge or Low levels on corresponding pins trigger interrupts.<br>1: 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)

Offset 0x410



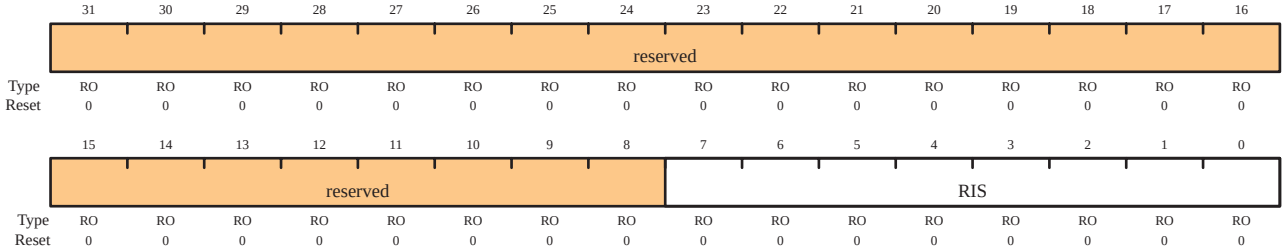

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | IME | R/W | 0x00 | GPIO Interrupt Mask Enable<br>0: Corresponding pin interrupt is masked.<br>1: Corresponding pin interrupt 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 121). 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)

Offset 0x414




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | RIS | RO | 0x00 | GPIO Interrupt Raw Status<br>Reflect the status of interrupt trigger condition detection on pins (raw, prior to masking).<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

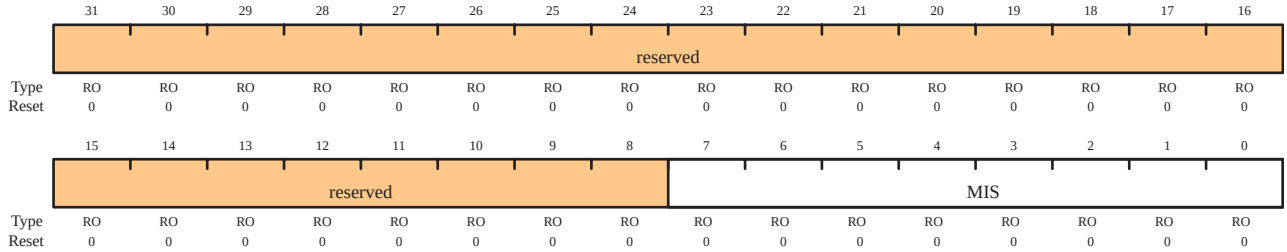
The **GPIOMIS** register is the masked interrupt status register. Bits read High in **GPIOMIS** reflect the status of input lines triggering an interrupt. Bits read as Low indicate that either no interrupt has been generated, or the interrupt is masked.

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 (**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 (see page 212) is configured to use the external trigger, an ADC conversion is initiated.

If no other PortB pins are being used to generate interrupts, the ARM Integrated Nested Vectored Interrupt Controller (NVIC) Interrupt Set Enable (SETNA) 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 B4, and wait for the ADC interrupt or the ADC interrupt needs to be disabled in the SETNA register and the PortB interrupt handler polls the ADC registers until the conversion is completed.

**GPIOMIS** is the state of the interrupt after masking.

GPIO Masked Interrupt Status (GPIOMIS)
Offset 0x418



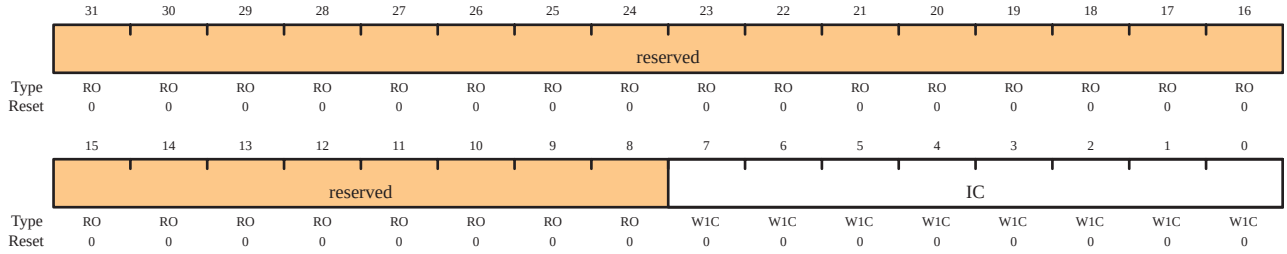

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | MIS | RO | 0x00 | GPIO Masked Interrupt Status<br>Masked value of interrupt due to corresponding pin.<br>0: Corresponding 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)

Offset 0x41C




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | IC | W1C | 0x00 | GPIO Interrupt Clear<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.

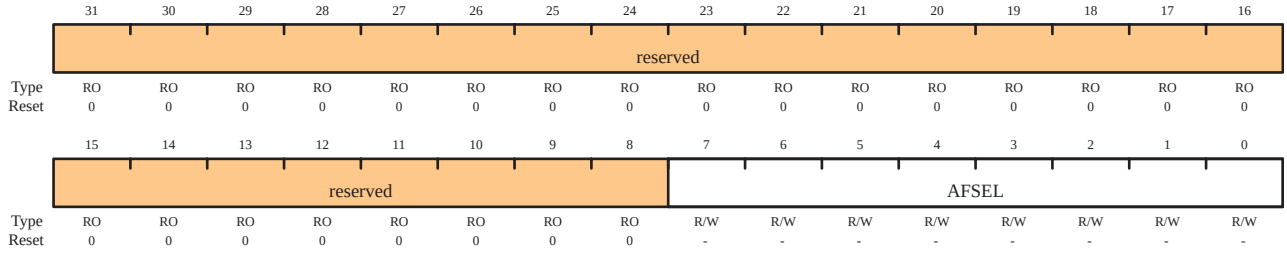
**Caution – All GPIO pins are inputs by default (GPIODIR=0 and GPIOAFSEL=0), with the exception of the five JTAG pins (PB7 and PC[3:0]). The JTAG pins default to their JTAG functionality (GPIOAFSEL=1). Asserting a Power-On-Reset (POR) or an external reset (RST) puts both groups of pins back to their default state.**

**If the JTAG pins are used as GPIOs in a design, PB7 and PC2 cannot have external pull-down resistors connected to both of them at the same time. If both pins are pulled Low during reset, the controller has unpredictable behavior. If this happens, remove one or both of the pull-down resistors, and apply RST or power-cycle the part.**

**In addition, 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)

Offset 0x420



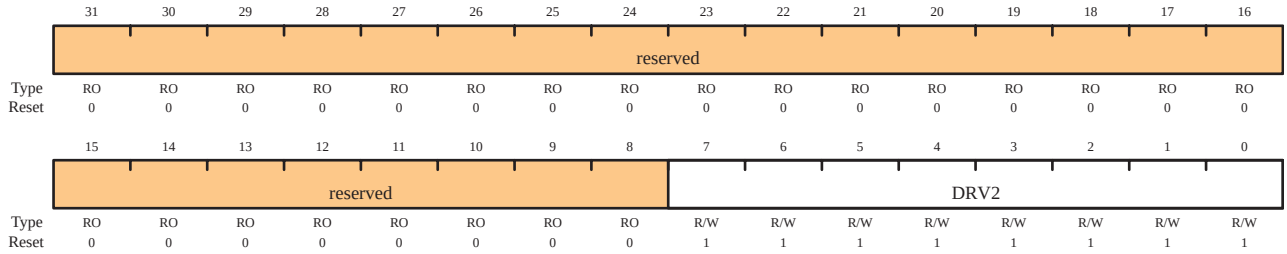

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | AFSEL | R/W | see note | GPIO Alternate Function Select<br>0: Software control of corresponding GPIO line (GPIO mode).<br>1: Hardware control of corresponding GPIO line (alternate hardware function).<br>**Note:** The default reset value for the **GPIOAFSEL** register is 0x00 for all GPIO pins, with the exception of the five JTAG pins (PB7 and PC[3:0]). These five pins default to JTAG functionality. Because of this, the default reset value of **GPIOAFSEL** for GPIO Port B is 0x80 while the default reset value of **GPIOAFSEL** for Port C is 0x0F. |

### 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)

Offset 0x500



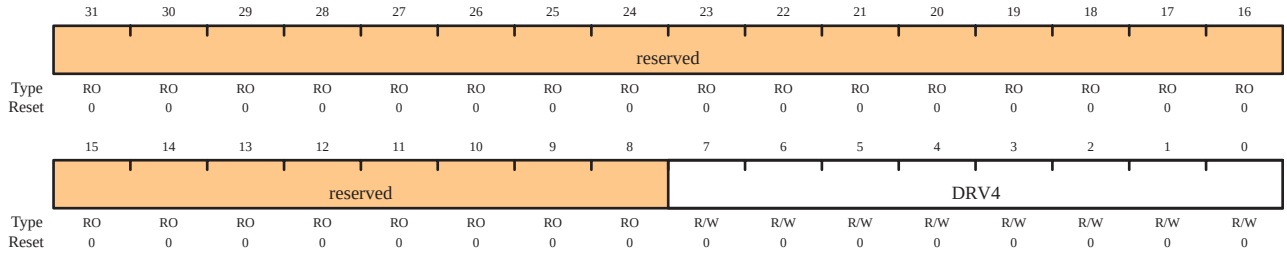

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | DRV2 | R/W | 0xFF | Output Pad 2-mA Drive Enable<br>A write of 1 to either **GPIODR4[n]** or **GPIODR8[n]** clears the corresponding 2-mA enable bit. The change is effective on the second clock cycle after the write. |

### 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)

Offset 0x504



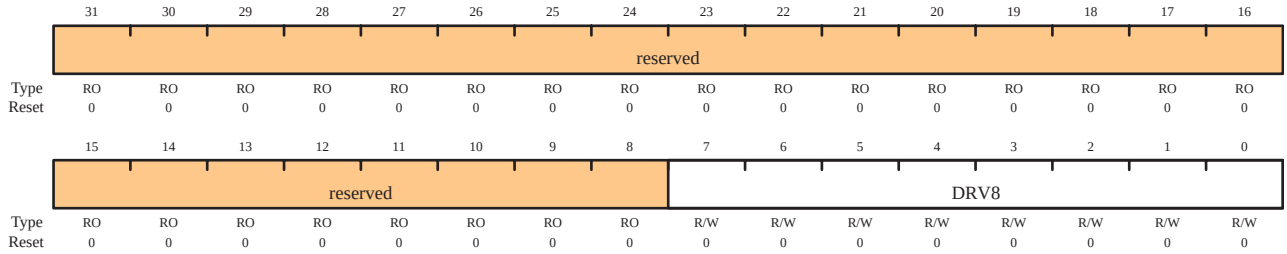

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | DRV4 | R/W | 0x00 | Output Pad 4-mA Drive Enable<br>A write of 1 to either **GPIODR2[n]** or **GPIODR8[n]** clears the corresponding 4-mA enable bit. The change is effective on the second clock cycle after the write. |

### 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.

GPIO 8-mA Drive Select (GPIODR8R)

Offset 0x508




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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. |

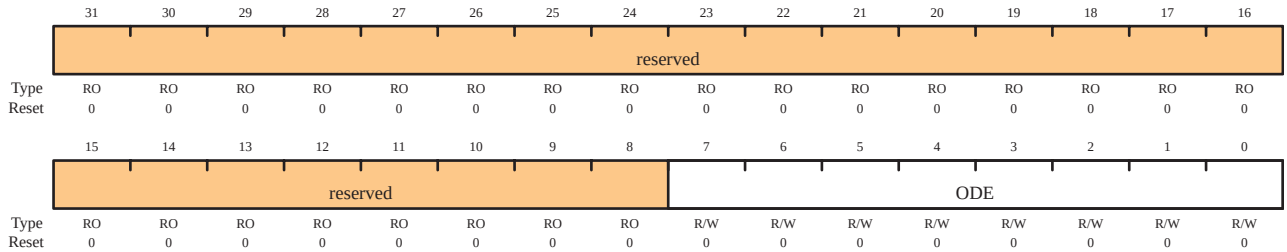
### Register 14: GPIO Open Drain Select (GPIOODR), offset 0x50C

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 Input Enable (GPIODEN)** register (see page 133). 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 set to 0; and as an open drain output when set to 1.

When using the I$^2$C module, the **GPIO Alternate Function Select (GPIOAFSEL)** register bit for `PB2` and `PB3` should be set to 1 (see examples in "Initialization and Configuration" on page 112).

GPIO Open Drain Select (GPIOODR)

Offset 0x50C

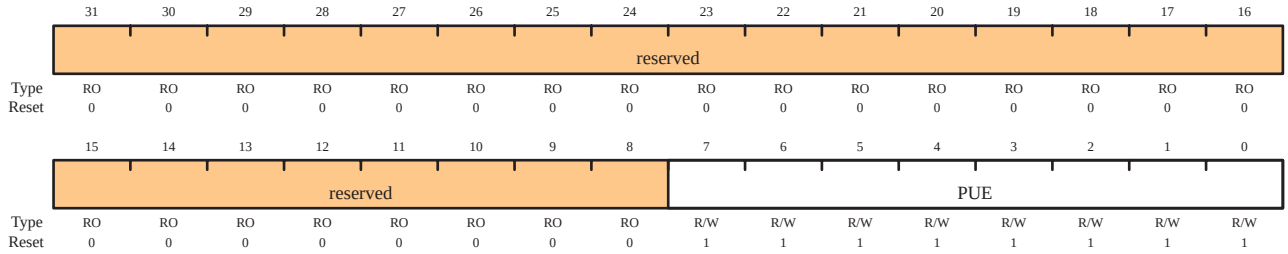



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | ODE | R/W | 0x00 | Output Pad Open Drain Enable<br>0: Open drain configuration is disabled.<br>1: Open drain configuration 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 131).

GPIO Pull-Up Select (GPIOPUR)
Offset 0x510

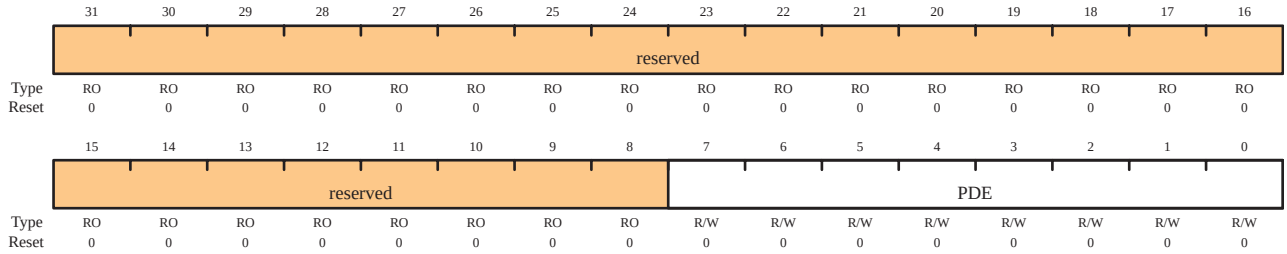



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PUE | R/W | 0xFF | Pad Weak Pull-Up Enable<br>A write of 1 to **GPIOPDR[n]** clears the corresponding **GPIOPUR[n]** enables. The change is effective on the second clock cycle after the write. |

### 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 130).

GPIO Pull-Down Select (GPIOPDR)
Offset 0x514



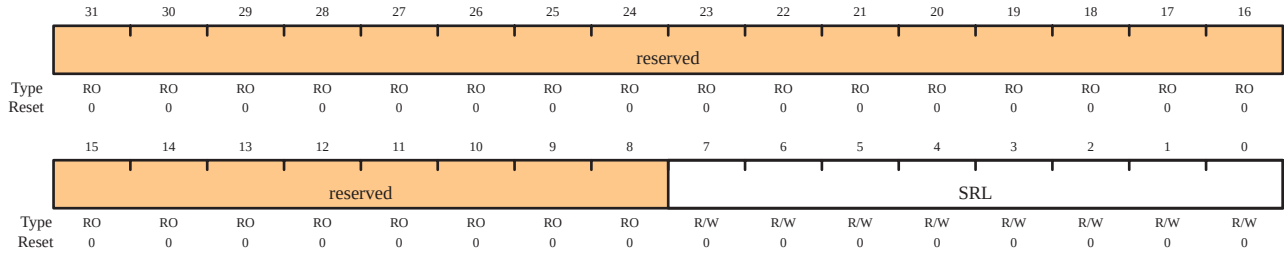

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PDE | R/W | 0x00 | Pad Weak Pull-Down Enable<br>A write of 1 to **GPIOPUR[n]** clears the corresponding **GPIOPDR[n]** 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 128).

GPIO Slew Rate Control Select (GPIOSLR)

Offset 0x518

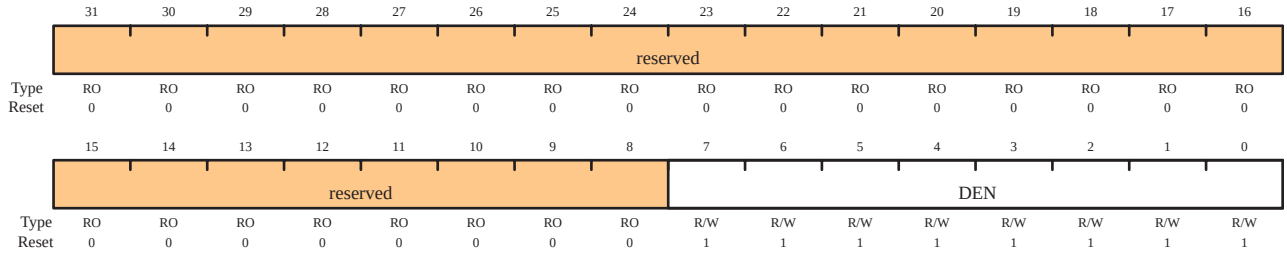



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | SRL | R/W | 0 | Slew Rate Limit Enable (8-mA drive only)<br>0: Slew rate control disabled.<br>1: Slew rate control enabled. |

### Register 18: GPIO Digital Input Enable (GPIODEN), offset 0x51C

The **GPIODEN** register is the digital input enable register. By default, all GPIO signals are configured as digital inputs at reset. The only time that a pin should not be configured as a digital input is when the GPIO pin is configured to be one of the analog input signals for the analog comparators.

GPIO Digital Input Enable (GPIODEN)
Offset 0x51C



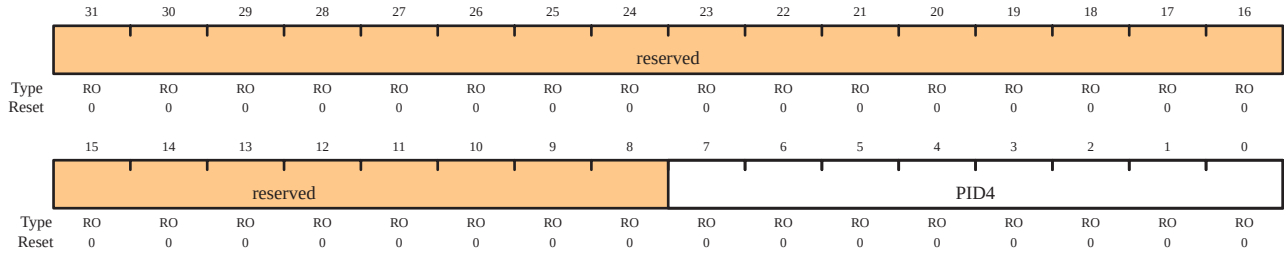

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | DEN | R/W | 0xFF | Digital-Input Enable<br>0: Digital input disabled<br>1: Digital input enabled |

### Register 19: 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)

Offset 0xFD0



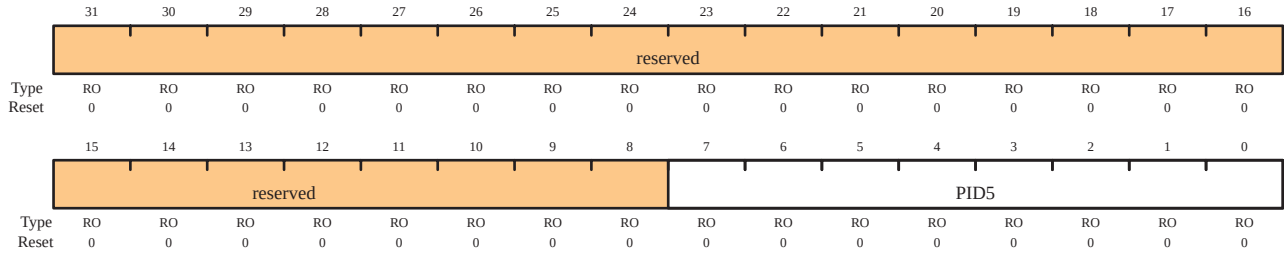

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID4 | RO | 0x00 | GPIO Peripheral ID Register[7:0] |

### Register 20: 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)

Offset 0xFD4



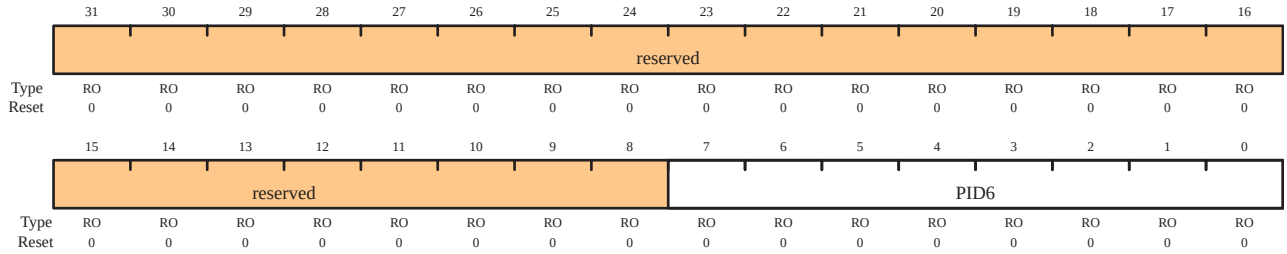

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID5 | RO | 0x00 | GPIO Peripheral ID Register[15:8] |

### Register 21: 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)

Offset 0xFD8



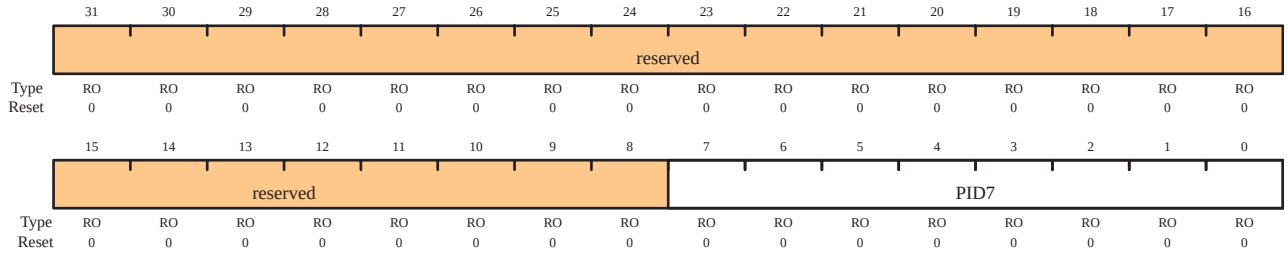

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID6 | RO | 0x00 | GPIO Peripheral ID Register[23:16] |

### Register 22: 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)

Offset 0xFDC



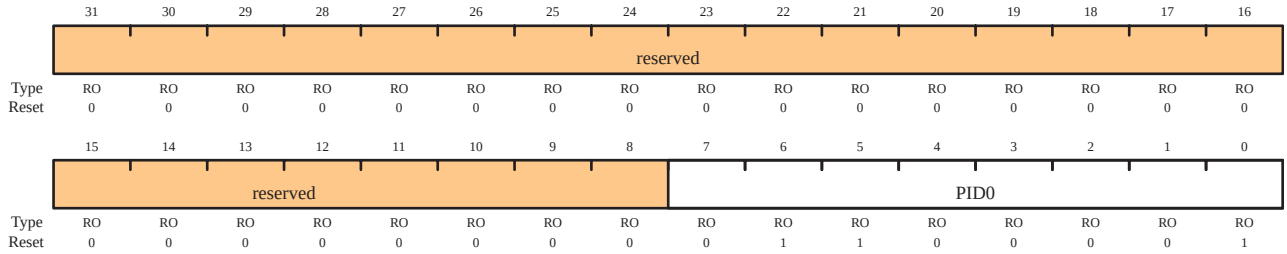

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID7 | RO | 0x00 | GPIO Peripheral ID Register[31:24] |

### Register 23: 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)

Offset 0xFE0



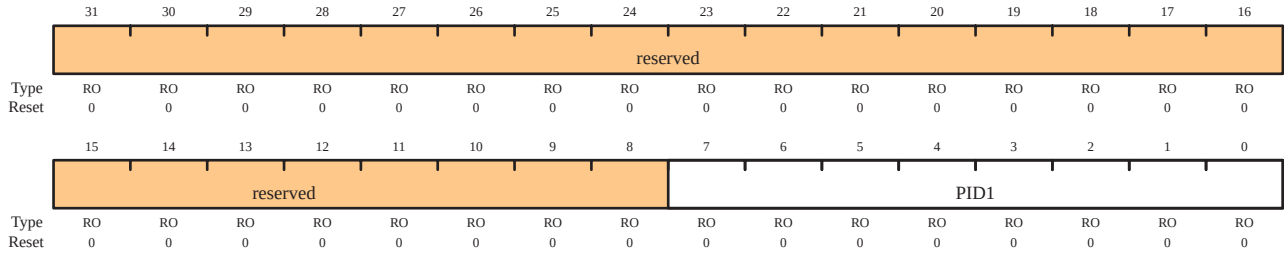

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID0 | RO | 0x61 | GPIO Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

### Register 24: 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)

Offset 0xFE4



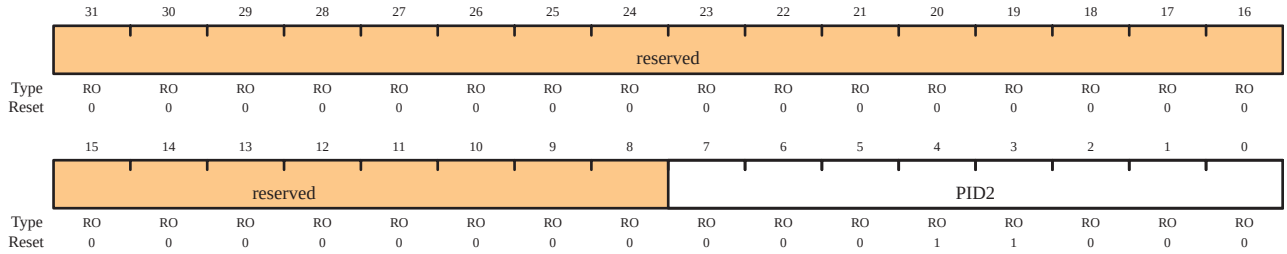

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID1 | RO | 0x00 | GPIO Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

### Register 25: 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)

Offset 0xFE8



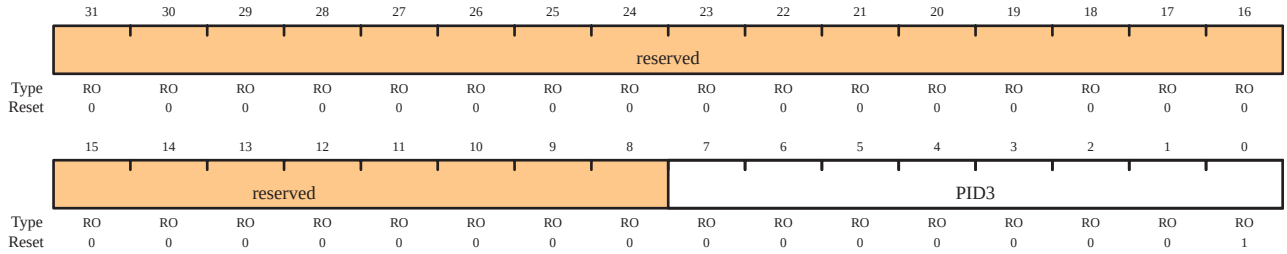

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID2 | RO | 0x18 | GPIO Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

### Register 26: 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)

Offset 0xFEC



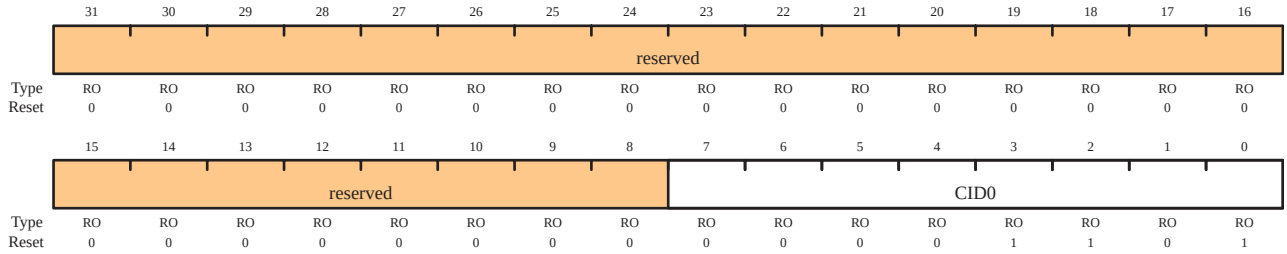

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID3 | RO | 0x01 | GPIO Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

### Register 27: 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)

Offset 0xFF0



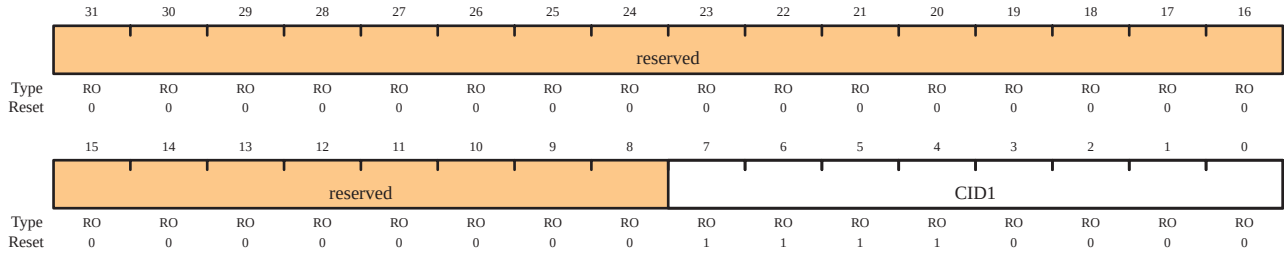

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID0 | RO | 0x0D | GPIO PrimeCell ID Register[7:0]<br>Provides software a standard cross-peripheral identification system. |

### Register 28: 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)

Offset 0xFF4



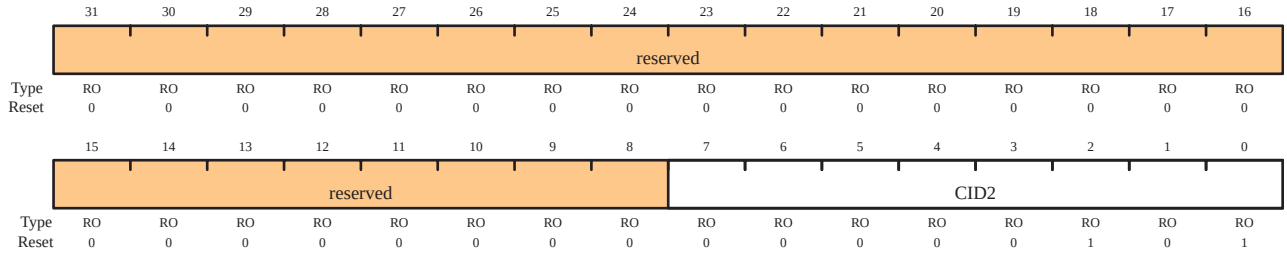

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID1 | RO | 0xF0 | GPIO PrimeCell ID Register[15:8]<br>Provides software a standard cross-peripheral identification system. |

### Register 29: 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)

Offset 0xFF8



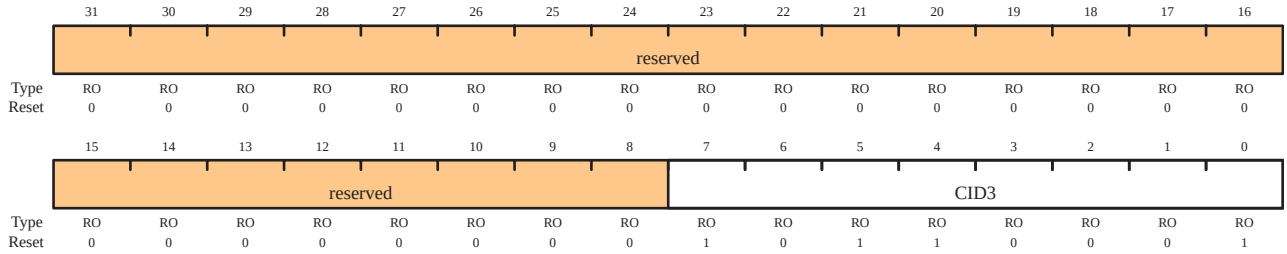

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID2 | RO | 0x05 | GPIO PrimeCell ID Register[23:16]<br>Provides software a standard cross-peripheral identification system. |

### Register 30: 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)

Offset 0xFFC




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID3 | RO | 0xB1 | GPIO PrimeCell ID Register[31:24]<br>Provides software a standard cross-peripheral identification system. |

# 9 General-Purpose Timers

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

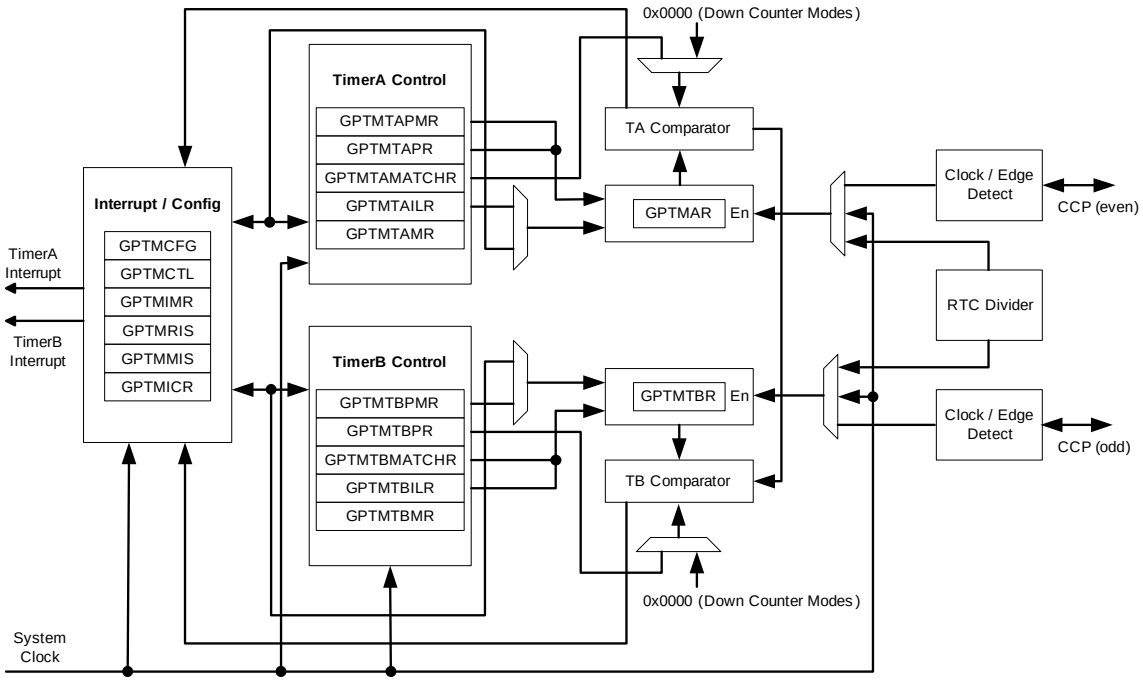
The LM3S316 controller General-Purpose Timer Module (GPTM) contains three GPTM blocks (Timer0, Timer1, and Timer 2). Each GPTM block provides two 16-bit timer/counters (referred to as TimerA and TimerB) that can be configured to operate independently as timers or event counters, or configured to operate as one 32-bit timer or one 32-bit Real-Time Clock (RTC). Timers can also be used to trigger analog-to-digital (ADC) conversions. The 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 following modes are supported:

- 32-bit Timer modes:
  - Programmable one-shot timer
  - Programmable periodic timer
  - Real-Time Clock using 32.768-KHz input clock
  - Software-controlled event stalling (excluding RTC mode)
- 16-bit Timer modes:
  - General-purpose timer function with an 8-bit prescaler
  - Programmable one-shot timer
  - Programmable periodic timer
  - Software-controlled event stalling
- 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

## 9.1 Block Diagram

**Figure 9-1. GPTM Module Block Diagram**




## 9.2 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, two prescaler 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 158), the **GPTM TimerA Mode (GPTMTAMR)** register (see page 159), and the **GPTM TimerB Mode (GPTMTBMR)** register (see page 160). 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.

### 9.2.1 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 168) and the **GPTM TimerB Interval Load (GPTMTBILR)** register (see page 169). The prescale counters are initialized to 0x00: the **GPTM TimerA Prescale (GPTMTAPR)** register (see page 172) and the **GPTM TimerB Prescale (GPTMTBPR)** register (see page 173).

### 9.2.2 32-Bit Timer Operating Modes

**Note:** Both the odd- and even-numbered CCP pins are used for 16-bit mode. Only the even-numbered CCP pins are used for 32-bit mode.

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 168
- **GPTM TimerB Interval Load (GPTMTBILR)** register [15:0], see page 169
- **GPTM TimerA (GPTMTAR)** register [15:0], see page 176
- **GPTM TimerB (GPTMTBR)** register [15:0], see page 177

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]`.

#### 9.2.2.1 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 159), 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 161), the timer begins counting down from its preloaded value. Once the 0x00000000 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 output triggers when it reaches the 0x0000000 state. The GPTM sets the `TATORIS` bit in the **GPTM Raw Interrupt Status (GPTMRIS)** register (see page 165), and holds it until it is cleared by writing the **GPTM Interrupt Clear (GPTMICR)** register (see page 167). If the time-out interrupt is enabled in the **GPTM Interrupt Mask (GPTIMR)** register (see page 163), the GPTM also sets the `TATOMIS` bit in the **GPTM Masked Interrupt Status (GPTMISR)** register (see page 166).

The output trigger is a one-clock-cycle pulse that is asserted when the counter hits the 0x00000000 state, and deasserted on the following clock cycle. It is enabled by setting the `TAOTE` bit in **GPTMCTL**, and can trigger SoC-level events such as ADC conversions.

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 asserted, the timer freezes counting until the signal is deasserted.

#### 9.2.2.2 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 0x00000001. All subsequent load values must be written to the **GPTM TimerA Match (GPTMTAMATCHR)** register (see page 170) by the controller.

The input clock on the CCP0, CCP2 or CCP4 pins 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 in **GPTMCTL**, the counter starts counting up from its preloaded value of 0x00000001. When the current count value matches the preloaded value in **GPTMTAMATCHR**, it rolls over to a value of 0x00000000 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 **GPTIMR**, the GPTM also sets the RTCMIS bit in **GPTMISR** 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**.

## 9.2.3 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 158). This section describes each of the GPTM 16-bit modes of operation. Timer A and Timer B have identical modes, so a single description is given using an **n** to reference both.

### 9.2.3.1 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 output 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 **GPTIMR**, the GPTM also sets the TnTOMIS bit in **GPTMISR** and generates a controller interrupt.

The output trigger is a one-clock-cycle pulse that is asserted when the counter hits the 0x0000 state, and deasserted on the following clock cycle. It is enabled by setting the TnOTE bit in the **GPTMCTL** register, and can trigger SoC-level events such as ADC conversions.

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 enabled, the timer freezes counting until the signal is deasserted.

The following example shows a variety of configurations for a 16-bit free running timer while using the prescaler. All values assume a 25-MHz clock with Tc=20 ns (clock period).

**Table 9-1. 16-Bit Timer With Prescaler Configurations**

| Prescale | #Clock ($T_C$)[a] | Max Time | Units |
|---|---|---|---|
| 00000000 | 1 | 2.6214 | mS |
| 00000001 | 2 | 5.2428 | mS |
| 00000010 | 3 | 7.8642 | mS |
| ---------- | -- | | |
| 11111100 | 254 | 665.8458 | mS |
| 11111110 | 255 | 668.4672 | mS |
| 11111111 | 256 | 671.0886 | mS |

a. $T_C$ is the clock period.

#### 9.2.3.2 16-Bit Input Edge Count Mode

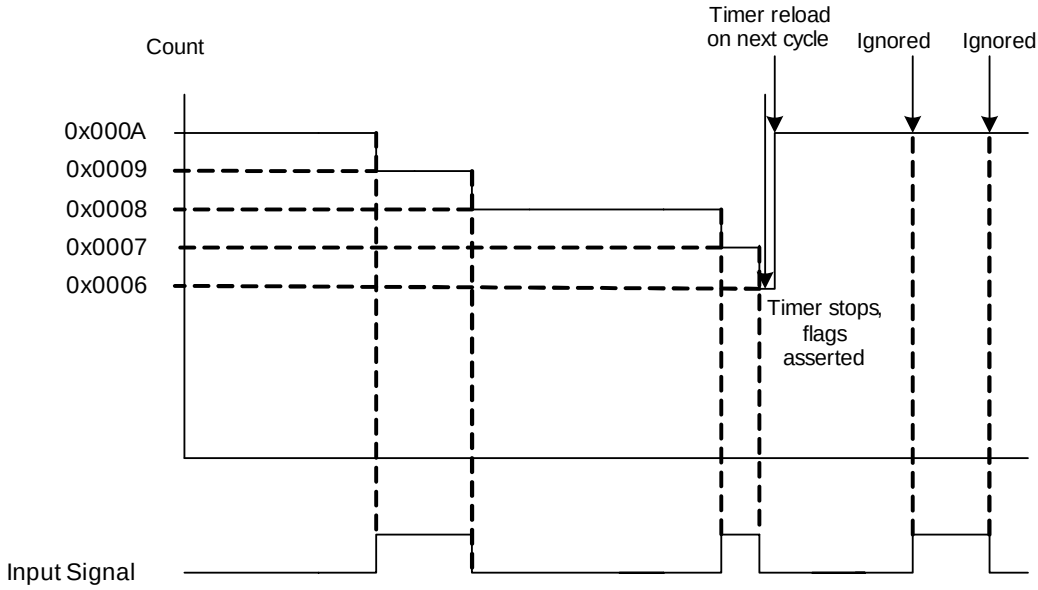
In Edge Count mode, the timer is configured as a down-counter capable of capturing three types of events: rising edge, falling edge, or both. To place the timer in Edge Count mode, the TnCMR bit of the **GPTMTnMR** register must be set to 0. The type of edge that the timer counts is determined by the 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 9-2 shows how input edge count mode works. In this case, the timer start value is set to **GPTMnILR**=0x000A and the match value is set to **GPTMnMATCHR**=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 **GPTMnMR** register.

**Figure 9-2. 16-Bit Input Edge Count Mode Example**




#### 9.2.3.3 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). This mode allows for event capture of both rising and falling edges. 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.

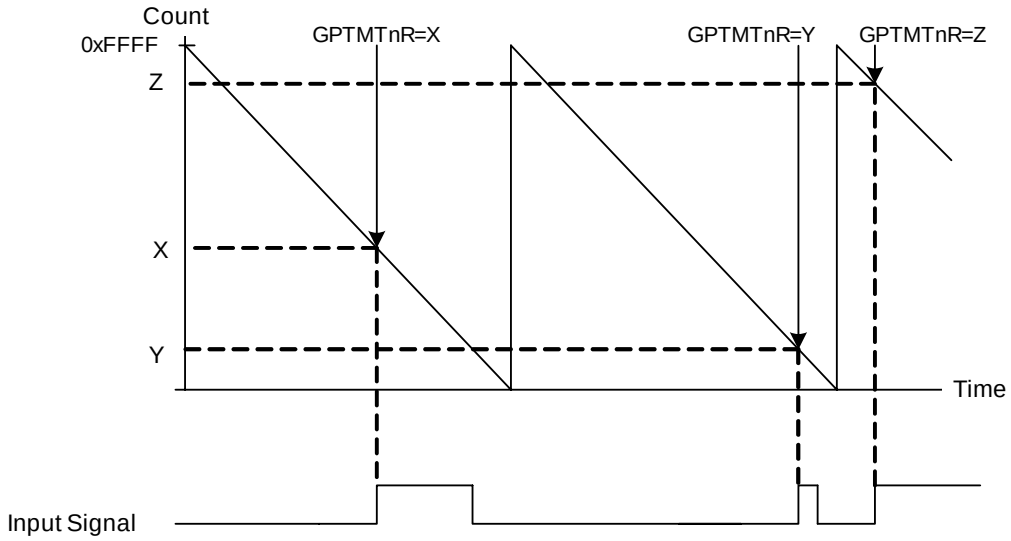
When software writes the TnEN bit in the **GPTMCTL** register, the timer is enabled for event capture. When the selected input event is detected, the current **Tn** counter value is captured in the **GPTMTnR** register and is available to be read by the controller. The GPTM then asserts the CnERIS bit (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 **GPTMnILR** register.

Figure 9-3 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 9-3. 16-Bit Input Edge Time Mode Example**




### 9.2.3.4 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**. 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.

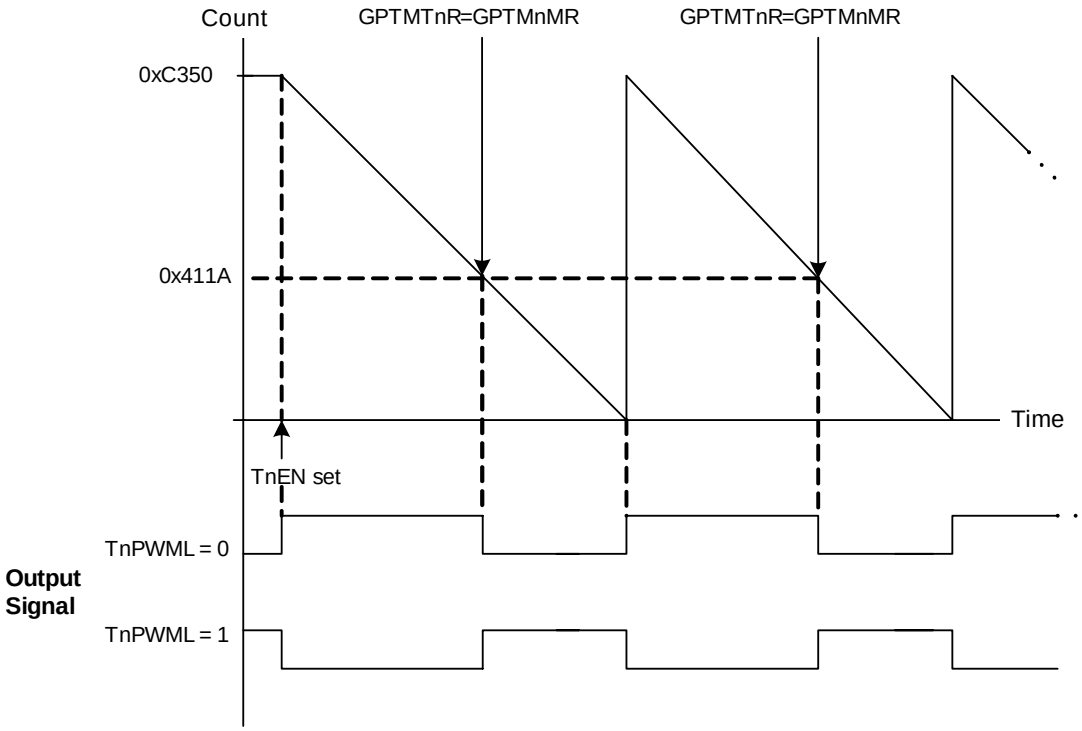
PWM mode can take advantage of the 8-bit prescaler by using the **GPTM Timern Prescale Register (GPTMTnPR)** and the **GPTM Timern Prescale Match Register (GPTMTnPMR)**. This effectively extends the range of the timer to 24 bits.

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 **GPTMTnPR** if using a prescaler) 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 (GPTMnMATCHR)**. Software has the capability of inverting the output PWM signal by setting the TnPWML bit in the **GPTMCTL** register.

Figure 9-4 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 **GPTMnIRL**=0xC350 and the match value is **GPTMnMR**=0x411A.

**Figure 9-4. 16-Bit PWM Mode Example**




## 9.3 Initialization and Configuration

To use the general purpose timers, the peripheral clock must be enabled by setting the GPTM0, GPTM1, and GPTM2 bits in the **RCGC1** register.

This section shows module initialization and configuration examples for each of the supported timer modes.

### 9.3.1 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. To re-enable the timer, repeat the sequence. A timer configured in Periodic mode does not stop counting after it times out.

### 9.3.2 32-Bit Real-Time Clock (RTC) Mode

To use the RTC mode, the timer must have a 32.768-KHz input signal on its CCP0, CCP2 or CCP4 pins. 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 counter is re-loaded with 0x00000000 and begins counting. If an interrupt is enabled, it does not have to be cleared.

### 9.3.3 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. To re-enable the timer, repeat the sequence. A timer configured in Periodic mode does not stop counting after it times out.

### 9.3.4 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 steps 4-9.

### 9.3.5 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.

### 9.3.6 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 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 **GPTM Timern Match (GPTMTnMATCHR)** register with the desired value.

7. If a prescaler is going to be used, configure the **GPTM Timern Prescale (GPTMTnPR)** register and the **GPTM Timern Prescale Match (GPTMTnPMR)** register.
8. 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.

## 9.4 Register Map

Table 9-1 lists the GPTM registers. The offset listed is a hexadecimal increment to the register's address, relative to that timer's base address:

- Timer0: 0x40030000
- Timer1: 0x40031000
- Timer2: 0x40032000

**Table 9-2. GPTM Register Map**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x000 | GPTMCFG | 0x00000000 | R/W | Configuration | 158 |
| 0x004 | GPTMTAMR | 0x00000000 | R/W | TimerA mode | 159 |
| 0x008 | GPTMTBMR | 0x00000000 | R/W | TimerB mode | 160 |
| 0x00C | GPTMCTL | 0x00000000 | R/W | Control | 161 |
| 0x018 | GPTMIMR | 0x00000000 | R/W | Interrupt mask | 163 |
| 0x01C | GPTMRIS | 0x00000000 | RO | Interrupt status | 165 |
| 0x020 | GPTMMIS | 0x00000000 | RO | Masked interrupt status | 166 |
| 0x024 | GPTMICR | 0x00000000 | W1C | Interrupt clear | 167 |
| 0x028 | GPTMTAILR | 0x0000FFFF[a]<br>0xFFFFFFFF | R/W | TimerA interval load | 168 |
| 0x02C | GPTMTBILR | 0x0000FFFF | R/W | TimerB interval load | 169 |
| 0x030 | GPTMTAMATCHR | 0x0000FFFF[a]<br>0xFFFFFFFF | R/W | TimerA match | 170 |
| 0x034 | GPTMTBMATCHR | 0x0000FFFF | R/W | TimerB match | 171 |
| 0x038 | GPTMTAPR | 0x00000000 | R/W | TimerA prescale | 172 |
| 0x03C | GPTMTBPR | 0x00000000 | R/W | TimerB prescale | 173 |
| 0x040 | GPTMTAPMR | 0x00000000 | R/W | TimerA prescale match | 174 |
| 0x044 | GPTMTBPMR | 0x00000000 | R/W | TimerB prescale match | 175 |

**Table 9-2. GPTM Register Map (Continued)**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x048 | GPTMTAR | 0x0000FFFF[a]<br>0xFFFFFFFF | RO | TimerA | 176 |
| 0x04C | GPTMTBR | 0x0000FFFF | RO | TimerB | 177 |

a. The default reset value for the **GPTMTAILR**, **GPTMTAMATCHR**, and **GPTMTAR** registers is 0x0000FFFF when in 16-bit mode and 0xFFFFFFFF when in 32-bit mode.

## 9.5 Register Descriptions

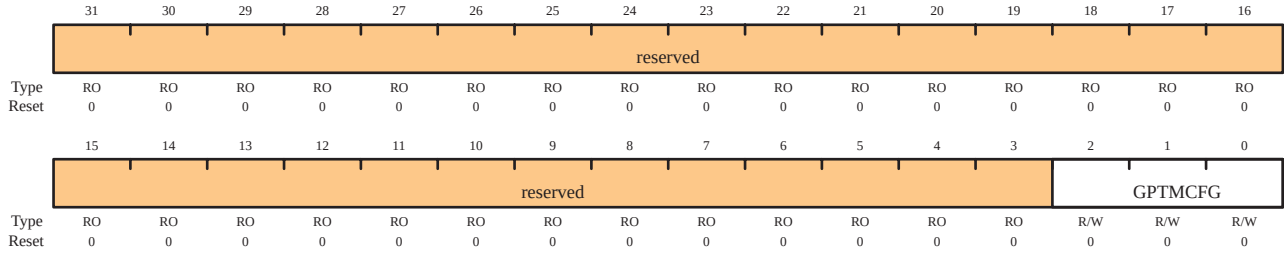
The remainder of this section lists and describes the GPTM registers, in numerical order by address offset.

### 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)

Offset 0x000

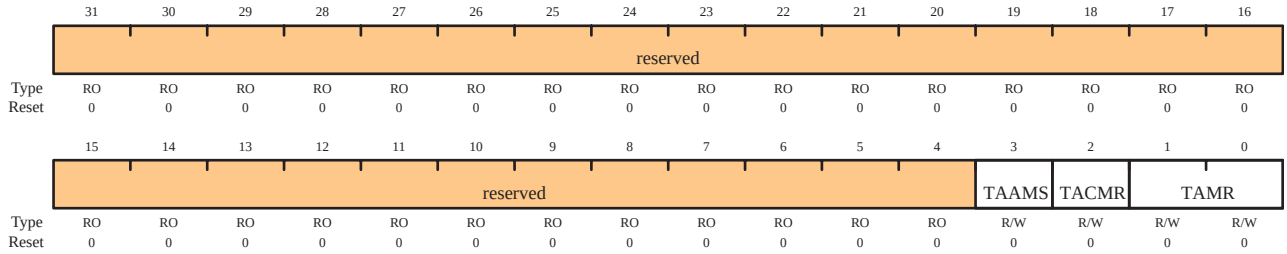



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 2:0 | GPTMCFG | R/W | 0 | GPTM Configuration<br>0x0: 32-bit timer configuration.<br>0x1: 32-bit real-time clock (RTC) counter 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)
Offset 0x004

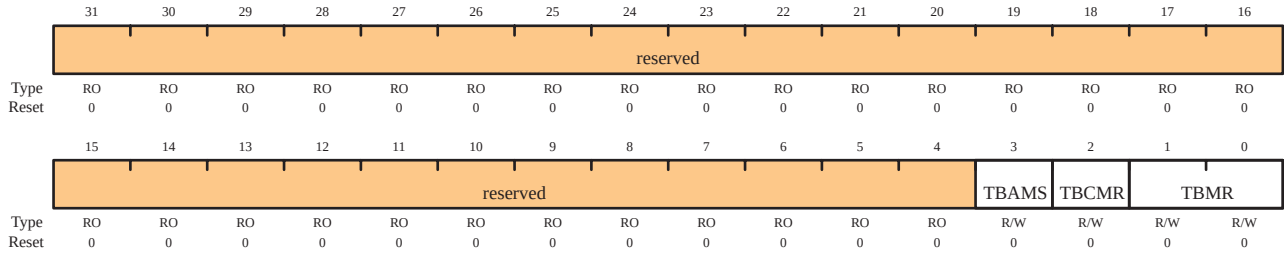



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | TAAMS | R/W | 0 | GPTM TimerA Alternate Mode Select<br>0: Capture mode is enabled.<br>1: PWM mode is enabled.<br>**Note:** 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>0: Edge-Count mode.<br>1: Edge-Time mode. |
| 1:0 | TAMR | R/W | 0 | GPTM TimerA Mode<br>0x0: Reserved.<br>0x1: One-Shot Timer mode.<br>0x2: Periodic Timer mode.<br>0x3: Capture mode.<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)
Offset 0x008

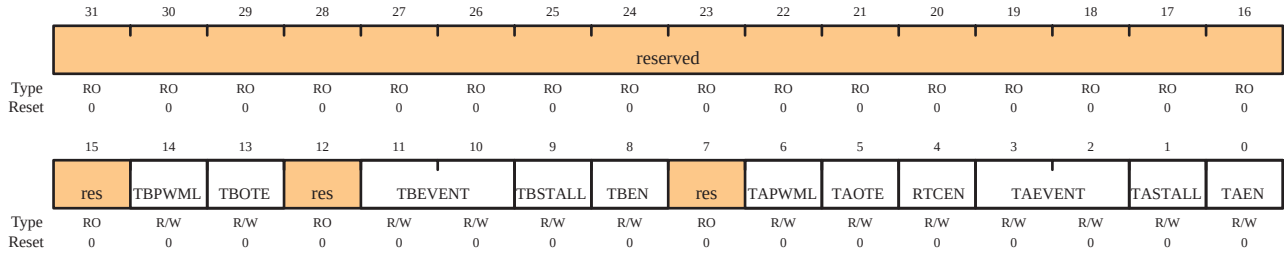



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | TBAMS | R/W | 0 | GPTM TimerB Alternate Mode Select<br>0: Capture mode is enabled.<br>1: PWM mode is enabled.<br>**Note:** 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>0: Edge-Count mode.<br>1: Edge-Time mode. |
| 1:0 | TBMR | R/W | 0 | GPTM TimerB Mode<br>0x0: Reserved.<br>0x1: One-Shot Timer mode.<br>0x2: Periodic Timer mode.<br>0x3: Capture mode.<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)
Offset 0x00C



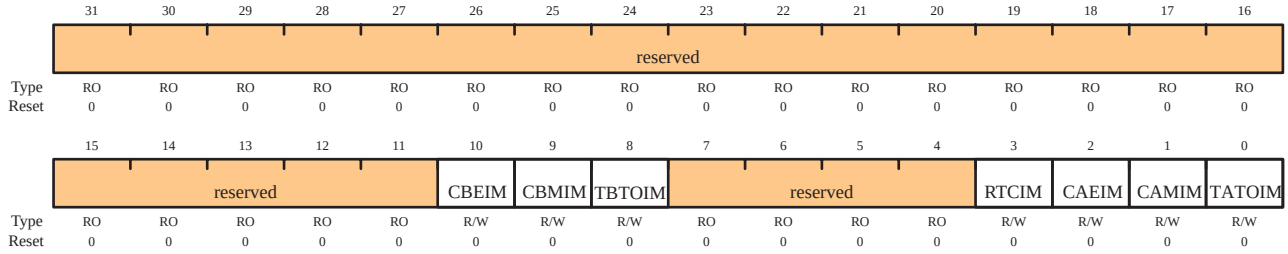

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:15 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 14 | TBPWML | R/W | 0 | GPTM TimerB PWM Output Level<br>0: Output is unaffected.<br>1: Output is inverted. |
| 13 | TBOTE | R/W | 0 | GPTM TimerB Output Trigger Enable<br>0: The output TimerB trigger is disabled.<br>1: The output TimerB trigger is enabled. |
| 12 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 11:10 | TBEVENT | R/W | 0 | GPTM TimerB Event Mode<br>00: Positive edge.<br>01: Negative edge.<br>10: Reserved.<br>11: Both edges. |
| 9 | TBSTALL | R/W | 0 | GPTM TimerB Stall Enable<br>0: TimerB stalling is disabled.<br>1: TimerB stalling is enabled. |
| 8 | TBEN | R/W | 0 | GPTM TimerB Enable<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 | Reserved bits return an indeterminate value, and should never be changed. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | TAPWML | R/W | 0 | GPTM TimerA PWM Output Level<br>0: Output is unaffected.<br>1: Output is inverted. |
| 5 | TAOTE | R/W | 0 | GPTM TimerA Output Trigger Enable<br>0: The output TimerA trigger is disabled.<br>1: The output TimerA trigger is enabled. |
| 4 | RTCEN | R/W | 0 | GPTM RTC Enable<br>0: RTC counting is disabled.<br>1: RTC counting is enabled. |
| 3:2 | TAEVENT | R/W | 0 | GPTM TimerA Event Mode<br>00: Positive edge.<br>01: Negative edge.<br>10: Reserved.<br>11: Both edges. |
| 1 | TASTALL | R/W | 0 | GPTM TimerA Stall Enable<br>0: TimerA stalling is disabled.<br>1: TimerA stalling is enabled. |
| 0 | TAEN | R/W | 0 | GPTM TimerA Enable<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)
Offset 0x018



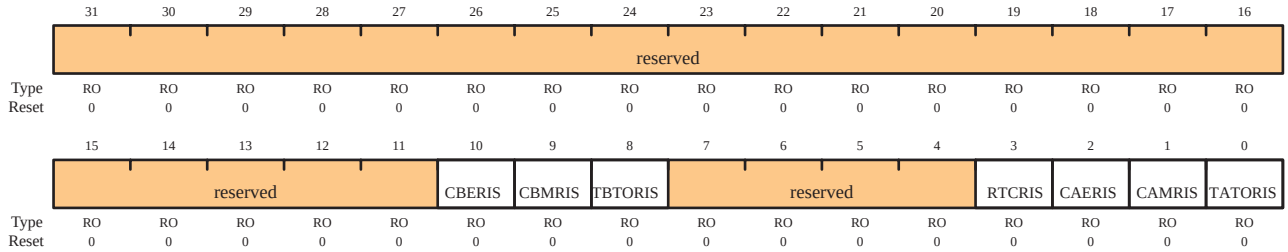

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 10 | CBEIM | R/W | 0 | GPTM CaptureB Event Interrupt Mask<br>0: Interrupt is disabled.<br>1: Interrupt is enabled. |
| 9 | CBMIM | R/W | 0 | GPTM CaptureB Match Interrupt Mask<br>0: Interrupt is disabled.<br>1: Interrupt is enabled. |
| 8 | TBTOIM | R/W | 0 | GPTM TimerB Time-Out Interrupt Mask<br>0: Interrupt is disabled.<br>1: Interrupt is enabled. |
| 7:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | RTCIM | R/W | 0 | GPTM RTC Interrupt Mask<br>0: Interrupt is disabled.<br>1: Interrupt is enabled. |
| 2 | CAEIM | R/W | 0 | GPTM CaptureA Event Interrupt Mask<br>0: Interrupt is disabled.<br>1: Interrupt is enabled. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | CAMIM | R/W | 0 | GPTM CaptureA Match Interrupt Mask<br>0: Interrupt is disabled.<br>1: Interrupt is enabled. |
| 0 | TATOIM | R/W | 0 | GPTM TimerA Time-Out Interrupt Mask<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)
Offset 0x01C

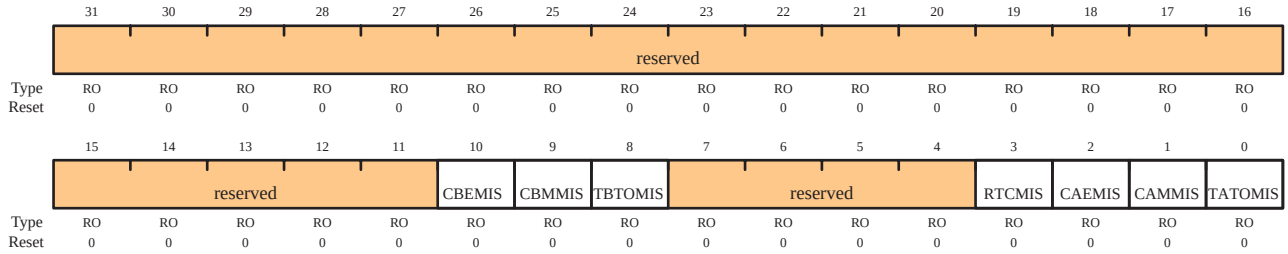



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 10 | CBERIS | RO | 0 | GPTM CaptureB Event Raw Interrupt<br>This is the CaptureB Event interrupt status prior to masking. |
| 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 | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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)
Offset 0x020

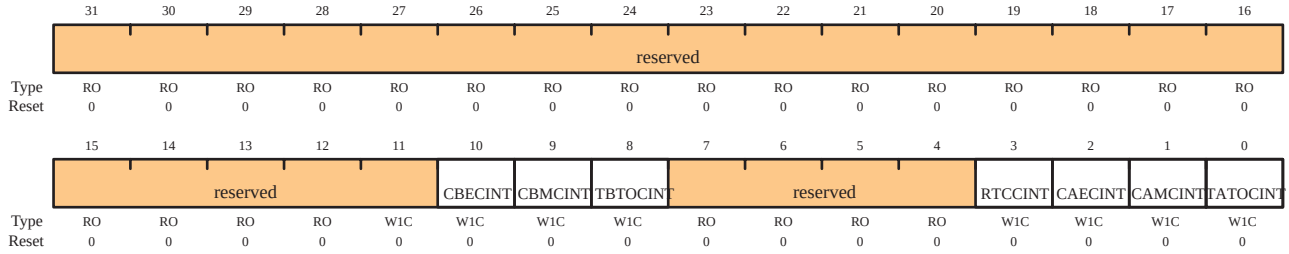



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 10 | CBEMIS | RO | 0 | GPTM CaptureB Event Masked Interrupt<br>This is the CaptureB event interrupt status after masking. |
| 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 | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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)
Offset 0x024



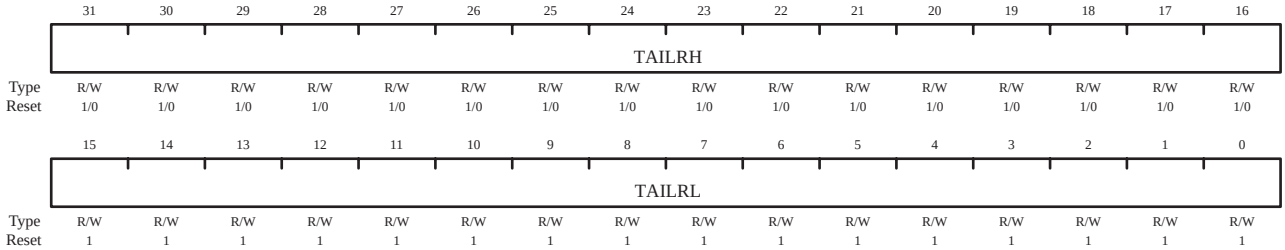

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 10 | CBECINT | W1C | 0 | GPTM CaptureB Event Interrupt Clear<br>0: The interrupt is unaffected.<br>1: The interrupt is cleared. |
| 9 | CBMCINT | W1C | 0 | GPTM CaptureB Match Interrupt Clear<br>0: The interrupt is unaffected.<br>1: The interrupt is cleared. |
| 8 | TBTOCINT | W1C | 0 | GPTM TimerB Time-Out Interrupt Clear<br>0: The interrupt is unaffected.<br>1: The interrupt is cleared. |
| 7:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | RTCCINT | W1C | 0 | GPTM RTC Interrupt Clear<br>0: The interrupt is unaffected.<br>1: The interrupt is cleared. |
| 2 | CAECINT | W1C | 0 | GPTM CaptureA Event Interrupt Clear<br>0: The interrupt is unaffected.<br>1: The interrupt is cleared. |
| 1 | CAMCINT | W1C | 0 | GPTM CaptureA Match Raw Interrupt<br>This is the CaptureA match interrupt status after masking. |
| 0 | TATOCINT | W1C | 0 | GPTM TimerA Time-Out Raw Interrupt<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)

Offset 0x028



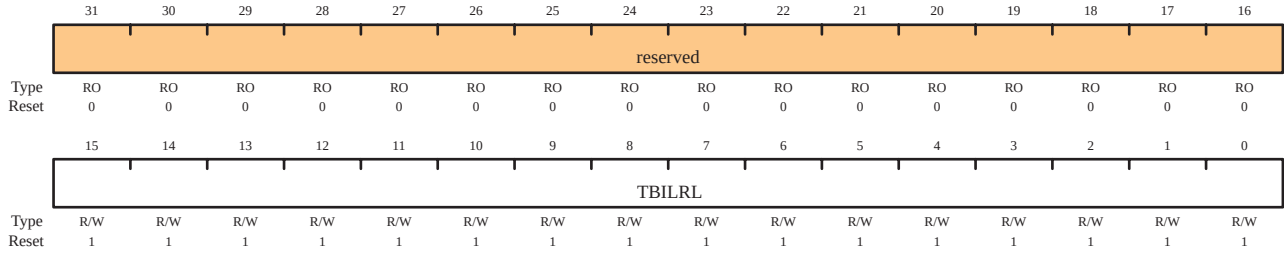

1/0 = 1 if timer is configured in 32-bit mode; 0 if timer is configured in 16-bit mode.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | TAILRH | R/W | 0xFFFF (32-bit mode)<br><br>0x0000 (16-bit mode) | GPTM TimerA Interval Load Register High<br><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><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><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)
Offset 0x02C



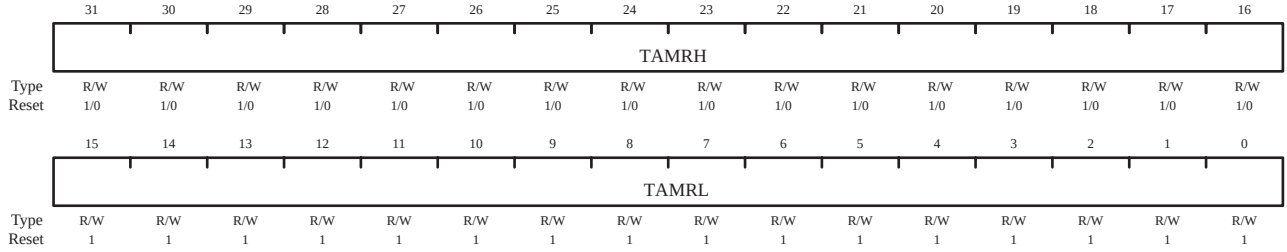

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:0 | TBILRL | R/W | 0xFFFF | GPTM TimerB Interval Load Register<br>When the GPTM is not configured as a 32-bit timer, a 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)

Offset 0x030




1/0 = 1 if timer is configured in 32-bit mode; 0 if timer is configured in 16-bit mode.

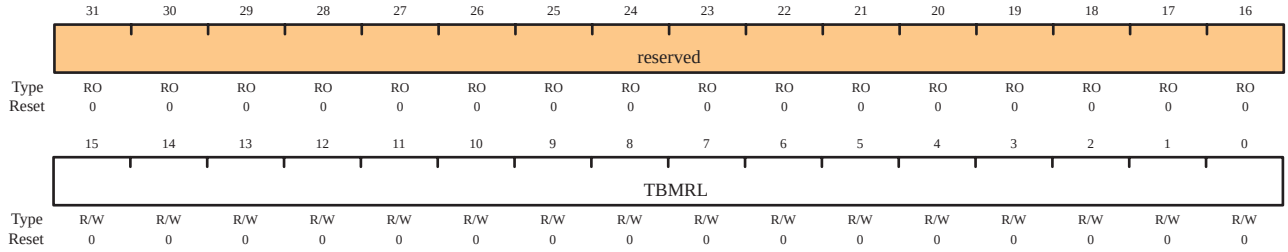
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | TAMRH | R/W | 0xFFFF (32-bit mode)<br><br>0x0000 (16-bit mode) | GPTM TimerA Match Register High<br><br>When configured for 32-bit Real-Time Clock (RTC) mode via the **GPTMCFG** register, this value is compared to the upper half of **GPTMTAR**, to determine match events.<br><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><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><br>When configured for PWM mode, this value along with **GPTMTAILR**, determines the duty cycle of the output PWM signal.<br><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 32-bit Real-Time Clock mode and 16-bit PWM and Input Edge Count modes.

GPTM TimerB Match (GPTMTBMATCHR)

Offset 0x034



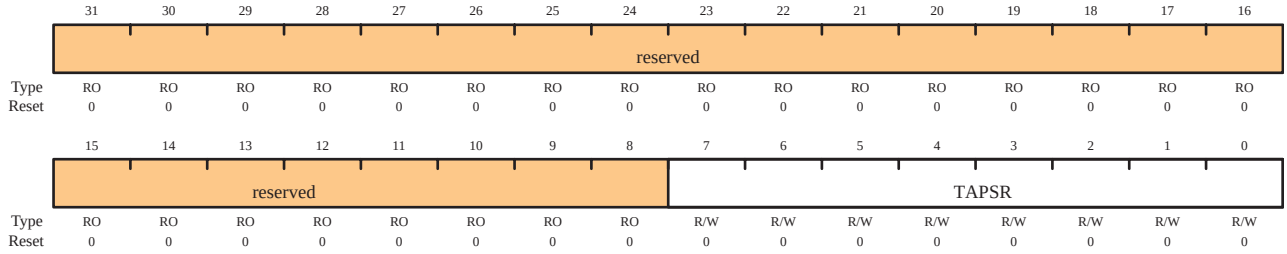

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:0 | TBMRL | R/W | 0xFFFF | GPTM TimerB Match Register Low<br><br>When configured for PWM mode, this value along with **GPTMTBILR**, determines the duty cycle of the output PWM signal.<br><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.

GPTM TimerA Prescale (GPTMTAPR)

Offset 0x038



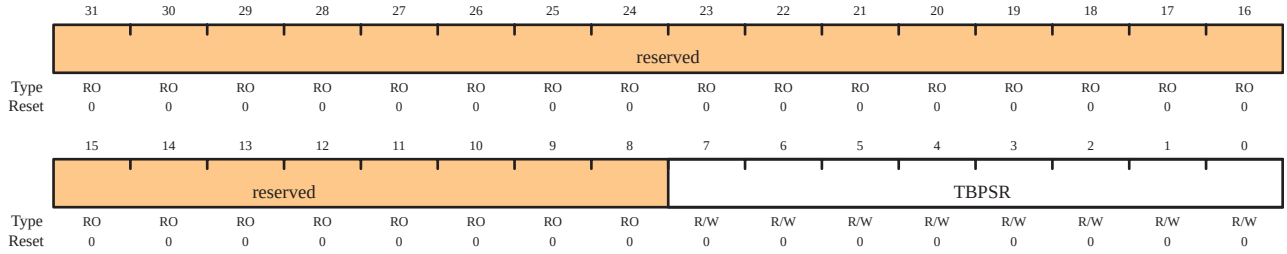

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | TAPSR | R/W | 0 | GPTM TimerA Prescale<br>The register loads this value on a write. A read returns the current value of the register.<br>Refer to Table 9-1 on page 150 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.

GPTM TimerB Prescale (GPTMTBPR)

Offset 0x03C



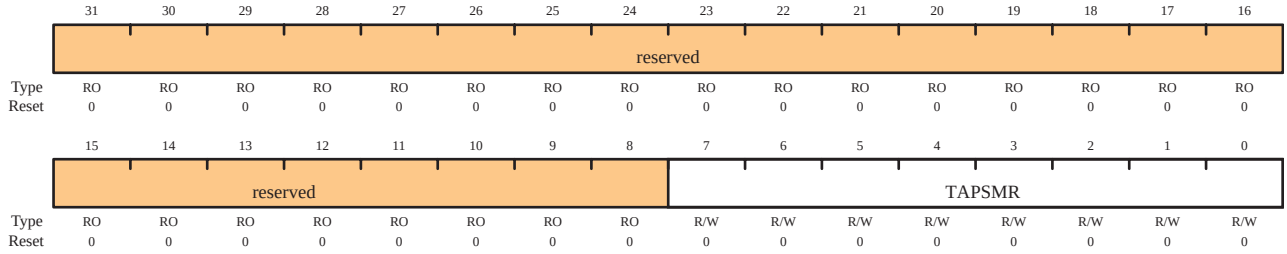

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | TBPSR | R/W | 0 | GPTM TimerB Prescale<br>The register loads this value on a write. A read returns the current value of this register.<br>Refer to Table 9-1 on page 150 for more details and an example. |

### Register 15: GPTM TimerA Prescale Match (GPTMTAPMR), offset 0x040

This register effectively extends the range of **GPTMTAMATCHR** to 24 bits.

GPTM TimerA Prescale Match (GPTMTAPMR)

Offset 0x040



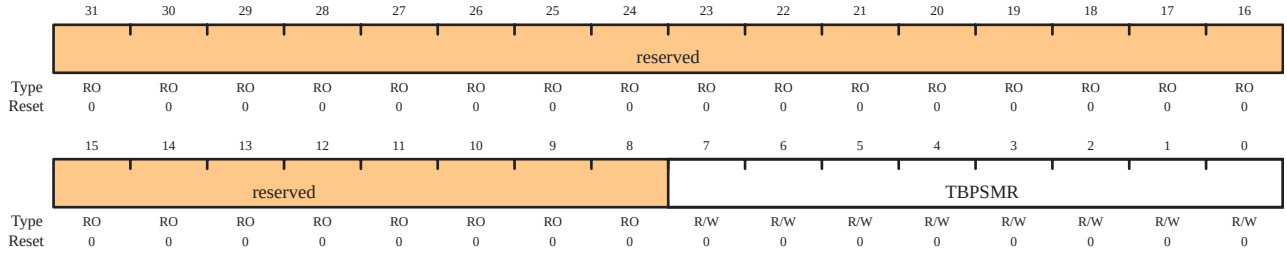

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | TAPSMR | R/W | 0 | GPTM TimerA Prescale Match<br>This value is used alongside **GPTMTAMATCHR** to detect timer match events while using a prescaler. |

### Register 16: GPTM TimerB Prescale Match (GPTMTBPMR), offset 0x044

This register effectively extends the range of **GPTMTBMATCHR** to 24 bits.

GPTM TimerB Prescale Match (GPTMTBPMR)

Offset 0x044




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | TBPSMR | R/W | 0 | GPTM TimerB Prescale Match<br>This value is used alongside **GPTMTBMATCHR** to detect timer match events while using a prescaler. |

### Register 17: 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 time at which the last edge event took place.

GPTM TimerA (GPTMTAR)

Offset 0x048

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TARH | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 | 1/0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TARL | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

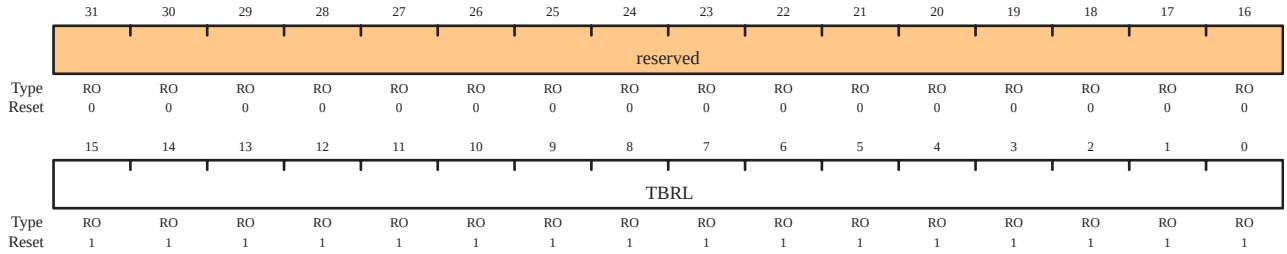
1/0 = 1 if timer is configured in 32-bit mode; 0 if timer is configured in 16-bit mode.

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | TARH | RO | 0xFFFF (32-bit mode)<br><br>0x0000 (16-bit mode) | GPTM TimerA Register High<br><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><br>A read returns the current value of the **GPTM TimerA Count Register**, except in Input Edge Count mode, when it returns the timestamp from the last edge event. |

### Register 18: 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 time at which the last edge event took place.

GPTM TimerB (GPTMTBR)
Offset 0x04C




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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 timestamp from the last edge event. |

# 10 Watchdog Timer

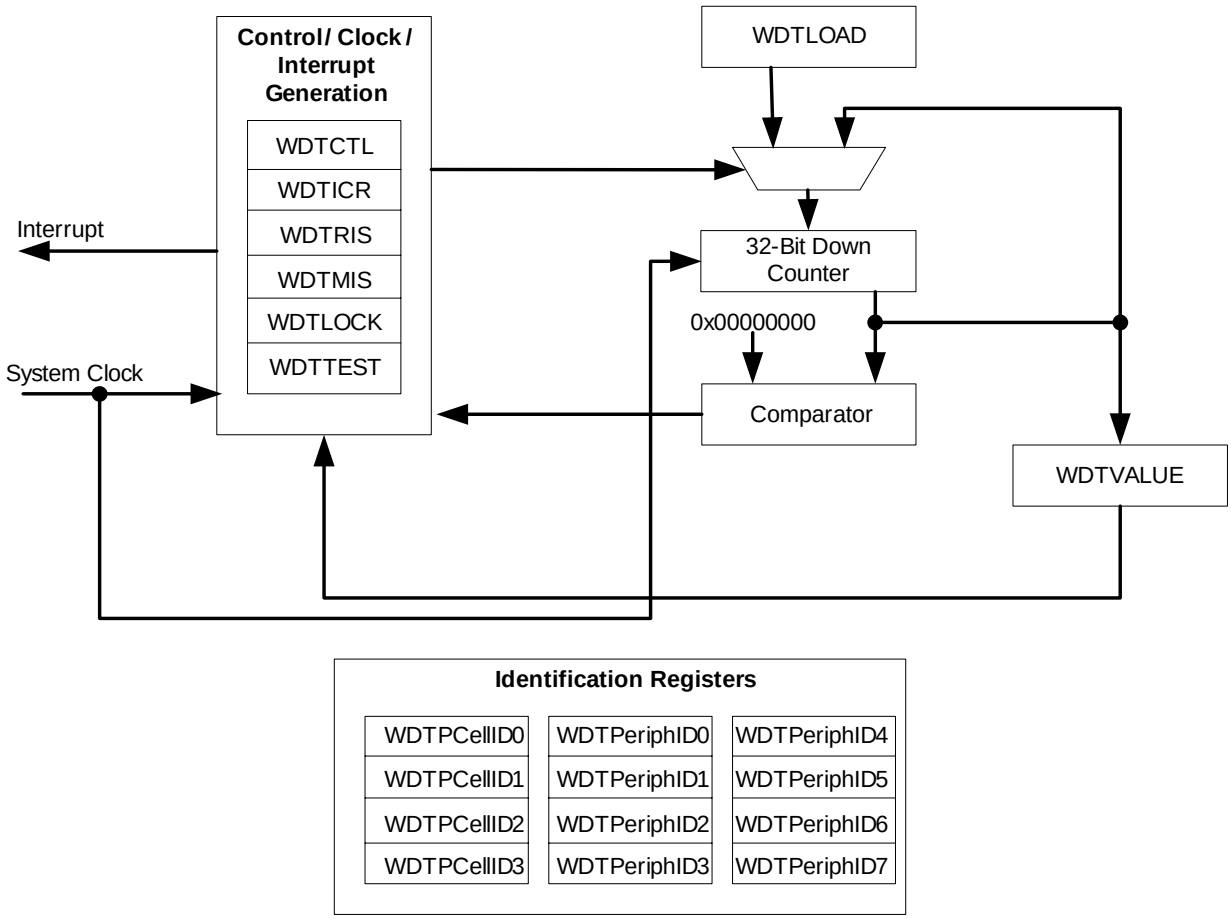
A watchdog timer can generate nonmaskable interrupts (NMIs) or a reset when a time-out value is reached. The watchdog timer is used to regain control when a system has failed due to a software error or due to the failure of an external device to respond in the expected way.

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

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.

## 10.1 Block Diagram

**Figure 10-1. WDT Module Block Diagram**



## 10.2 Functional Description

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

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.

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.

## 10.3 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 0x1ACCE551.

## 10.4 Register Map

Table 10-1 lists the Watchdog registers. The offset listed is a hexadecimal increment to the register's address, relative to the Watchdog Timer base address of 0x40000000.

**Table 10-1. WDT Register Map**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x000 | WDTLOAD | 0xFFFFFFFF | R/W | Load | 181 |
| 0x004 | WDTVALUE | 0xFFFFFFFF | RO | Current value | 182 |
| 0x008 | WDTCTL | 0x00000000 | R/W | Control | 183 |

**Table 10-1. WDT Register Map (Continued)**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x00C | WDTICR | - | WO | Interrupt clear | 184 |
| 0x010 | WDTRIS | 0x00000000 | RO | Raw interrupt status | 185 |
| 0x014 | WDTMIS | 0x00000000 | RO | Masked interrupt status | 186 |
| 0x418 | WDTTEST | 0x00000000 | R/W | Watchdog stall enable | 188 |
| 0xC00 | WDTLOCK | 0x00000000 | R/W | Lock | 187 |
| 0xFD0 | WDTPeriphID4 | 0x00000000 | RO | Peripheral identification 4 | 189 |
| 0xFD4 | WDTPeriphID5 | 0x00000000 | RO | Peripheral identification 5 | 190 |
| 0xFD8 | WDTPeriphID6 | 0x00000000 | RO | Peripheral identification 6 | 191 |
| 0xFDC | WDTPeriphID7 | 0x00000000 | RO | Peripheral identification 7 | 192 |
| 0xFE0 | WDTPeriphID0 | 0x00000005 | RO | Peripheral identification 0 | 193 |
| 0xFE4 | WDTPeriphID1 | 0x00000018 | RO | Peripheral identification 1 | 194 |
| 0xFE8 | WDTPeriphID2 | 0x00000018 | RO | Peripheral identification 2 | 195 |
| 0xFEC | WDTPeriphID3 | 0x00000001 | RO | Peripheral identification 3 | 196 |
| 0xFF0 | WDTPCellID0 | 0x0000000D | RO | PrimeCell identification 0 | 197 |
| 0xFF4 | WDTPCellID1 | 0x000000F0 | RO | PrimeCell identification 1 | 198 |
| 0xFF8 | WDTPCellID2 | 0x00000005 | RO | PrimeCell identification 2 | 199 |
| 0xFFC | WDTPCellID3 | 0x000000B1 | RO | PrimeCell identification 3 | 200 |

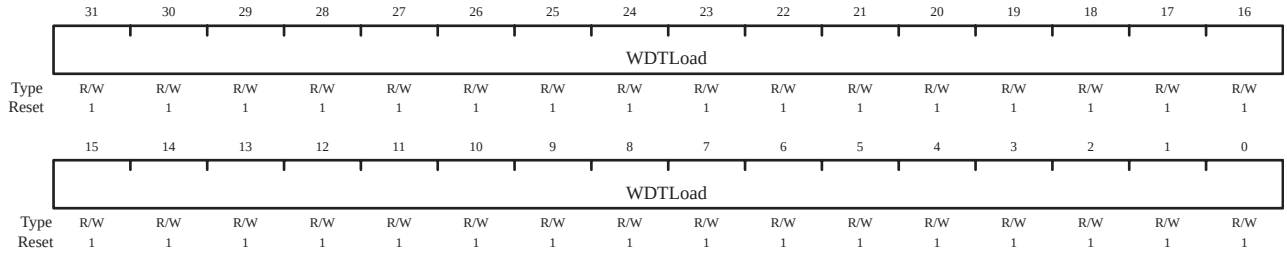
## 10.5 Register Descriptions

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 0x00000000, an interrupt is immediately generated.

Watchdog Load (WDTLOAD)
Offset 0x000

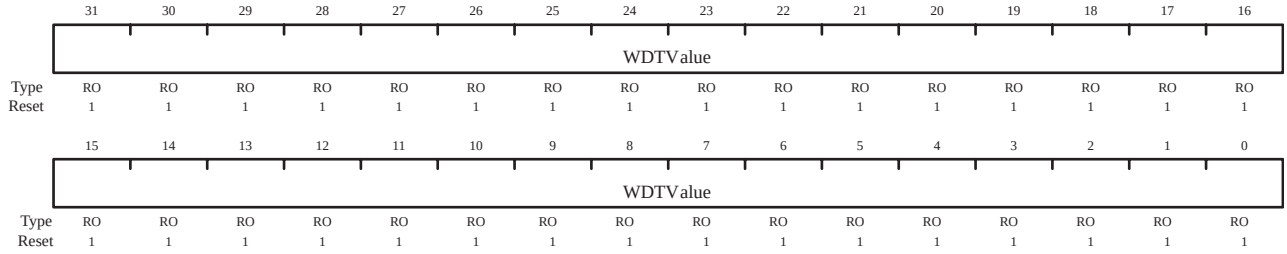



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTLoad | R/W | 0xFFFFFFFF | Watchdog Load Value |

### Register 2: Watchdog Value (WDTVALUE), offset 0x004

This register contains the current count value of the timer.

Watchdog Value (WDTVALUE)
Offset 0x004




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTValue | RO | 0xFFFFFFFF | Watchdog Value<br>Current value of the 32-bit down counter. |

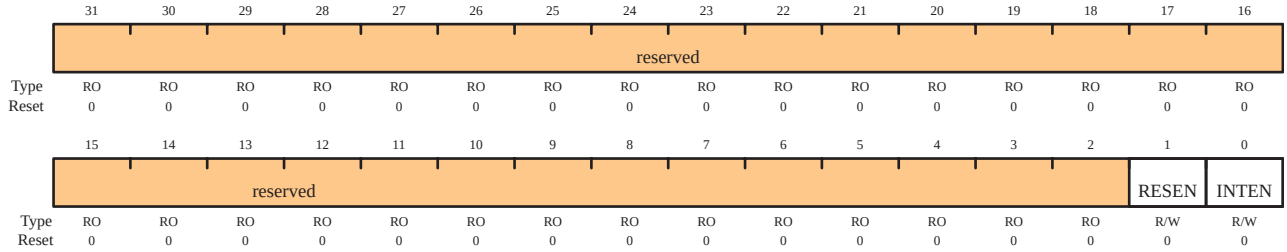
### Register 3: Watchdog Control (WDTCTL), offset 0x008

This register is the watchdog control register. The watchdog timer can be configured to generate a reset signal (upon 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)

Offset 0x008

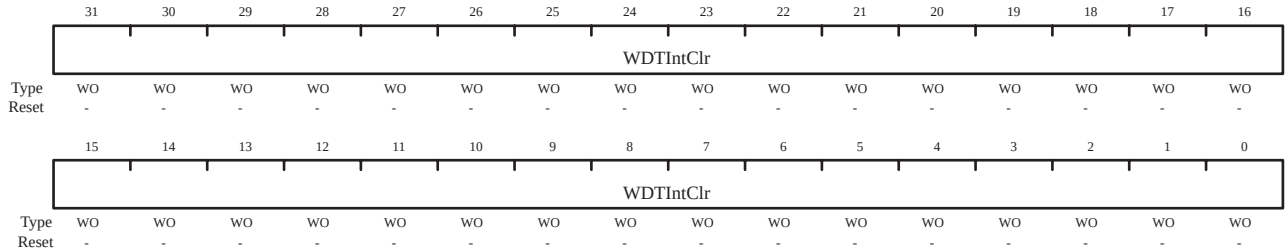



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | RESEN | R/W | 0 | Watchdog Reset Enable<br>0: Disabled.<br>1: Enable the Watchdog module reset output. |
| 0 | INTEN | R/W | 0 | Watchdog Interrupt Enable<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)
Offset 0x00C

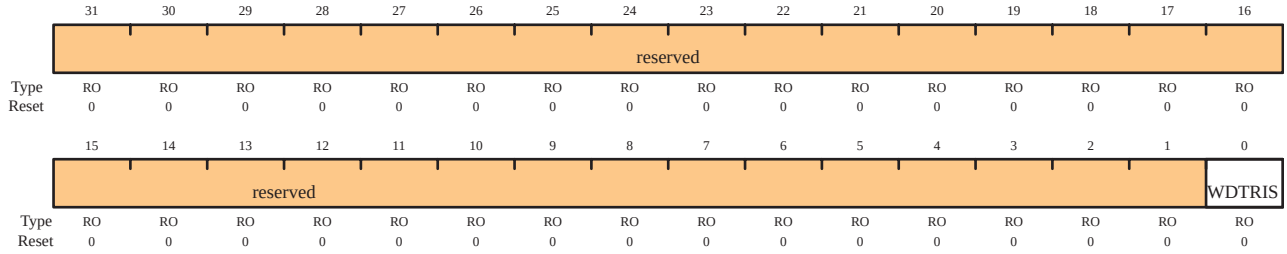



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTIntClr | WO | - | Watchdog Interrupt Clear |

### 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)
Offset 0x010

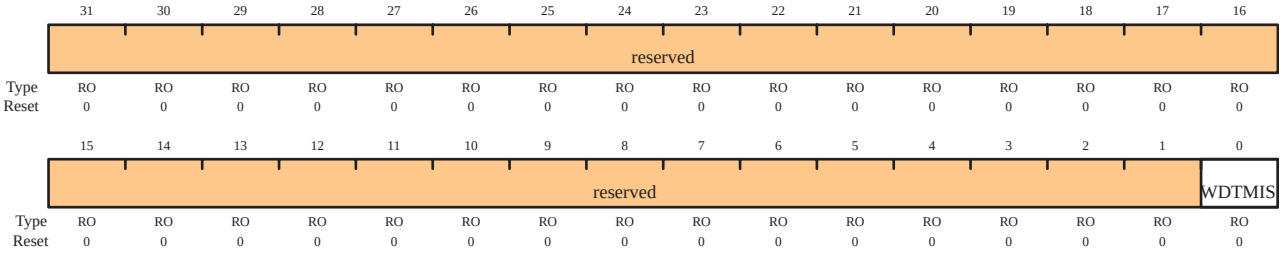



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | WDTRIS | RO | 0 | Watchdog Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of **WDTINTR**. |

### 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)
Offset 0x014

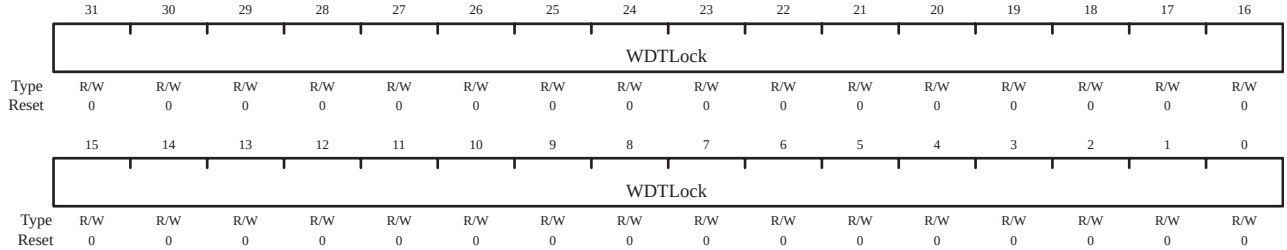



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | WDTMIS | RO | 0 | Watchdog Masked Interrupt Status<br>Gives the masked interrupt state (after masking) of the **WDTINTR** interrupt. |

### Register 7: Watchdog Lock (WDTLOCK), offset 0xC00

Writing 0x1ACCE551 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 0x00000001 (when locked; otherwise, the returned value is 0x00000000 (unlocked)).

Watchdog Lock (WDTLOCK)
Offset 0xC00

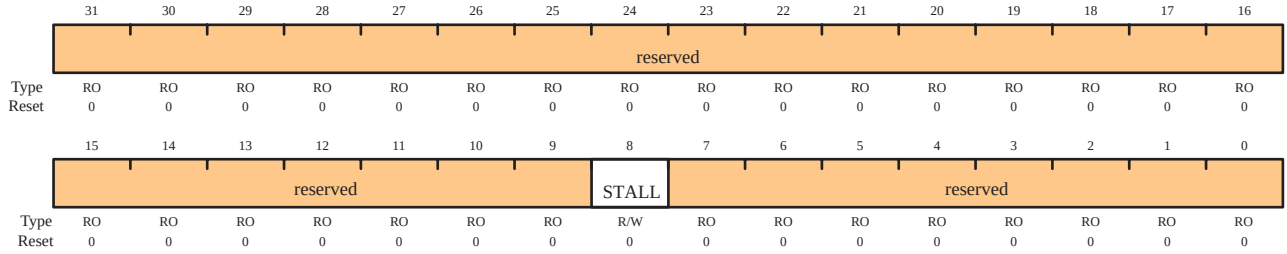



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTLock | R/W | 0x0000 | Watchdog Lock<br>A write of the value 0x1ACCE551 unlocks the watchdog registers for write access. A write of any other value reapplies the lock, preventing any register updates.<br>A read of this register returns the following values:<br>Locked: 0x00000001<br>Unlocked: 0x00000000 |

### Register 8: Watchdog Test (WDTTEST), offset 0x418

This register provides user-enabled stalling when the microcontroller asserts the CPU halt flag during debug.

Watchdog Test (WDTTEST)
Offset 0x418



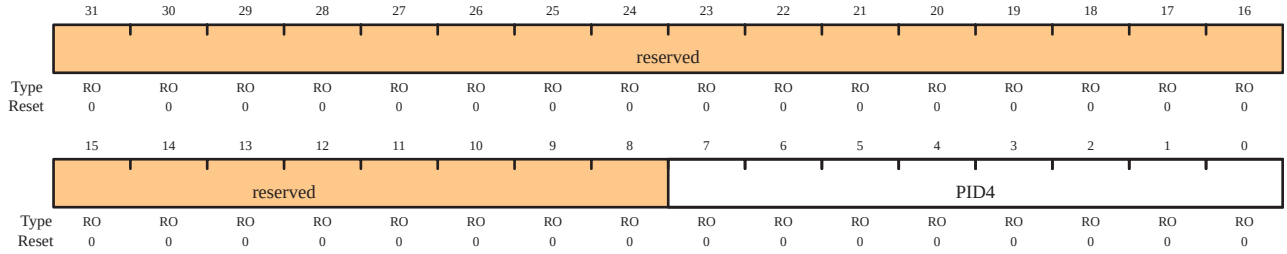

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:9 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 8 | STALL | R/W | 0 | Watchdog Stall Enable<br>When set to 1, if the Stellaris microcontroller is stopped with a debugger, the watchdog timer stops counting. Once the microcontroller is restarted, the watchdog timer resumes counting. |
| 7:0 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |

### 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)

Offset 0xFD0

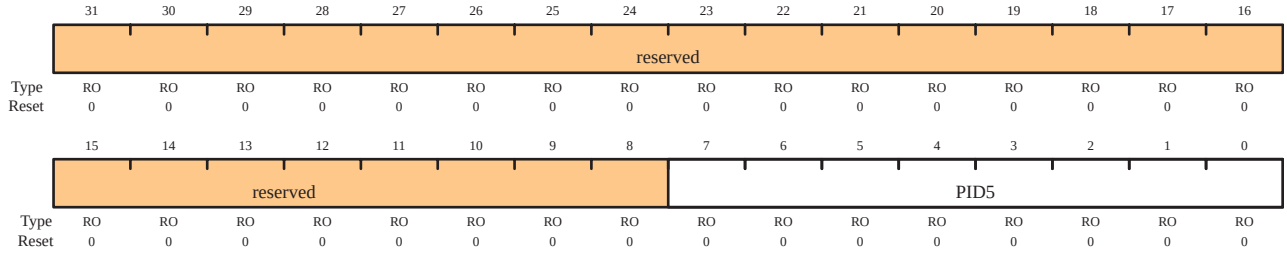



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID4 | RO | 0x00 | WDT Peripheral ID Register[7:0] |

### 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)
Offset 0xFD4

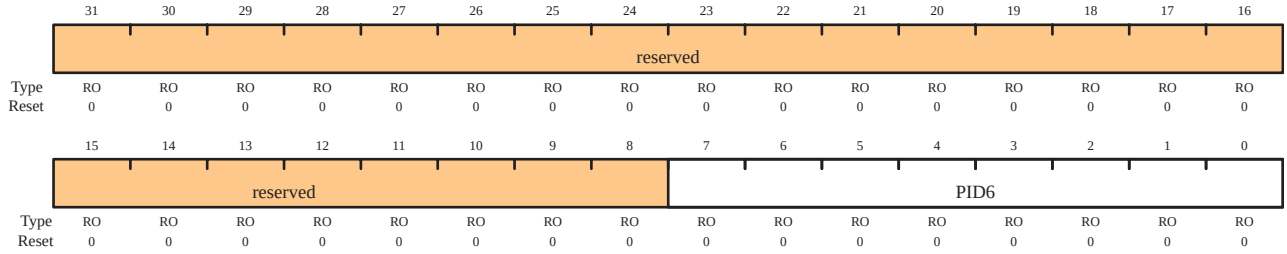



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID5 | RO | 0x00 | WDT Peripheral ID Register[15:8] |

### 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)
Offset 0xFD8



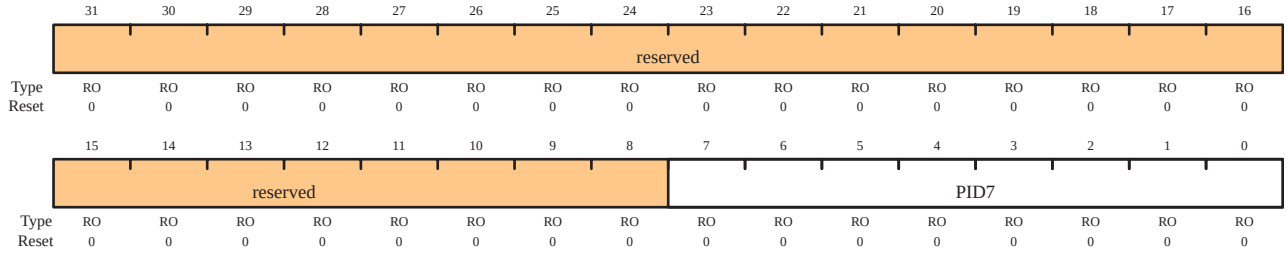

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID6 | RO | 0x00 | WDT Peripheral ID Register[23:16] |

### 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)

Offset 0xFDC

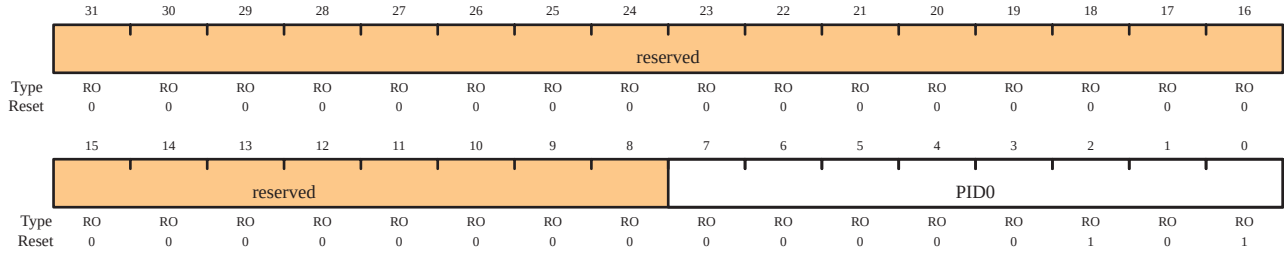



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID7 | RO | 0x00 | WDT Peripheral ID Register[31:24] |

### 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)
Offset 0xFE0

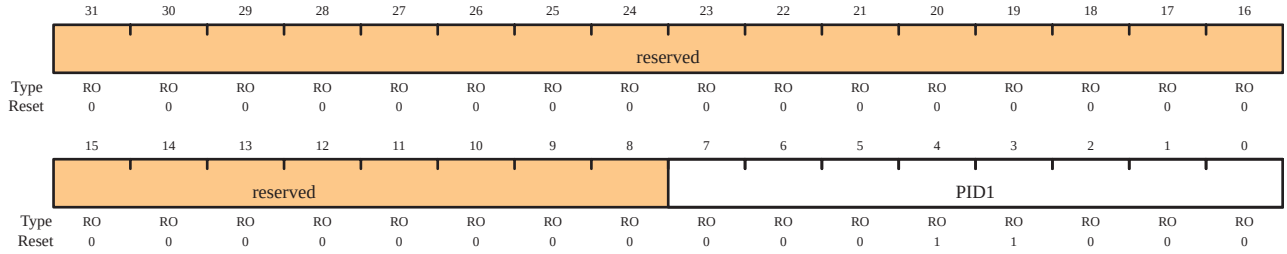



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID0 | RO | 0x05 | Watchdog Peripheral ID Register[7:0] |

### 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)
Offset 0xFE4

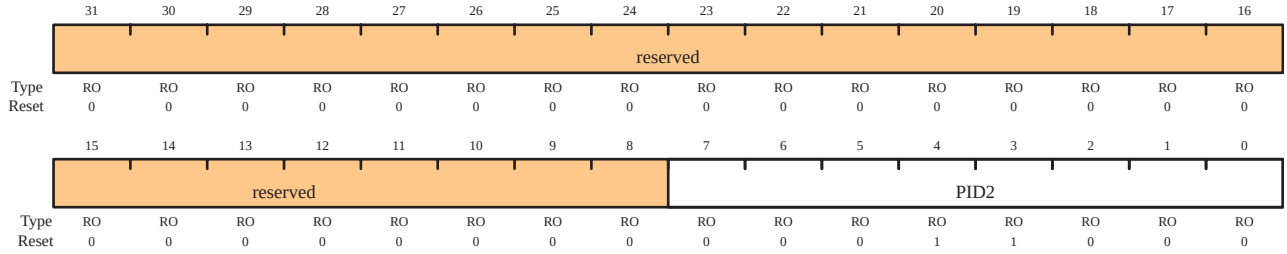



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID1 | RO | 0x18 | Watchdog Peripheral ID Register[15:8] |

### 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)
Offset 0xFE8



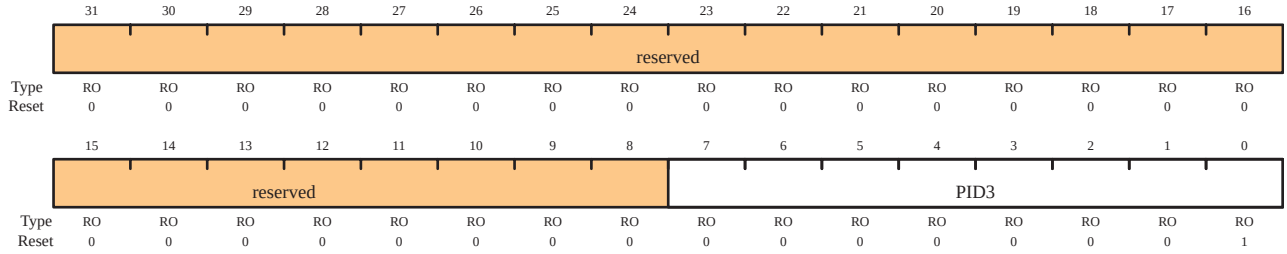

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID2 | RO | 0x18 | Watchdog Peripheral ID Register[23:16] |

### 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)

Offset 0xFEC

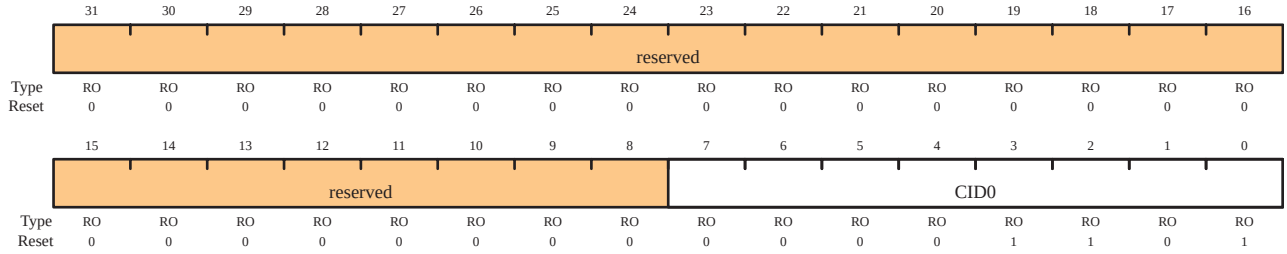



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID3 | RO | 0x01 | Watchdog Peripheral ID Register[31:24] |

### 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)
Offset 0xFF0

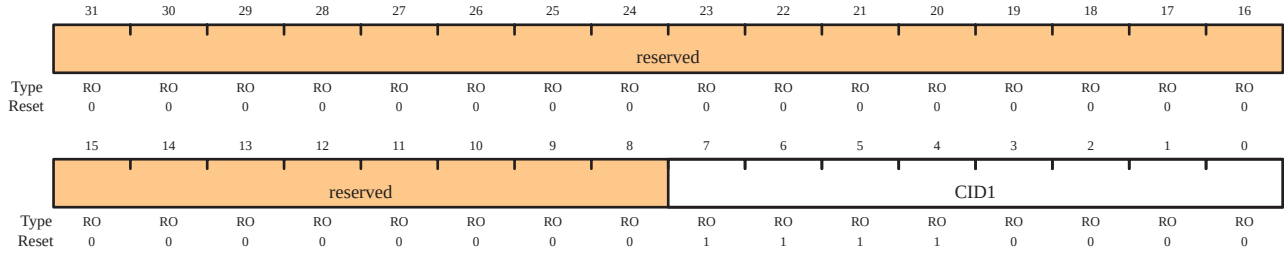



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID0 | RO | 0x0D | Watchdog PrimeCell ID Register[7:0] |

### 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)
Offset 0xFF4

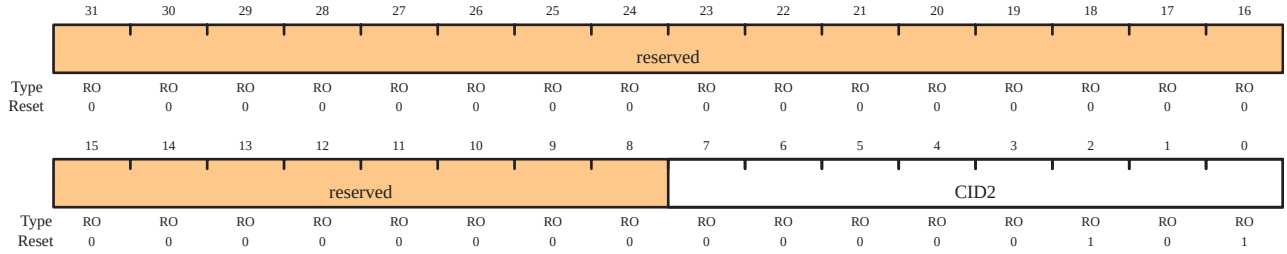



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID1 | RO | 0xF0 | Watchdog PrimeCell ID Register[15:8] |

### 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)
Offset 0xFF8

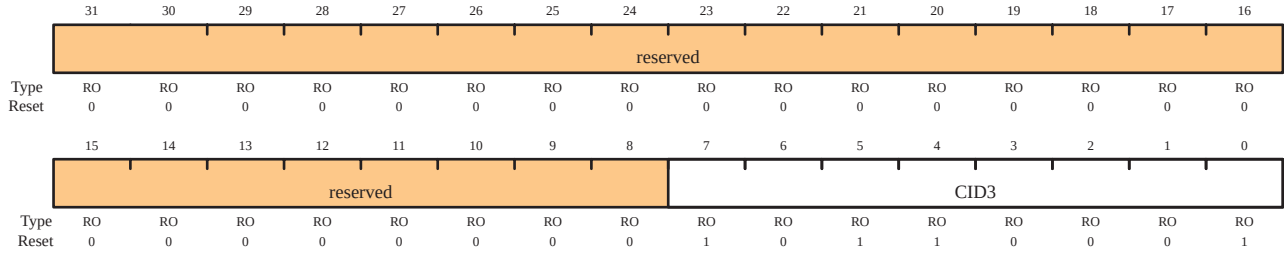



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID2 | RO | 0x05 | Watchdog PrimeCell ID Register[23:16] |

### 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)
Offset 0xFFC




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID3 | RO | 0xB1 | Watchdog PrimeCell ID Register[31:24] |

# 11 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.

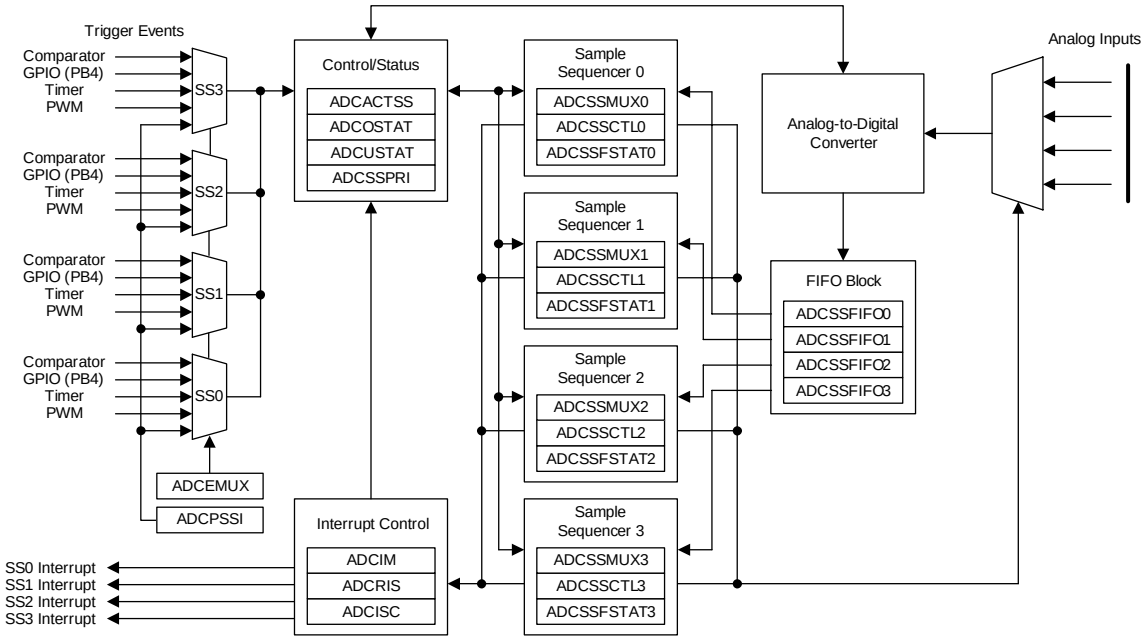
The Stellaris ADC module features 10-bit conversion resolution and supports four input channels, plus an internal temperature sensor. The ADC module contains a programmable sequencer which allows for the sampling of multiple analog input sources without controller intervention. Each sample sequence provides flexible programming with fully configurable input source, trigger events, interrupt generation, and sequence priority.

The Stellaris ADC provides the following features:

- Four analog input channels
- Single-ended and differential-input configurations
- Internal temperature sensor
- Sample rate of 250 thousand samples/second
- Four programmable sample conversion sequences from one to eight entries long, with corresponding conversion result FIFOs
- Flexible trigger control
  - Controller (software)
  - Timers
  - Analog Comparators
  - PWM
  - GPIO
- Hardware averaging of up to 64 samples for improved accuracy

## 11.1 Block Diagram

**Figure 11-1. ADC Module Block Diagram**




## 11.2 Functional Description

The Stellaris ADC collects sample data by using a programmable sequence-based approach instead of the traditional single or double-sampling approach 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.

### 11.2.1 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 11-1 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 11-1. 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, but can 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 Interrupt Enable (IE) 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.

### 11.2.2 Module Control

Outside of the Sample Sequencers, the remainder of the control logic is responsible for tasks such as interrupt generation, sequence prioritization, and 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.

#### 11.2.2.1 Interrupts

The Sample Sequencers dictate the events that cause interrupts, but they don't have control over whether the interrupt is actually sent to the interrupt controller. The ADC module's interrupt signal is 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 a Sample Sequencer's interrupt signal, and the **ADC Interrupt Status and Clear (ADCISC)** register, which shows the logical AND of the **ADCRIS** register's INR bit and the **ADCIM** register's MASK bits. Interrupts are cleared by writing a 1 to the corresponding IN bit in **ADCISC**.

#### 11.2.2.2 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.

#### 11.2.2.3 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 CH bits in the **ADC Processor Sample Sequence Initiate (ADCPSSI)** register.

When using the "Always" trigger, care must be taken. If a sequence's priority is too high, it is possible to starve other lower priority sequences.

### 11.2.3 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 216). There is a single averaging circuit and all input channels receive the same amount of averaging whether they are single-ended or differential.

### 11.2.4 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.

### 11.2.5 Test Modes

There is a user-available test mode that allows for loopback operation within the digital portion of the ADC module. This can be useful for debugging software without having to provide actual analog stimulus. This mode is available through the **ADC Test Mode Loopback (ADCTMLB)** register (see page 229).

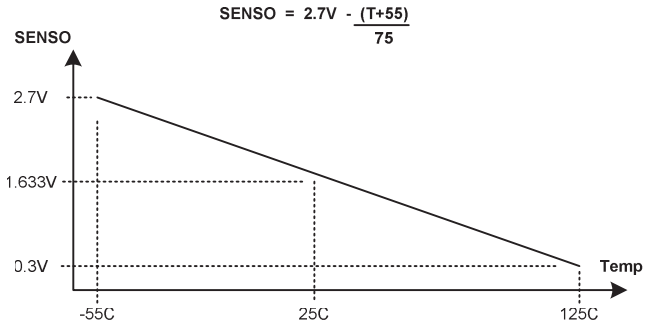
### 11.2.6 Internal Temperature Sensor

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 11-2 on page 204.

**Figure 11-2. Internal Temperature Sensor Characteristic**




## 11.3 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 on page 80). Using unsupported frequencies can cause faulty operation in the ADC module.

### 11.3.1 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 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 0x00010000 to the **RCGC1** register in the System Control module.
2. 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.

### 11.3.2 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 ASEN 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 ASEN bit in the **ADCACTSS** register.

## 11.4 Register Map

Table 11-2 lists the ADC registers. The offset listed is a hexadecimal increment to the register's address, relative to the ADC base address of 0x40038000.

**Table 11-2. ADC Register Map**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x000 | ADCACTSS | 0x00000000 | R/W | Active sample sequencer | 207 |
| 0x004 | ADCRIS | 0x00000000 | RO | Raw interrupt status and clear | 208 |
| 0x008 | ADCIM | 0x00000000 | R/W | Interrupt mask | 209 |
| 0x00C | ADCISC | 0x00000000 | R/W1C | Interrupt status and clear | 210 |
| 0x010 | ADCOSTAT | 0x00000000 | R/W1C | Overflow status | 211 |
| 0x014 | ADCEMUX | 0x00000000 | R/W | Event multiplexer select | 212 |
| 0x018 | ADCUSTAT | 0x00000000 | R/W1C | Underflow status | 213 |

**Table 11-2. ADC Register Map (Continued)**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x020 | ADCSSPRI | 0x00003210 | R/W | Sample sequencer priority | 214 |
| 0x028 | ADCPSSI | - | WO | Processor sample sequence initiate | 215 |
| 0x030 | ADCSAC | 0x00000000 | R/W | Sample averaging control | 216 |
| 0x040 | ADCSSMUX0 | 0x00000000 | R/W | Sample sequence input multiplexer select 0 | 217 |
| 0x044 | ADCSSCTL0 | 0x00000000 | R/W | Sample sequence control 0 | 219 |
| 0x048 | ADCSSFIFO0 | 0x00000000 | RO | Sample sequence result FIFO 0 | 221 |
| 0x04C | ADCSSFSTAT0 | 0x00000100 | RO | Sample sequence FIFO 0 status | 222 |
| 0x060 | ADCSSMUX1 | 0x00000000 | R/W | Sample sequence input multiplexer select 1 | 223 |
| 0x064 | ADCSSCTL1 | 0x00000000 | R/W | Sample sequence control 1 | 224 |
| 0x068 | ADCSSFIFO1 | 0x00000000 | RO | Sample sequence result FIFO 1 | 224 |
| 0x06C | ADCSSFSTAT1 | 0x00000100 | RO | Sample sequence FIFO 1 status | 224 |
| 0x080 | ADCSSMUX2 | 0x00000000 | R/W | Sample sequence input multiplexer select 2 | 225 |
| 0x084 | ADCSSCTL2 | 0x00000000 | R/W | Sample sequence control 2 | 226 |
| 0x088 | ADCSSFIFO2 | 0x00000000 | RO | Sample sequence result FIFO 2 | 226 |
| 0x08C | ADCSSFSTAT2 | 0x00000100 | RO | Sample sequence FIFO 2 status | 226 |
| 0x0A0 | ADCSSMUX3 | 0x00000000 | R/W | Sample sequence input multiplexer select 3 | 227 |
| 0x064 | ADCSSCTL3 | 0x00000002 | R/W | Sample sequence control 3 | 228 |
| 0x0A8 | ADCSSFIFO3 | 0x00000000 | RO | Sample sequence result FIFO 3 | 228 |
| 0x0AC | ADCSSFSTAT3 | 0x00000100 | RO | Sample sequence FIFO 3 status | 228 |
| 0x100 | ADCTMLB | 0x00000000 | R/W | Test mode loopback | 229 |

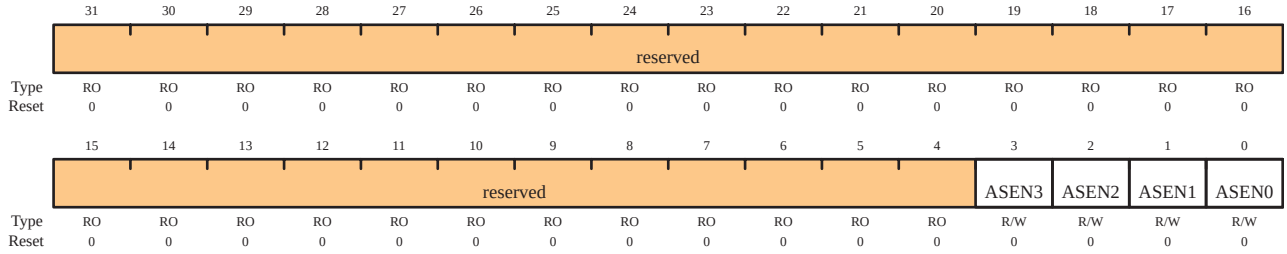
## 11.5 Register Descriptions

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/disabled independently.

ADC Active Sample Sequencer (ADCACTSS)
Offset 0x000

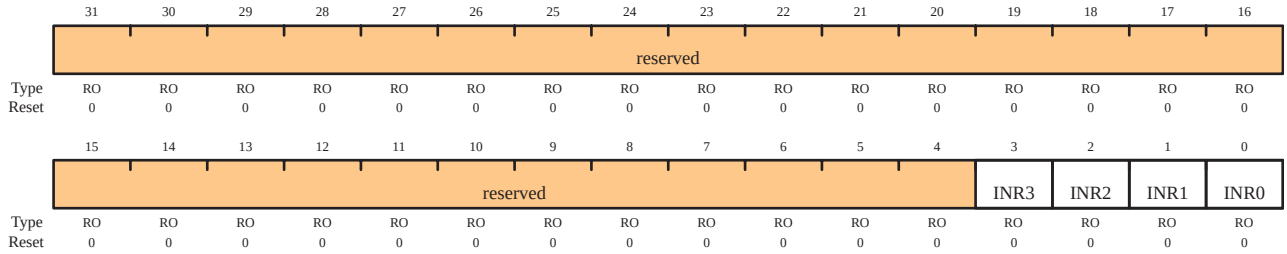



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | ASEN3 | R/W | 0 | Specifies whether Sample Sequencer 3 is enabled. If set, the sample sequence logic for Sequencer 3 is active. Otherwise, the Sequencer is inactive. |
| 2 | ASEN2 | R/W | 0 | 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 | 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 | 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)
Offset 0x004

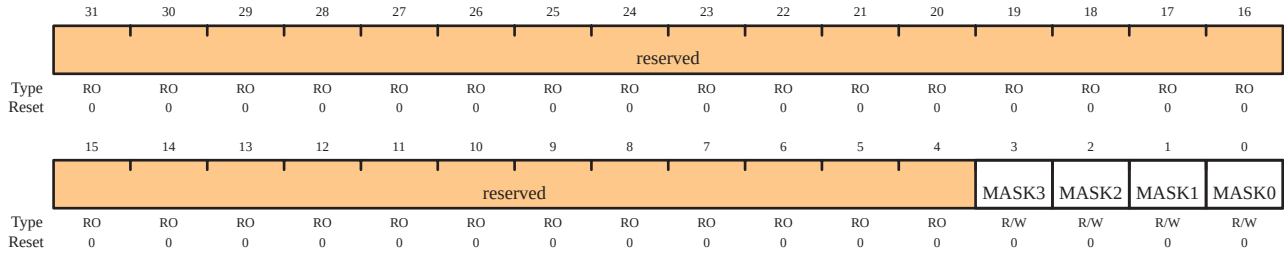



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | INR3 | RO | 0 | Set by hardware when a sample with its respective **ADCSSCTL3** `IE` bit has completed conversion. This bit is cleared by writing a 1 to the **ADCISC** `IN3` bit. |
| 2 | INR2 | RO | 0 | Set by hardware when a sample with its respective **ADCSSCTL2** `IE` bit has completed conversion. This bit is cleared by writing a 1 to the **ADCISC** `IN2` bit. |
| 1 | INR1 | RO | 0 | Set by hardware when a sample with its respective **ADCSSCTL1** `IE` bit has completed conversion. This bit is cleared by writing a 1 to the **ADCISC** `IN1` bit. |
| 0 | INR0 | RO | 0 | Set by hardware when a sample with its respective **ADCSSCTL0** `IE` bit has completed conversion. This bit is cleared by writing a 1 to the **ADCISC** `IN0` bit. |

### Register 3: ADC Interrupt Mask (ADCIM), offset 0x008

This register controls whether the Sample Sequencer raw interrupt signals are promoted to controller interrupts. The raw interrupt signal for each Sample Sequencer can be masked independently.

ADC Interrupt Mask (ADCIM)
Offset 0x008



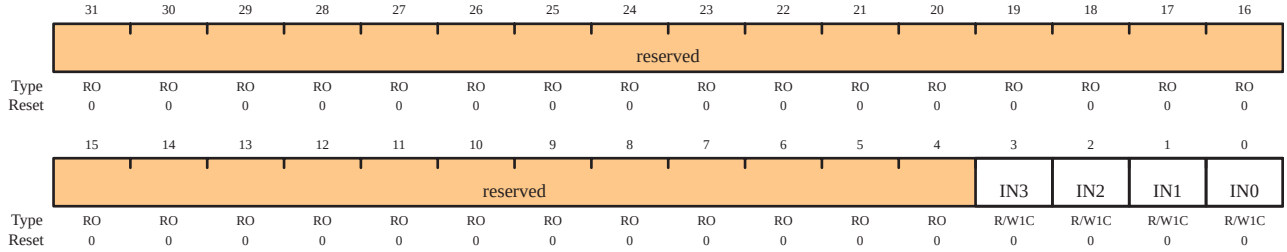

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | MASK3 | R/W | 0 | Specifies whether the raw interrupt signal from Sample Sequencer 3 (**ADCRIS** register `INR3` bit) is promoted to a controller interrupt. If set, the raw interrupt signal is promoted to a controller interrupt. Otherwise, it is not. |
| 2 | MASK2 | R/W | 0 | Specifies whether the raw interrupt signal from Sample Sequencer 2 (**ADCRIS** register `INR2` bit) is promoted to a controller interrupt. If set, the raw interrupt signal is promoted to a controller interrupt. Otherwise, it is not. |
| 1 | MASK1 | R/W | 0 | Specifies whether the raw interrupt signal from Sample Sequencer 1 (**ADCRIS** register `INR0` bit) is promoted to a controller interrupt. If set, the raw interrupt signal is promoted to a controller interrupt. Otherwise, it is not. |
| 0 | MASK0 | R/W | 0 | Specifies whether the raw interrupt signal from Sample Sequencer 0 (**ADCRIS** register `INR0` bit) is promoted to a controller interrupt. If set, the raw interrupt signal is promoted to a controller interrupt. Otherwise, it is not. |

### Register 4: ADC Interrupt Status and Clear (ADCISC), offset 0x00C

This register provides the mechanism for clearing 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. Interrupts are cleared by writing a 1 to the corresponding bit position. If software is polling the **ADCRIS** instead of generating interrupts, the INR bits are still cleared via the **ADCISC** register, even if the IN bit is not set.

ADC Interrupt Status and Clear (ADCISC)

Offset 0x00C

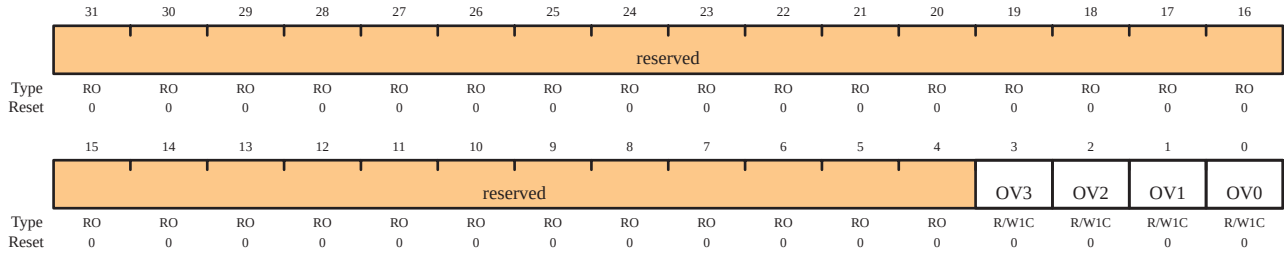



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | IN3 | R/W1C | 0 | This bit is set by hardware when the MASK3 and INR3 bits are both 1, providing a level-based interrupt to the controller. It is cleared by writing a 1, and also clears the INR3 bit. |
| 2 | IN2 | R/W1C | 0 | This bit is set by hardware when the MASK2 and INR2 bits are both 1, providing a level based interrupt to the controller. It is cleared by writing a 1, and also clears the INR2 bit. |
| 1 | IN1 | R/W1C | 0 | This bit is set by hardware when the MASK1 and INR1 bits are both 1, providing a level based interrupt to the controller. It is cleared by writing a 1, and also clears the INR1 bit. |
| 0 | IN0 | R/W1C | 0 | This bit is set by hardware when the MASK0 and INR0 bits are both 1, providing a level based interrupt to the controller. It is cleared by writing a 1, and 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)
Offset 0x010

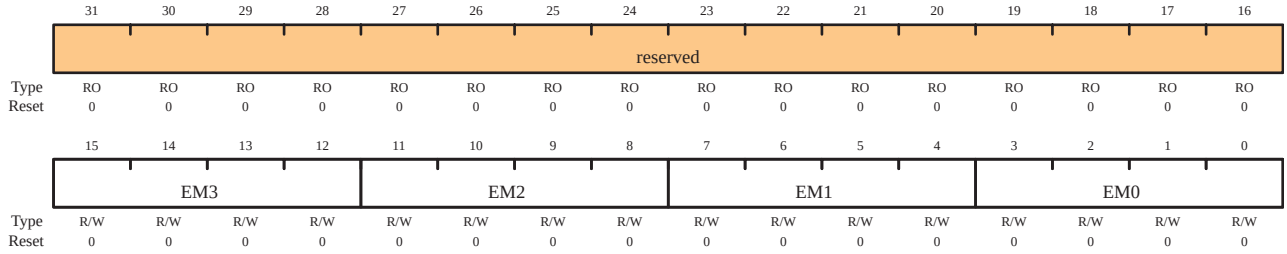



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | OV3 | R/W1C | 0 | This bit specifies that the FIFO for Sample Sequencer 3 has hit an overflow condition where the FIFO is full and a write was requested. When an overflow is detected, the most recent write is dropped and this bit is set by hardware to indicate the occurrence of dropped data. This bit is cleared by writing a 1. |
| 2 | OV2 | R/W1C | 0 | 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 and this bit is set by hardware to indicate the occurrence of dropped data. This bit is cleared by writing a 1. |
| 1 | OV1 | R/W1C | 0 | 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 and this bit is set by hardware to indicate the occurrence of dropped data. This bit is cleared by writing a 1. |
| 0 | OV0 | R/W1C | 0 | 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 and this bit is set by hardware to indicate the occurrence of dropped data. 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)
Offset 0x014

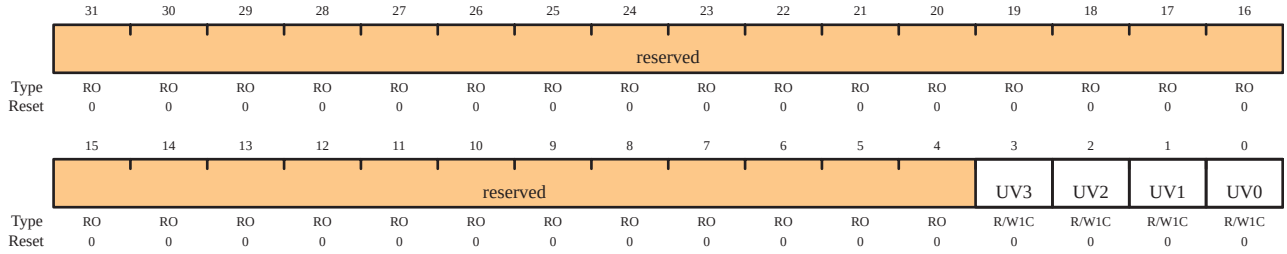



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:12 | EM3 | R/W | 0 | This field selects the trigger source for Sample Sequencer 3.<br>The valid configurations for this field are:<br>EM Binary Value — Event<br>0000 — Controller (default)<br>0001 — Analog Comparator 0<br>0010 — Reserved<br>0011 — Reserved<br>0100 — External (GPIO PB4)<br>0101 — Timer<br>0110 — PWM0<br>0111 — PWM1<br>1000 — Reserved<br>1001-1110 — reserved<br>1111 — Always (continuously sample) |
| 11:8 | EM2 | R/W | 0 | This field selects the trigger source for Sample Sequencer 2. The encodings are the same as those for EM3. |
| 7:4 | EM1 | R/W | 0 | This field selects the trigger source for Sample Sequencer 1. The encodings are the same as those for EM3. |
| 3:0 | EM0 | R/W | 0 | This field selects the trigger source for Sample Sequencer 0. The encodings are the same as those for EM3. |

### Register 7: ADC Underflow Status (ADCUSTAT), offset 0x018

This register indicates underflow conditions in the Sample Sequencer FIFOs. The corresponding underflow condition can be cleared by writing a 1 to the relevant bit position.

ADC Underflow Status (ADCUSTAT)
Offset 0x010

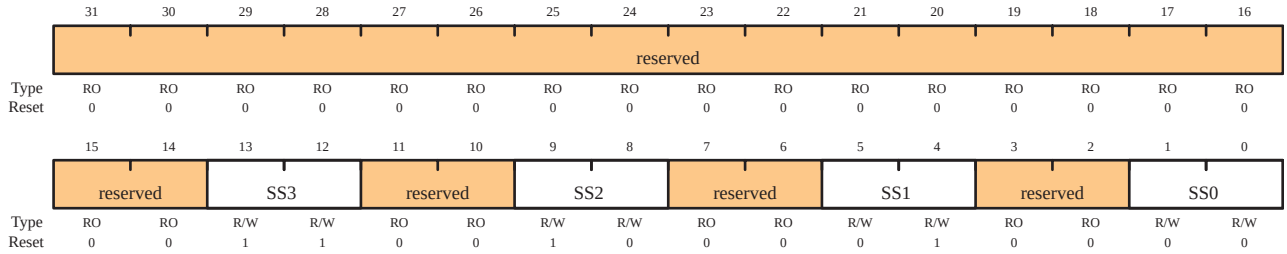



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | UV3 | R/W1C | 0 | This bit specifies that the FIFO for Sample Sequencer 3 has hit an underflow condition where the FIFO is empty and a read was requested. The problematic read does not move the FIFO pointers, and 0s are returned. This bit is cleared by writing a 1. |
| 2 | UV2 | R/W1C | 0 | 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. This bit is cleared by writing a 1. |
| 1 | UV1 | R/W1C | 0 | 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. This bit is cleared by writing a 1. |
| 0 | UV0 | R/W1C | 0 | 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. 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 sample sequence 3 has the lowest priority. When reconfiguring sequence priorities, each sequence must have a unique priority or the ADC behavior is inconsistent.

ADC Sample Sequencer Priority (ADCSSPRI)
Offset 0x020

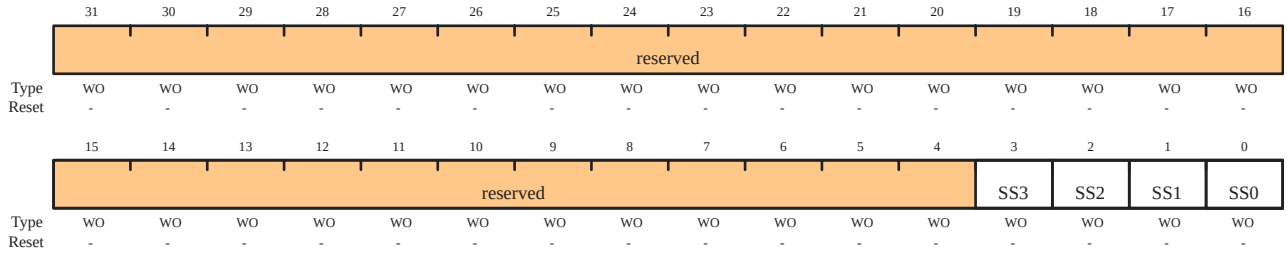



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:14 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 13:12 | SS3 | R/W | 0x3 | The SS3 field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 3. A priority encoding of 0 is highest and 3 is lowest. The priorities assigned to the Sequencers must be uniquely mapped. ADC behavior is not consistent if two or more fields are equal. |
| 11:10 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 9:8 | SS2 | R/W | 0x2 | The SS2 field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 2. |
| 7:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 5:4 | SS1 | R/W | 0x1 | The SS1 field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 1. |
| 3:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1:0 | SS0 | R/W | 0x0 | The SS0 field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 0. |

### 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)
Offset 0x028

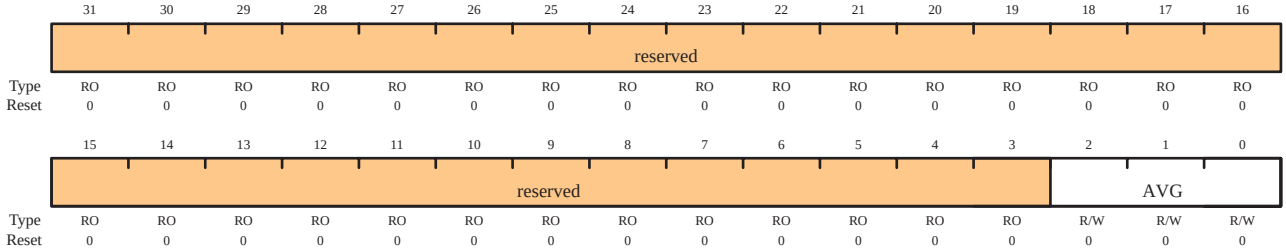



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | WO | - | Only a write by software is valid; a read of the register returns no meaningful data. |
| 3 | SS3 | WO | - | Only a write by software is valid; a read of the register returns no meaningful data. When set by software, sampling is triggered on Sample Sequencer 3, assuming the Sequencer is enabled in the **ADCACTSS** register. |
| 2 | SS2 | WO | - | Only a write by software is valid; a read of the register returns no meaningful data. When set by software, sampling is triggered on Sample Sequencer 2, assuming the Sequencer is enabled in the **ADCACTSS** register. |
| 1 | SS1 | WO | - | Only a write by software is valid; a read of the register returns no meaningful data. When set by software, sampling is triggered on Sample Sequencer 1, assuming the Sequencer is enabled in the **ADCACTSS** register. |
| 0 | SS0 | WO | - | Only a write by software is valid; a read of the register returns no meaningful data. When set by software, sampling is triggered on Sample Sequencer 0, assuming the Sequencer is enabled in the **ADCACTSS** register. |

### 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 is 6, 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)
Offset 0x030




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 2:0 | AVG | R/W | 0 | 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. |

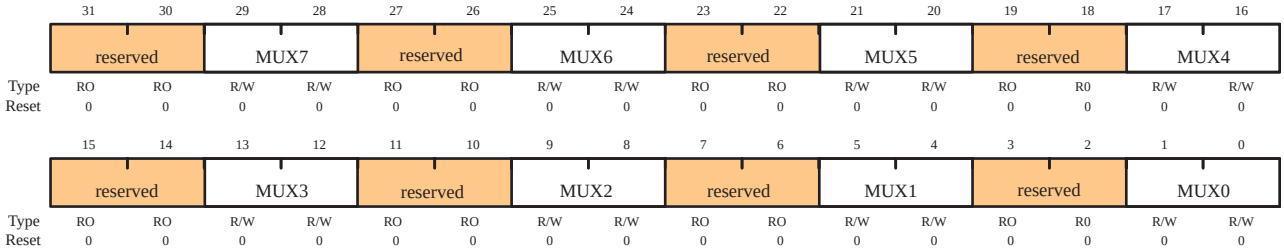
### Register 11: ADC Sample Sequence Input Multiplexer Select 0 (ADCSSMUX0), offset 0x040

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)

Offset 0x040




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 29:28 | MUX7 | R/W | 0 | The `MUX7` field is used during the eighth sample of a sequence executed with Sample Sequencer 0. 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 | Reserved bits return an indeterminate value, and should never be changed. |
| 25:24 | MUX6 | R/W | 0 | The `MUX6` field is used during the seventh sample of a sequence executed with Sample Sequencer 0 and specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 23:22 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 21:20 | MUX5 | R/W | 0 | The `MUX5` field is used during the sixth sample of a sequence executed with Sample Sequencer 0 and specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 19:18 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 17:16 | MUX4 | R/W | 0 | The `MUX4` field is used during the fifth sample of a sequence executed with Sample Sequencer 0 and specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 15:14 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 13:12 | MUX3 | R/W | 0 | The `MUX3` field is used during the fourth sample of a sequence executed with Sample Sequencer 0 and specifies which of the analog inputs is sampled for the analog-to-digital conversion. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 11:10 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 9:8 | MUX2 | R/W | 0 | The MUX2 field is used during the third sample of a sequence executed with Sample Sequencer 0 and specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 7:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 5:4 | MUX1 | R/W | 0 | The MUX1 field is used during the second sample of a sequence executed with Sample Sequencer 0 and specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 3:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1:0 | MUX0 | R/W | 0 | The MUX0 field is used during the first sample of a sequence executed with Sample Sequencer 0 and 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 Sample Sequencer 0. 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)

Offset 0x044

| | 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 |

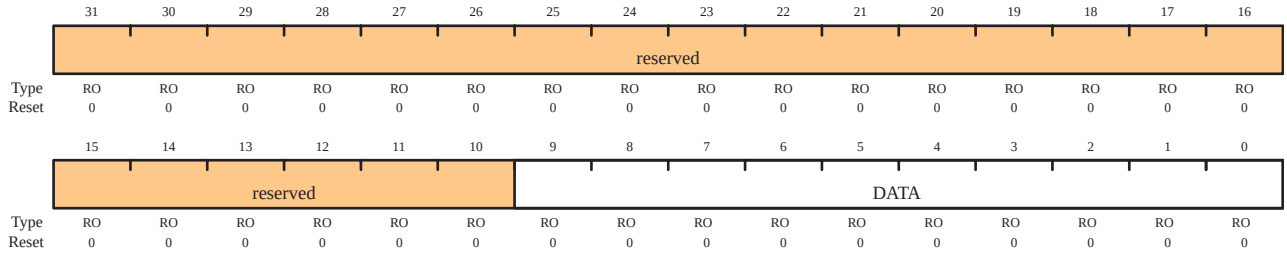
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | TS7 | R/W | 0 | The TS7 bit is used during the eighth sample of the sample sequence and specifies the input source of the sample. If set, the temperature sensor is read. Otherwise, the input pin specified by the **ADCAMUX** register is read. |
| 30 | IE7 | R/W | 0 | The IE7 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. When this bit is set, the raw interrupt is asserted, otherwise it is not. It is legal to have multiple samples within a sequence generate interrupts. |
| 29 | END7 | R/W | 0 | 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 | 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 | Same definition as TS7 but used during the seventh sample. |
| 26 | IE6 | R/W | 0 | Same definition as IE7 but used during the seventh sample. |
| 25 | END6 | R/W | 0 | Same definition as END7 but used during the seventh sample. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 24 | D6 | R/W | 0 | Same definition as D7 but used during the seventh sample. |
| 23 | TS5 | R/W | 0 | Same definition as TS7 but used during the sixth sample. |
| 22 | IE5 | R/W | 0 | Same definition as IE7 but used during the sixth sample. |
| 21 | END5 | R/W | 0 | Same definition as END7 but used during the sixth sample. |
| 20 | D5 | R/W | 0 | Same definition as D7 but used during the sixth sample. |
| 19 | TS4 | R/W | 0 | Same definition as TS7 but used during the fifth sample. |
| 18 | IE4 | R/W | 0 | Same definition as IE7 but used during the fifth sample. |
| 17 | END4 | R/W | 0 | Same definition as END7 but used during the fifth sample. |
| 16 | D4 | R/W | 0 | Same definition as D7 but used during the fifth sample. |
| 15 | TS3 | R/W | 0 | Same definition as TS7 but used during the fourth sample. |
| 14 | IE3 | R/W | 0 | Same definition as IE7 but used during the fourth sample. |
| 13 | END3 | R/W | 0 | Same definition as END7 but used during the fourth sample. |
| 12 | D3 | R/W | 0 | Same definition as D7 but used during the fourth sample. |
| 11 | TS2 | R/W | 0 | Same definition as TS7 but used during the third sample. |
| 10 | IE2 | R/W | 0 | Same definition as IE7 but used during the third sample. |
| 9 | END2 | R/W | 0 | Same definition as END7 but used during the third sample. |
| 8 | D2 | R/W | 0 | Same definition as D7 but used during the third sample. |
| 7 | TS1 | R/W | 0 | Same definition as TS7 but used during the second sample. |
| 6 | IE1 | R/W | 0 | Same definition as IE7 but used during the second sample. |
| 5 | END1 | R/W | 0 | Same definition as END7 but used during the second sample. |
| 4 | D1 | R/W | 0 | Same definition as D7 but used during the second sample. |
| 3 | TS0 | R/W | 0 | Same definition as TS7 but used during the first sample. |
| 2 | IE0 | R/W | 0 | Same definition as IE7 but used during the first sample. |
| 1 | END0 | R/W | 0 | Same definition as END7 but used during the first sample. Since this sequencer has only one entry, this bit must be set. |
| 0 | D0 | R/W | 0 | Same definition as D7 but used during the first sample. |

### Register 13: ADC Sample Sequence Result FIFO 0 (ADCSSFIFO0), offset 0x048

This register contains the conversion results for samples collected with Sample Sequencer 0. 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)
Offset 0x048

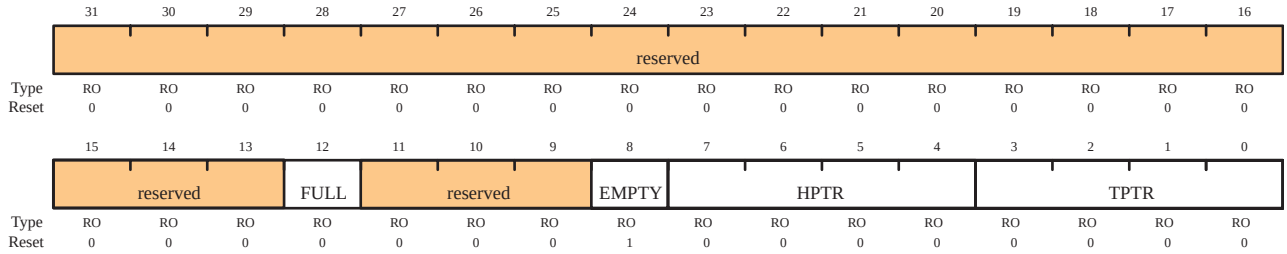



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 9:0 | DATA | RO | 0 | Conversion result data. |

### Register 14: ADC Sample Sequence FIFO 0 Status (ADCSSFSTAT0), offset 0x04C

This register provides a window into the Sample Sequencer FIFO 0, 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.

ADC Sample Sequence FIFO 0 Status (ADCSSFSTAT0)
Offset 0x04C




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:13 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 12 | FULL | RO | 0 | When set, indicates that the FIFO is currently full. |
| 11:9 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 8 | EMPTY | RO | 1 | When set, indicates that the FIFO is currently empty. |
| 7:4 | HPTR | RO | 0 | This field contains the current "head" pointer index for the FIFO, that is, the next entry to be written. |
| 3:0 | TPTR | RO | 0 | This field contains the current "tail" pointer index for the FIFO, that is, the next entry to be read. |

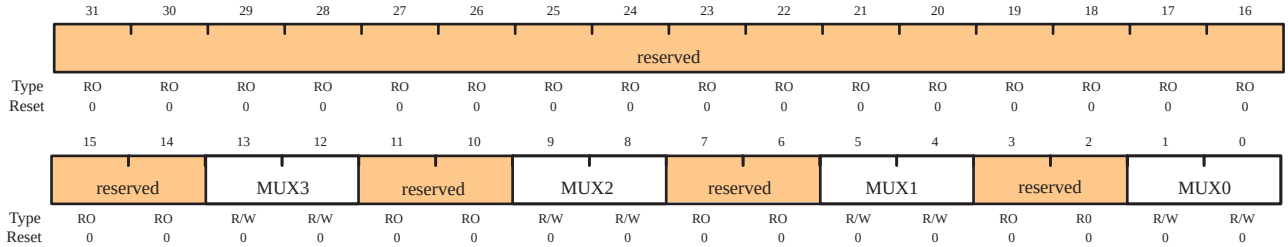
### Register 15: ADC Sample Sequence Input Multiplexer Select 1 (ADCSSMUX1), offset 0x060

This register defines the analog input configuration for each sample in a sequence executed with Sample Sequencer 1.

This register is 16-bits wide and contains information for four possible samples. This register's bit fields are as shown in the diagram below. Bit field definitions are the same as those in the **ADCSSMUX0** register (see page 217) but are for Sample Sequencer 1.

ADC Sample Sequence Input Multiplexer Select 1 (ADCSSMUX1)

Offset 0x060



### Register 16: ADC Sample Sequence Control 1 (ADCSSCTL1), offset 0x064

This register contains the configuration information for each sample for a sequence executed with Sample Sequencer 1. 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 16-bits wide and contains information for four possible samples. This register's bit fields are as shown in the diagram below. Bit field definitions are the same as those in the **ADCSSCTL0** register (see page 219) but are for Sample Sequencer 1.

ADC Sample Sequence Control 1 (ADCSSCTL1)
Offset 0x064

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 |

### Register 17: ADC Sample Sequence Result FIFO 1 (ADCSSFIFO1), offset 0x068

This register contains the conversion results for samples collected with Sample Sequencer 1. 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.

Bit fields and definitions are the same as **ADCSSFIFO0** (see page 221) but are for FIFO 1.

### Register 18: ADC Sample Sequence FIFO 1 Status (ADCSSFSTAT1), offset 0x06C

This register provides a window into the Sample Sequencer FIFO 1, 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.

This register has the same bit fields and definitions as **ADCSSFSTAT0** (see page 222) but is for FIFO 1.

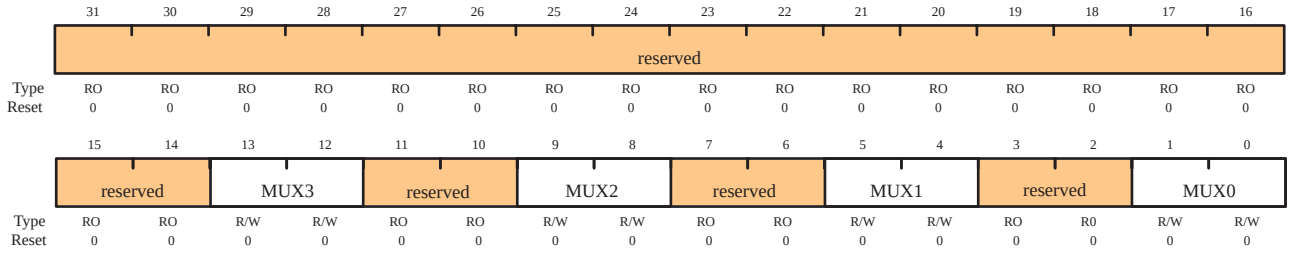
### Register 19: ADC Sample Sequence Input Multiplexer Select 2 (ADCSSMUX2), offset 0x080

This register defines the analog input configuration for each sample in a sequence executed with Sample Sequencer 2.

This register is 16-bits wide and contains information for four possible samples. This register's bit fields are as shown in the diagram below. Bit field definitions are the same as those in the **ADCSSMUX0** register (see page 217) but are for Sample Sequencer 2.

ADC Sample Sequence Input Multiplexer Select 2 (ADCSSMUX2)

Offset 0x080



### Register 20: ADC Sample Sequence Control 2 (ADCSSCTL2), offset 0x084

This register contains the configuration information for each sample for a sequence executed with Sample Sequencer 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.

This register is 16-bits wide and contains information for four possible samples. This register's bit fields are as shown in the diagram below. Bit field definitions are the same as those in the **ADCSSCTL0** register (see page 219) but are for Sample Sequencer 2.

ADC Sample Sequence Control 2 (ADCSSCTL2)

Offset 0x084

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 |

### Register 21: ADC Sample Sequence Result FIFO 2 (ADCSSFIFO2), offset 0x088

This register contains the conversion results for samples collected with Sample Sequencer 2. 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.

Bit fields and definitions are the same as **ADCSSFIFO0** (see page 221) but are for FIFO 2.

### Register 22: ADC Sample Sequence FIFO 2 Status (ADCSSFSTAT2), offset 0x08C

This register provides a window into the Sample Sequencer FIFO 2, 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.

This register has the same bit fields and definitions as **ADCSSFSTAT0** (see page 222) but is for FIFO 2.

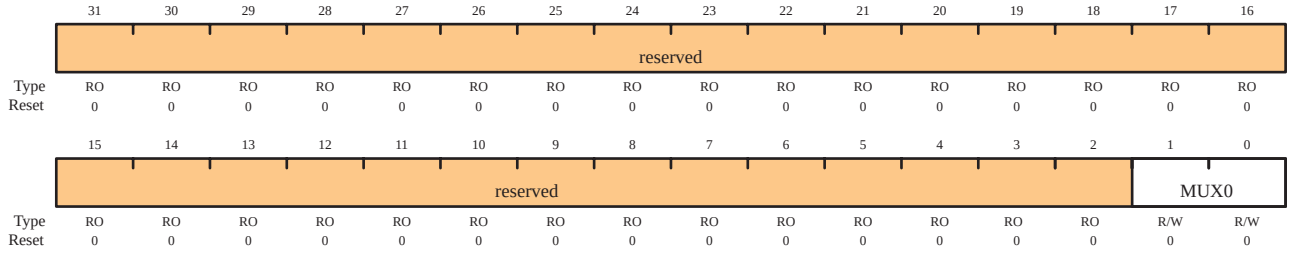
### Register 23: ADC Sample Sequence Input Multiplexer Select 3 (ADCSSMUX3), offset 0x0A0

This register defines the analog input configuration for each sample in a sequence executed with Sample Sequencer 3.

This register is 4-bits wide and contains information for one possible sample. This register's bit fields are as shown in the diagram below. Bit field definitions are the same as those in the **ADCSSMUX0** register ( see page 217) but are for Sample Sequencer 3.

ADC Sample Sequence Input Multiplexer Select 3 (ADCSSMUX3)

Offset 0x0A0



### Register 24: ADC Sample Sequence Control 3 (ADCSSCTL3), offset 0x064

This register contains the configuration information for each sample for a sequence 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. This register's bit fields are as shown in the diagram below. Bit field definitions are the same as those in the **ADCSSCTL0** register (see page 219) but are for Sample Sequencer 3.

ADC Sample Sequence Control 3 (ADCSSCTL3)
Offset 0x0A4

| | 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 | | | | | | | | | | | | TS0 | IE0 | END0 | D0 |
| 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 | 1 | 0 |

### Register 25: ADC Sample Sequence Result FIFO 3 (ADCSSFIFO3), offset 0x0A8

This register contains the conversion results for samples collected with Sample Sequencer 3. Reads of this register return the conversion result data. If the FIFO is not properly handled by software, overflow and underflow conditions are registered in the **ADCOSTAT** and **ADCUSTAT** registers.

Bit fields and definitions are the same as **ADCSSFIFO0** (see page 221) but are for FIFO 3.

### Register 26: ADC Sample Sequence FIFO 3 Status (ADCSSFSTAT3), offset 0x0AC

This register provides a window into the Sample Sequencer FIFO 3, 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.

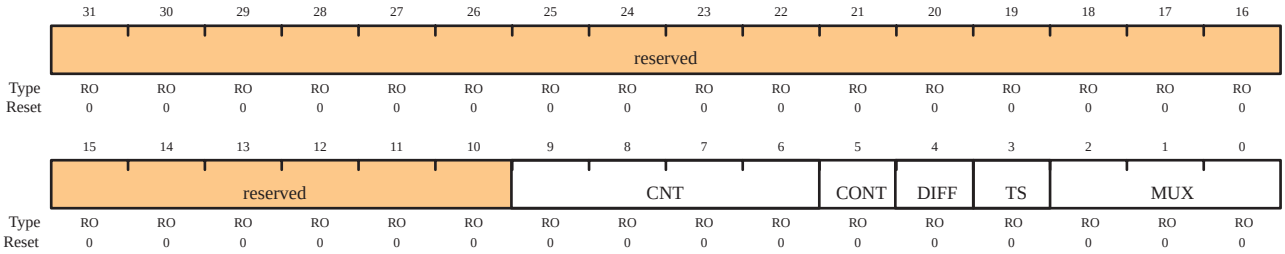
This register has the same bit fields and definitions as **ADCSSFSTAT0** (see page 222) but is for FIFO 3.

### Register 27: ADC Test Mode Loopback (ADCTMLB), offset 0x100

This register provides loopback operation within the digital logic of the ADC, which can be useful in debugging software without having to provide actual analog stimulus. This test mode is entered by writing a value of 0x00000001 to this register. When data is read from the FIFO in loopback mode, the read-only portion of this register is returned.

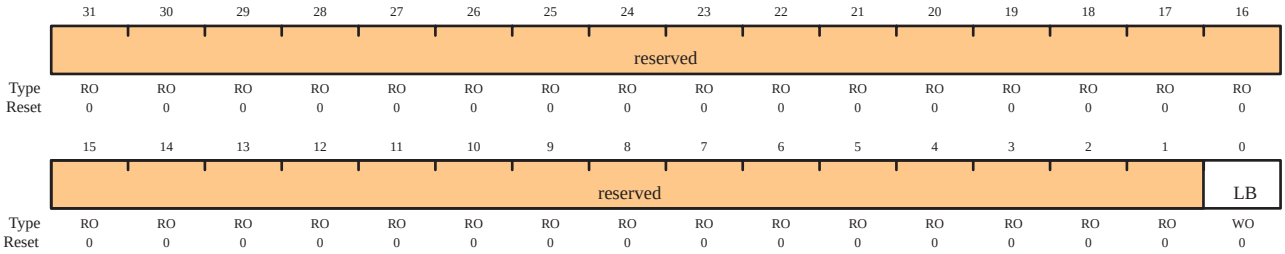
ADC Test Mode Loopback (ADCTMLB): Read

Offset 0x100




ADC Test Mode Loopback (ADCTMLB):Write

Offset 0x100




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| **Read-Only Register** | | | | |
| 31:10 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 9:6 | CNT | RO | 0 | Continuous sample counter that is initialized to 0 and counts each sample as it processed. This helps provide a unique value for the data received. |
| 5 | CONT | RO | 0 | When set, indicates that this is a continuation sample. For example if two sequencers were to run back-to-back, this indicates that the controller kept continuously sampling at full rate. |
| 4 | DIFF | RO | 0 | When set, indicates that this was to be a differential sample. |
| 3 | TS | RO | 0 | When set, indicates that this was to be a temperature sensor sample. |
| 2:0 | MUX | RO | 0 | Indicate which analog input was to be sampled. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| **Write-Only Register** | | | | |
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | LB | WO | 0 | When set, forces a loopback within the digital block to provide information on input and unique numbering.<br>The 10-bit loopback data is defined as shown in the read for bits 9:0 below. |

# 12 Universal Asynchronous Receivers/Transmitters (UARTs)

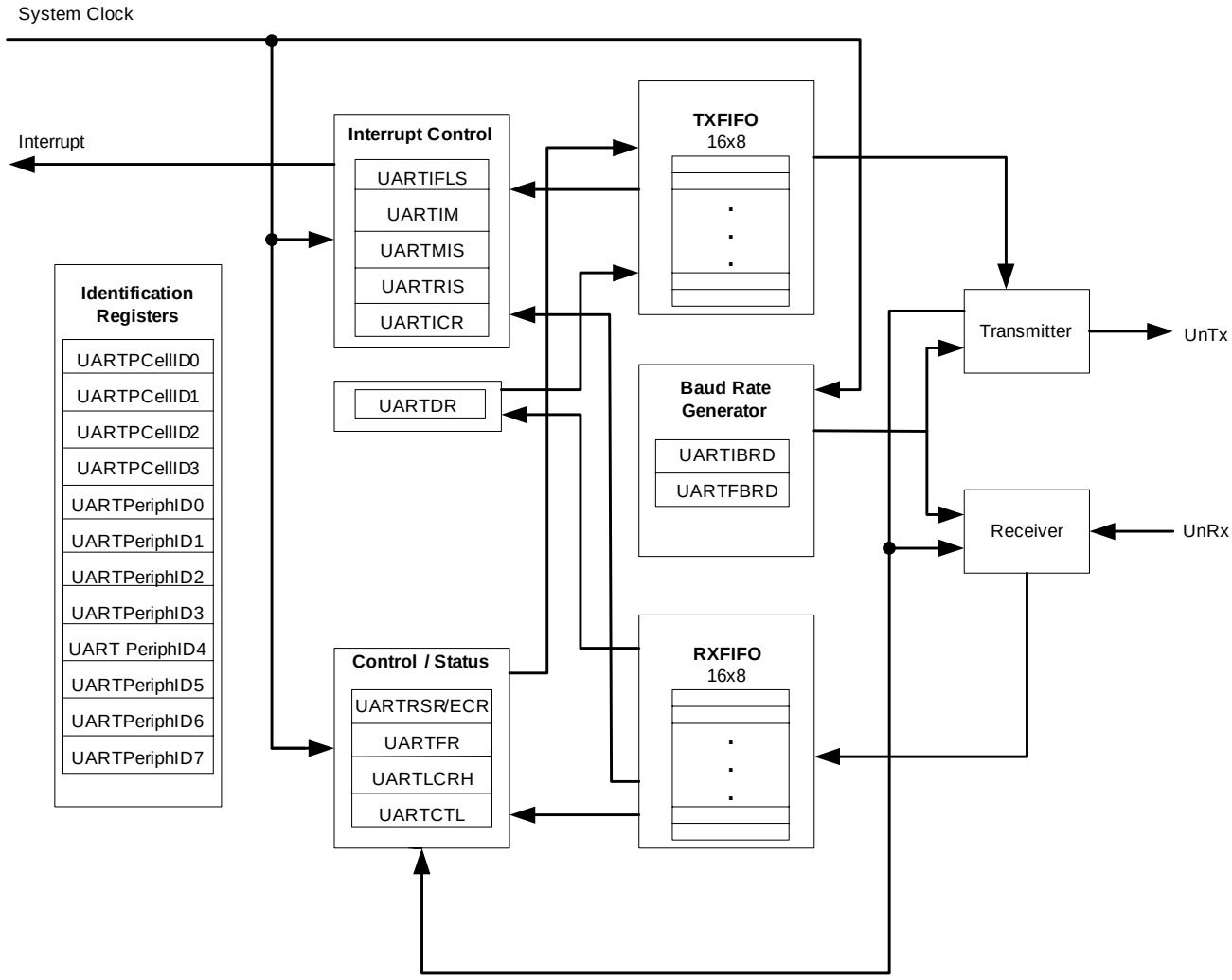
The Universal Asynchronous Receivers/Transmitters (UARTs) provide fully programmable, 16C550-type serial interface characteristics. The LM3S316 controller is equipped with two UART modules.

Each UART has the following features:

- Separate transmit and receive FIFOs
- 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
- Programmable baud-rate generator allowing rates up to 460.8 Kbps
- Standard asynchronous communication bits for start, stop and parity
- False start bit detection
- 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

## 12.1 Block Diagram

**Figure 12-1. UART Module Block Diagram**




## 12.2 Functional Description

The 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 248). 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.

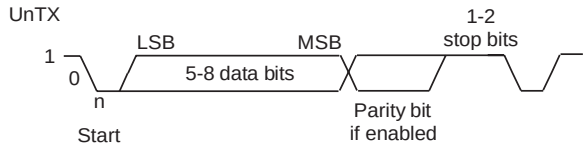
### 12.2.1 Transmit/Receive Logic

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 12-2 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 12-2. UART Character Frame**




## 12.2.2 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 244) and the 6-bit fractional part is loaded with the **UART Fractional Baud-Rate Divisor (UARTFBRD)** register (see page 245). 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 = SysClk / (16 * Baud Rate)
```

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 246), 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

### 12.2.3 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 242) 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 U0Rx or U1Rx 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 232).

The start bit is valid if U0Rx or U1Rx is still low on the eighth cycle of Baud16, otherwise a false start bit is detected and it is ignored. Start bit errors can be viewed in the **UART Receive Status (UARTRSR)** register (see page 240). If the start bit was valid, 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 U0Rx or U1Rx 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.

### 12.2.4 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 238). 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 246).

FIFO status can be monitored via the **UART Flag (UARTFR)** register (see page 242) 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 249). Both FIFOs can be individually configured to trigger interrupts at different levels. Available configurations include 1/8, 1/4, 1/2, 3/4 and 7/8. For example, if the 1/4 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 1/2 mark.

### 12.2.5 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 253).

The interrupt events that can trigger a controller-level interrupt are defined in the **UART Interrupt Mask (UARTIM**) register (see page 250) 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 252).

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 254).

### 12.2.6 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 248). In loopback mode, data transmitted on U0Tx is received on the U0Rx input, and data transmitted on U1Tx is received on the U1Rx input.

## 12.3 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 for using a UART module. For this example, the system 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 233, 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 244) should be set to 10. The value to be loaded into the **UARTFBRD** register (see page 245) 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 0x00000060).
5. Enable the UART by setting the UARTEN bit in the **UARTCTL** register.

## 12.4 Register Map

Table 12-1 lists the UART registers. The offset listed is a hexadecimal increment to the register's address, relative to that UART's base address:

- UART0: 0x4000C000
- UART1: 0x4000D000

**Note:** The UART must be disabled (see the UARTEN bit in the **UARTCTL** register on page 248) 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 12-1. UART Register Map**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x000 | UARTDR | 0x00000000 | R/W | Data | 238 |
| 0x004 | UARTRSR<br>UARTECR | 0x00000000 | R/W | Receive Status (read)<br>Error Clear (write) | 240 |
| 0x018 | UARTFR | 0x00000090 | RO | Flag Register (read only) | 242 |
| 0x024 | UARTIBRD | 0x00000000 | R/W | Integer Baud-Rate Divisor | 244 |
| 0x028 | UARTFBRD | 0x00000000 | R/W | Fractional Baud-Rate Divisor | 245 |
| 0x02C | UARTLCRH | 0x00000000 | R/W | Line Control Register, High byte | 246 |
| 0x030 | UARTCTL | 0x00000300 | R/W | Control Register | 248 |
| 0x034 | UARTIFLS | 0x00000012 | R/W | Interrupt FIFO Level Select | 249 |
| 0x038 | UARTIM | 0x00000000 | R/W | Interrupt Mask | 250 |
| 0x03C | UARTRIS | 0x0000000F | RO | Raw Interrupt Status | 252 |
| 0x040 | UARTMIS | 0x00000000 | RO | Masked Interrupt Status | 253 |
| 0x044 | UARTICR | 0x00000000 | W1C | Interrupt Clear | 254 |
| 0xFD0 | UARTPeriphID4 | 0x00000000 | RO | Peripheral identification 4 | 255 |
| 0xFD4 | UARTPeriphID5 | 0x00000000 | RO | Peripheral identification 5 | 256 |
| 0xFD8 | UARTPeriphID6 | 0x00000000 | RO | Peripheral identification 6 | 257 |
| 0xFDC | UARTPeriphID7 | 0x00000000 | RO | Peripheral identification 7 | 258 |
| 0xFE0 | UARTPeriphID0 | 0x00000011 | RO | Peripheral identification 0 | 259 |
| 0xFE4 | UARTPeriphID1 | 0x00000000 | RO | Peripheral identification 1 | 260 |
| 0xFE8 | UARTPeriphID2 | 0x00000018 | RO | Peripheral identification 2 | 261 |

**Table 12-1. UART Register Map (Continued)**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0xFEC | UARTPeriphID3 | 0x00000001 | RO | Peripheral identification 3 | 262 |
| 0xFF0 | UARTPCellID0 | 0x0000000D | RO | PrimeCell identification 0 | 263 |
| 0xFF4 | UARTPCellID1 | 0x000000F0 | RO | PrimeCell identification 1 | 264 |
| 0xFF8 | UARTPCellID2 | 0x00000005 | RO | PrimeCell identification 2 | 265 |
| 0xFFC | UARTPCellID3 | 0x000000B1 | RO | PrimeCell identification 3 | 266 |

## 12.5 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

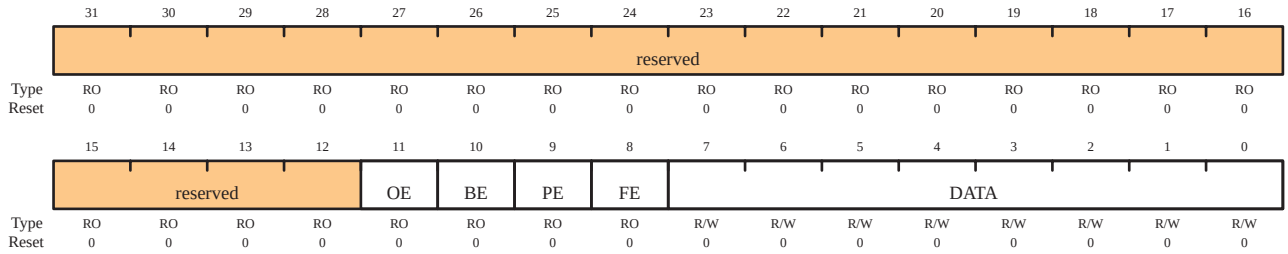
This register is the data register (the interface to the FIFOs).

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)

Offset 0x000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:12 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 11 | OE | RO | 0 | UART Overrun Error<br>1=New data was received when the FIFO was full, resulting in data loss.<br>0=There has been no data loss due to a FIFO overrun. |
| 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. |
| 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. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 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 | 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

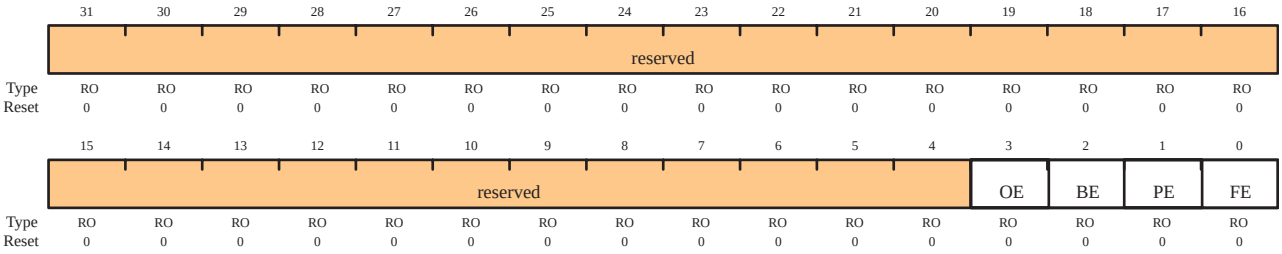
The **UARTRSR/UARTECR** register is the receive status register/error clear register.

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.

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.

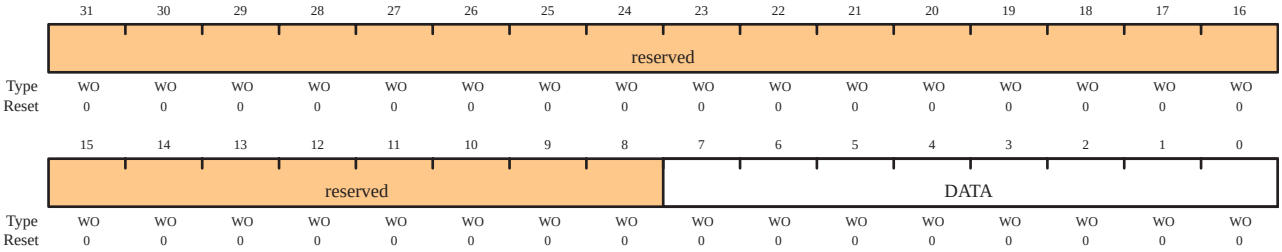
UART Receive Status (UARTRSR): Read

Offset 0x004




UART Error Clear (UARTECR): Write

Offset 0x004




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| **Read-Only Receive Status (UARTRSR) Register** | | | | |
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. The **UARTRSR** register cannot be written. |
| 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. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 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. |
| 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. |

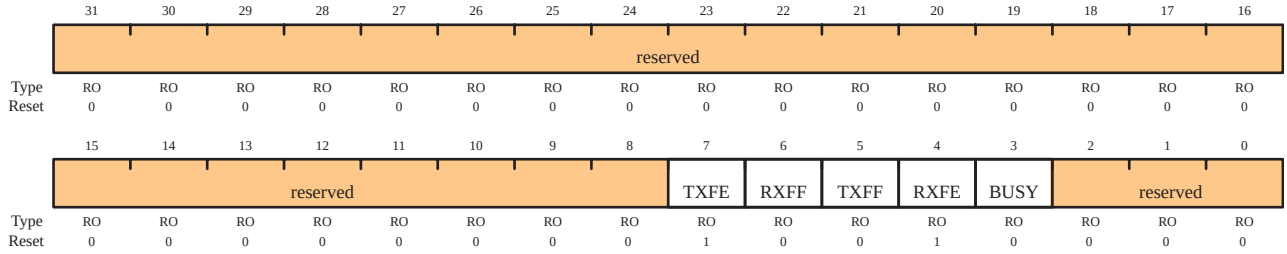
**Write-Only Error Clear (UARTECR) Register**

| | | | | |
|---|---|---|---|---|
| 31:8 | reserved | WO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | DATA | WO | 0 | 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)
Offset 0x018




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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. |

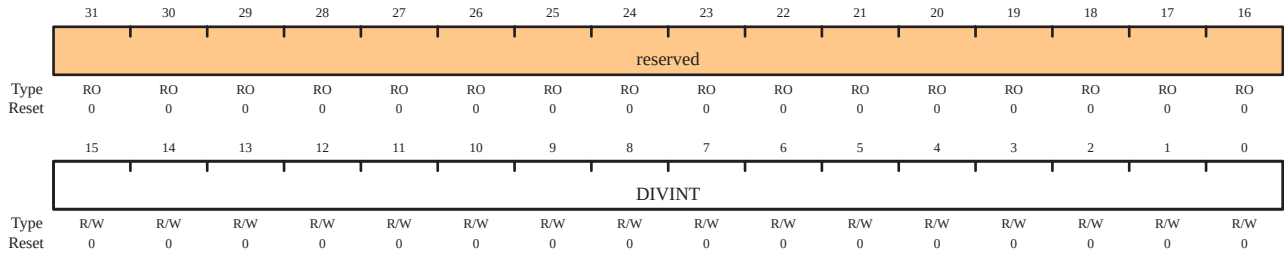
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | RXFE | RO | 1 | UART Receive FIFO Empty<br>The meaning of this bit depends on the state of the FEN bit in the **UARTLCRH** register.<br>If the FIFO is disabled, this bit is set when the receive holding register is empty.<br>If the FIFO is enabled, this bit is set when the receive FIFO is empty. |
| 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 | Reserved bits return an indeterminate value, and should never be changed. |

### Register 4: 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 233 for configuration details.

UART Integer Baud-Rate Divisor

Offset 0x024



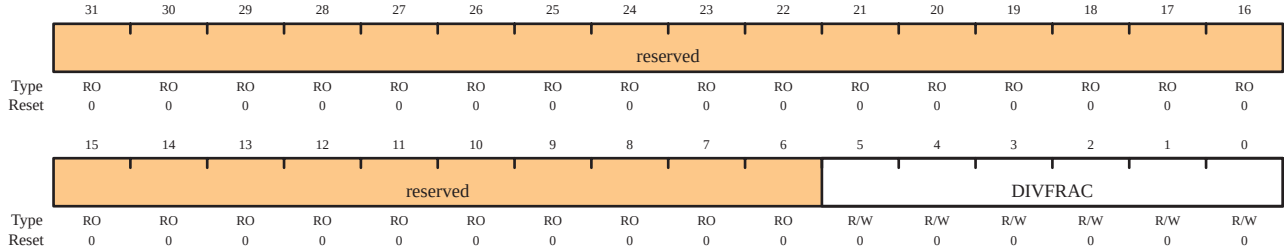

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:0 | DIVINT | R/W | 0x0000 | Integer Baud-Rate Divisor |

### Register 5: 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 233 for configuration details.

UART Fractional Baud-Rate Divisor (UARTFBRD)

Offset 0x028



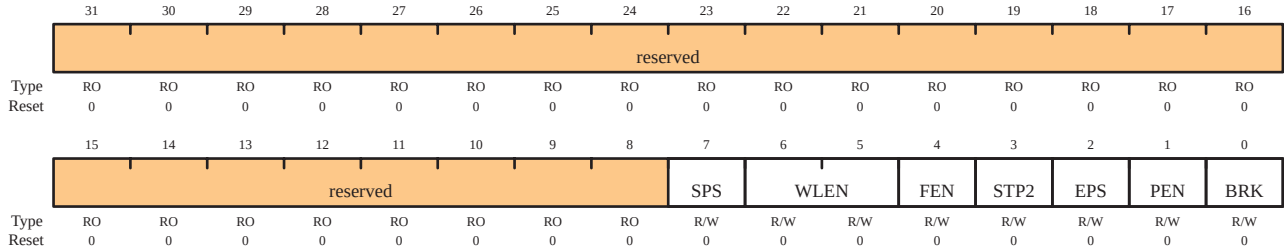

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 5:0 | DIVFRAC | R/W | 0x00 | Fractional Baud-Rate Divisor |

### Register 6: 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)
Offset 0x02C




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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>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. |

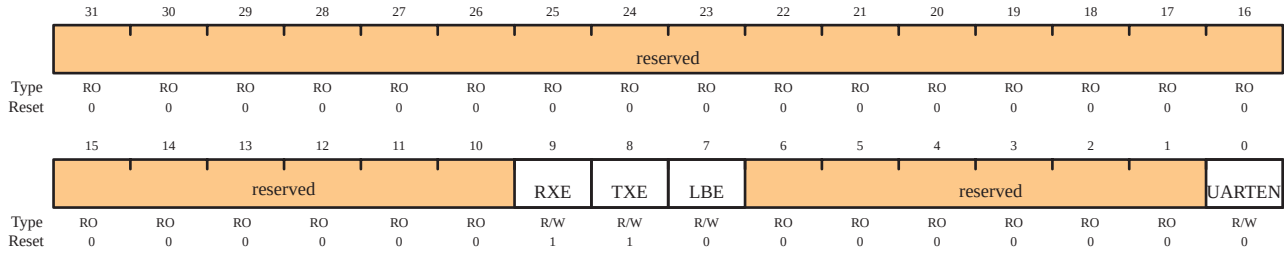
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | EPS | R/W | 0 | UART Even Parity Select<br>If this bit is set to 1, even parity generation and checking is performed during transmission and reception, which checks for an even number of 1s in data and parity bits.<br>When 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 7: 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.

UART Control (UARTCR)
Offset 0x030




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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. |
| 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. |
| 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. |
| 6:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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 8: UART Interrupt FIFO Level Select (UARTIFLS), offset 0x034

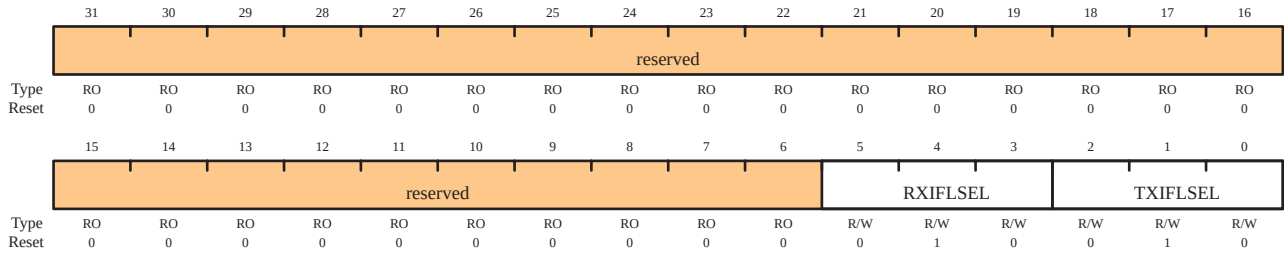
The **UARTIFLS** register is the interrupt FIFO level select register. You can use this register to define the FIFO level at which the TXRIS and RXRIS bits in the **UARTRIS** register are triggered.

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)

Offset 0x034




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 5:3 | RXIFLSEL | R/W | 0X2 | UART Receive Interrupt FIFO Level Select<br>The trigger points for the receive interrupt are as follows:<br>000: RX FIFO ≥ 1/8 full<br>001: RX FIFO ≥ 1/4 full<br>010: RX FIFO ≥ 1/2 full (default)<br>011: RX FIFO ≥ 3/4 full<br>100: RX FIFO ≥ 7/8 full<br>101-111: Reserved |
| 2:0 | TXIFLSEL | R/W | 0X2 | UART Transmit Interrupt FIFO Level Select<br>The trigger points for the transmit interrupt are as follows:<br>000: TX FIFO ≤ 1/8 full<br>001: TX FIFO ≤ 1/4 full<br>010: TX FIFO ≤ 1/2 full (default)<br>011: TX FIFO ≤ 3/4 full<br>100: TX FIFO ≤ 7/8 full<br>101-111: Reserved |

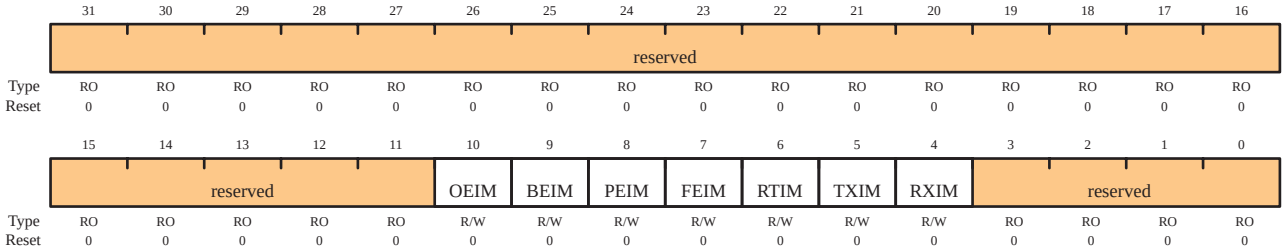
### Register 9: UART Interrupt Mask (UARTIM), offset 0x038

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)

Offset 0x038



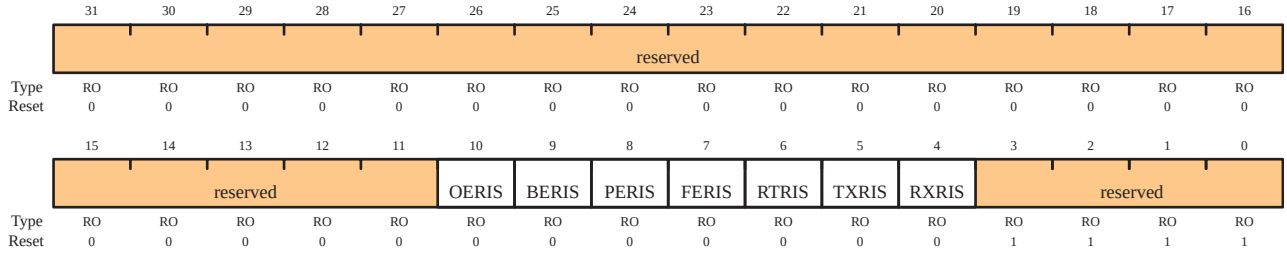

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 10 | OEIM | R/W | 0 | UART Overrun Error Interrupt Mask<br>On a read, the current mask for the `OEIM` interrupt is returned.<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. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 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. |
| 3:0 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |

### Register 10: 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)
Offset 0x03C

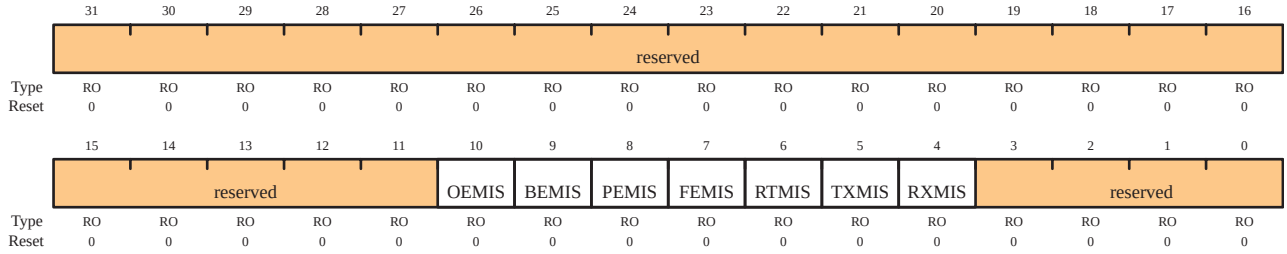



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 10 | OERIS | RO | 0 | UART Overrun Error Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this 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 | 0xF | This reserved bit is read-only and has a reset value of 0xF. |

### Register 11: 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)
Offset 0x040

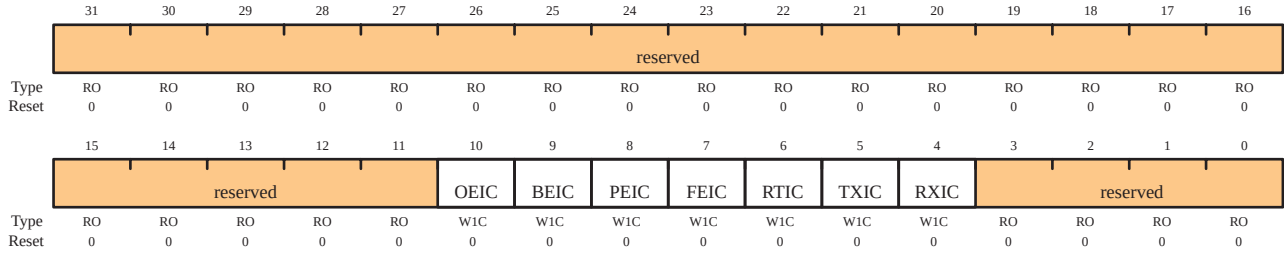



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 10 | OEMIS | RO | 0 | UART Overrun Error Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 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 | Reserved bits return an indeterminate value, and should never be changed. |

### Register 12: 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)
Offset 0x044



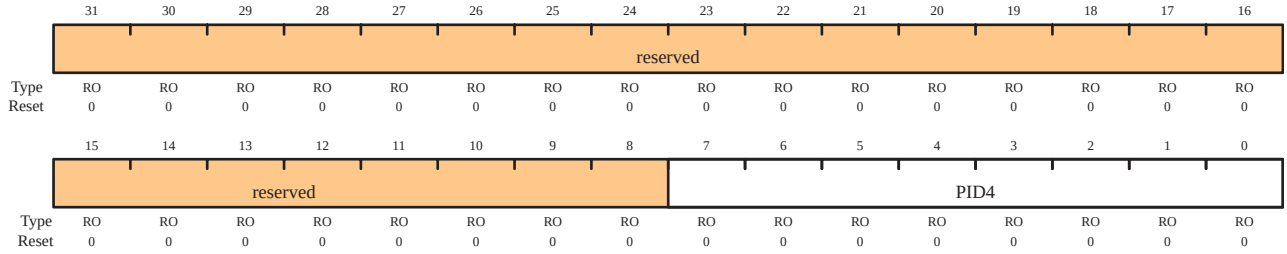

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 10 | OEIC | W1C | 0 | Overrun Error Interrupt Clear<br>0: No effect on the interrupt.<br>1: Clears interrupt. |
| 9 | BEIC | W1C | 0 | Break Error Interrupt Clear<br>0: No effect on the interrupt.<br>1: Clears interrupt. |
| 8 | PEIC | W1C | 0 | Parity Error Interrupt Clear<br>0: No effect on the interrupt.<br>1: Clears interrupt. |
| 7 | FEIC | W1C | 0 | Framing Error Interrupt Clear<br>0: No effect on the interrupt.<br>1: Clears interrupt. |
| 6 | RTIC | W1C | 0 | Receive Time-Out Interrupt Clear<br>0: No effect on the interrupt.<br>1: Clears interrupt. |
| 5 | TXIC | W1C | 0 | Transmit Interrupt Clear<br>0: No effect on the interrupt.<br>1: Clears interrupt. |
| 4 | RXIC | W1C | 0 | Receive Interrupt Clear<br>0: No effect on the interrupt.<br>1: Clears interrupt. |
| 3:0 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |

### Register 13: 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)

Offset 0xFD0



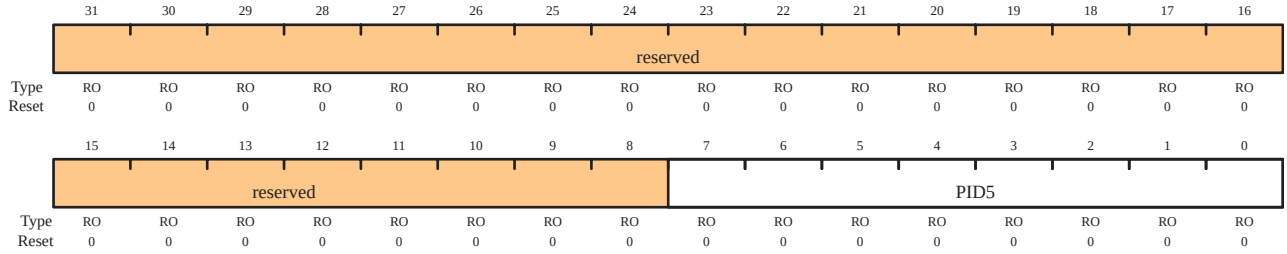

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID4 | RO | 0x00 | UART Peripheral ID Register[7:0] |

### Register 14: 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)

Offset 0xFD4



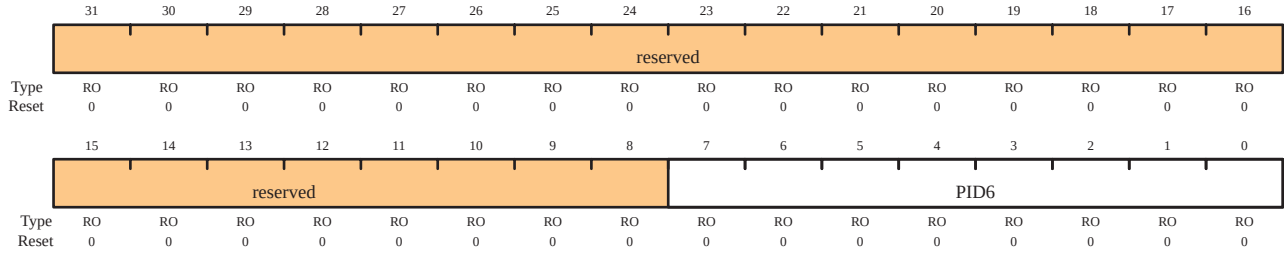

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID5 | RO | 0x00 | UART Peripheral ID Register[15:8] |

### Register 15: 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)

Offset 0xFD8



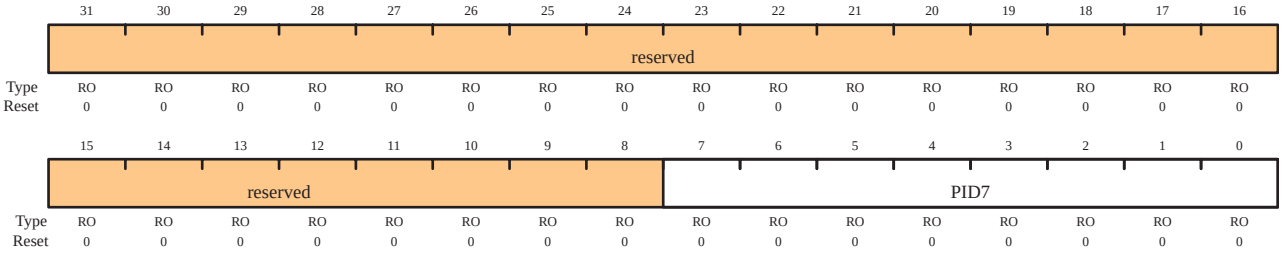

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID6 | RO | 0x00 | UART Peripheral ID Register[23:16] |

### Register 16: 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)

Offset 0xFDC



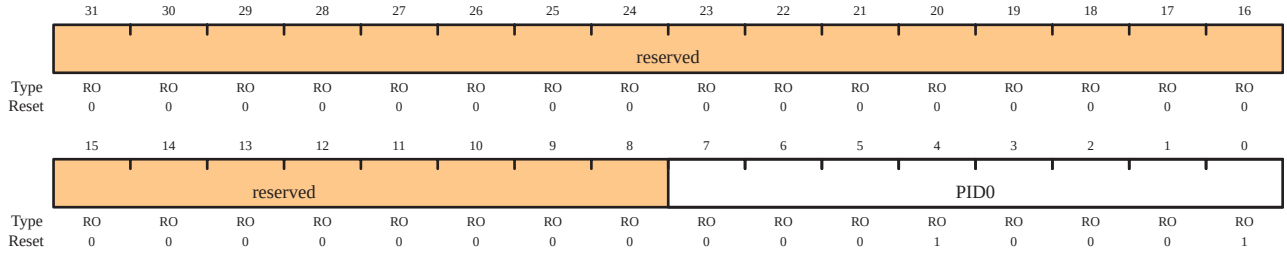

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID7 | RO | 0x00 | UART Peripheral ID Register[31:24] |

### Register 17: 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)

Offset 0xFE0



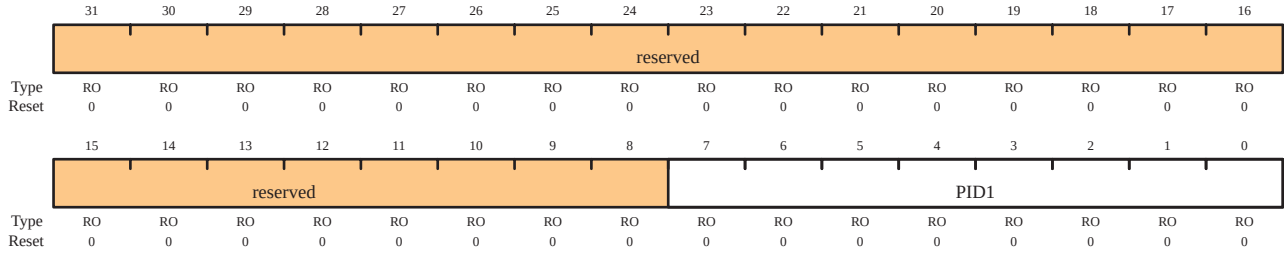

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID0 | RO | 0x11 | UART Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

### Register 18: 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)

Offset 0xFE4



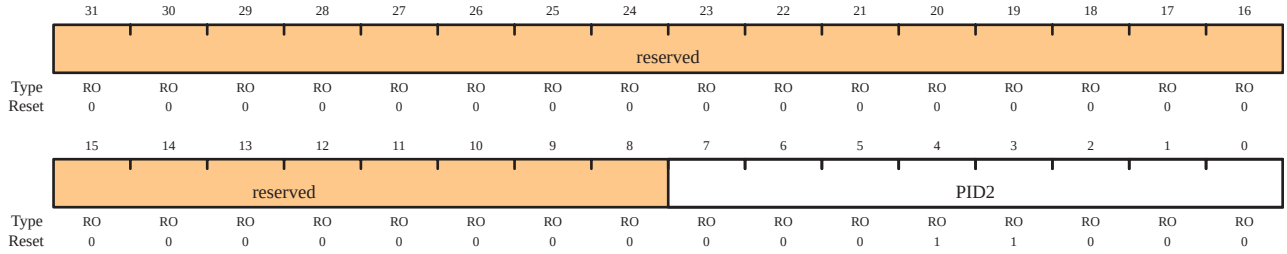

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID1 | RO | 0x00 | UART Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

### Register 19: 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)

Offset 0xFE8



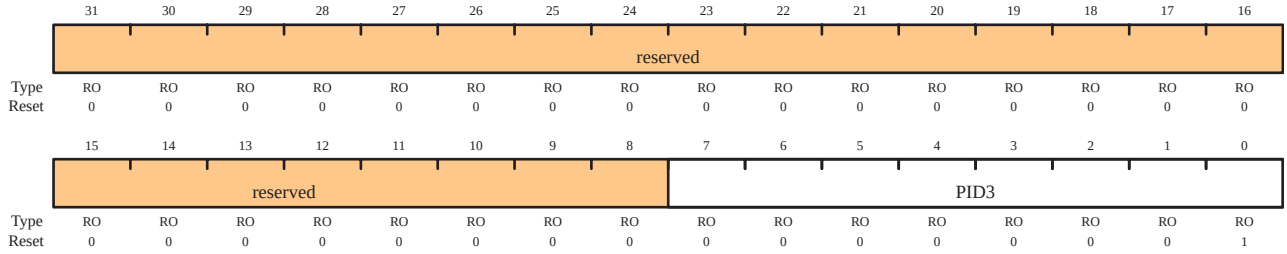

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID2 | RO | 0x18 | UART Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

### Register 20: 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)

Offset 0xFEC



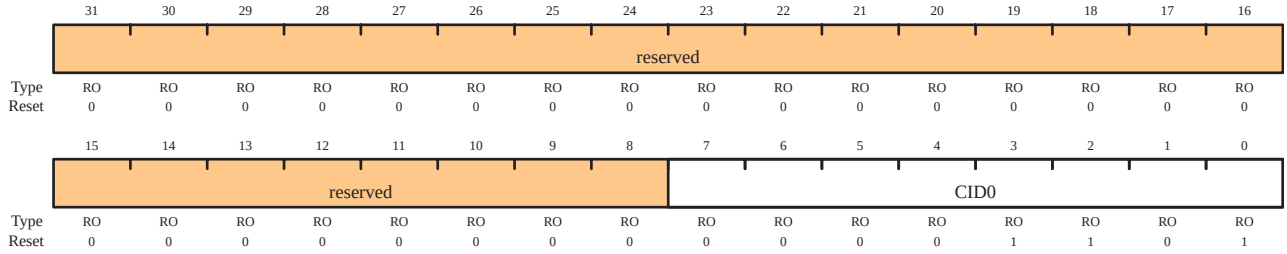

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID3 | RO | 0x01 | UART Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

### Register 21: 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)

Offset 0xFF0



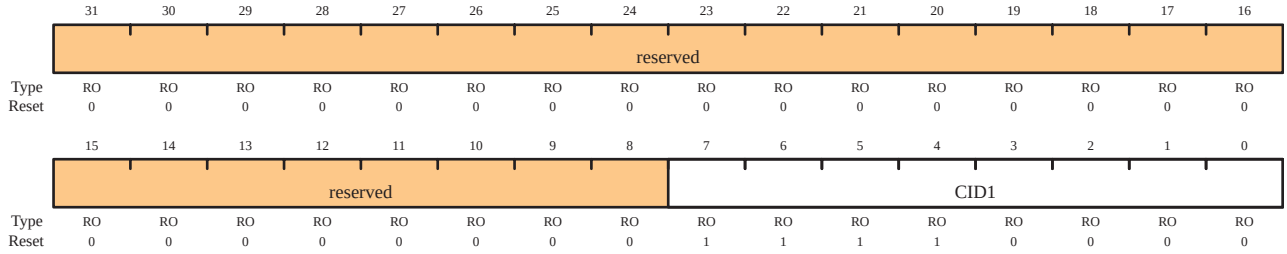

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID0 | RO | 0x0D | UART PrimeCell ID Register[7:0]<br>Provides software a standard cross-peripheral identification system. |

### Register 22: 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)

Offset 0xFF4



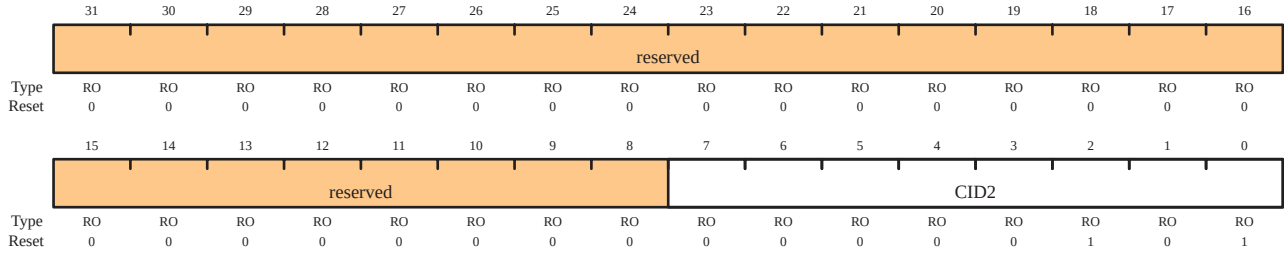

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID1 | RO | 0xF0 | UART PrimeCell ID Register[15:8]<br>Provides software a standard cross-peripheral identification system. |

### Register 23: 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)

Offset 0xFF8



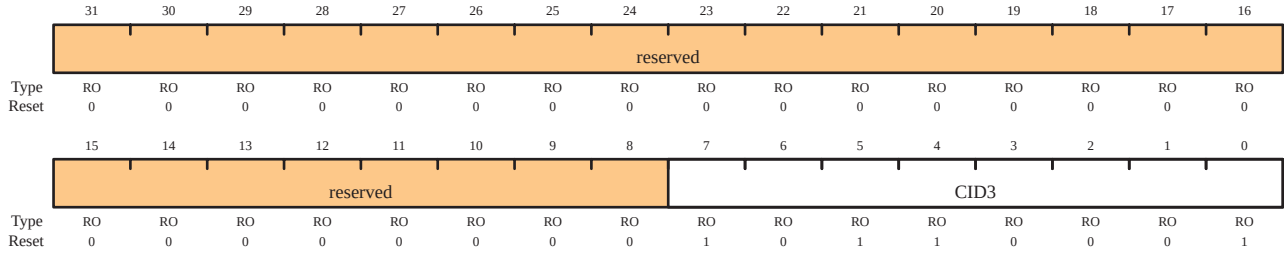

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID2 | RO | 0x05 | UART PrimeCell ID Register[23:16]<br>Provides software a standard cross-peripheral identification system. |

### Register 24: 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)

Offset 0xFFC




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID3 | RO | 0xB1 | UART PrimeCell ID Register[31:24]<br>Provides software a standard cross-peripheral identification system. |

# 13 Synchronous Serial Interface (SSI)

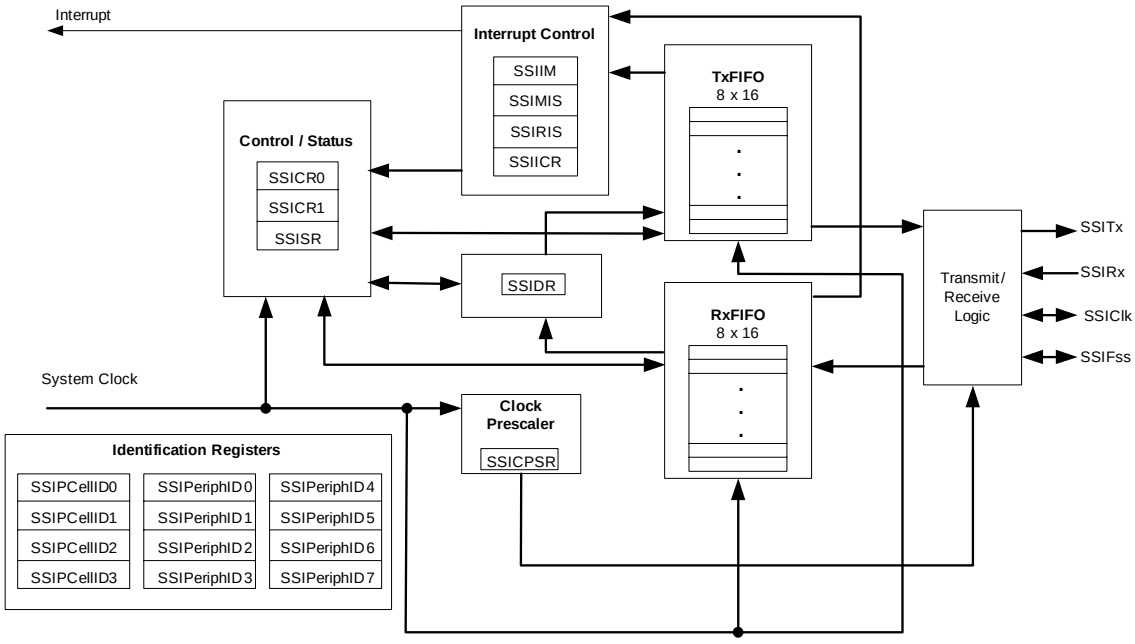
The Stellaris Synchronous Serial Interface (SSI) is a master or slave interface for synchronous serial communication with peripheral devices that have either Freescale SPI, MICROWIRE, or Texas Instruments synchronous serial interfaces.

The Stellaris SSI has the following features:

- Master or slave operation
- 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

## 13.1 Block Diagram

**Figure 13-1. SSI Module Block Diagram**



## 13.2 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.

### 13.2.1 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 25-MHz input clock. 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 285). 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 279).

The frequency of the output clock `SSIClk` is defined by:

```
FSSIClk = FSysClk / (CPSDVSR * (1 + SCR))
```

Note that although the `SSIClk` transmit clock can theoretically be 12.5 MHz, the module may not be able to operate at that speed. For transmit operations, the system clock must be at least two times faster than the `SSIClk`. For receive operations, the system clock must be at least 12 times faster than the `SSIClk`.

See “Electrical Characteristics” on page 389 to view SSI timing parameters.

### 13.2.2 FIFO Operation

#### 13.2.2.1 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 283), 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.

#### 13.2.2.2 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.

### 13.2.3 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 286). 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 287 and page 288, respectively).

### 13.2.4 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.

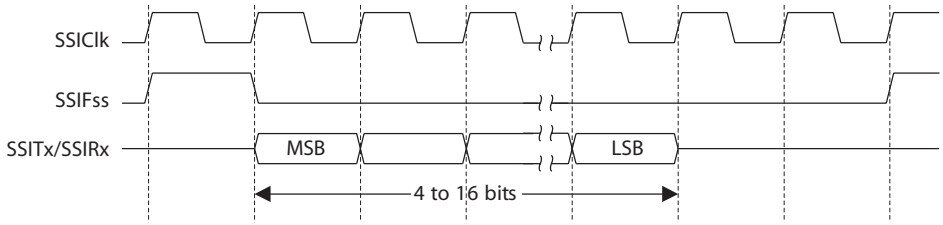
For Texas Instruments synchronous serial frame format, the `SSIFss` pin is pulsed for one serial clock period starting at its rising edge, prior to the transmission of each frame. For this frame format, both the SSI and the off-chip slave device drive their output data on the rising edge of `SSIClk`, 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.

#### 13.2.4.1 Texas Instruments Synchronous Serial Frame Format

Figure 13-2 shows the Texas Instruments synchronous serial frame format for a single transmitted frame.

**Figure 13-2. TI Synchronous Serial Frame Format (Single Transfer)**

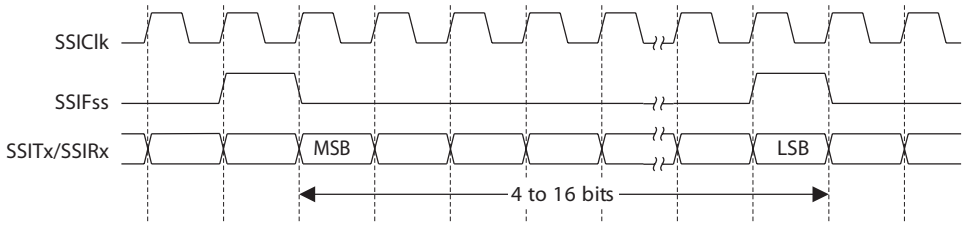


In this mode, SSIClk and SSIFss are forced Low, and the transmit data line SSITx is tristated whenever the SSI is idle. Once the bottom entry of the transmit FIFO contains data, SSIFss is pulsed High for one SSIClk period. The value to be transmitted is also transferred from the transmit FIFO to the 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 13-3 shows the Texas Instruments synchronous serial frame format when back-to-back frames are transmitted.

**Figure 13-3. TI Synchronous Serial Frame Format (Continuous Transfer)**




### 13.2.4.2 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.

### 13.2.4.3 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 13-4 and Figure 13-5.

**Figure 13-4. Freescale SPI Format (Single Transfer) with SPO=0 and SPH=0**

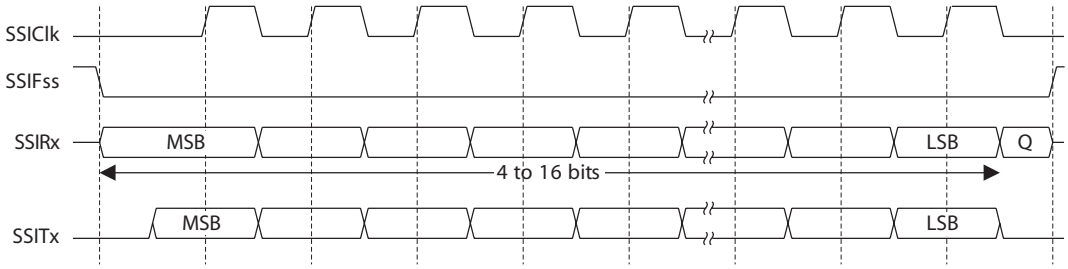
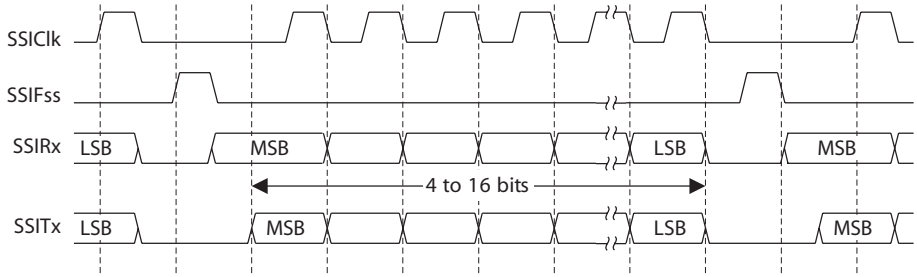



**Figure 13-5. Freescale SPI Format (Continuous Transfer) with SPO=0 and SPH=0**




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.

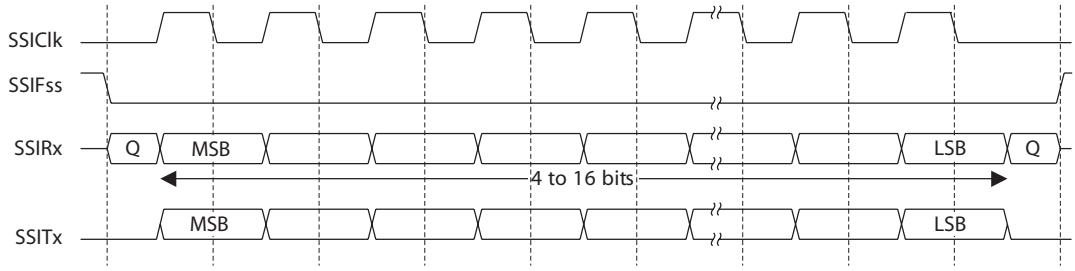
In the case of a single word transmission, after all bits of the data word have been transferred, the SSIFss line is returned to its idle High state one SSIClk period after the last bit has been captured.

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.

#### 13.2.4.4 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 13-6, which covers both single and continuous transfers.

**Figure 13-6. Freescale SPI Frame Format with SPO=0 and SPH=1**




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.

### 13.2.4.5 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 13-7 and Figure 13-8.

**Figure 13-7. Freescale SPI Frame Format (Single Transfer) with SPO=1 and SPH=0**

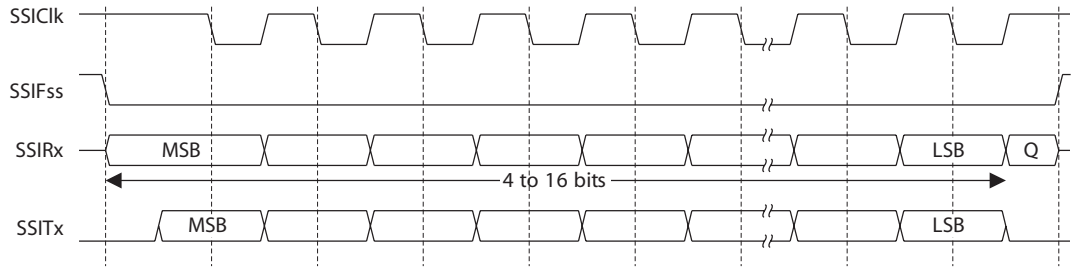
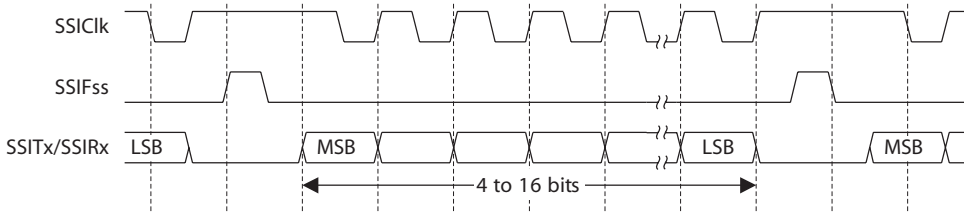


**Figure 13-8. Freescale SPI Frame Format (Continuous Transfer) with SPO=1 and SPH=0**




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.

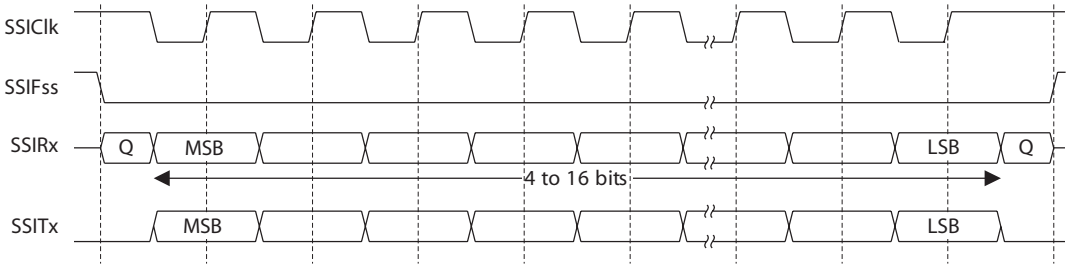
In the case of a single word transmission, after all bits of the data word are transferred, the SSIFss line is returned to its idle High state one SSIClk period after the last bit has been captured.

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.

### 13.2.4.6 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 13-9, which covers both single and continuous transfers.

**Figure 13-9. Freescale SPI Frame Format with SPO=1 and SPH=1**




**Note:** Q is undefined in Figure 13-9.

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.

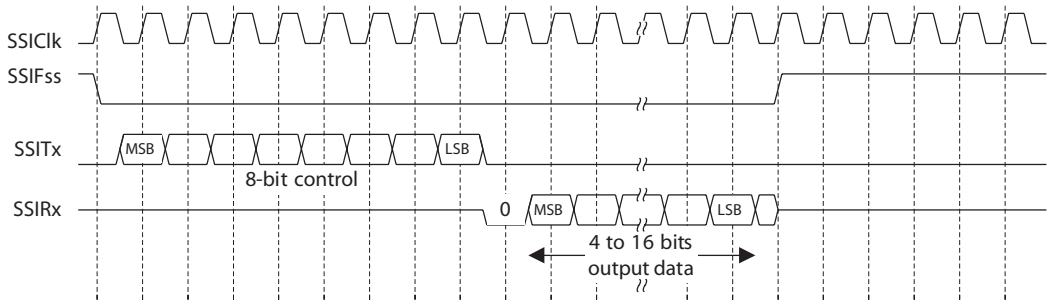
For continuous back-to-back transmissions, the SSIFss pin remains in its active Low state, until the final bit of the last word has been captured, and then returns to its idle state as described above.

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.

#### 13.2.4.7 MICROWIRE Frame Format

Figure 13-10 shows the MICROWIRE frame format, again for a single frame. Figure 13-11 shows the same format when back-to-back frames are transmitted.

**Figure 13-10. 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

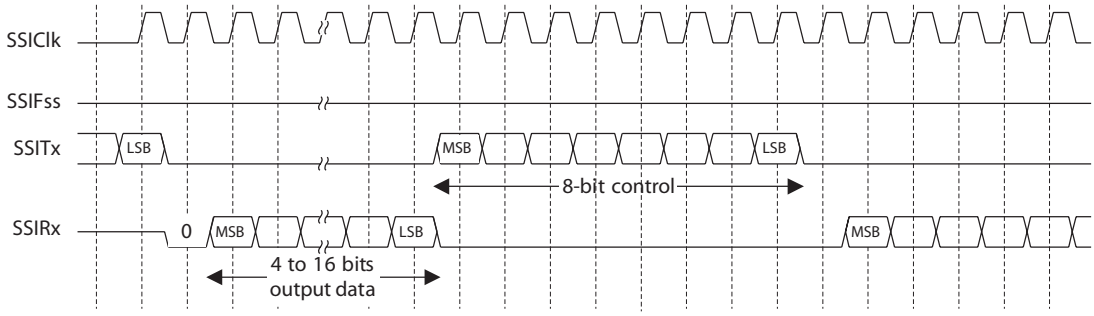
A transmission is triggered by writing a control byte to the transmit FIFO. The falling edge of SSIFss causes the value contained in the bottom entry of the transmit FIFO to be transferred to the serial shift register of the transmit logic, and the MSB of the 8-bit control frame to be shifted out onto 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.

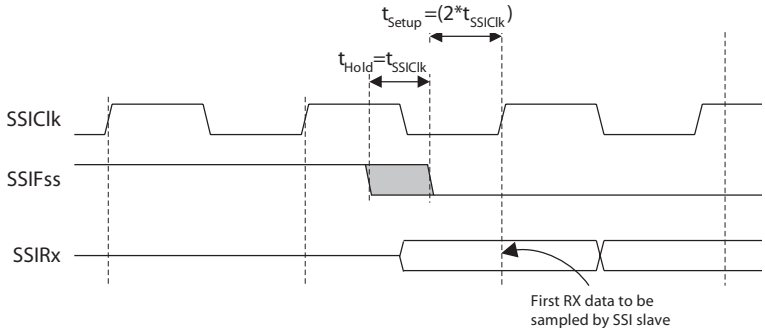
**Figure 13-11. MICROWIRE Frame Format (Continuous Transfer)**




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 13-12 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 13-12. MICROWIRE Frame Format, SSIFss Input Setup and Hold Requirements**




## 13.3 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 0x00000000.
   b. For slave mode (output enabled), set the **SSICR1** register to 0x00000004.
   c. For slave mode (output disabled), set the **SSICR1** register to 0x0000000C.
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. 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)) ' 1x106 = 20x106 / (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 0x00000000.

3. Write the **SSICPSR** register with a value of 0x00000002.
4. Write the **SSICR0** register with a value of 0x000009C7.
5. The SSI is then enabled by setting the SSE bit in the **SSICR1** register to 1.

## 13.4 Register Map

Table 13-1 lists the SSI registers. The offset listed is a hexadecimal increment to the register's address, relative to the SSI base address of 0x40008000.

**Note:** The SSI must be disabled (see the SSE bit in the **SSICR1** register) before any of the control registers are reprogrammed.

**Table 13-1. SSI Register Map**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x000 | SSICR0 | 0x00000000 | RW | Control 0 | 279 |
| 0x004 | SSICR1 | 0x00000000 | RW | Control 1 | 281 |
| 0x008 | SSIDR | 0x00000000 | RW | Data | 283 |
| 0x00C | SSISR | 0x00000003 | RO | Status | 284 |
| 0x010 | SSICPSR | 0x00000000 | RW | Clock prescale | 285 |
| 0x014 | SSIIM | 0x00000000 | RW | Interrupt mask | 286 |
| 0x018 | SSIRIS | 0x00000008 | RO | Raw interrupt status | 287 |
| 0x01C | SSIMIS | 0x00000000 | RO | Masked interrupt status | 288 |
| 0x020 | SSIICR | 0x00000000 | W1C | Interrupt clear | 289 |
| 0xFD0 | SSIPeriphID4 | 0x00000000 | RO | Peripheral identification 4 | 290 |
| 0xFD4 | SSIPeriphID5 | 0x00000000 | RO | Peripheral identification 5 | 291 |
| 0xFD8 | SSIPeriphID6 | 0x00000000 | RO | Peripheral identification 6 | 292 |
| 0xFDC | SSIPeriphID7 | 0x00000000 | RO | Peripheral identification 7 | 293 |
| 0xFE0 | SSIPeriphID0 | 0x00000022 | RO | Peripheral identification 0 | 294 |
| 0xFE4 | SSIPeriphID1 | 0x00000000 | RO | Peripheral identification 1 | 295 |
| 0xFE8 | SSIPeriphID2 | 0x00000018 | RO | Peripheral identification 2 | 296 |
| 0xFEC | SSIPeriphID3 | 0x00000001 | RO | Peripheral identification 3 | 297 |
| 0xFF0 | SSIPCellID0 | 0x0000000D | RO | PrimeCell identification 0 | 298 |
| 0xFF4 | SSIPCellID1 | 0x000000F0 | RO | PrimeCell identification 1 | 299 |
| 0xFF8 | SSIPCellID2 | 0x00000005 | RO | PrimeCell identification 2 | 300 |
| 0xFFC | SSIPCellID3 | 0x000000B1 | RO | PrimeCell identification 3 | 301 |

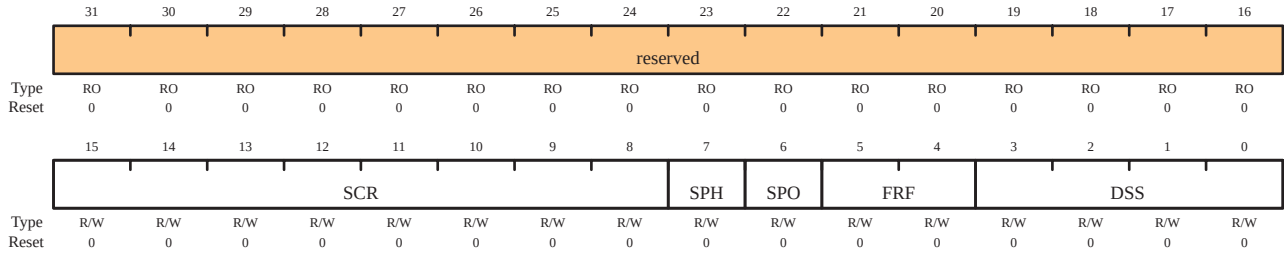
## 13.5 Register Descriptions

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)
Offset 0x000



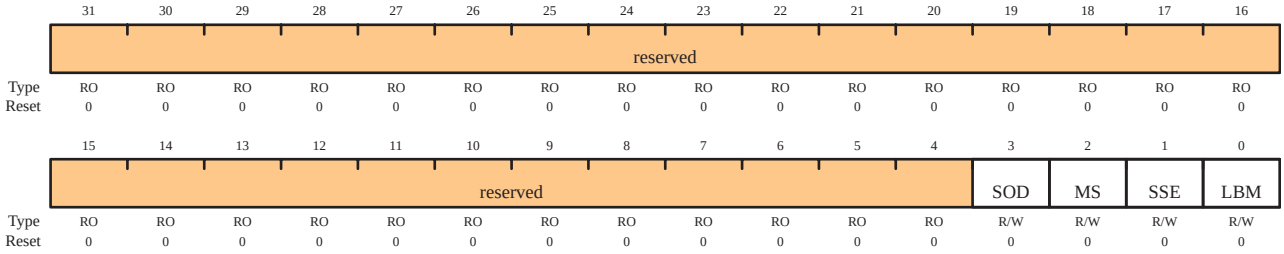

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:8 | SCR | R/W | 0 | SSI Serial Clock Rate<br>The value SCR is used to generate the transmit and receive bit rate of the SSI. The bit rate is:<br>BR= $F_{SSICLK}$/(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. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 5:4 | FRF | R/W | 0 | SSI Frame Format Select.<br>The FRF values are defined as follows:<br>FRF Value — Frame Format<br>00 — Freescale SPI Frame Format<br>01 — Texas Instruments Synchronous Serial Frame Format<br>10 — MICROWIRE Frame Format<br>11 — Reserved |
| 3:0 | DSS | R/W | 0 | SSI Data Size Select<br>The DSS values are defined as follows:<br>DSS Value — Data Size<br>0000-0010 — Reserved<br>0011 — 4-bit data<br>0100 — 5-bit data<br>0101 — 6-bit data<br>0110 — 7-bit data<br>0111 — 8-bit data<br>1000 — 9-bit data<br>1001 — 10-bit data<br>1010 — 11-bit data<br>1011 — 12-bit data<br>1100 — 13-bit data<br>1101 — 14-bit data<br>1110 — 15-bit data<br>1111 — 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 (SSCR1)
Offset 0x004




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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>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>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>0: SSI operation disabled.<br>1: SSI operation enabled.<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>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

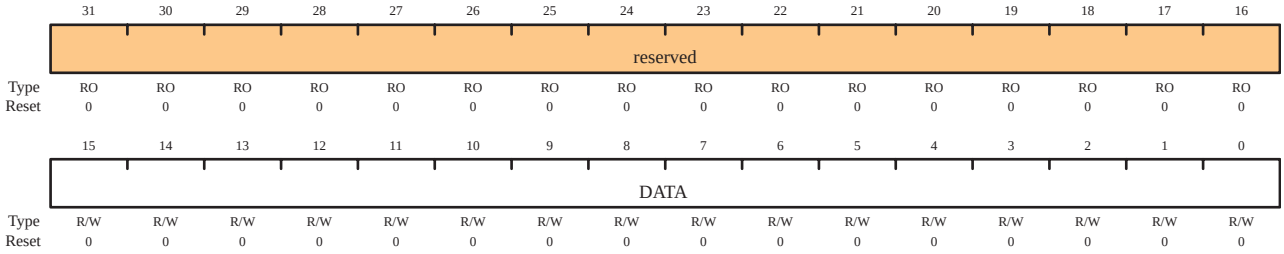
**SSIDR** is the data register and is 16-bits wide. When **SSIDR** is read, the entry in the receive FIFO (pointed to by the current FIFO read pointer) is accessed. As data values are removed by the SSI receive logic from the incoming data frame, they are placed into the entry in the receive FIFO (pointed 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)
Offset 0x008

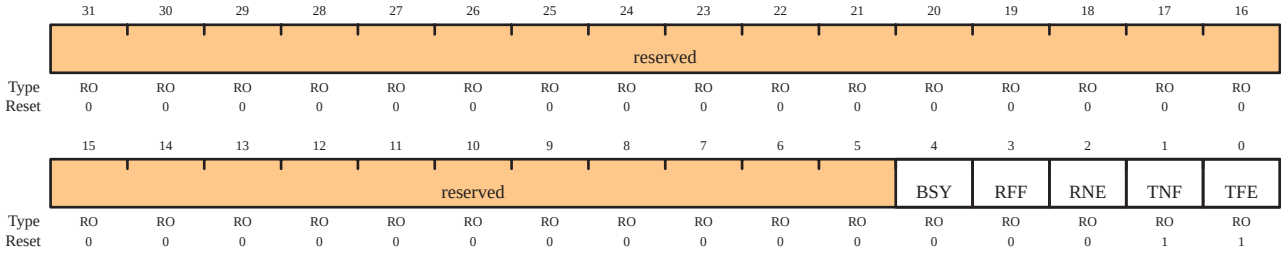



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:0 | DATA | R/W | 0 | SSI Receive/Transmit Data<br>A read operation reads the receive FIFO. A write operation writes the 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)
Offset 0x00C




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 4 | BSY | RO | 0 | SSI Busy Bit<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>0: Receive FIFO is not full.<br>1: Receive FIFO is full. |
| 2 | RNE | RO | 0 | SSI Receive FIFO Not Empty<br>0: Receive FIFO is empty.<br>1: Receive FIFO is not empty. |
| 1 | TNF | RO | 1 | SSI Transmit FIFO Not Full<br>0: Transmit FIFO is full.<br>1: Transmit FIFO is not full. |
| 0 | TFE | R0 | 1 | SSI Transmit FIFO Empty<br>0: Transmit FIFO is not empty.<br>1: Transmit FIFO is empty. |

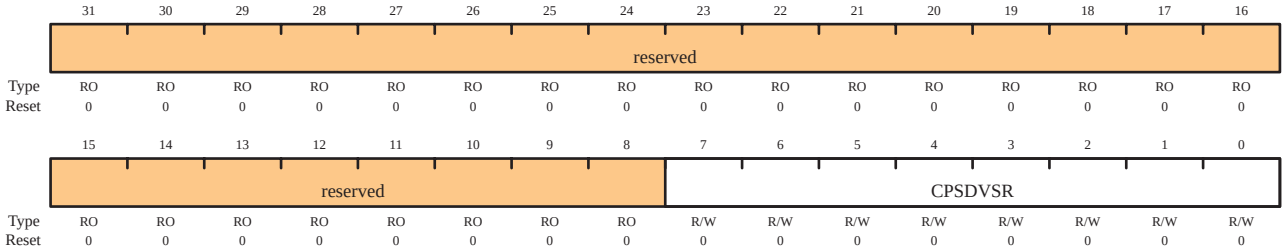
### Register 5: SSI Clock Prescale (SSICPSR), offset 0x010

**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)

Offset 0x010



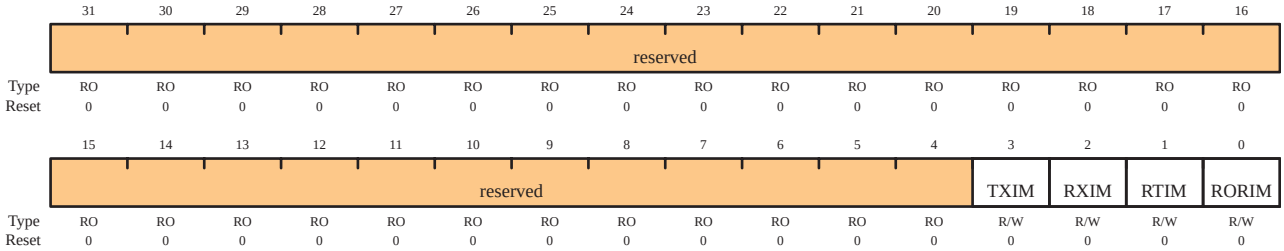

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CPSDVSR | R/W | 0 | SSI Clock Prescale Divisor<br>This value must be an even number from 2 to 254, depending on the 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)
Offset 0x014

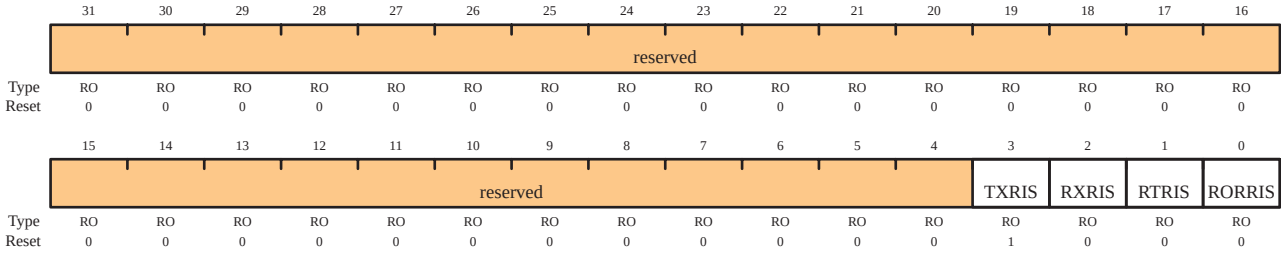



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | TXIM | R/W | 0 | SSI Transmit FIFO Interrupt Mask<br>0: TX FIFO half-full or less condition interrupt is masked.<br>1: TX FIFO half-full or less condition interrupt is not masked. |
| 2 | RXIM | R/W | 0 | SSI Receive FIFO Interrupt Mask<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>0: RX FIFO time-out interrupt is masked.<br>1: RX FIFO time-out interrupt is not masked. |
| 0 | RORIM | R/W | 0 | SSI Receive Overrun Interrupt Mask<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)
Offset 0x018

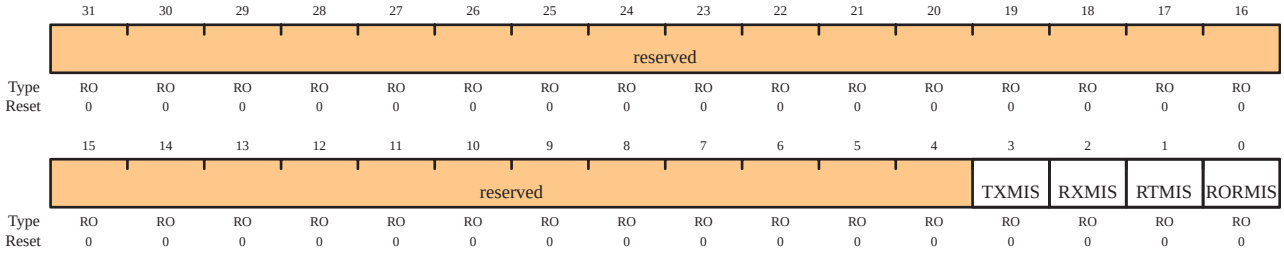



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | TXRIS | RO | 1 | SSI Transmit FIFO Raw Interrupt Status<br>Indicates that the transmit FIFO is half full or less, when 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)
Offset 0x01C

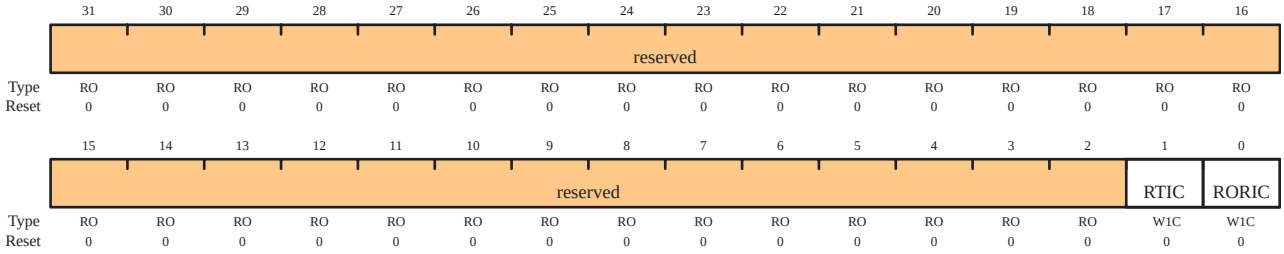



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | TXMIS | RO | 0 | SSI Transmit FIFO Masked Interrupt Status<br>Indicates that the transmit FIFO is half full or less, when 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)
Offset 0x020



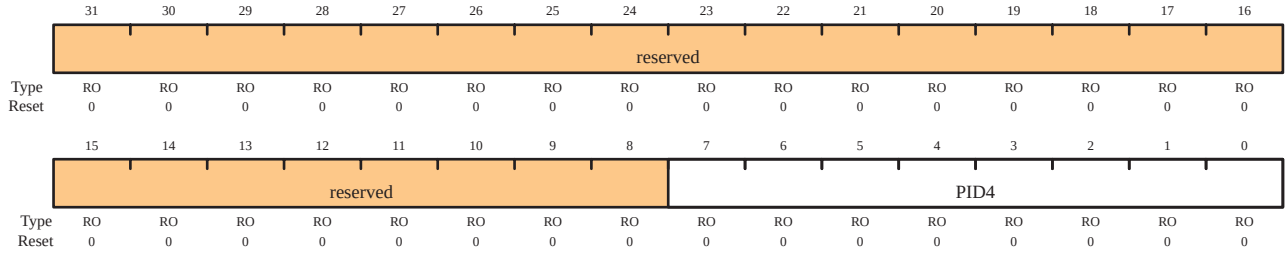

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | RTIC | W1C | 0 | SSI Receive Time-Out Interrupt Clear<br>0: No effect on interrupt.<br>1: Clears interrupt. |
| 0 | RORIC | W1C | 0 | SSI Receive Overrun Interrupt Clear<br>0: No effect on interrupt.<br>1: Clears interrupt. |

### Register 10: 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)

Offset 0xFD0

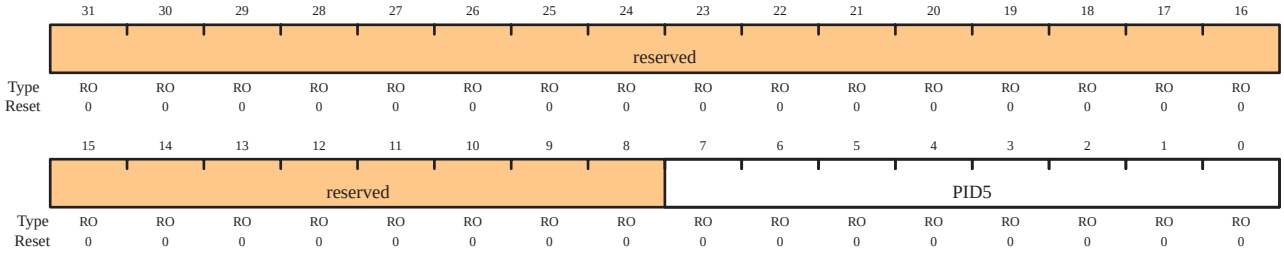



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID4 | RO | 0x00 | SSI Peripheral ID Register[7:0] |

### Register 11: 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)
Offset 0xFD4



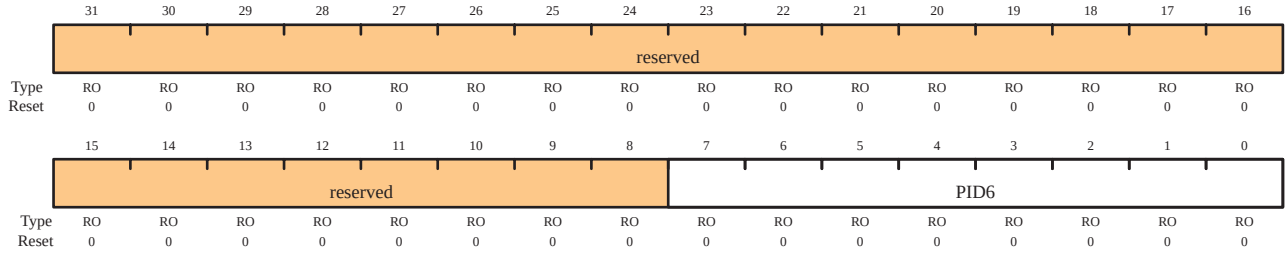

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID5 | RO | 0x00 | SSI Peripheral ID Register[15:8] |

### Register 12: 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)

Offset 0xFD8



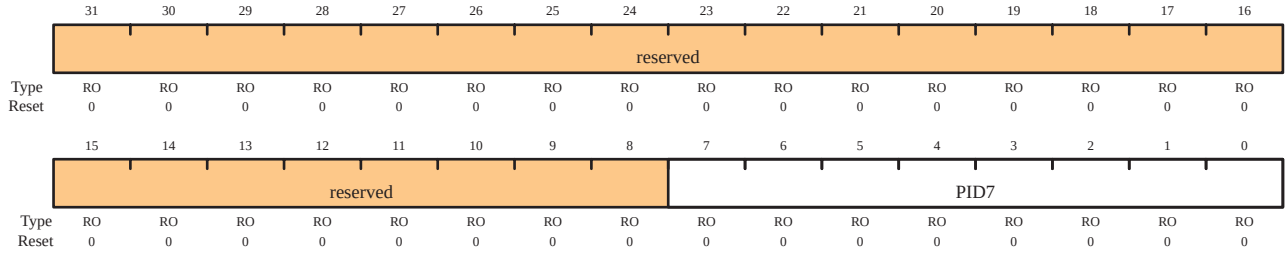

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID6 | RO | 0x00 | SSI Peripheral ID Register[23:16] |

### Register 13: 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)

Offset 0xFDC



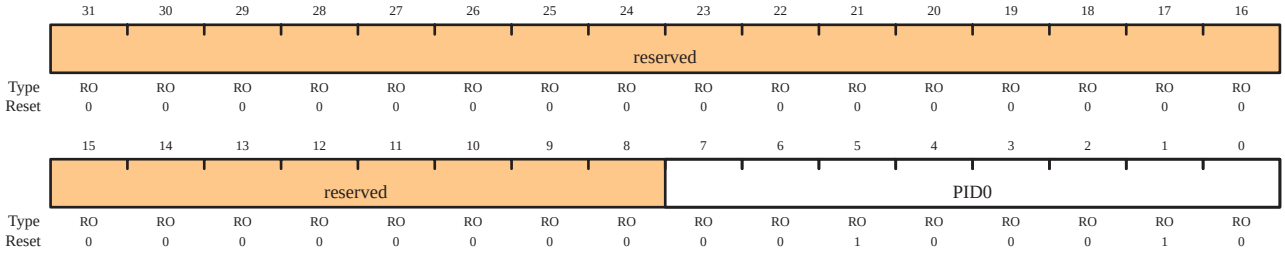

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID7 | RO | 0x00 | SSI Peripheral ID Register[31:24] |

### Register 14: 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)

Offset 0xFEO



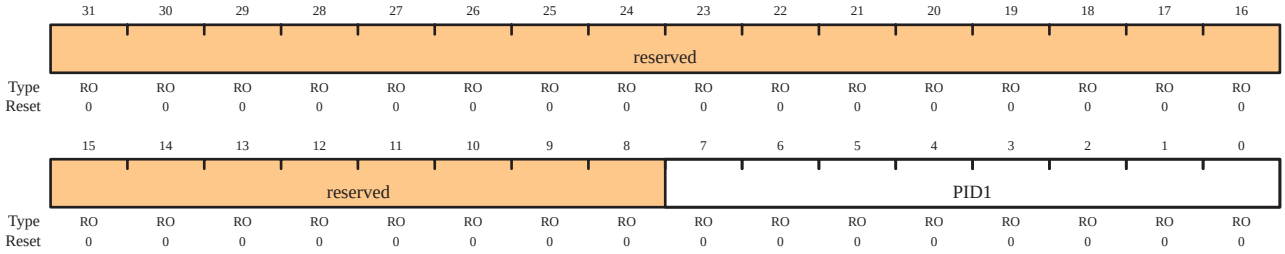

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID0 | RO | 0x22 | SSI Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

### Register 15: 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)

Offset 0xFE4



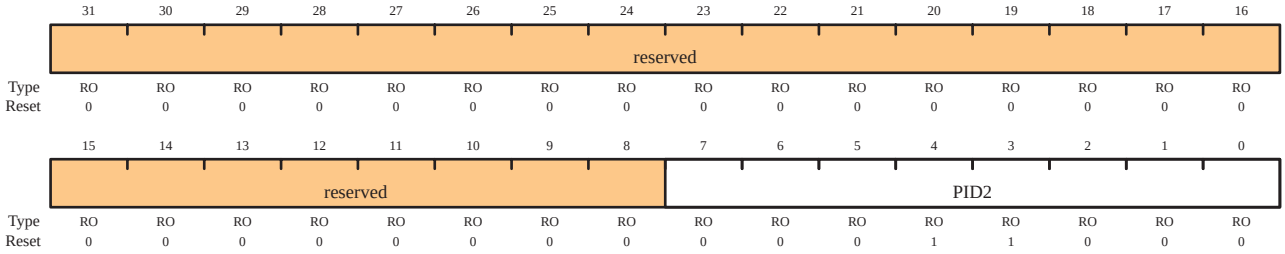

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID1 | RO | 0x00 | SSI Peripheral ID Register [15:8]<br>Can be used by software to identify the presence of this peripheral. |

### Register 16: 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)

Offset 0xFE8



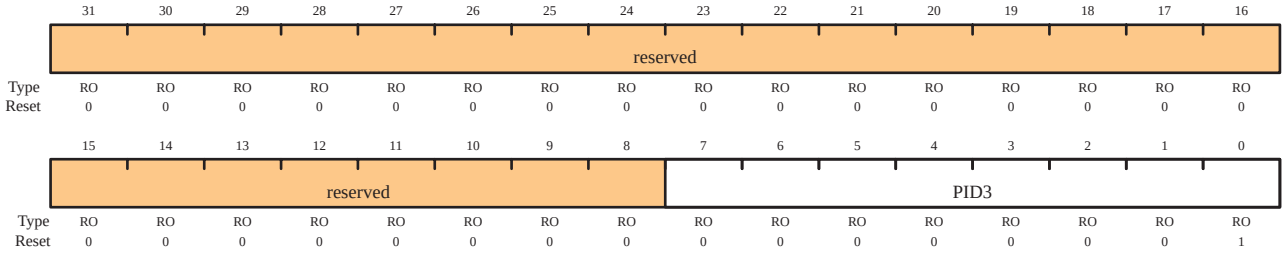

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID2 | RO | 0x18 | SSI Peripheral ID Register [23:16]<br>Can be used by software to identify the presence of this peripheral. |

### Register 17: 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)

Offset 0xFEC

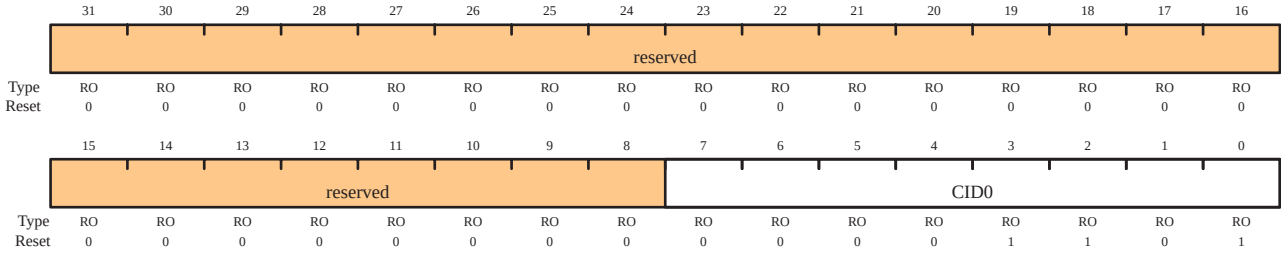



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | PID3 | RO | 0x01 | SSI Peripheral ID Register [31:24]<br>Can be used by software to identify the presence of this peripheral. |

### Register 18: 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)
Offset 0xFF0



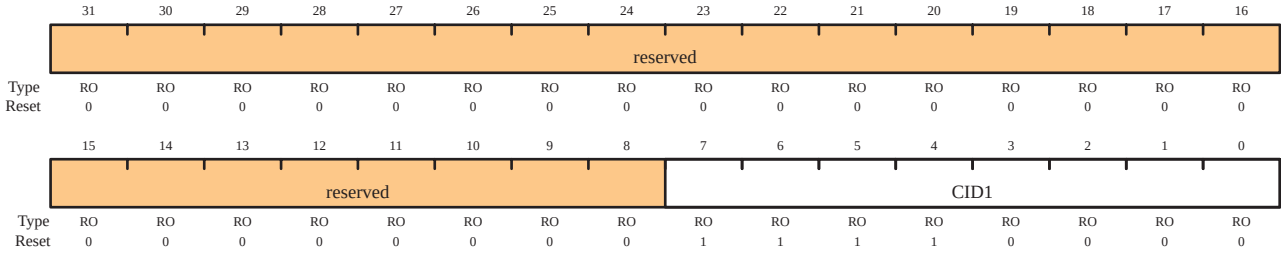

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID0 | RO | 0x0D | SSI PrimeCell ID Register [7:0]<br>Provides software a standard cross-peripheral identification system. |

### Register 19: 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)

Offset 0xFF4

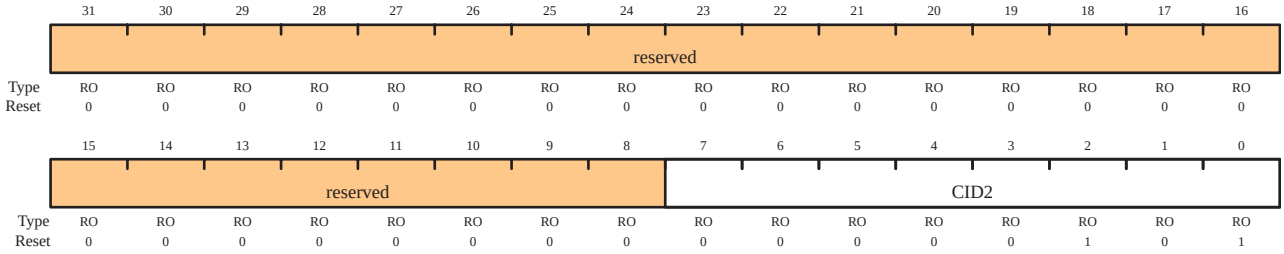



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID1 | RO | 0xF0 | SSI PrimeCell ID Register [15:8]<br>Provides software a standard cross-peripheral identification system. |

### Register 20: 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)
Offset 0xFF8



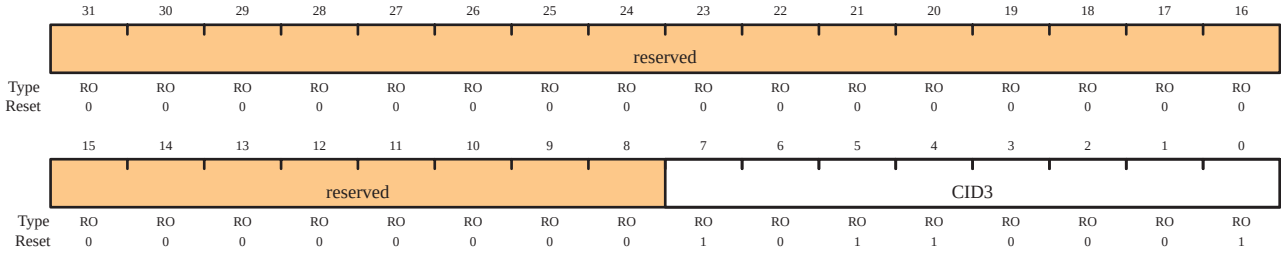

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID2 | RO | 0x05 | SSI PrimeCell ID Register [23:16]<br>Provides software a standard cross-peripheral identification system. |

### Register 21: 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)

Offset 0xFFC




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | CID3 | RO | 0xB1 | SSI PrimeCell ID Register [31:24]<br>Provides software a standard cross-peripheral identification system. |

# 14 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 SDL and a serial clock line SCL).

The $I^2C$ bus 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 Stellaris $I^2C$ module provides the ability to communicate to other IC devices over an $I^2C$ bus. The $I^2C$ bus supports devices that can both transmit and receive (write and read) data.

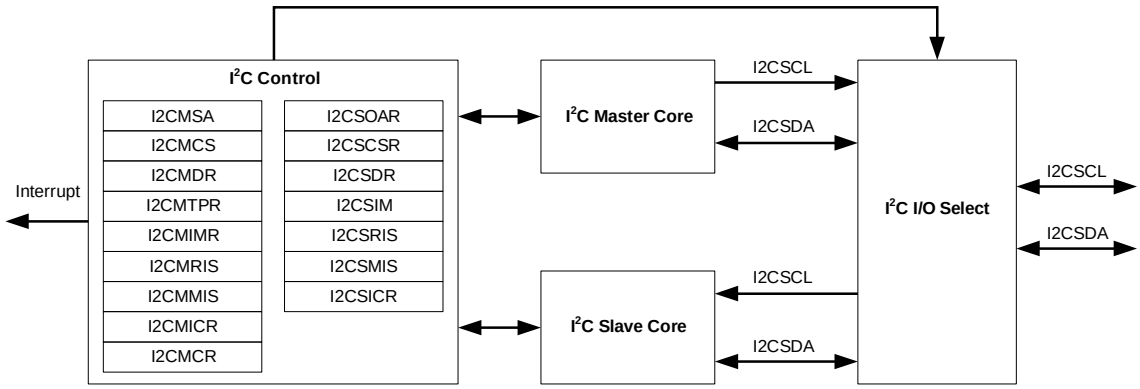
Devices on the $I^2C$ bus can be designated as either a master or a slave. The $I^2C$ module supports both sending and receiving data as either a master or a slave, and also supports the simultaneous operation as both a master and a slave. The four $I^2C$ modes are: Master Transmit, Master Receive, Slave Transmit, and Slave Receive.

The Stellaris $I^2C$ module can operate at two speeds: Standard (100 Kbps) and Fast (400 Kbps).

Both the $I^2C$ master and slave can generate interrupts. The $I^2C$ master generates interrupts when a transmit or receive operation completes (or aborts due to an error). The $I^2C$ slave generates interrupts when data has been sent or requested by a master.

## 14.1 Block Diagram

**Figure 14-1. $I^2C$ Block Diagram**

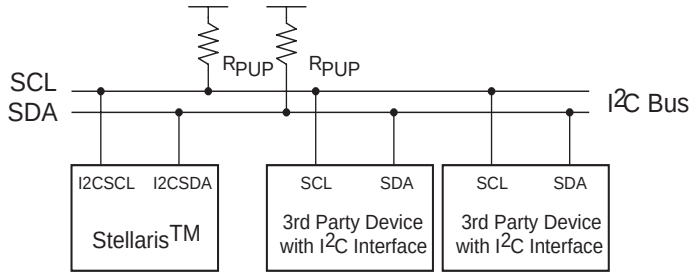



## 14.2 Functional Description

The $I^2C$ module is comprised of both a master and slave function. The master and slave functions are implemented as separate peripherals. The $I^2C$ module must be connected to bi-directional Open-Drain pads. A typical $I^2C$ bus configuration is shown in Figure 14-2.

See “I2C Timing” on page 395 for $I^2C$ timing diagrams.

**Figure 14-2. I²C Bus Configuration**




## 14.2.1 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.

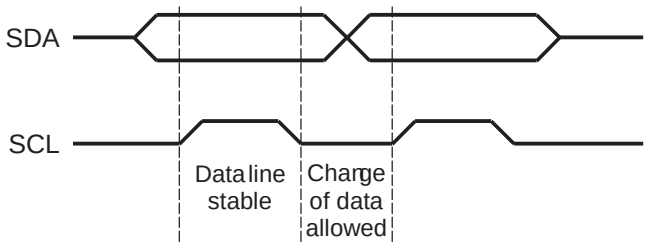
### 14.2.1.1 Data Transfers

Both the SDA and SCL lines are bi-directional, connected to the positive supply via pull-up resistors. The bus is idle or free, when both lines are High. The output devices (pad drivers) must have an open-drain configuration. Data on the I²C bus can be transferred at rates up to 100 Kbps in Standard mode and up to 400 Kbps in Fast mode.

### 14.2.1.2 Data Validity

The data on the SDA line must be stable during the High period of the clock. The data line can only change when the clock SCL is in its Low state (see Figure 14-3).

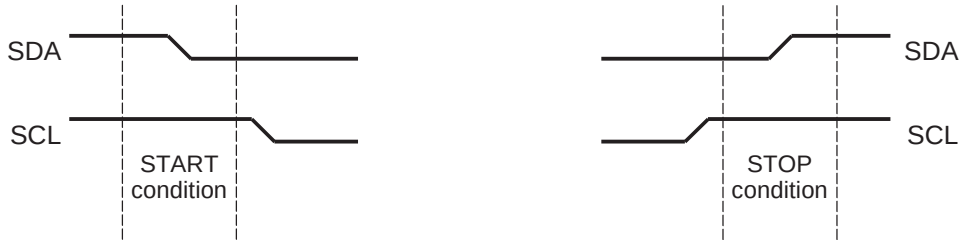
**Figure 14-3. Data Validity During Bit Transfer on the I²C Bus**




### 14.2.1.3 START and STOP Conditions

The protocol of the I²C bus defines two states: START and STOP. A High-to-Low transition on the SDA line while the SCL is High is a START condition. 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. The bus is considered free after a STOP condition. See Figure 14-4.

**Figure 14-4. START and STOP Conditions**



#### 14.2.1.4 Byte Format

Every byte put out on the SDA line must be 8-bits long. The number of bytes per transfer is unrestricted. Each byte has to be followed by an Acknowledge bit. Data is transferred with the 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.

#### 14.2.1.5 Acknowledge

Data transfer with an acknowledge is obligatory. The acknowledge-related clock pulse is generated by the master. The transmitter releases the SDA line during the acknowledge clock pulse.

The receiver must pull down SDA during the acknowledge clock pulse such that it remains stable (Low) during the High period of the acknowledge clock pulse.

When a slave receiver does not acknowledge the slave address, the data line must be left in a High state by the slave. The master can then generate a STOP condition to abort the current transfer.

If the master receiver is involved in the transfer, it must signal the end of data to the slave-transmitter by not generating an acknowledge on the last byte that was clocked out of the slave. The slave-transmitter must release the SDA line to allow the master to generate the STOP or a repeated START condition.

#### 14.2.1.6 Arbitration

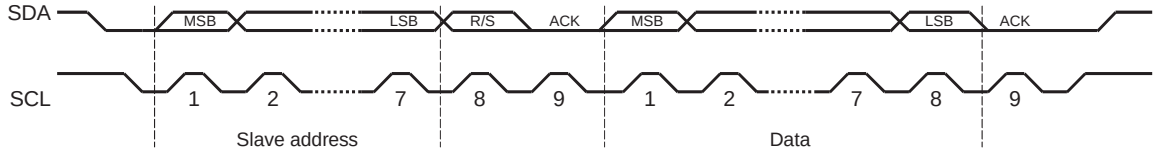
A master may start a transfer only if the bus is idle. Two or more masters may generate a START condition within minimum hold time of the START condition. Arbitration takes place on the SDA line, while SCL is in the High state, in such a manner that the master transmitting a High level (while another master is transmitting a Low level) will switch off its data output stage.

Arbitration can be over several bits. Its first stage is a comparison of address bits. If both masters are trying to address the same device, arbitration continues with comparison of data bits.

#### 14.2.1.7 Data Format with 7-Bit Address

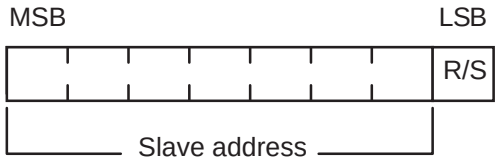
Data transfers follow the format shown in Figure 14-5. After the START condition, a slave address is sent. This address is 7-bits long followed by an eighth bit, which is a data direction bit (R/S bit in the **I2CMSA** register). A zero indicates a transmission (Send); a one indicates a request for data (Receive). A data transfer is always terminated by a STOP condition generated by the master. However, a master can still communicate 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 such a transfer.

**Figure 14-5. Complete Data Transfer with a 7-Bit Address**




The first seven bits of the first byte make up the slave address (see Figure 14-6). 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) information to a selected slave. A one in this position means that the master will receive information from the slave.

**Figure 14-6. R/S Bit in First Byte**




#### 14.2.1.8 I²C Master Command Sequences

Figure 14-7 through Figure 14-12 present the command sequences available for the I²C master.

**Figure 14-7. Master Single SEND**

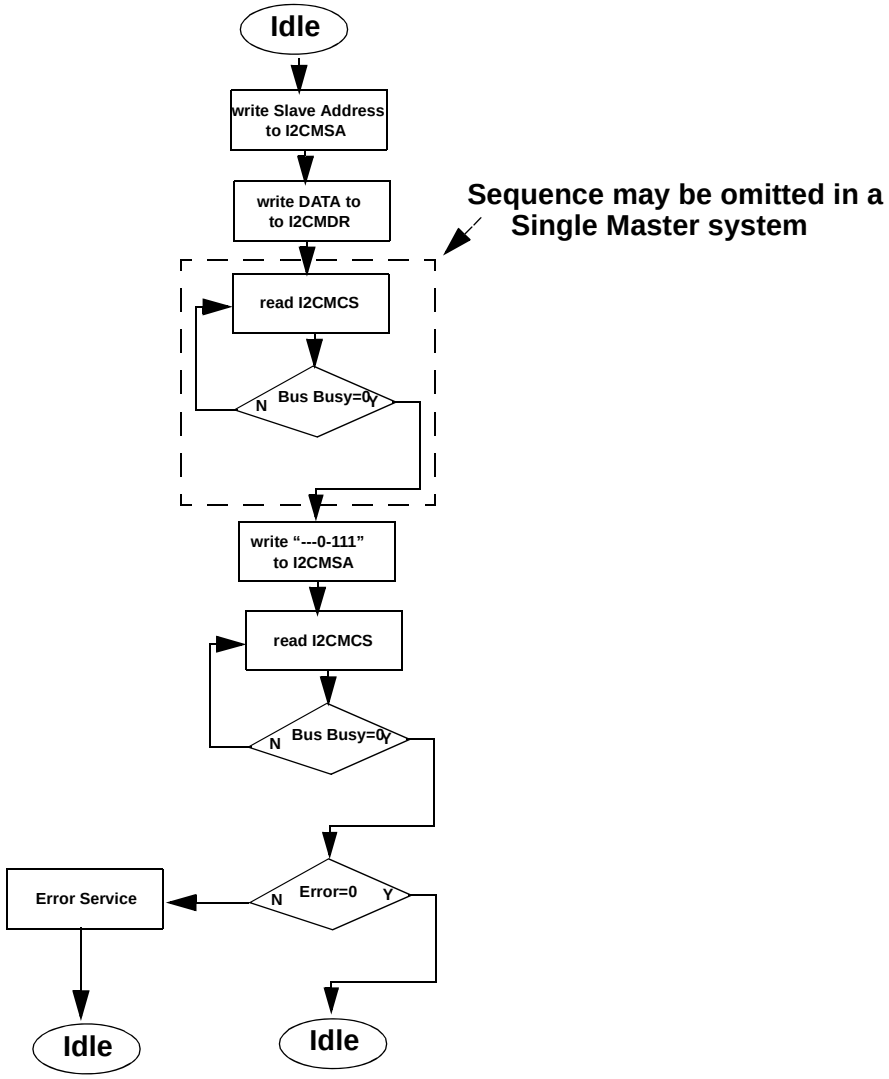


**Figure 14-8. Master Single RECEIVE**

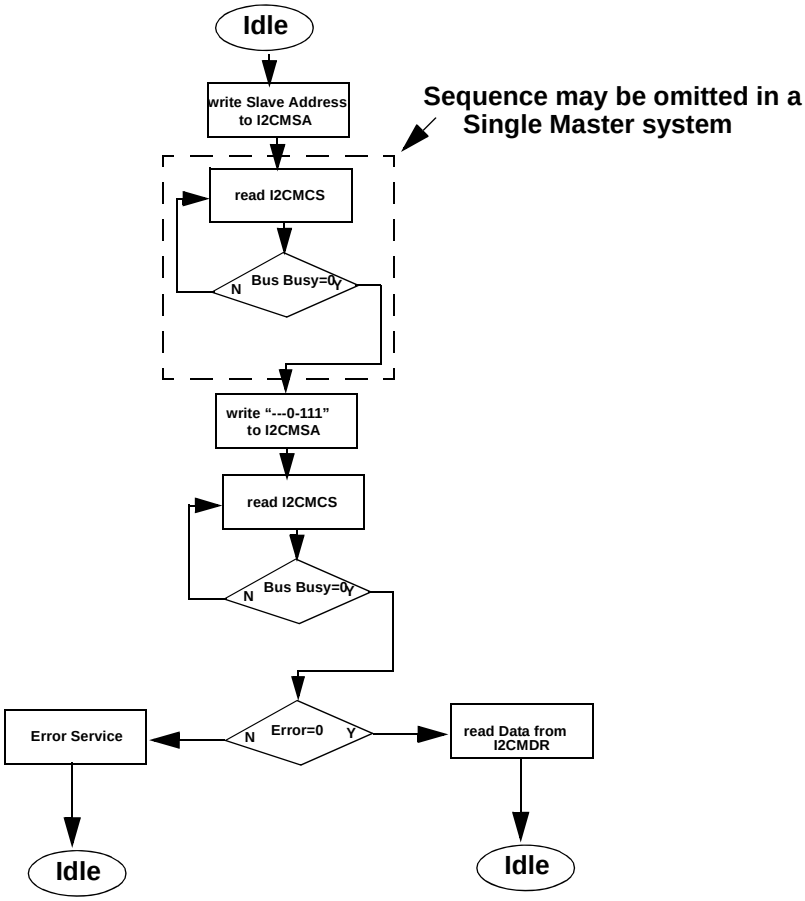


**Figure 14-9. Master Burst SEND**

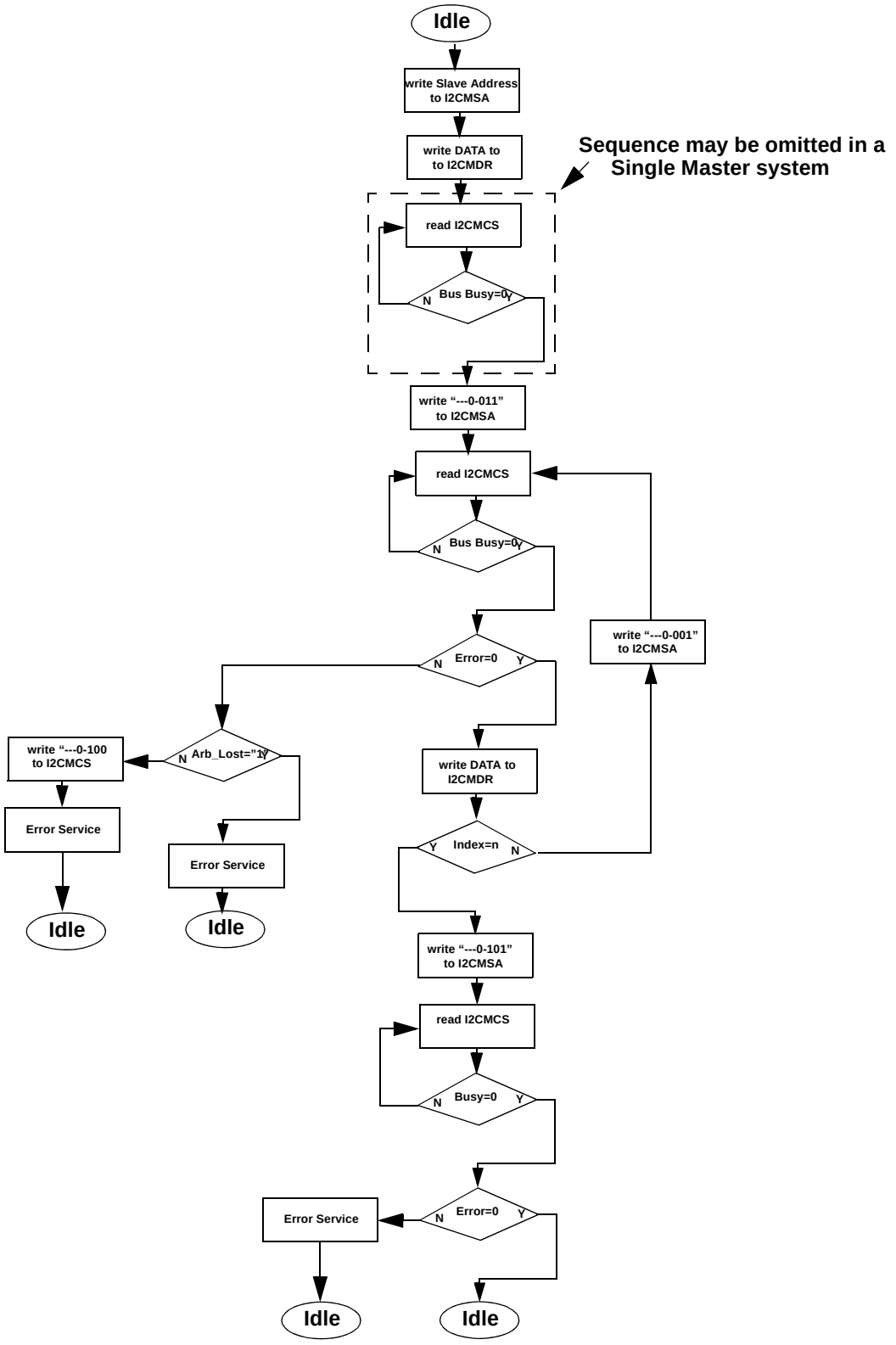


**Figure 14-10. Master Burst RECEIVE**

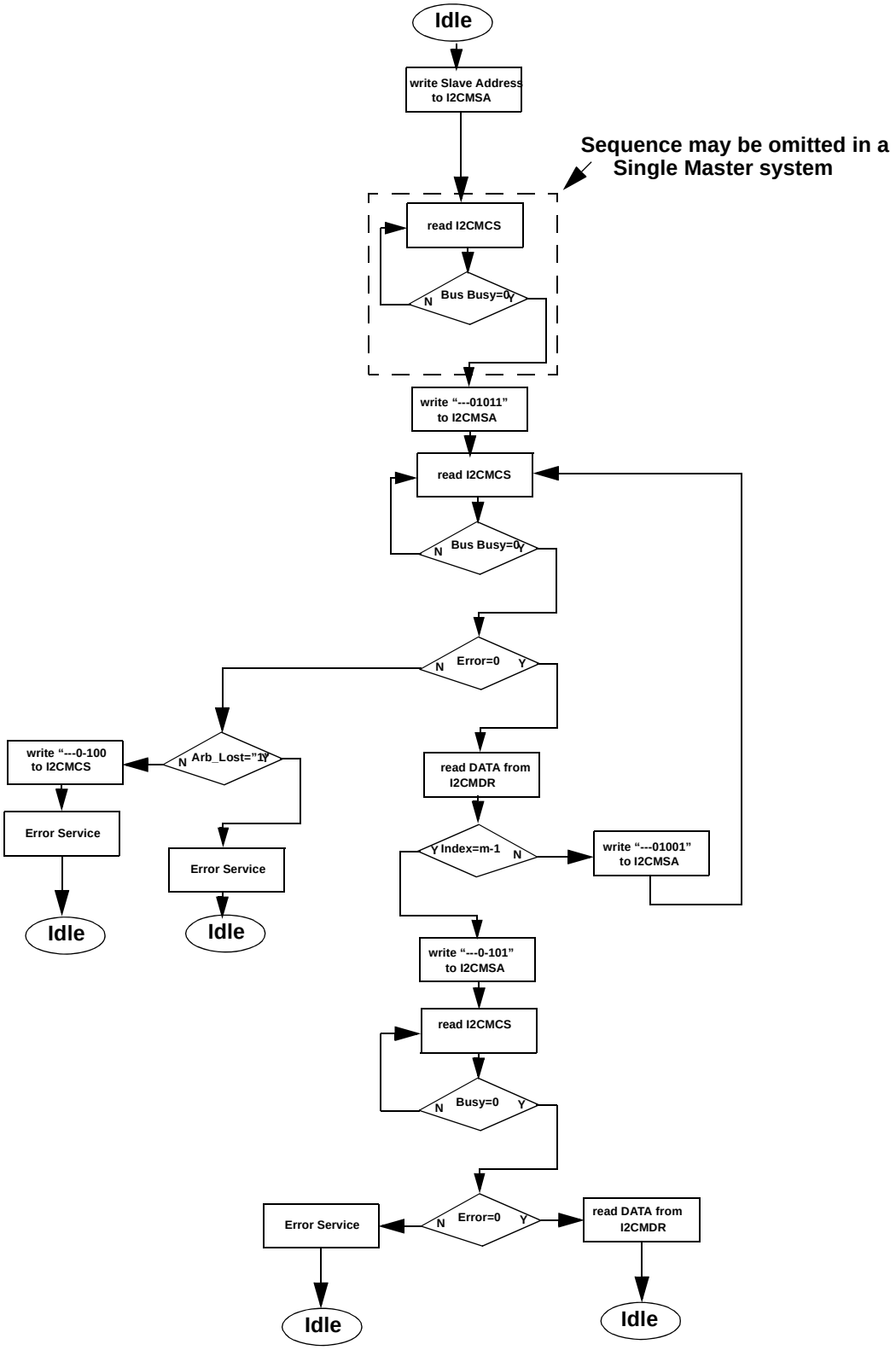


**Figure 14-11. Master Burst RECEIVE after Burst SEND**

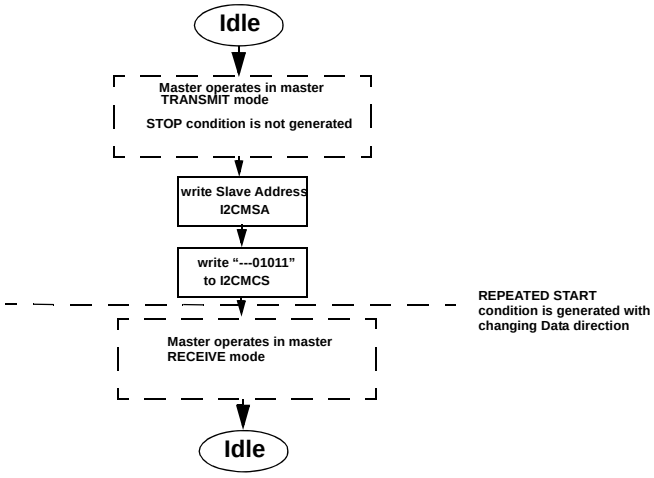
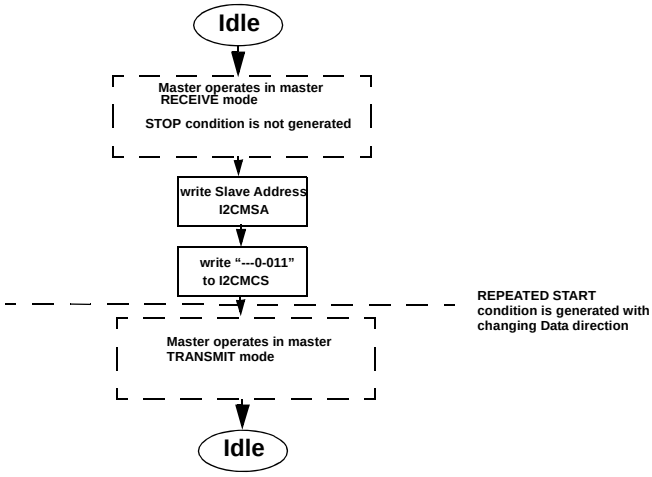


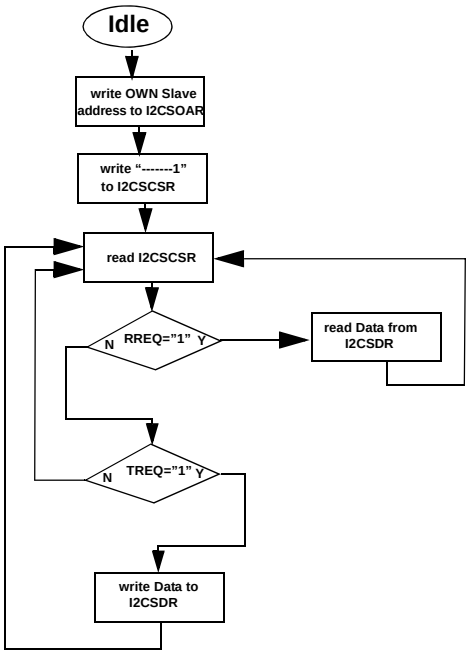

**Figure 14-12. Master Burst SEND after Burst RECEIVE**




#### 14.2.1.9 I²C Slave Command Sequences

Figure 14-13 presents the command sequence available for the I²C slave.

**Figure 14-13. Slave Command Sequence**




## 14.2.2 Available Speed Modes

The SCL 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 the SCL clock (fixed at 6)

SCL_HP is the High phase of the SCL clock (fixed at 4)

TIMER_PRD is the programmed value in the **I2C Master Timer Period (I2CMTPR)** register (see page 320).

The SCL 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 14-1 gives examples of Timer period, system clock, and speed mode (Standard or Fast).

**Table 14-1. 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 |

## 14.3 Initialization and Configuration

The following example shows how to configure the I²C module to send a single byte as a master. This assumes the system clock is 20 MHz.

1. Enable the I²C clock by writing a value of 0x00001000 to the **RCGC1** register in the System Control module.
2. 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.
3. Initialize the I²C Master by writing the **I2CMCR** register with a value of 0x00000020.
4. 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 0x00000009.
5. 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 0x00000076. This sets the slave address to 0x3B.
6. Place data (byte) to be sent in the data register by writing the **I2CMDR** register with the desired data.
7. Initiate a single byte send of the data from Master to Slave by writing the **I2CMCS** register with a value of 0x00000007 (STOP, START, RUN).
8. Wait until the transmission completes by polling the **I2CMCS** register's `BUSBSY` bit until it has been cleared.

## 14.4 Register Map

Table 14-2 lists the I²C registers. All addresses given are relative to the I²C base addresses for the master and slave:

- I²C Master: 0x40020000

- I$^2$C Slave: 0x40020800

**Table 14-2. I$^2$C Register Map**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x000 | I2CMSA | 0x00000000 | R/W | Master slave address | 313 |
| 0x004 | I2CMCS | 0x00000000 | R/W | Master control/status | 314 |
| 0x008 | I2CMDR | 0x00000000 | R/W | Master data | 319 |
| 0x00C | I2CMTPR | 0x00000001 | R/W | Master timer period | 320 |
| 0x010 | I2CMIMR | 0x00000000 | R/W | Master interrupt mask | 321 |
| 0x014 | I2CMRIS | 0x00000000 | RO | Master raw interrupt status | 322 |
| 0x018 | I2CMMIS | 0x00000000 | RO | Master masked interrupt status | 322 |
| 0x01C | I2CMICR | 0x00000000 | WO | Master interrupt clear | 323 |
| 0x020 | I2CMCR | 0x00000000 | R/W | Master configuration | 324 |
| 0x000 | I2CSOAR | 0x00000000 | R/W | Slave address | 326 |
| 0x004 | I2CSCSR | 0x00000000 | RO | Slave control/status | 327 |
| 0x008 | I2CSDR | 0x00000000 | R/W | Slave data | 329 |
| 0x00C | I2CSIMR | 0x00000000 | R/W | Slave interrupt mask | 330 |
| 0x010 | I2CSRIS | 0x00000000 | RO | Slave raw interrupt status | 331 |
| 0x014 | I2CSMIS | 0x00000000 | RO | Slave masked interrupt status | 332 |
| 0x018 | I2CSICR | 0x00000000 | WO | Slave interrupt clear | 333 |

## 14.5 Register Descriptions (I$^2$C Master)

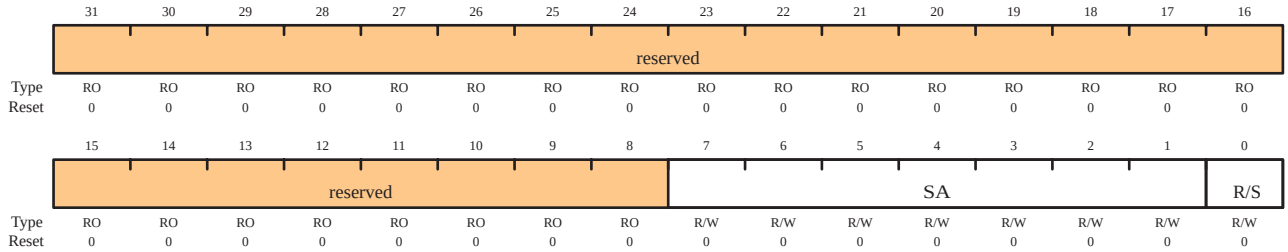
The remainder of this section lists and describes the I$^2$C master registers, in numerical order by address offset. See also “Register Descriptions (I2C Slave)” on page 326.

### 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)

Offset 0x000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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>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.

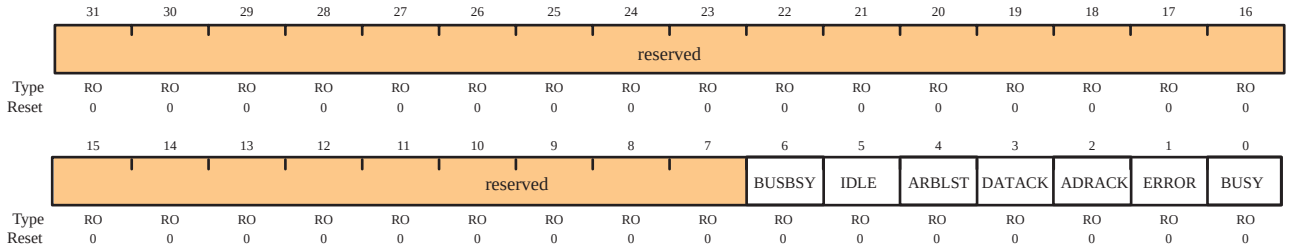
The control register consists of four bits: the RUN, START, STOP, and ACK bits.

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.

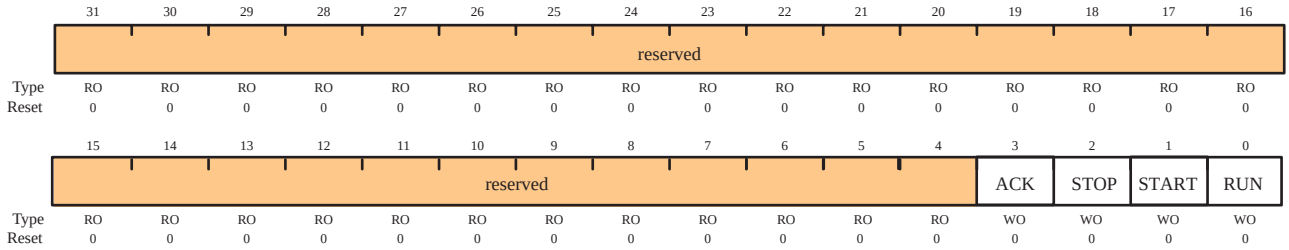
I2C Master Status (I2CMCS): Read

Offset 0x004




I2C Master Control (I2CMCS): Write

Offset 0x004




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| **Read-Only Status Register** | | | | |
| 31:7 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 6 | BUSBSY | R | 0 | 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 | R | 0 | This bit specifies the I²C controller state. If set, the controller is idle; otherwise the controller is not idle. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | ARBLST | R | 0 | This bit specifies the result of bus arbitration. If set, the controller lost arbitration; otherwise, the controller won arbitration. |
| 3 | DATACK | R | 0 | 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 | R | 0 | 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 | R | 0 | 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, the transmit data not being acknowledged, or because the controller lost arbitration. |
| 0 | BUSY | R | 0 | 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. |

**Write-Only Control Register**

| | | | | |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 6-4 | reserved | W | 0 | Write reserved. |
| 3 | ACK | W | 0 | When set, causes received data byte to be acknowledged automatically by the master. See field decoding in Table 14-3 on page 316. |
| 2 | STOP | W | 0 | When set, causes the generation of the STOP condition. See field decoding in Table 14-3. |
| 1 | START | W | 0 | When set, causes the generation of a START or repeated START condition. See field decoding in Table 14-3. |
| 0 | RUN | W | 0 | When set, allows the master to send or receive data. See field decoding in Table 14-3. |

**Table 14-3. 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. |

**Table 14-3. Write Field Decoding for I2CMCS[3:0] Field (Sheet 2 of 3)**

| Current State | I2CMSA[0] | I2CMCS[3:0] | | | | Description |
|---|---|---|---|---|---|---|
| | R/S | ACK | STOP | START | RUN | |
| 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 14-3. Write Field Decoding for I2CMCS[3:0] Field (Sheet 3 of 3)**

| 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. |

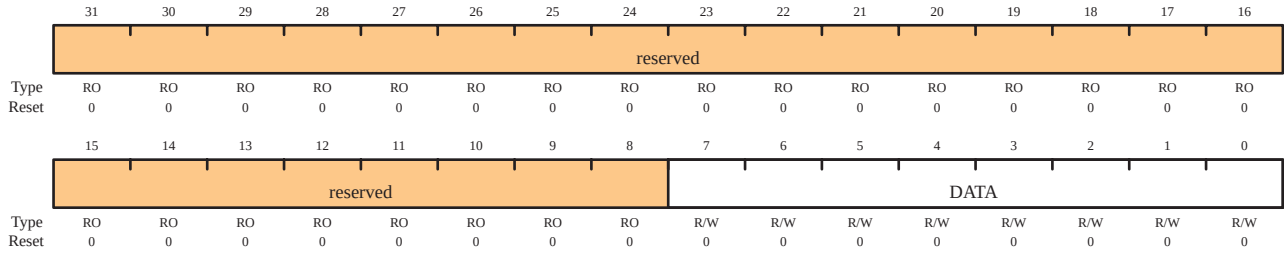
a. An X in a table cell indicates that applies to a bit set to 0 or 1.
b. In Master Receive mode, a STOP condition should be generated only after a Data Negative Acknowledge executed by the master or an Address Negative Acknowledge executed by the slave.

### Register 3: I²C Master Data (I2CMDR), offset 0x008

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)

Offset 0x008



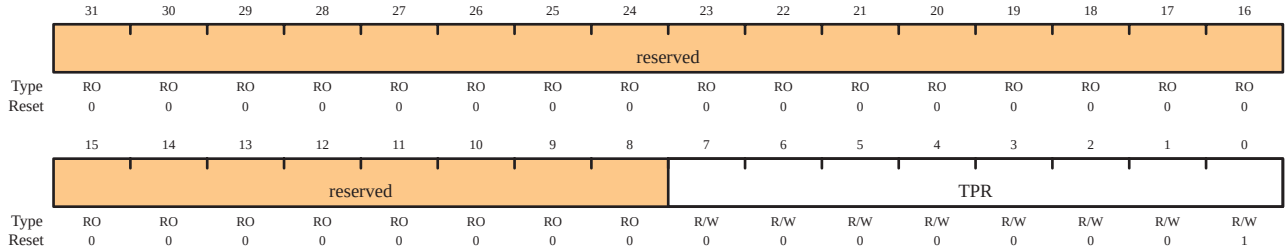

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | DATA | R/W | 0x00 | Data transferred during transaction. |

### Register 4: I²C Master Timer Period (I2CMTPR), offset 0x00C

This register specifies the period of the SCL clock

I2C Master Timer Period (I2CMTPR)

Offset 0x00C



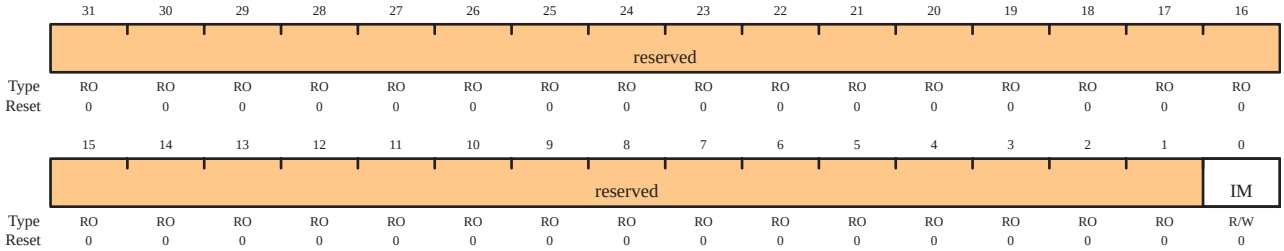

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | TPR | R/W | 0x1 | This field specifies the period of the SCL clock.<br>`SCL_PRD = 2*(1 + TPR)*(SCL_LP + SCL_HP)*CLK_PRD`<br>where:<br>`SCL_PRD` is the SCL line period (I²C clock).<br>`TPR` is the Timer Period register value (range of 1 to 255).<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)

Offset 0x010



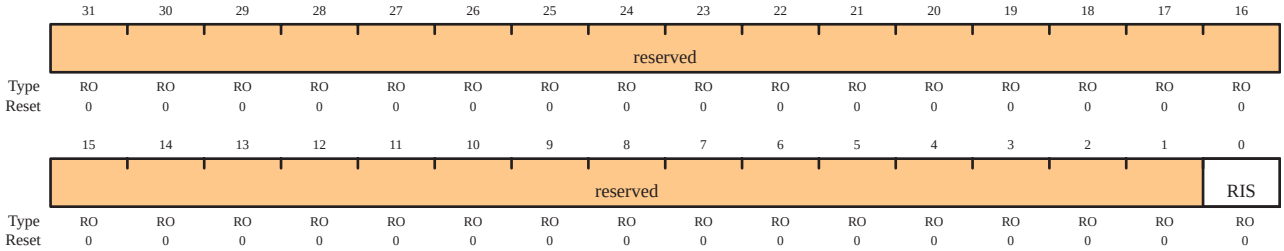

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | IM | R/W | 0 | This bit controls whether a raw interrupt is promoted to a controller interrupt. If set, the interrupt 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)

Offset 0x014



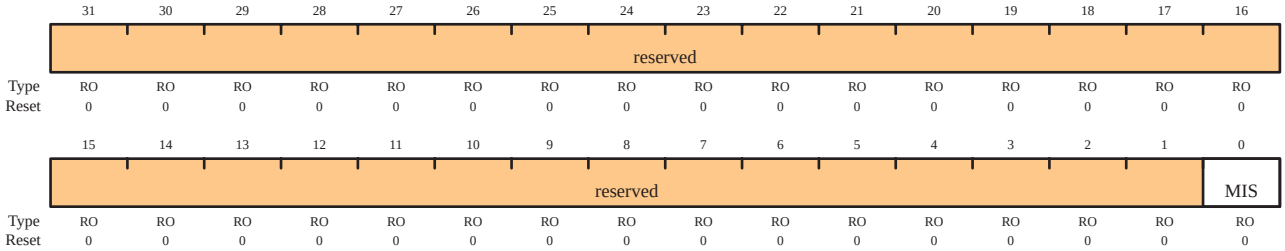

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | RIS | RO | 0 | This bit specifies the raw interrupt state (prior to masking) of the I²C master block. If set, an interrupt 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)

Offset 0x018

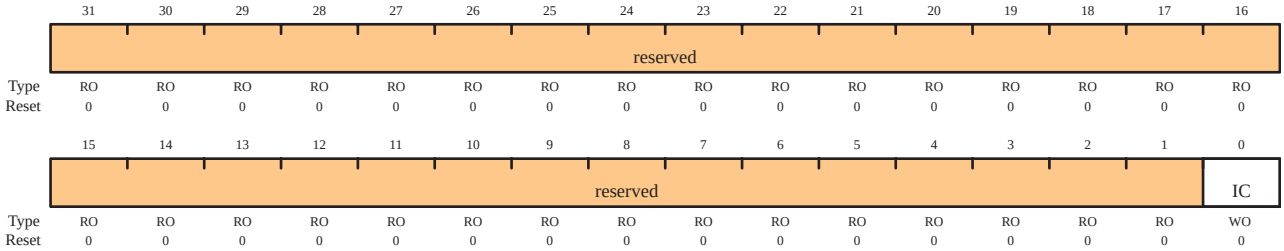



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | MIS | RO | 0 | This bit specifies the raw interrupt state (after masking) of the I²C master block. If set, an interrupt 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)
Offset 0x01C

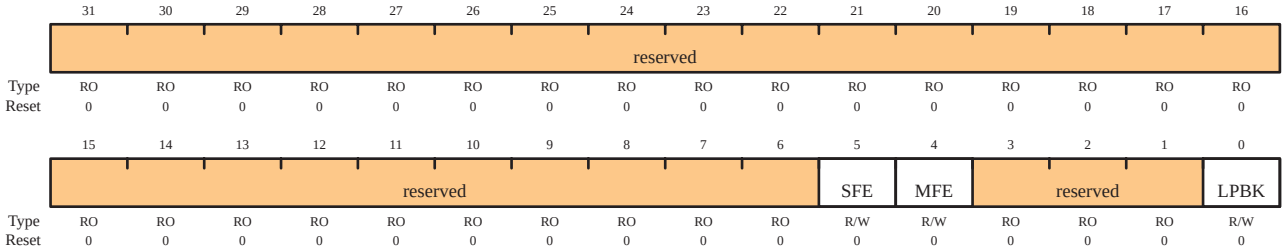



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | IC | WO | 0 | Interrupt Clear<br>This bit controls the clearing of the raw interrupt. A write of 1 clears the interrupt; 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)
Offset 0x020




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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 | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 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. |

## 14.6 Register Descriptions ($I^2C$ Slave)

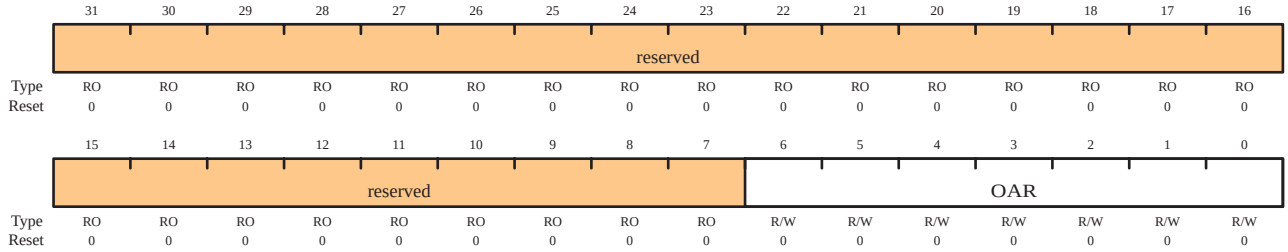
The remainder of this section lists and describes the $I^2C$ slave registers, in numerical order by address offset. See also “Register Descriptions (I2C Master)” on page 312.

### Register 10: $I^2C$ Slave Own Address (I2CSOAR), offset 0x000

This register consists of seven address bits that identify the Stellaris $I^2C$ device on the $I^2C$ bus.

I2C Slave Own Address Register (I2CSOAR)

Offset 0x000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 6:0 | OAR | R/W | 0 | $I^2C$ Slave Own Address<br>This field specifies bits A6 through A0 of the slave address. |

### Register 11: I²C Slave Control/Status (I2CSCSR), offset 0x004

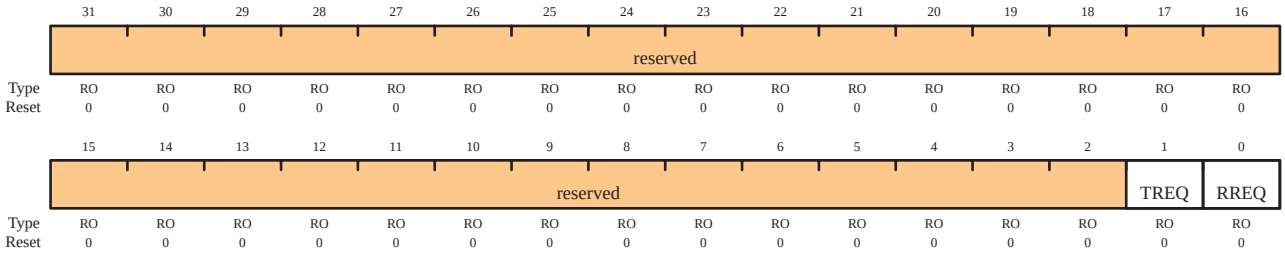
This register accesses one control bit when written, and two status bits when read.

The read-only Status register consists of two bits: the `RREQ` bit and the `TREQ` bit. 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. 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.

The write-only Control register consists of one bit: the `DA` bit. The `DA` bit enables and disables the Stellaris I²C slave operation.

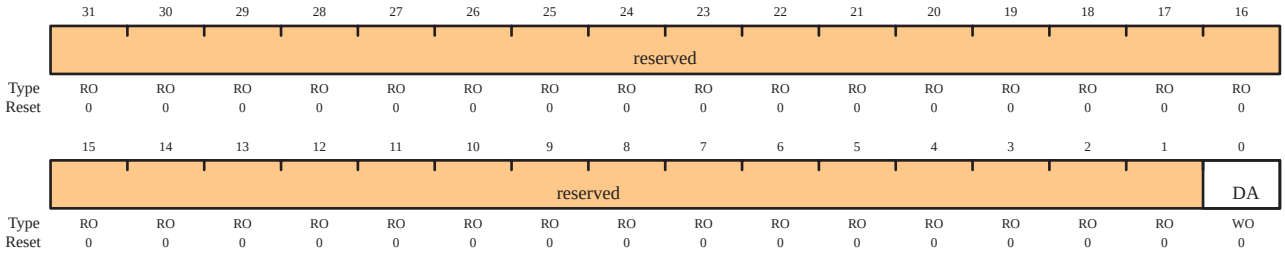
I2C Slave Status Register (I2CSCSR): Read

Offset 0x004




I2C Slave Control Register (I2CSCSR): Write

Offset 0x004




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| **Read-Only Status Register** | | | | |
| 31:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | TREQ | RO | 0 | 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. |

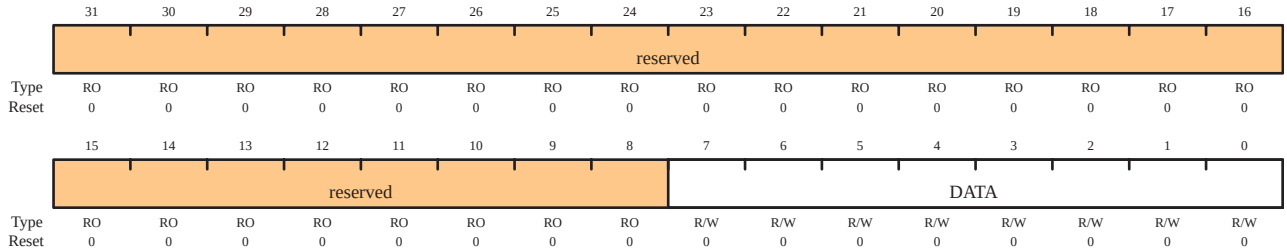
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | RREQ | RO | 0 | Receive Request<br>This bit specifies the status of the $I^2C$ slave with regards to outstanding receive requests. If set, the $I^2C$ unit has outstanding receive data from the $I^2C$ master and uses clock stretching to delay the master until the data has been read from the **I2CSDR** register. Otherwise, no receive data is outstanding. |
| **Write-Only Control Register** | | | | |
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | DA | WO | 0 | Device Active<br>1=Enables the $I^2C$ slave operation.<br>0=Disables the $I^2C$ slave operation. |

### Register 12: I²C Slave Data (I2CSDR), offset 0x008

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)

Offset 0x008



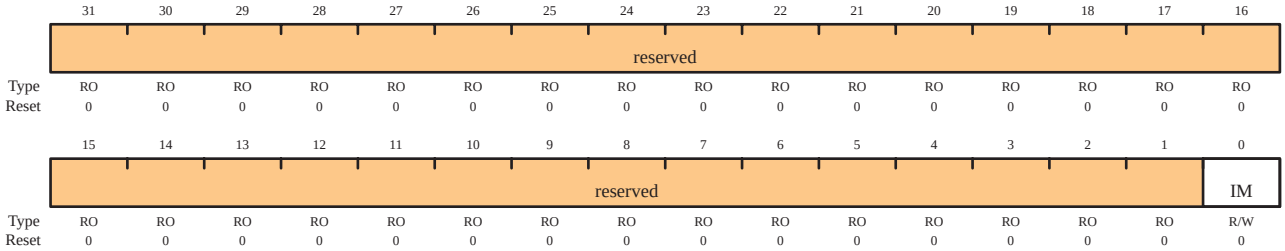

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7:0 | DATA | R/W | 0x0 | This field contains the data for transfer during a slave receive or transmit operation. |

### Register 13: I²C Slave Interrupt Mask (I2CSIMR), offset 0x00C

This register controls whether a raw interrupt is promoted to a controller interrupt.

I2C Slave Interrupt Mask (I2CSIMR)

Offset 0x00C



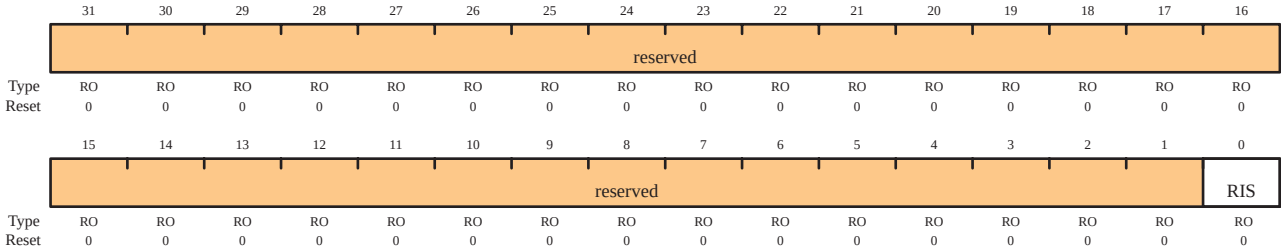

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | IM | R/W | 0 | This bit controls whether a raw interrupt is promoted to a controller interrupt. If set, the interrupt is not masked and the interrupt is promoted; otherwise, the interrupt is masked. |

### Register 14: I²C Slave Raw Interrupt Status (I2CSRIS), offset 0x010

This register specifies whether an interrupt is pending.

I2C Slave Raw Interrupt Status (I2CSRIS)

Offset 0x010



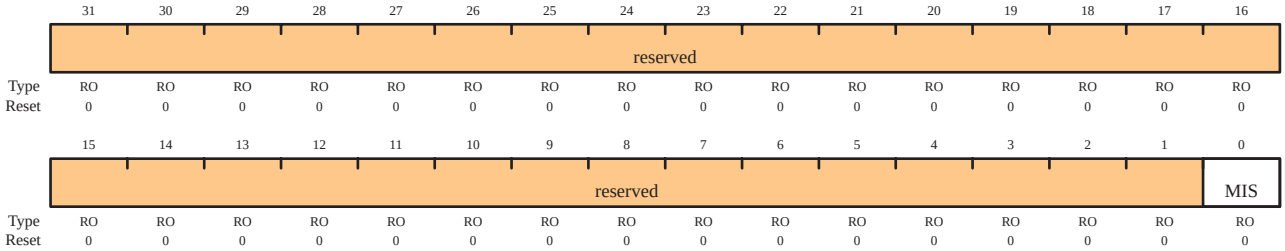

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | RIS | RO | 0 | This bit specifies the raw interrupt state (prior to masking) of the I²C slave block. If set, an interrupt is pending; otherwise, an interrupt is not pending. |

### Register 15: I$^2$C Slave Masked Interrupt Status (I2CSMIS), offset 0x014

This register specifies whether an interrupt was signaled.

I2C Slave Masked Interrupt Status (I2CSMIS)

Offset 0x014



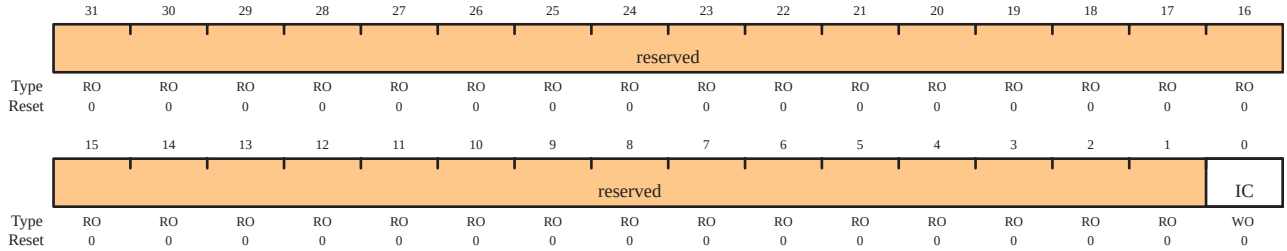

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | MIS | RO | 0 | This bit specifies the raw interrupt state (after masking) of the I$^2$C slave block. If set, an interrupt was signaled; otherwise, an interrupt has not been generated since the bit was last cleared. |

### Register 16: I²C Slave Interrupt Clear (I2CSICR), offset 0x018

This register clears the raw interrupt.

I2C Slave Interrupt Clear (I2CSICR)

Offset 0x018




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | IC | WO | 0 | 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. |

# 15 Analog Comparator

An analog comparator is a peripheral that compares two analog voltages, and provides a logical output that signals the comparison result.

The LM3S316 controller provides one analog comparator that can be configured to drive an output or generate an interrupt or ADC event.

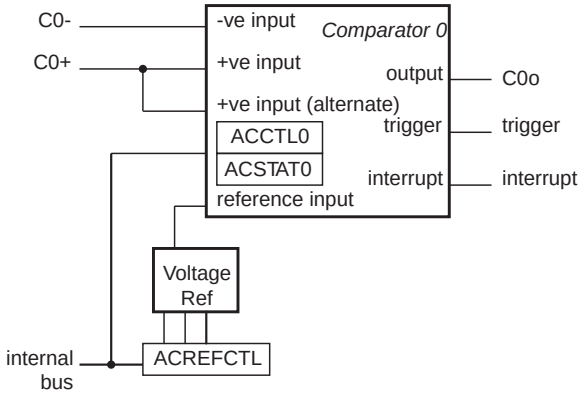
A comparator can compare a test voltage against any one of these voltages:

- An individual external reference voltage
- A shared single external reference voltage
- A shared internal reference voltage

The comparator can provide its output to a device pin, acting as a replacement for an analog comparator on the board, or it can be used to signal the application via interrupts or triggers to the ADC to cause it to start capturing a sample sequence. The interrupt generation and ADC triggering logic is separate. This means, for example, that an interrupt can be generated on a rising edge and the ADC triggered on a falling edge.

## 15.1 Block Diagram

**Figure 15-1. Analog Comparator Module Block Diagram**



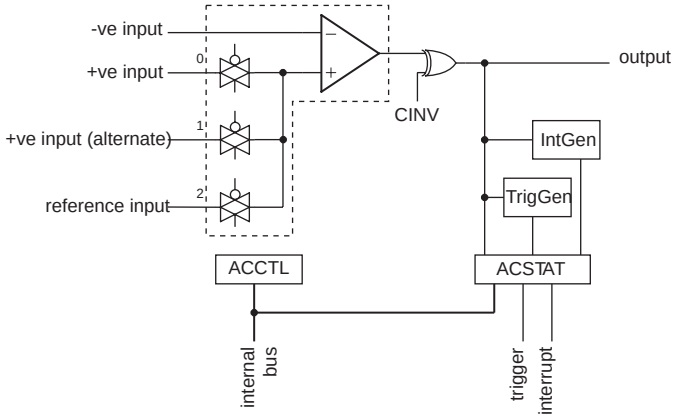

## 15.2 Functional Description

**Important:** It is recommended that the Digital-Input enable (the GPIODEN bit in the GPIO module) for the analog input pin be disabled to prevent excessive current draw from the I/O pads.

The comparator compares the VIN- and VIN+ inputs to produce an output, VOUT.

As shown in Figure 15-2, the input source for VIN- is an external input. In addition to an external input, input sources for VIN+ can be the +ve input of comparator 0 or an internal reference.

**Figure 15-2. Structure of Comparator Unit**




A comparator is configured through two status/control registers (**ACCTL** and **ACSTAT**). The internal reference is configured through one control register (**ACREFCTL**). Interrupt status and control is configured through three registers (**ACMIS**, **ACRIS**, and **ACINTEN**). The operating modes of the comparators are shown in Table 15-1.

Typically, the comparator output is used internally to generate controller interrupts. It may also be used to drive an external pin or generate an analog-to-digital converter (ADC) trigger.

**Important:** Certain register bit values must be set before using the analog comparators. The proper pad configuration for the comparator input and output pins are described in Table 8-1 on page 113.

**Table 15-1. Comparator 0 Operating Modes**

| ACCNTL0 | Comparator 0 | | | | |
|---|---|---|---|---|---|
| **ASRCP** | **VIN-** | **VIN+** | **Output** | **Interrupt** | **ADC Trigger** |
| 00 | C0- | C0+ | C0o | yes | yes |
| 01 | C0- | C0+ | C0o | yes | yes |
| 10 | C0- | Vref | C0o | yes | yes |
| 11 | C0- | reserved | C0o | yes | yes |

## 15.2.1 Internal Reference Programming

The structure of the internal reference is shown in Figure 15-3. This is controlled by a single configuration register (**ACREFCTL**). Table 15-2 shows the programming options to develop specific internal reference values, to compare an external voltage against a particular voltage generated internally.

**Figure 15-3. Comparator Internal Reference Structure**

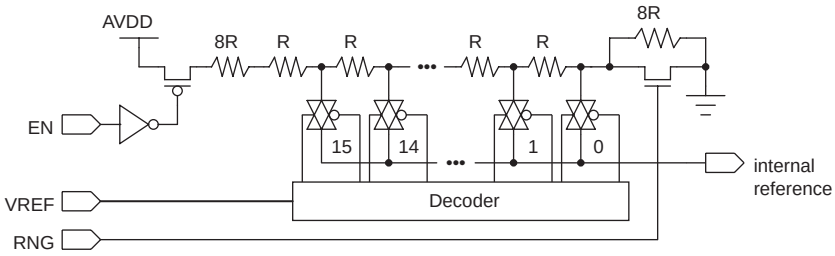



**Table 15-2. Internal Reference Voltage and ACREFCTL Field Values**

| ACREFCTL Register | | Output Reference Voltage Based on VREF Field Value |
|---|---|---|
| EN Bit Value | RNG Bit Value | |
| EN=0 | RNG=X | 0 V (GND) for any value of VREF; however, it is recommended that RNG=1 and VREF=0 for the least noisy ground reference. |
| EN=1 | RNG=0 | Total resistance in ladder is 32 R.<br>$V_{REF} = AV_{DD} \times \frac{R_{VREF}}{R_T}$<br>$V_{REF} = AV_{DD} \times \frac{(VREF + 8)}{32}$<br>$V_{REF} = 0.825 + 0.103 \cdot VREF$<br>The range of internal reference in this mode is 0.825–2.37 V. |
| | RNG=1 | Total resistance in ladder is 24 R.<br>$V_{REF} = AV_{DD} \times \frac{R_{VREF}}{R_T}$<br>$V_{REF} = AV_{DD} \times \frac{(VREF)}{24}$<br>$V_{REF} = 0.1375 \cdot VREF$<br>The range of internal reference for this mode is 0.0–2.0625 V. |

## 15.3 Initialization and Configuration

The following example shows how to configure analog comparator to read back its output value from an internal register.

1. Enable the analog comparator 0 clock by writing a value of 0x00100000 to the **RCGC1** register in the System Control module.
2. In the GPIO module, enable the GPIO port/pin associated with `C0-` as a GPIO input.
3. Configure the internal voltage reference to 1.65 V by writing the **ACREFCTL** register with the value 0x0000030C.
4. Configure comparator 0 to use the internal voltage reference and to *not* output a value on the `C0O` pin by writing the **ACCTL0** register with the value of 0x0000040C.

5. Delay for some time.
6. Read the comparator output value by reading the **ACSTAT0** register's OVAL value.

Change the level of the signal input on C0- to see the OVAL value change.

## 15.4 Register Map

Table 15-3 lists the comparator registers. The offset listed is a hexadecimal increment to the register's address, relative to the Analog Comparator base address of 0x4003C000.

**Table 15-3. Analog Comparator Register Map**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x00 | ACMIS | 0x00000000 | RO | Interrupt status | 338 |
| 0X04 | ACRIS | 0x00000000 | RO | Raw interrupt status | 339 |
| 0X08 | ACINTEN | 0x00000000 | R/W | Interrupt enable | 340 |
| 0x10 | ACREFCTL | 0x00000000 | R/W | Reference voltage control | 341 |
| 0x20 | ACSTAT0 | 0x00000000 | RO | Comparator 0 status | 342 |
| 0x24 | ACCTL0 | 0x00000000 | RW | Comparator 0 control | 343 |

## 15.5 Register Descriptions

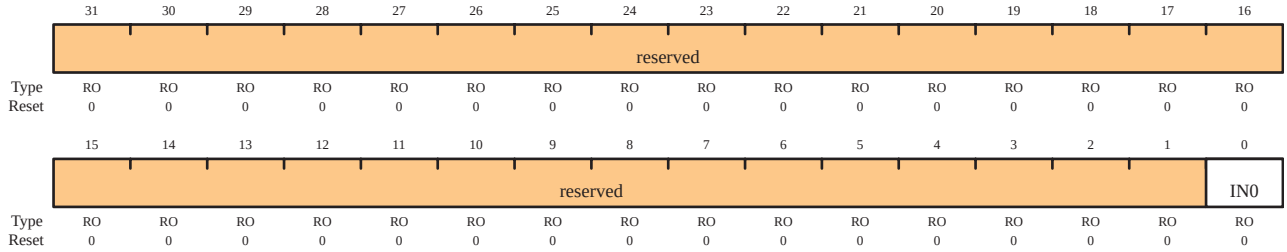
The remainder of this section lists and describes the Analog Comparator registers, in numerical order by address offset.

### Register 1: Analog Comparator Masked Interrupt Status (ACMIS), offset 0x00

This register provides a summary of the interrupt status (masked) of the comparator.

Analog Comparator Masked Interrupt Status (ACMIS)

Offset 0x000



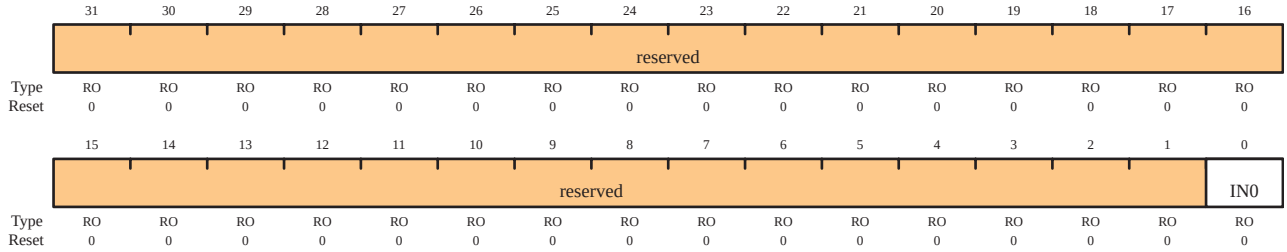

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | IN0 | RO | 0 | Comparator 0 Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |

### Register 2: Analog Comparator Raw Interrupt Status (ACRIS), offset 0x04

This register provides a summary of the interrupt status (raw) of the comparator.

Analog Comparator Raw Interrupt Status (ACRIS)

Offset 0x04



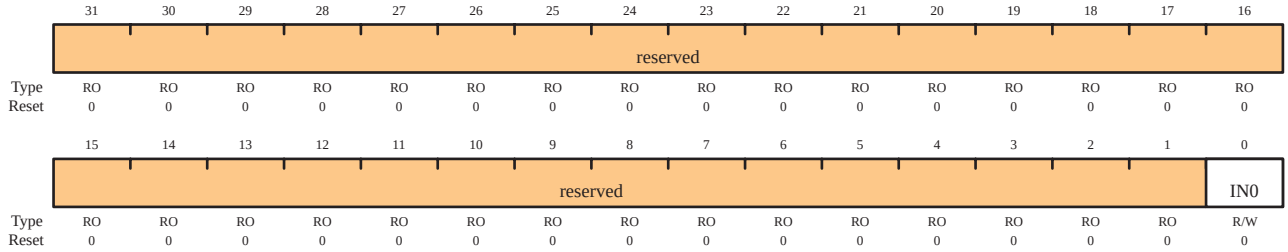

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | IN0 | RO | 0 | When set, indicates that an interrupt has been generated by comparator 0. |

### Register 3: Analog Comparator Interrupt Enable (ACINTEN), offset 0x08

This register provides the interrupt enable for the comparator.

Analog Comparator Interrupt Enable (ACINTEN)

Offset 0x08

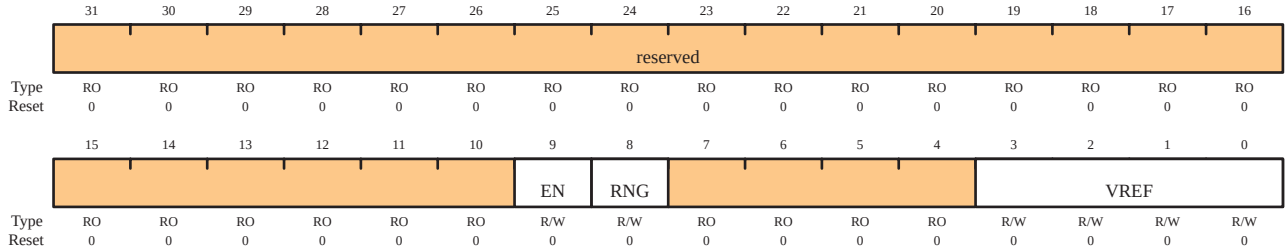



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | IN0 | R/W | 0 | When set, enables the controller interrupt from the comparator 0 output. |

### Register 4: Analog Comparator Reference Voltage Control (ACREFCTL), offset 0x10

This register specifies whether the resistor ladder is powered on as well as the range and tap.

Analog Comparator Reference Voltage Control (ACREFCTL)
Offset 0x010



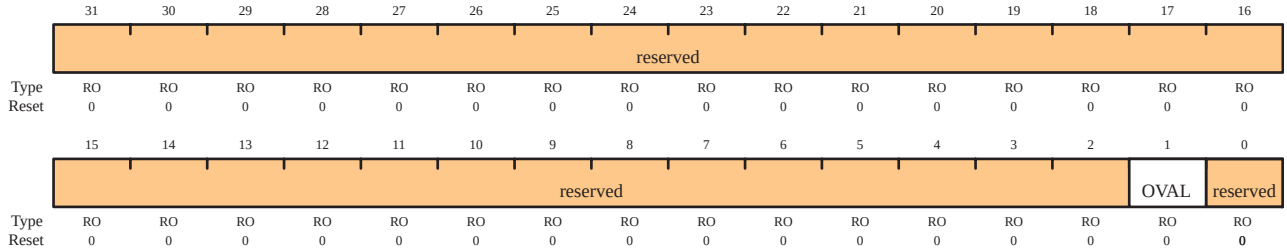

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 9 | EN | R/W | 0 | The EN bit specifies whether the resistor ladder is powered on. If 0, the resistor ladder is unpowered. If 1, the resistor ladder is connected to the analog $V_{DD}$.<br>This bit is reset to 0 so that the internal reference consumes the least amount of power if not used and programmed. |
| 8 | RNG | R/W | 0 | The RNG bit specifies the range of the resistor ladder. If 0, the resistor ladder has a total resistance of 32 R. If 1, the resistor ladder has a total resistance of 24 R. |
| 7:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3:0 | VREF | R/W | 0 | The VREF bit field specifies the resistor ladder tap that is passed through an analog multiplexer. The voltage corresponding to the tap position is the internal reference voltage available for comparison. See Table 15-2 on page 336 for some output reference voltage examples. |

### Register 5: Analog Comparator Status 0 (ACSTAT0), offset 0x20

This register specifies the current output value of that comparator.

Analog Comparator Status 0 (ACSTAT0)

Offset 0x020

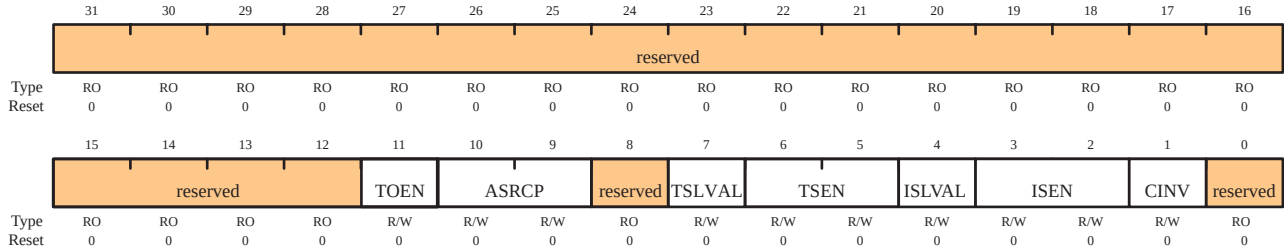



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | OVAL | RO | 0 | The OVAL bit specifies the current output value of the comparator. |
| 0 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |

### Register 6: Analog Comparator Control 0 (ACCTL0), offset 0x24

This register configures that comparator's input and output.

Analog Comparator Control 0 (ACCTL0)
Offset 0x024




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:12 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 11 | TOEN | R/W | 0 | The TOEN bit enables the ADC event transmission to the ADC. If 0, the event is suppressed and not sent to the ADC. If 1, the event is transmitted to the ADC. |
| 10:9 | ASRCP | R/W | 0 | The ASRCP field specifies the source of input voltage to the VIN+ terminal of the comparator. The encodings for this field are as follows:<br>ASRCP — Function<br>00 — Pin value<br>01 — Pin value of C0+<br>10 — Internal voltage reference<br>11 — Reserved |
| 8 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 7 | TSLVAL | R/W | 0 | The TSLVAL bit specifies the sense value of the input that generates an ADC event if in Level Sense mode. If 0, an ADC event is generated if the comparator output is Low. Otherwise, an ADC event is generated if the comparator output is High. |
| 6:5 | TSEN | R/W | 0 | The TSEN field specifies the sense of the comparator output that generates an ADC event. The sense conditioning is as follows:<br>TSEN — Function<br>00 — Level sense, see TSLVAL<br>01 — Falling edge<br>10 — Rising edge<br>11 — Either edge |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | ISLVAL | R/W | 0 | The ISLVAL bit specifies the sense value of the input that generates an interrupt if in Level Sense mode. If 0, an interrupt is generated if the comparator output is Low. Otherwise, an interrupt is generated if the comparator output is High. |
| 3:2 | ISEN | R/W | 0 | The ISEN field specifies the sense of the comparator output that generates an interrupt. The sense conditioning is as follows:<br>ISEN — Function<br>00 — Level sense, see ISLVAL<br>01 — Falling edge<br>10 — Rising edge<br>11 — Either edge |
| 1 | CINV | R/W | 0 | The CINV bit conditionally inverts the output of the comparator. If 0, the output of the comparator is unchanged. If 1, the output of the comparator is inverted prior to being processed by hardware. |
| 0 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |

# 16 Pulse Width Modulator (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.

The LM3S316 PWM module consists of two PWM generator blocks and a control block. Each PWM generator block contains one timer (16-bit down or up/down counter), two comparators, a PWM signal generator, a dead-band generator, and an interrupt/ADC-trigger selector. The control block determines the polarity of the PWM signals, and which signals are passed through to the pins.

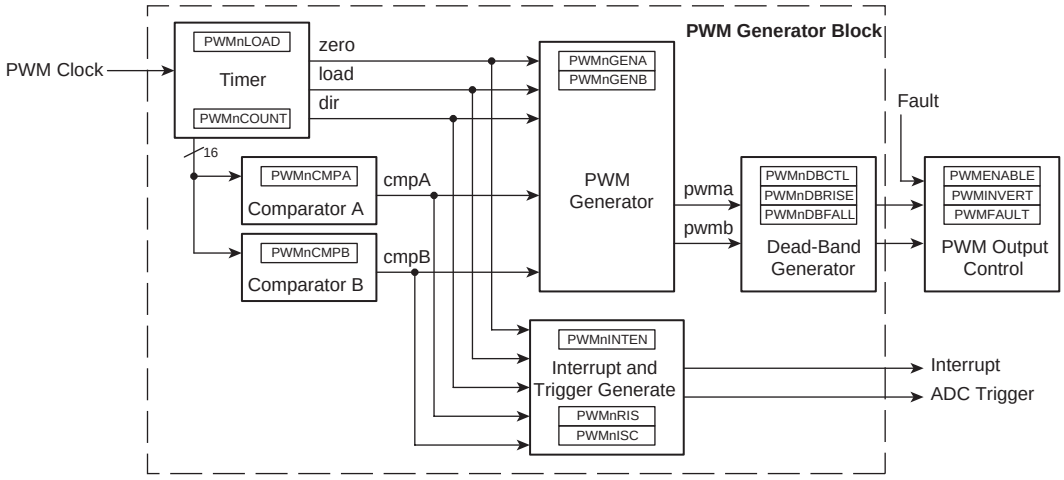
Each PWM generator block produces two PWM signals that can either be independent signals (other than being based on the same timer and therefore having the same frequency) or a single pair of complementary signals with dead-band delays inserted. The output of the PWM generation blocks are managed by the output control block before being passed to the device pins.

The LM3S316 PWM module provides a great deal of flexibility. It can generate simple PWM signals, such as those required by a simple charge pump. It can also generate paired PWM signals with dead-band delays, such as those required by a half-H bridge driver.

## 16.1 Block Diagram

Figure 16-1 provides a block diagram of a Stellaris PWM module. The LM3S316 controller contains two generator blocks (PWM0 and PWM1) and generates four independent PWM signals or two paired PWM signals with dead-band delays inserted.

**Figure 16-1. PWM Module Block Diagram**




## 16.2 Functional Description

### 16.2.1 PWM Timer

The timer in each PWM generator runs in one of two modes: Count-Down mode or Count-Up/Down mode. In Count-Down mode, the timer counts from the load value to zero, goes back to the load value, and continues counting down. In Count-Up/Down mode, the timer counts from zero up to the load value, back down to zero, back up to the load value, and so on. Generally, Count-Down

mode is used for generating left- or right-aligned PWM signals, while the Count-Up/Down mode is used for generating center-aligned PWM signals.

The timers output three signals that are used in the PWM generation process: the direction signal (this is always Low in Count-Down mode, but alternates between Low and High in Count-Up/Down mode), a single-clock-cycle-width High pulse when the counter is zero, and a single-clock-cycle-width High pulse when the counter is equal to the load value. Note that in Count-Down mode, the zero pulse is immediately followed by the load pulse.

### 16.2.2 PWM Comparators

There are two comparators in each PWM generator that monitor the value of the counter; when either match the counter, they output a single-clock-cycle-width High pulse. When in Count-Up/Down mode, these comparators match both when counting up and when counting down; they are therefore qualified by the counter direction signal. These qualified pulses are used in the PWM generation process. If either comparator match value is greater than the counter load value, then that comparator never outputs a High pulse.

Figure 16-2 shows the behavior of the counter and the relationship of these pulses when the counter is in Count-Down mode. Figure 16-3 shows the behavior of the counter and the relationship of these pulses when the counter is in Count-Up/Down mode.

**Figure 16-2. PWM Count-Down Mode**

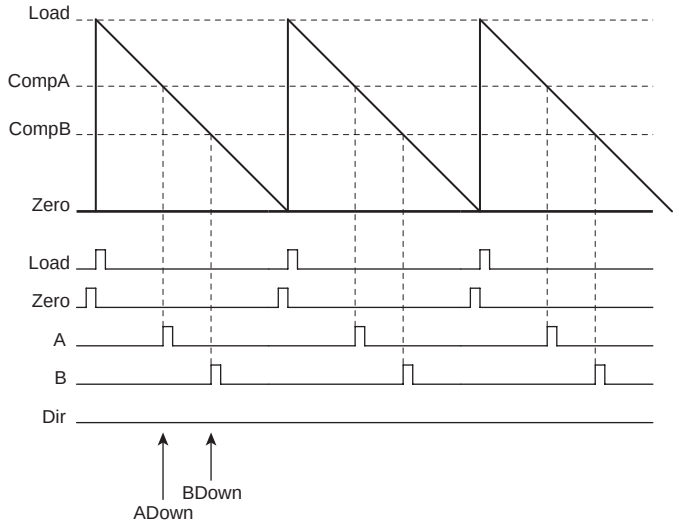
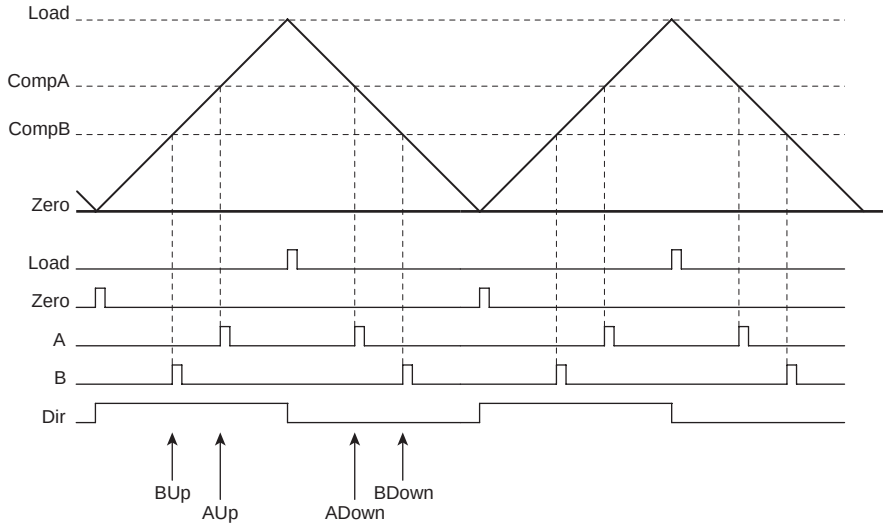


**Figure 16-3. PWM Count-Up/Down Mode**

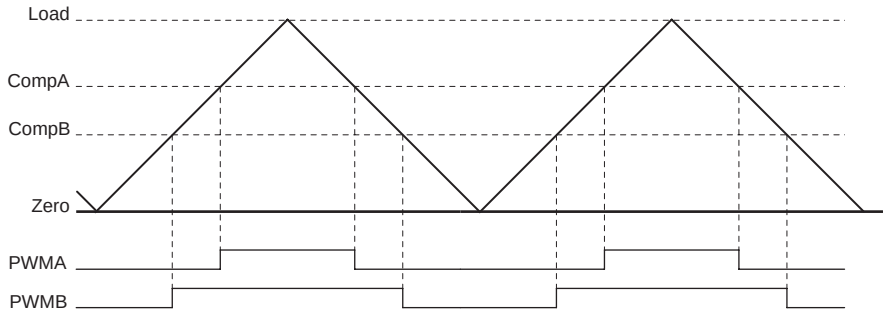



## 16.2.3 PWM Signal Generator

The PWM generator takes these pulses (qualified by the direction signal), and generates two PWM signals. In Count-Down mode, there are four events that can affect the PWM signal: zero, load, match A down, and match B down. In Count-Up/Down mode, there are six events that can affect the PWM signal: zero, load, match A down, match A up, match B down, and match B up. The match A or match B events are ignored when they coincide with the zero or load events. If the match A and match B events coincide, the first signal, PWMA, is generated based only on the match A event, and the second signal, PWMB, is generated based only on the match B event.

For each event, the effect on each output PWM signal is programmable: it can be left alone (ignoring the event), it can be toggled, it can be driven Low, or it can be driven High. These actions can be used to generate a pair of PWM signals of various positions and duty cycles, which do or do not overlap. Figure 16-4 shows the use of Count-Up/Down mode to generate a pair of center-aligned, overlapped PWM signals that have different duty cycles.

**Figure 16-4. PWM Generation Example In Count-Up/Down Mode**



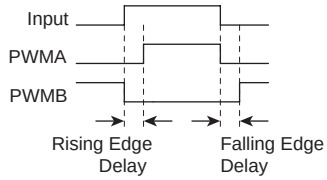

In this example, the first generator is set to drive High on match A up, drive Low on match A down, and ignore the other four events. The second generator is set to drive High on match B up, drive Low on match B down, and ignore the other four events. Changing the value of comparator A changes the duty cycle of the PWMA signal, and changing the value of comparator B changes the duty cycle of the PWMB signal.

### 16.2.4 Dead-Band Generator

The two PWM signals produced by the PWM generator are passed to the dead-band generator. If disabled, the PWM signals simply pass through unmodified. If enabled, the second PWM signal is lost and two PWM signals are generated based on the first PWM signal. The first output PWM signal is the input signal with the rising edge delayed by a programmable amount. The second output PWM signal is the inversion of the input signal with a programmable delay added between the falling edge of the input signal and the rising edge of this new signal.

This is therefore a pair of active High signals where one is always High, except for a programmable amount of time at transitions where both are Low. These signals are therefore suitable for driving a half-H bridge, with the dead-band delays preventing shoot-through current from damaging the power electronics. Figure 16-5 shows the effect of the dead-band generator on an input PWM signal.

**Figure 16-5. PWM Dead-Band Generator**




### 16.2.5 Interrupt/ADC-Trigger Selector

The PWM generator also takes the same four (or six) counter events and uses them to generate an interrupt or an ADC trigger. Any of these events or a set of these events can be selected as a source for an interrupt; when any of the selected events occur, an interrupt is generated. Additionally, the same event, a different event, the same set of events, or a different set of events can be selected as a source for an ADC trigger; when any of these selected events occur, an ADC trigger pulse is generated. The selection of events allows the interrupt or ADC trigger to occur at a specific position within the PWM signal. Note that interrupts and ADC triggers are based on the raw events; delays in the PWM signal edges caused by the dead-band generator are not taken into account.

### 16.2.6 Synchronization Methods

There is a global reset capability that can synchronously reset any or all of the counters in the PWM generator. If multiple PWM generators are configured with the same counter load value, this can be used to guarantee that they also have the same count value (this does imply that the PWM generators must be configured before they are synchronized). With this, more than two PWM signals can be produced with a known relationship between the edges of those signals since the counters always have the same values.

The counter load values and comparator match values of the PWM generator can be updated in two ways. The first is immediate update mode, where a new value is used as soon as the counter reaches zero. By waiting for the counter to reach zero, a guaranteed behavior is defined, and overly short or overly long output PWM pulses are prevented.

The other update method is synchronous, where the new value is not used until a global synchronized update signal is asserted, at which point the new value is used as soon as the counter reaches zero. This second mode allows multiple items in multiple PWM generators to be updated simultaneously without odd effects during the update; everything runs from the old values until a point at which they all run from the new values. The Update mode of the load and comparator match values can be individually configured in each PWM generator block. It only makes sense to use the synchronous update mechanism across PWM generator blocks when the

timers in those blocks are synchronized, though this is not required in order for this mechanism to function properly.

### 16.2.7 Fault Conditions

There are two external conditions that affect the PWM block; the signal input on the Fault pin and the stalling of the controller by a debugger. There are two mechanisms available to handle such conditions: the output signals can be forced into an inactive state and/or the PWM timers can be stopped.

Each output signal has a fault bit. If set, a fault input signal causes the corresponding output signal to go into the inactive state. If the inactive state is a safe condition for the signal to be in for an extended period of time, this keeps the output signal from driving the outside world in a dangerous manner during the fault condition. A fault condition can also generate a controller interrupt.

Each PWM generator can also be configured to stop counting during a stall condition. The user can select for the counters to run until they reach zero then stop, or to continue counting and reloading. A stall condition does not generate a controller interrupt.

### 16.2.8 Output Control Block

With each PWM generator block producing two raw PWM signals, the output control block takes care of the final conditioning of the PWM signals before they go to the pins. Via a single register, the set of PWM signals that are actually enabled to the pins can be modified; this can be used, for example, to perform commutation of a brushless DC motor with a single register write (and without modifying the individual PWM generators, which are modified by the feedback control loop). Similarly, fault control can disable any of the PWM signals as well. A final inversion can be applied to any of the PWM signals, making them active Low instead of the default active High.

## 16.3 Initialization and Configuration

The following example shows how to initialize the PWM Generator 0 with a 25-KHz frequency, and with a 25% duty cycle on the PWM0 pin and a 75% duty cycle on the PWM1 pin. This example assumes the system clock is 20 MHz.

1. Enable the PWM clock by writing a value of 0x00100000 to the **RCGC0** register in the System Control module.
2. In the GPIO module, enable the appropriate pins for their alternate function using the **GPIOAFSEL** register.
3. Configure the **Run-Mode Clock Configuration (RCC)** register in the System Control module to use the PWM divide (USEPWMDIV) and set the divider (PWMDIV) to divide by 2 (000).
4. Configure the PWM generator for countdown mode with immediate updates to the parameters.
   - Write the **PWM0CTL** register with a value of 0x00000000.
   - Write the **PWM0GENA** register with a value of 0x0000008C.
   - Write the **PWM0GENB** register with a value of 0x0000080C.
5. Set the period. For a 25-KHz frequency, the period = 1/25,000, or 40 microseconds. The PWM clock source is 10 MHz; the system clock divided by 2. This translates to 400 clock ticks per period. Use this value to set the **PWM0LOAD** register. In Count-Down mode, set the LOAD field in the **PWM0LOAD** register to the requested period minus one.
   - Write the **PWM0LOAD** register with a value of 0x0000018F.
6. Set the pulse width of the PWM0 pin for a 25% duty cycle.

- Write the **PWM0CMPA** register with a value of 0x0000012B.

7. Set the pulse width of the PWM1 pin for a 75% duty cycle.
   - Write the **PWM0CMPB** register with a value of 0x00000063.
8. Start the timers in PWM generator 0.
   - Write the **PWM0CTL** register with a value of 0x00000001.
9. Enable PWM outputs.
   - Write the **PWMENABLE** register with a value of 0x00000003.

## 16.4 Register Map

Table 16-2 lists the PWM registers. The offset listed is a hexadecimal increment to the register's address, relative to the PWM base address of 0x40028000.

**Table 16-1. PWM Register Map (Sheet 1 of 2)**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| **PWM Module Control** | | | | | |
| 0x000 | PWMCTL | 0x00000000 | R/W | Master control of the PWM module | 352 |
| 0x004 | PWMSYNC | 0x00000000 | R/W | Counter synchronization for the PWM generators | 353 |
| 0x008 | PWMENABLE | 0x00000000 | R/W | Master enable for the PWM output pins | 354 |
| 0x00C | PWMINVERT | 0x00000000 | R/W | Inversion control for the PWM output pins | 355 |
| 0x010 | PWMFAULT | 0x00000000 | R/W | Fault handling for the PWM output pins | 356 |
| 0x014 | PWMINTEN | 0x00000000 | R/W | Interrupt enable | 357 |
| 0x018 | PWMRIS | 0x00000000 | RO | Raw interrupt status | 358 |
| 0x01C | PWMISC | 0x00000000 | R/W1C | Interrupt status and clear | 359 |
| 0x020 | PWMSTATUS | 0x00000000 | RO | Value of the Fault input signal | 360 |
| **PWM Generator 0** | | | | | |
| 0x040 | PWM0CTL | 0x00000000 | R/W | Master control of the PWM0 generator block | 361 |
| 0x044 | PWM0INTEN | 0x00000000 | R/W | Interrupt and trigger enable | 363 |
| 0x048 | PWM0RIS | 0x00000000 | RO | Raw interrupt status | 365 |
| 0x04C | PWM0ISC | 0x00000000 | R/W1C | Interrupt status and clear | 366 |
| 0x050 | PWM0LOAD | 0x00000000 | R/W | Load value for the counter | 367 |
| 0x054 | PWM0COUNT | 0x00000000 | RO | Current counter value | 367 |
| 0x058 | PWM0CMPA | 0x00000000 | R/W | Comparator A value | 369 |
| 0x05C | PWM0CMPB | 0x00000000 | R/W | Comparator B value | 370 |
| 0x060 | PWM0GENA | 0x00000000 | R/W | Controls PWM generator A | 371 |

**Table 16-1. PWM Register Map (Sheet 2 of 2)**

| Offset | Name | Reset | Type | Description | See page |
|---|---|---|---|---|---|
| 0x064 | PWM0GENB | 0x00000000 | R/W | Controls PWM generator B | 373 |
| 0x068 | PWM0DBCTL | 0x00000000 | R/W | Control the dead-band generator | 374 |
| 0x06C | PWM0DBRISE | 0x00000000 | R/W | Dead-band rising-edge delay count | 375 |
| 0x070 | PWM0DBFALL | 0x00000000 | R/W | Dead-band falling-edge delay count | 376 |
| **PWM Generator 1** | | | | | |
| 0x080 | PWM1CTL | 0x00000000 | R/W | Master control of the PWM1 generator block | 361 |
| 0x084 | PWM1INTEN | 0x00000000 | R/W | Interrupt and trigger enable | 363 |
| 0x088 | PWM1RIS | 0x00000000 | RO | Raw interrupt status | 365 |
| 0x08C | PWM1ISC | 0x00000000 | R/W1C | Interrupt status and clear | 366 |
| 0x090 | PWM1LOAD | 0x00000000 | R/W | Load value for the counter | 367 |
| 0x094 | PWM1COUNT | 0x00000000 | RO | Current counter value | 368 |
| 0x098 | PWM1CMPA | 0x00000000 | R/W | Comparator A value | 369 |
| 0x09C | PWM1CMPB | 0x00000000 | R/W | Comparator B value | 370 |
| 0x0A0 | PWM1GENA | 0x00000000 | R/W | Controls PWM generator A | 371 |
| 0x0A4 | PWM1GENB | 0x00000000 | R/W | Controls PWM generator B | 373 |
| 0x0A8 | PWM1DBCTL | 0x00000000 | R/W | Control the dead-band generator | 374 |
| 0x0AC | PWM1DBRISE | 0x00000000 | R/W | Dead-band rising-edge delay count | 375 |
| 0x0B0 | PWM1DBFALL | 0x00000000 | R/W | Dead-band falling-edge delay count | 376 |

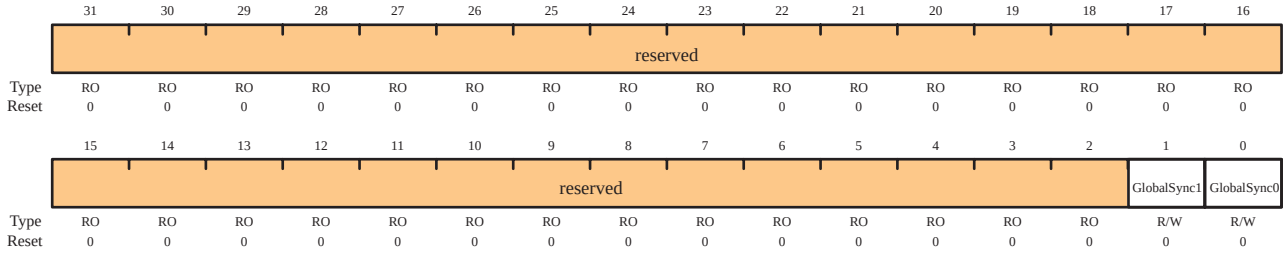
## 16.5 Register Descriptions

The remainder of this section lists and describes the PWM registers, in numerical order by address offset.

### Register 1: PWM Master Control (PWMCTL), offset 0x000

This register provides master control over the PWM generation blocks.

PWM Master Control (PWMCTL)
Offset 0x000



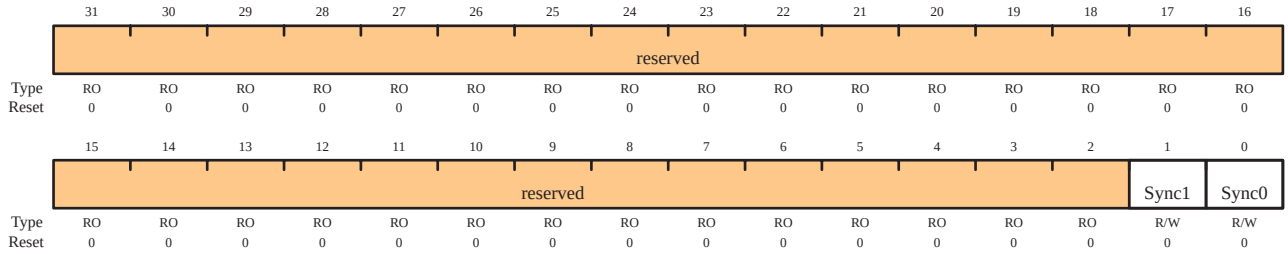

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | GlobalSync1 | R/W | 0 | Same as `GlobalSync0` but for PWM generator 1. |
| 0 | GlobalSync0 | R/W | 0 | Setting this bit causes any queued update to a load or comparator register in PWM generator 0 to be applied the next time the corresponding counter becomes zero. This bit automatically clears when the updates have completed; it cannot be cleared by software. |

### Register 2: PWM Time Base Sync (PWMSYNC), offset 0x004

This register provides a method to perform synchronization of the counters in the PWM generation blocks. Writing a bit in this register to 1 causes the specified counter to reset back to 0; writing multiple bits resets multiple counters simultaneously. The bits auto-clear after the reset has occurred; reading them back as zero indicates that the synchronization has completed.

PWM Time Base Sync (PWMSYNC)

Offset 0x004

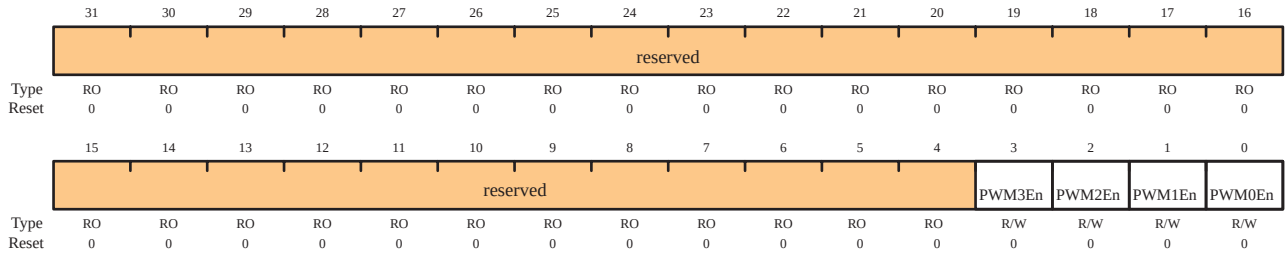



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | Sync1 | R/W | 0 | Performs a reset of the PWM generator 1 counter. |
| 0 | Sync0 | R/W | 0 | Performs a reset of the PWM generator 0 counter. |

### Register 3: PWM Output Enable (PWMENABLE), offset 0x008

This register provides a master control of which generated PWM signals are output to device pins. By disabling a PWM output, the generation process can continue (for example when the time bases are synchronized) without driving PWM signals to the pins. When bits in this register are set, the corresponding PWM signal is passed through to the output stage, which is controlled by the **PWMINVERT** register. When bits are not set, the PWM signal is replaced by a zero value which is also passed to the output stage.

PWM Output Enable (PWMENABLE)
Offset 0x008

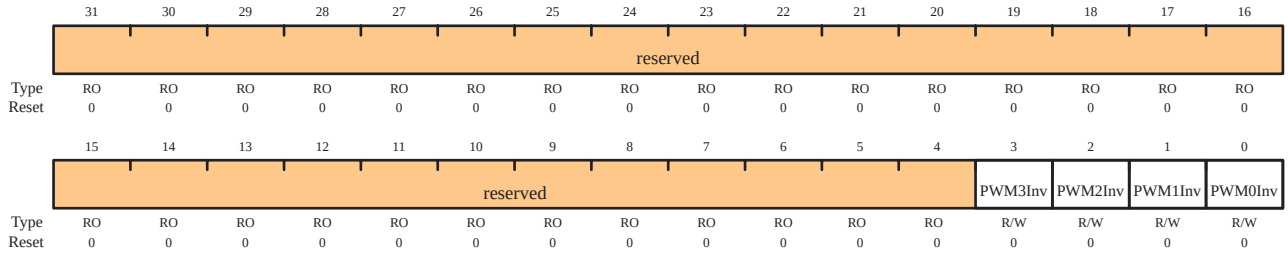



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | PWM3En | R/W | 0 | When set, allows the generated PWM3 signal to be passed to the device pin. |
| 2 | PWM2En | R/W | 0 | When set, allows the generated PWM2 signal to be passed to the device pin. |
| 1 | PWM1En | R/W | 0 | When set, allows the generated PWM1 signal to be passed to the device pin. |
| 0 | PWM0En | R/W | 0 | When set, allows the generated PWM0 signal to be passed to the device pin. |

### Register 4: PWM Output Inversion (PWMINVERT), offset 0x00C

This register provides a master control of the polarity of the PWM signals on the device pins. The PWM signals generated by the dead-band block are active High; they can optionally be made active Low via this register. Disabled PWM channels are also passed through the output inverter (if so configured) so that inactive channels maintain the correct polarity.

PWM Output Inversion (PWMINVERT)
Offset 0x00C




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | PWM3Inv | R/W | 0 | When set, the generated PWM3 signal is inverted. |
| 2 | PWM2Inv | R/W | 0 | When set, the generated PWM2 signal is inverted. |
| 1 | PWM1Inv | R/W | 0 | When set, the generated PWM1 signal is inverted. |
| 0 | PWM0Inv | R/W | 0 | When set, the generated PWM0 signal is inverted. |

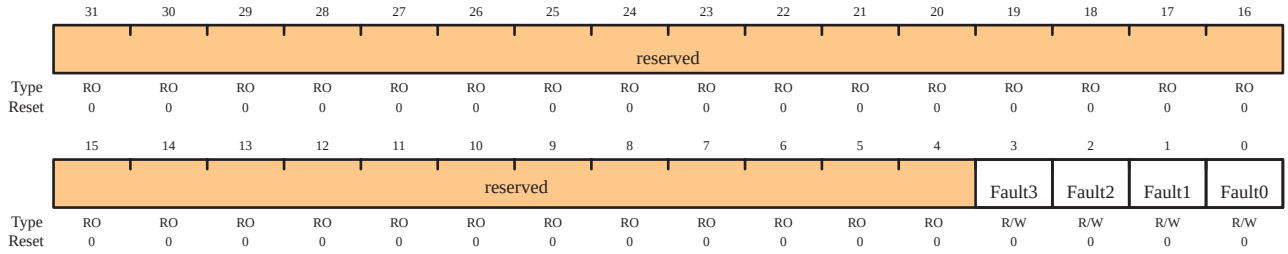
### Register 5: PWM Output Fault (PWMFAULT), offset 0x010

This register controls the behavior of the PWM outputs in the presence of fault conditions. Both the fault input and debug events are considered fault conditions. On a fault condition, each PWM signal can either be passed through unmodified or driven Low. For outputs that are configured for pass-through, the debug event handling on the corresponding PWM generator also determines if the PWM signal continues to be generated.

Fault condition control happens before the output inverter, so PWM signals driven Low on fault are inverted if the channel is configured for inversion (therefore, the pin is driven High on a fault condition).

PWM Output Fault (PWMFAULT)

Offset 0x010

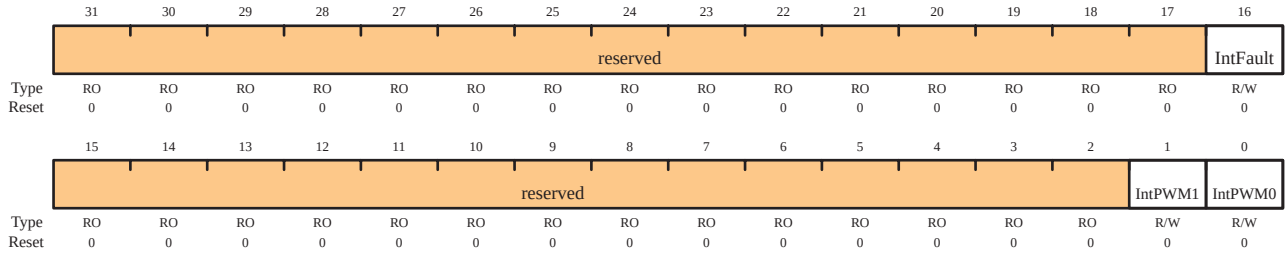



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 3 | Fault3 | R/W | 0 | When set, the PWM3 output signal is driven Low on a fault condition. |
| 2 | Fault2 | R/W | 0 | When set, the PWM2 output signal is driven Low on a fault condition. |
| 1 | Fault1 | R/W | 0 | When set, the PWM1 output signal is driven Low on a fault condition. |
| 0 | Fault0 | R/W | 0 | When set, the PWM0 output signal is driven Low on a fault condition. |

### Register 6: PWM Interrupt Enable (PWMINTEN), offset 0x014

This register controls the global interrupt generation capabilities of the PWM module. The events that can cause an interrupt are the fault input and the individual interrupts from the PWM generators.

PWM Interrupt Enable (PWMINTEN)
Offset 0x014

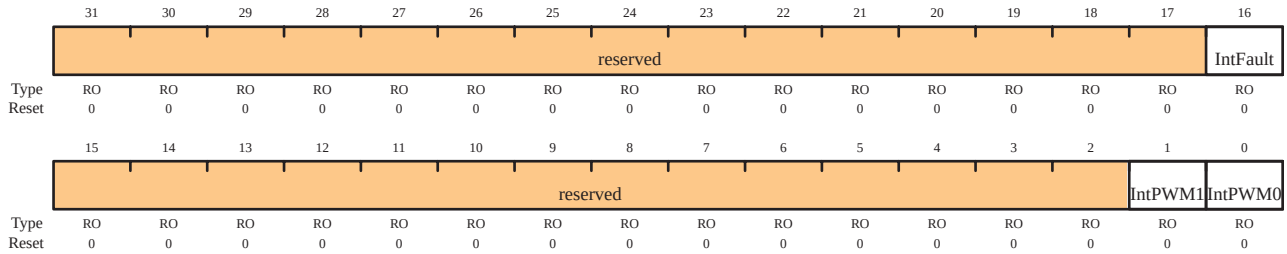



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 16 | IntFault | R/W | 0 | When 1, an interrupt occurs when the fault input is asserted. |
| 15:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | IntPWM1 | R/W | 0 | When 1, an interrupt occurs when the PWM generator 1 block asserts an interrupt. |
| 0 | IntPWM0 | R/W | 0 | When 1, an interrupt occurs when the PWM generator 0 block asserts an interrupt. |

### Register 7: PWM Raw Interrupt Status (PWMRIS), offset 0x018

This register provides the current set of interrupt sources that are asserted, regardless of whether they cause an interrupt to be asserted to the controller. The fault interrupt is latched on detection; it must be cleared through the **PWM Interrupt Status and Clear (PWMISC)** register (see page 359). The PWM generator interrupts simply reflect the status of the PWM generators; they are cleared via the interrupt status register in the PWM generator blocks. Bits set to 1 indicate the events that are active; a zero bit indicates that the event in question is not active.

PWM Raw Interrupt Status (PWMRIS)
Offset 0x018

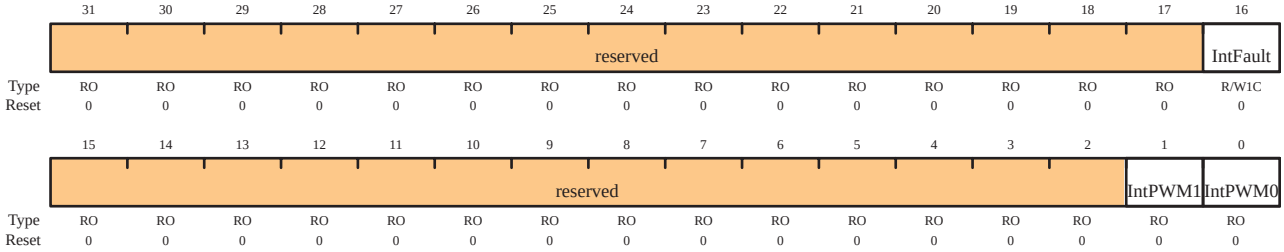



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 16 | IntFault | RO | 0 | Indicates that the fault input has been asserted. |
| 15:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | IntPWM1 | RO | 0 | Indicates that the PWM generator 1 block is asserting its interrupt. |
| 0 | IntPWM0 | RO | 0 | Indicates that the PWM generator 0 block is asserting its interrupt. |

### Register 8: PWM Interrupt Status and Clear (PWMISC), offset 0x01C

This register provides a summary of the interrupt status of the PWM generator blocks. A bit set to 1 indicates that the corresponding generator block is asserting an interrupt. The individual interrupt status registers in each block must be consulted to determine the reason for the interrupt, and used to clear the interrupt. For the fault interrupt, a write of 1 to that bit position clears the latched interrupt status.

PWM Interrupt Status and Clear (PWMISC)
Offset 0x01C

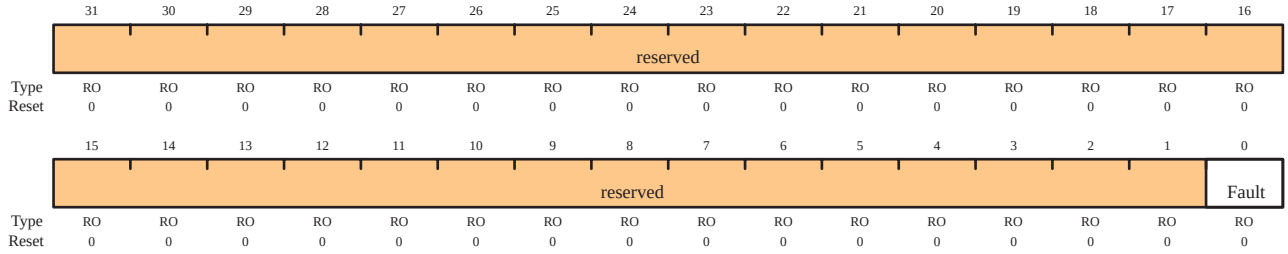



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 16 | IntFault | R/W1C | 0 | Indicates if the fault input is asserting an interrupt. |
| 15:2 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 1 | IntPWM1 | RO | 0 | Indicates if the PWM generator 1 block is asserting an interrupt. |
| 0 | IntPWM0 | RO | 0 | Indicates if the PWM generator 0 block is asserting an interrupt. |

### Register 9: PWM Status (PWMSTATUS), offset 0x020

This register provides the status of the Fault input signal.

PWM Status (PWMSTATUS)
Offset 0x020




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | Fault | RO | 0 | When set to 1, indicates the fault input is asserted. |

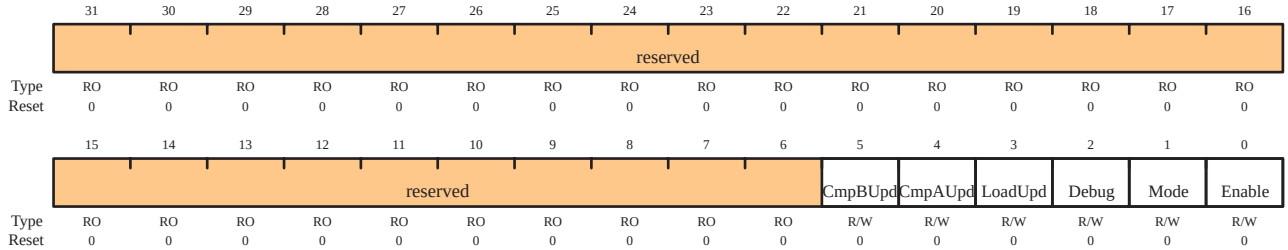
### Register 10: PWM0 Control (PWM0CTL), offset 0x040

### Register 11: PWM1 Control (PWM1CTL), offset 0x080

These registers configure the PWM signal generation blocks (**PWM0CTL** controls the PWM generator 0 block, and so on). The Register Update mode, Debug mode, Counting mode, and Block Enable mode are all controlled via these registers. The block produce the PWM signals, which can be either two independent PWM signals (from the same counter), or a paired set of PWM signals with dead-band delays added.

The PWM0 block produces the PWM0 and PWM1 outputs, and the PWM1 block produces the PWM2 and PWM3 outputs.

PWMn Control (PWMnCTL)




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 5 | CmpBUpd | R/W | 0 | Same as `CmpAUpd` but for the comparator B register. |
| 4 | CmpAUpd | R/W | 0 | The Update mode for the comparator A register. If 0, updates to the register are reflected to the comparator the next time the counter is 0. If 1, updates to the register are delayed until the next time the counter is 0 after a synchronous update has been requested through the **PWM Master Control (PWMCTL)** register (see page 352). |
| 3 | LoadUpd | R/W | 0 | The Update mode for the load register. If 0, updates to the register are reflected to the counter the next time the counter is 0. If 1, updates to the register are delayed until the next time the counter is 0 after a synchronous update has been requested through the **PWM Master Control (PWMCTL)** register. |
| 2 | Debug | R/W | 0 | The behavior of the counter in Debug mode. If 0, the counter stops running when it next reaches 0, and continues running again when no longer in Debug mode. If 1, the counter always runs. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | Mode | R/W | 0 | The mode for the counter. If 0, the counter counts down from the load value to 0 and then wraps back to the load value (Count-Down mode). If 1, the counter counts up from 0 to the load value, back down to 0, and then repeats (Count-Up/Down mode). |
| 0 | Enable | R/W | 0 | Master enable for the PWM generation block. If 0, the entire block is disabled and not clocked. If 1, the block is enabled and produces PWM signals. |

### Register 12: PWM0 Interrupt/Trigger Enable (PWM0INTEN), offset 0x044

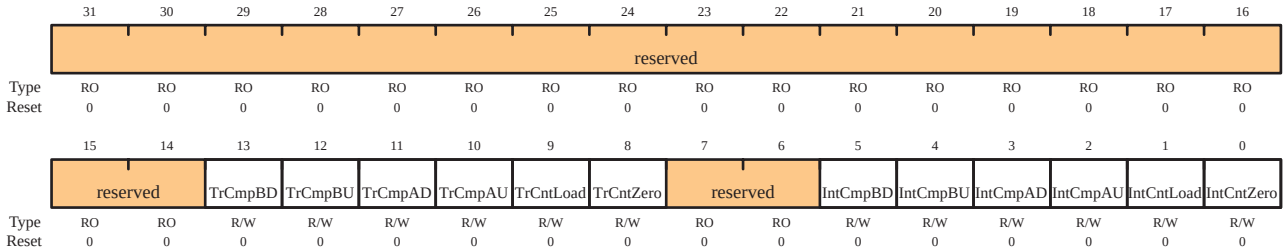
### Register 13: PWM1 Interrupt/Trigger Enable (PWM1INTEN), offset 0x084

These registers control the interrupt and ADC trigger generation capabilities of the PWM generators (PWM0INTEN controls the PWM generator 0 block, and so on). The events that can cause an interrupt or an ADC trigger are:

- The counter being equal to the load register
- The counter being equal to zero
- The counter being equal to the comparator A register while counting up
- The counter being equal to the comparator A register while counting down
- The counter being equal to the comparator B register while counting up
- The counter being equal to the comparator B register while counting down

Any combination of these events can generate either an interrupt or an ADC trigger, though no determination can be made as to the actual event that caused an ADC trigger.

PWMn Interrupt/Trigger Enable (PWMnINTEN)




| Bit/Field | Name | Reset | Type | Description |
|---|---|---|---|---|
| 31:14 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 13 | TrCmpBD | R/W | 0 | When 1, a trigger pulse is output when the counter matches the comparator B value and the counter is counting down. |
| 12 | TrCmpBU | R/W | 0 | When 1, a trigger pulse is output when the counter matches the comparator B value and the counter is counting up. |
| 11 | TrCmpAD | R/W | 0 | When 1, a trigger pulse is output when the counter matches the comparator A value and the counter is counting down. |
| 10 | TrCmpAU | R/W | 0 | When 1, a trigger pulse is output when the counter matches the comparator A value and the counter is counting up. |
| 9 | TrCntLoad | R/W | 0 | When 1, a trigger pulse is output when the counter matches the **PWMnLOAD** register. |
| 8 | TrCntZero | R/W | 0 | When 1, a trigger pulse is output when the counter is 0. |
| 7:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |

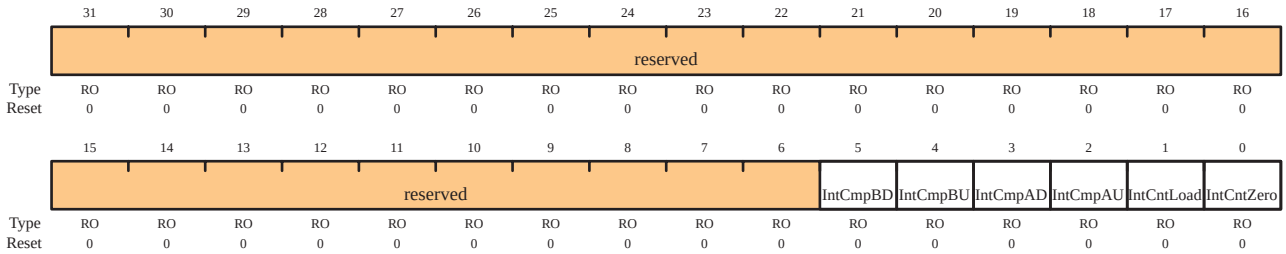
| Bit/Field | Name | Reset | Type | Description |
|---|---|---|---|---|
| 5 | IntCmpBD | R/W | 0 | When 1, an interrupt occurs when the counter matches the comparator B value and the counter is counting down. |
| 4 | IntCmpBU | R/W | 0 | When 1, an interrupt occurs when the counter matches the comparator B value and the counter is counting up. |
| 3 | IntCmpAD | R/W | 0 | When 1, an interrupt occurs when the counter matches the comparator A value and the counter is counting down. |
| 2 | IntCmpAU | R/W | 0 | When 1, an interrupt occurs when the counter matches the comparator A value and the counter is counting up. |
| 1 | IntCntLoad | R/W | 0 | When 1, an interrupt occurs when the counter matches the **PWMnLOAD** register. |
| 0 | IntCntZero | R/W | 0 | When 1, an interrupt occurs when the counter is 0. |

### Register 14: PWM0 Raw Interrupt Status (PWM0RIS), offset 0x048

### Register 15: PWM1 Raw Interrupt Status (PWM1RIS), offset 0x088

These registers provide the current set of interrupt sources that are asserted, regardless of whether they cause an interrupt to be asserted to the controller (**PWM0RIS** controls the PWM generator 0 block, and so on). Bits set to 1 indicate the latched events that have occurred; a 0 bit indicates that the event in question has not occurred.

PWMn Raw Interrupt Status (PWMnRIS)



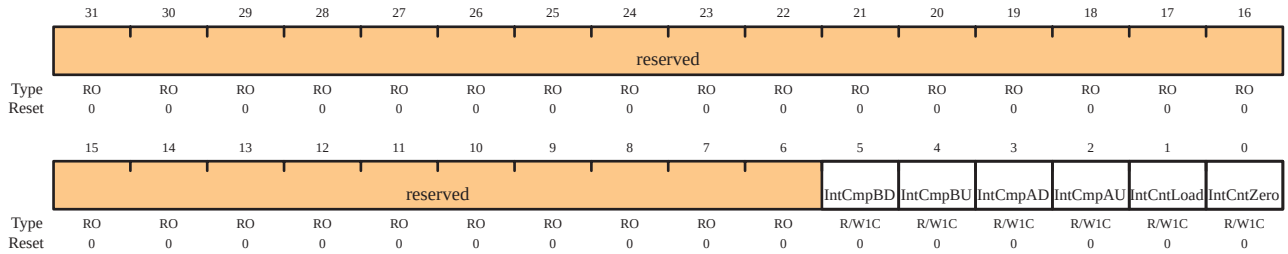

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 5 | IntCmpBD | RO | 0 | Indicates that the counter has matched the comparator B value while counting down. |
| 4 | IntCmpBU | RO | 0 | Indicates that the counter has matched the comparator B value while counting up. |
| 3 | IntCmpAD | RO | 0 | Indicates that the counter has matched the comparator A value while counting down. |
| 2 | IntCmpAU | RO | 0 | Indicates that the counter has matched the comparator A value while counting up. |
| 1 | IntCntLoad | RO | 0 | Indicates that the counter has matched the **PWMnLOAD** register. |
| 0 | IntCntZero | RO | 0 | Indicates that the counter has matched 0. |

### Register 16: PWM0 Interrupt Status and Clear (PWM0ISC), offset 0x04C

### Register 17: PWM1 Interrupt Status and Clear (PWM1ISC), offset 0x08C

These registers provide the current set of interrupt sources that are asserted to the controller (**PWM0ISC** controls the PWM generator 0 block, and so on). Bits set to 1 indicate the latched events that have occurred; a 0 bit indicates that the event in question has not occurred. These are R/W1C registers; writing a 1 to a bit position clears the corresponding interrupt reason.

PWMn Interrupt Status (PWMnISC)



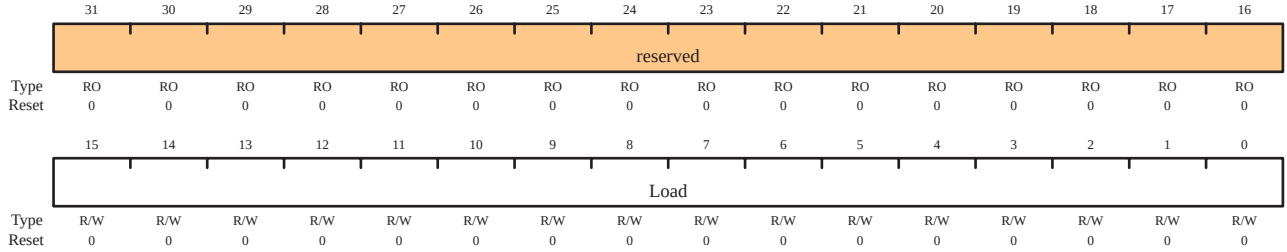

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 5 | IntCmpBD | R/W1C | 0 | Indicates that the counter has matched the comparator B value while counting down. |
| 4 | IntCmpBU | R/W1C | 0 | Indicates that the counter has matched the comparator B value while counting up. |
| 3 | IntCmpAD | R/W1C | 0 | Indicates that the counter has matched the comparator A value while counting down. |
| 2 | IntCmpAU | R/W1C | 0 | Indicates that the counter has matched the comparator A value while counting up. |
| 1 | IntCntLoad | R/W1C | 0 | Indicates that the counter has matched the **PWMnLOAD** register. |
| 0 | IntCntZero | R/W1C | 0 | Indicates that the counter has matched 0. |

### Register 18: PWM0 Load (PWM0LOAD), offset 0x050

### Register 19: PWM1 Load (PWM1LOAD), offset 0x090

These registers contain the load value for the PWM counter (**PWM0LOAD** controls the PWM generator 0 block, and so on). Based on the counter mode, either this value is loaded into the counter after it reaches zero, or it is the limit of up-counting after which the counter decrements back to zero. If the Load Value Update mode is immediate, this value is used the next time the counter reaches zero; if the mode is synchronous, it is used the next time the counter reaches zero after a synchronous update has been requested through the **PWM Master Control (PWMCTL)** register (see page 352). If this register is re-written before the actual update occurs, the previous value is never used and is lost.

PWMn Load (PWMnLOAD)



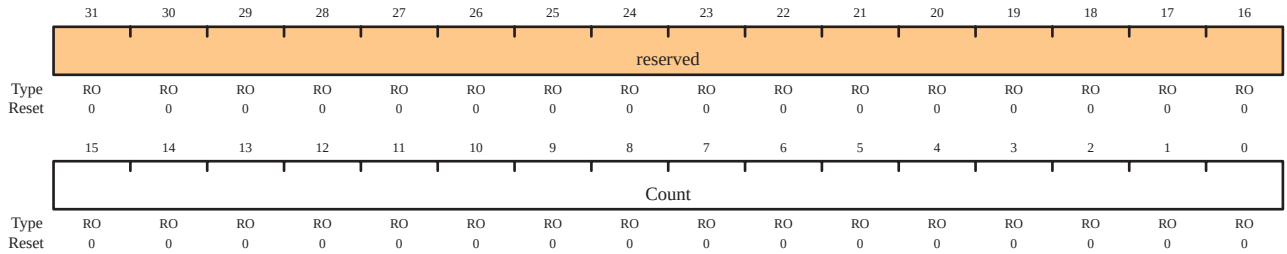

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:0 | Load | R/W | 0 | The counter load value. |

### Register 20: PWM0 Counter (PWM0COUNT), offset 0x054

### Register 21: PWM1 Counter (PWM1COUNT), offset 0x094

These registers contain the current value of the PWM counter (**PWM0COUNT** controls the PWM generator 0 block, and so on). When this value matches the load register, a pulse is output; this can drive the generation of a PWM signal (via the **PWMnGENA/PWMnGENB** registers, see page 371 and 373) or drive an interrupt or ADC trigger (via the **PWMnINTEN** register, see page 363). A pulse with the same capabilities is generated when this value is zero.

PWMn Counter (PWMnCOUNT)




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:0 | Count | RO | 0 | The current value of the counter. |

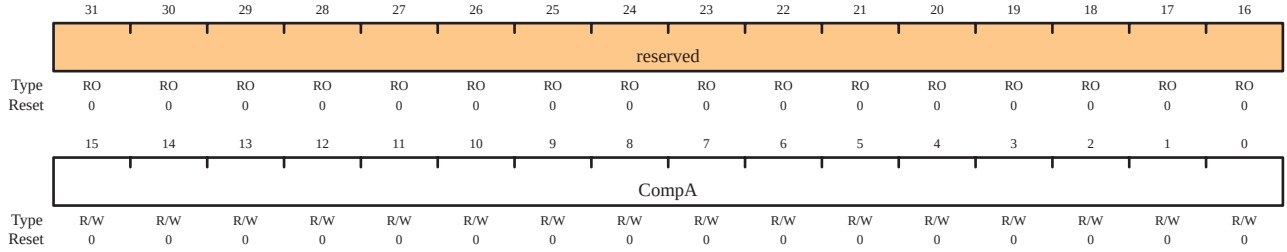
### Register 22: PWM0 Compare A (PWM0CMPA), offset 0x058

### Register 23: PWM1 Compare A (PWM1CMPA), offset 0x098

These registers contain a value to be compared against the counter (**PWM0CMPA** controls the PWM generator 0 block, and so on). When this value matches the counter, a pulse is output; this can drive the generation of a PWM signal (via the **PWMnGENA/PWMnGENB** registers) or drive an interrupt or ADC trigger (via the **PWMnINTEN** register). If the value of this register is greater than the **PWMnLOAD** register (see page 367), then no pulse is ever output.

For comparator A, if the update mode is immediate (based on the `CmpAUpd` bit in the **PWMnCTL** register), then this 16-bit `CompA` value is used the next time the counter reaches zero. If the update mode is synchronous, it is used the next time the counter reaches zero after a synchronous update has been requested through the **PWM Master Control (PWMCTL)** register (see page 352). If this register is rewritten before the actual update occurs, the previous value is never used and is lost.

PWMn Compare A (PWMnCMPA)




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:0 | CompA | R/W | 0 | The value to be compared against the counter. |

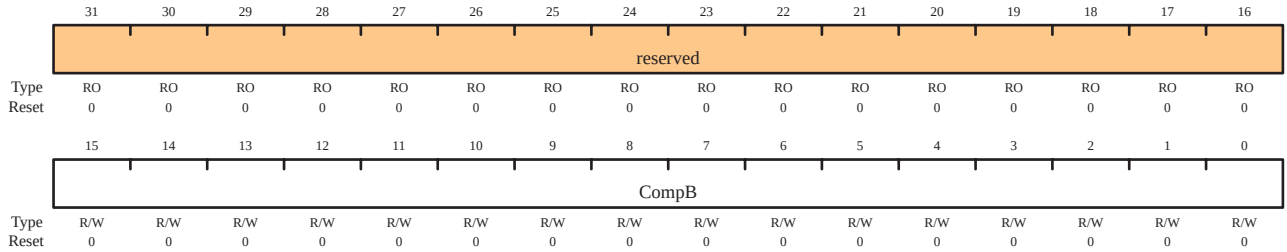
### Register 24: PWM0 Compare B (PWM0CMPB), offset 0x05C

### Register 25: PWM1 Compare B (PWM1CMPB), offset 0x09C

These registers contain a value to be compared against the counter (**PWM0CMPB** controls the PWM generator 0 block, and so on). When this value matches the counter, a pulse is output; this can drive the generation of a PWM signal (via the **PWMnGENA/PWMnGENB** registers) or drive an interrupt or ADC trigger (via the **PWMnINTEN** register). If the value of this register is greater than the **PWMnLOAD** register, then no pulse is ever output.

For comparator B, if the update mode is immediate (based on the `CmpBUpd` bit in the **PWMnCTL** register), then this 16-bit `CompB` value is used the next time the counter reaches zero after a synchronous update has been requested through the **PWM Master Control (PWMCTL)** register (see page 352). If this register is rewritten before the actual update occurs, the previous value is never used and is lost.

PWMn Compare B (PWMnCMPB)




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 15:0 | CompB | R/W | 0 | The value to be compared against the counter. |

### Register 26: PWM0 Generator A Control (PWM0GENA), offset 0x060

### Register 27: PWM1 Generator A Control (PWM1GENA), offset 0x0A0

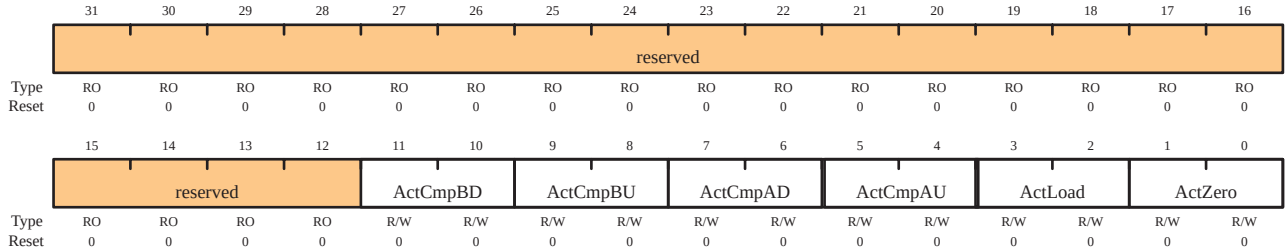
These registers control the generation of the PWMnA signal based on the load and zero output pulses from the counter, as well as the compare A and compare B pulses from the comparators (**PWM0GENA** controls the PWM generator 0 block, and so on). When the counter is running in Count-Down mode, only four of these events occur; when running in Count-Up/Down mode, all six occur. These events provide great flexibility in the positioning and duty cycle of the PWM signal that is produced.

The **PWM0GENA** register controls generation of the PWM0A signal; **PWM1GENA**, the PWM1A signal.

Each field in these registers can take on one of the values defined in Table 16-2, which defines the effect of the event on the output signal.

If a zero or load event coincides with a compare A or compare B event, the zero or load action is taken and the compare A or compare B action is ignored. If a compare A event coincides with a compare B event, the compare A action is taken and the compare B action is ignored.

PWMn Generator A Control (PWMnGENA)




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:12 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 11:10 | ActCmpBD | R/W | 0 | The action to be taken when the counter matches comparator B while counting down. |
| 9:8 | ActCmpBU | R/W | 0 | The action to be taken when the counter matches comparator B while counting up. Occurs only when the Mode bit in the **PWMnCTL** register (see page 361) is set to 1. |
| 7:6 | ActCmpAD | R/W | 0 | The action to be taken when the counter matches comparator A while counting down. |
| 5:4 | ActCmpAU | R/W | 0 | The action to be taken when the counter matches comparator A while counting up.Occurs only when the Mode bit in the **PWMnCTL** register is set to 1. |
| 3:2 | ActLoad | R/W | 0 | The action to be taken when the counter matches the load value. |
| 1:0 | ActZero | R/W | 0 | The action to be taken when the counter is zero. |

**Table 16-2. PWM Generator Action Encodings**

| Value | Description |
|---|---|
| 00 | Do nothing. |
| 01 | Invert the output signal. |
| 10 | Set the output signal to 0. |
| 11 | Set the output signal to 1. |

### Register 28: PWM0 Generator B Control (PWM0GENB), offset 0x064

### Register 29: PWM1 Generator B Control (PWM1GENB), offset 0x0A4

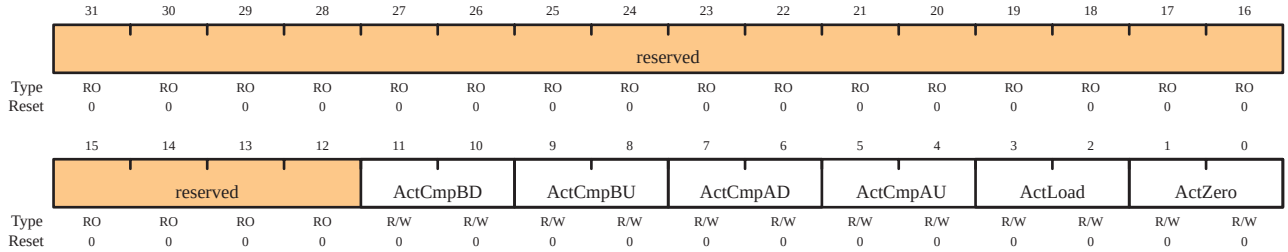
These registers control the generation of the PWMnB signal based on the load and zero output pulses from the counter, as well as the compare A and compare B pulses from the comparators (**PWM0GENB** controls the PWM generator 0 block, and so on). When the counter is running in Down mode, only four of these events occur; when running in Up/Down mode, all six occur. These events provide great flexibility in the positioning and duty cycle of the PWM signal that is produced.

The **PWM0GENB** register controls generation of the PWM0B signal; **PWM1GENB**, the PWM1B signal.

Each field in these registers can take on one of the values defined in Table 16-2 on page 372, which defines the effect of the event on the output signal.

If a zero or load event coincides with a compare A or compare B event, the zero or load action is taken and the compare A or compare B action is ignored. If a compare A event coincides with a compare B event, the compare B action is taken and the compare A action is ignored.

PWMn Generator B Control (PWMnGENB)



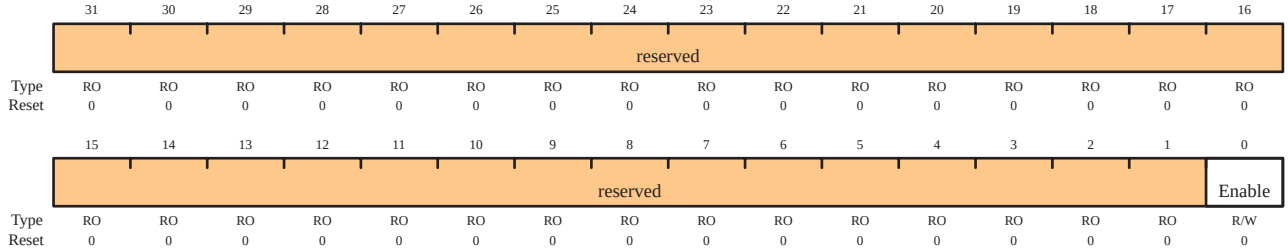

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:12 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 11:10 | ActCmpBD | R/W | 0 | The action to be taken when the counter matches comparator B while counting down. |
| 9:8 | ActCmpBU | R/W | 0 | The action to be taken when the counter matches comparator B while counting up. Occurs only when the Mode bit in the **PWMnCTL** register (see page 361) is set to 1. |
| 7:6 | ActCmpAD | R/W | 0 | The action to be taken when the counter matches comparator A while counting down. |
| 5:4 | ActCmpAU | R/W | 0 | The action to be taken when the counter matches comparator A while counting up. Occurs only when the Mode bit in the **PWMnCTL** register is set to 1. |
| 3:2 | ActLoad | R/W | 0 | The action to be taken when the counter matches the load value. |
| 1:0 | ActZero | R/W | 0 | The action to be taken when the counter is 0. |

### Register 30: PWM0 Dead-Band Control (PWM0DBCTL), offset 0x068

### Register 31: PWM1 Dead-Band Control (PWM1DBCTL), offset 0x0A8

The **PWM0DBCTL** register controls the dead-band generator, which produces the PWM0 and PWM1 signals based on the PWM0A and PWM0B signals. When disabled, the PWM0A signal passes through to the PWM0 signal and the PWM0B signal passes through to the PWM1 signal. When enabled, the PWM0B signal is ignored; the PWM0 signal is generated by delaying the rising edge(s) of the PWM0A signal by the value in the **PWM0DBRISE** register (see page 375), and the PWM1 signal is generated by delaying the falling edge(s) of the PWM0A signal by the value in the **PWM0DBFALL** register (see page 376). In a similar manner, PWM2 and PWM3 are produced from the PWM1A and PWM1B signals.

PWMn Dead-Band Control (PWMnDBCTL)



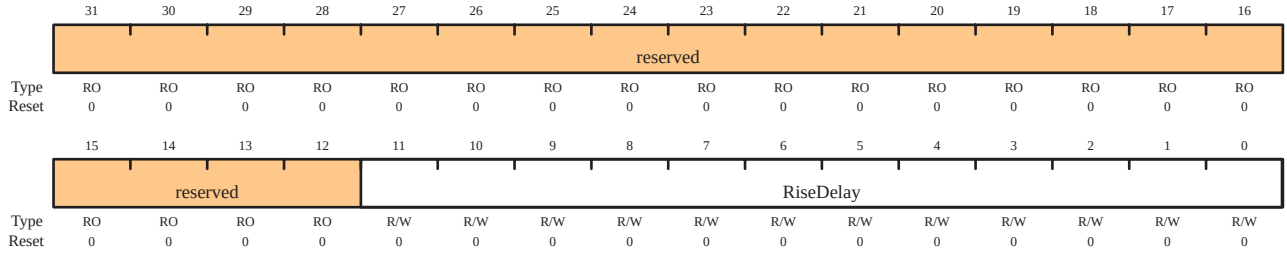

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 0 | Enable | R/W | 0 | When set, the dead-band generator inserts dead bands into the output signals; when clear, it simply passes the PWM signals through. |

### Register 32: PWM0 Dead-Band Rising-Edge Delay (PWM0DBRISE), offset 0x06C

### Register 33: PWM1 Dead-Band Rising-Edge Delay (PWM1DBRISE), offset 0x0AC

The **PWM0DBRISE** register contains the number of clock ticks to delay the rising edge of the PWM0A signal when generating the PWM0 signal. If the dead-band generator is disabled through the **PWMnDBCTL** register, the **PWM0DBRISE** register is ignored. If the value of this register is larger than the width of a High pulse on the input PWM signal, the rising-edge delay consumes the entire High time of the signal, resulting in no High time on the output. Care must be taken to ensure that the input High time always exceeds the rising-edge delay. In a similar manner, PWM2 is generated from PWM1A with its rising edge delayed.

PWMn Dead-Band Rising-Edge Delay (PWMnDBRISE)



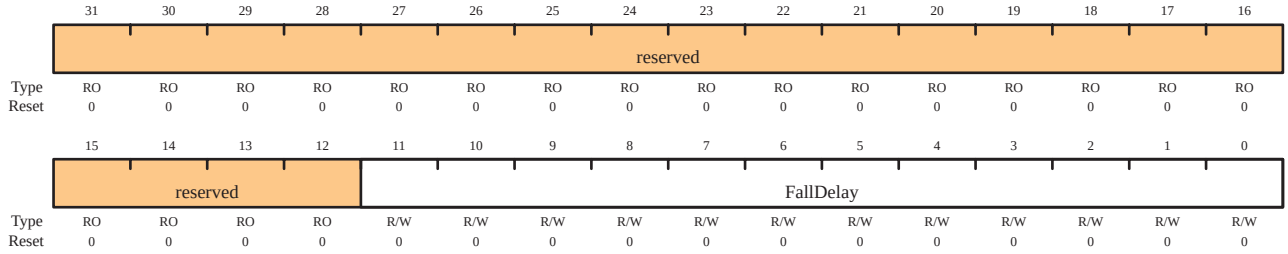

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:12 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 11:0 | RiseDelay | R/W | 0 | The number of clock ticks to delay the rising edge. |

### Register 34: PWM0 Dead-Band Falling-Edge-Delay (PWM0DBFALL), offset 0x070

### Register 35: PWM1 Dead-Band Falling-Edge-Delay (PWM1DBFALL), offset 0x0B0

The **PWM0DBFALL** register contains the number of clock ticks to delay the falling edge of the PWM0A signal when generating the PWM1 signal. If the dead-band generator is disabled, this register is ignored. If the value of this register is larger than the width of a Low pulse on the input PWM signal, the falling-edge delay consumes the entire Low time of the signal, resulting in no Low time on the output. Care must be taken to ensure that the input Low time always exceeds the falling-edge delay. In a similar manner, PWM3 is generated from PWM1A with its falling edge delayed.

PWMn Dead-Band Falling-Edge-Delay Register (PWMnDBFALL)

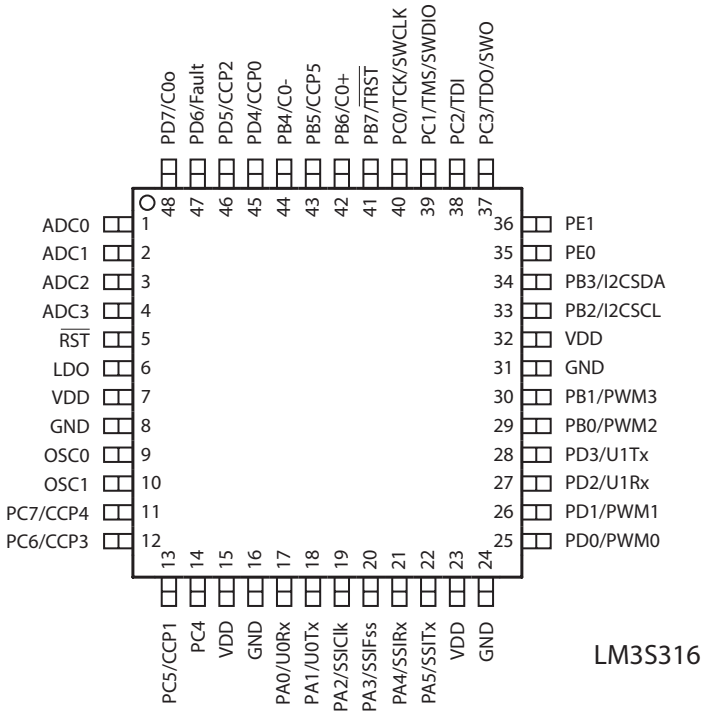



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:12 | reserved | RO | 0 | Reserved bits return an indeterminate value, and should never be changed. |
| 11:0 | FallDelay | R/W | 0 | The number of clock ticks to delay the falling edge. |

# 17 Pin Diagram

Figure 17-1 shows the pin diagram and pin-to-signal-name mapping.

**Figure 17-1. Pin Connection Diagram**



# 18 Signal Tables

The following tables list the signals available for each pin. Functionality is enabled by software with the **GPIOAFSEL** register (see page 125).

**Important:** All multiplexed pins are GPIOs by default, with the exception of the five JTAG pins (PB7 and PC[3:0]) which default to the JTAG functionality.

Table 18-1 shows the pin-to-signal-name mapping, including functional characteristics of the signals. Table 18-2 lists the signals in alphabetical order by signal name. Table 18-3 groups the signals by functionality, except for GPIOs. Table 18-4 lists the GPIO pins and their alternate functionality.

**Table 18-1. Signals by Pin Number (Sheet 1 of 4)**

| Pin Number | Pin Name | Pin Type | Buffer Type | Description |
|---|---|---|---|---|
| 1 | ADC0 | I | Analog | Analog-to-digital converter input 0. |
| 2 | ADC1 | I | Analog | Analog-to-digital converter input 1. |
| 3 | ADC2 | I | Analog | Analog-to-digital converter input 2. |
| 4 | ADC3 | I | Analog | Analog-to-digital converter input 3. |
| 5 | $\overline{\text{RST}}$ | I | TTL | System reset input. |
| 6 | LDO | - | Power | The low drop-out regulator output voltage. This pin requires an external capacitor between the pin and GND of 1 µF or greater. |
| 7 | VDD | - | Power | Positive supply for logic and I/O pins. |
| 8 | GND | - | Power | Ground reference for logic and I/O pins. |
| 9 | OSC0 | I | Analog | Oscillator crystal input or an external clock reference input. |
| 10 | OSC1 | O | Analog | Oscillator crystal output. |
| 11 | PC7 | I/O | TTL | GPIO port C bit 7. |
| | CCP4 | I/O | TTL | Timer 2 capture input, compare output, or PWM output channel 4. |
| 12 | PC6 | I/O | TTL | GPIO port C bit 6. |
| | CCP3 | I/O | TTL | Timer 1 capture input, compare output, or PWM output channel 3. |
| 13 | PC5 | I/O | TTL | GPIO port C bit 5. |
| | CCP1 | I/O | TTL | Timer 0 capture input, compare output, or PWM output channel 1. |
| 14 | PC4 | I/O | TTL | GPIO port C bit 4. |
| 15 | VDD | - | Power | Positive supply for logic and I/O pins. |
| 16 | GND | - | Power | Ground reference for logic and I/O pins. |
| 17 | PA0 | I/O | TTL | GPIO port A bit 0. |
| | U0Rx | I | TTL | UART0 receive data input. |

**Table 18-1. Signals by Pin Number (Sheet 2 of 4)**

| Pin Number | Pin Name | Pin Type | Buffer Type | Description |
|---|---|---|---|---|
| 18 | PA1 | I/O | TTL | GPIO port A bit 1. |
| | U0Tx | O | TTL | UART0 transmit data output. |
| 19 | PA2 | I/O | TTL | GPIO port A bit 2. |
| | SSIClk | I/O | TTL | SSI clock reference (input when in slave mode and output in master mode). |
| 20 | PA3 | I/O | TTL | GPIO port A bit 3. |
| | SSIFss | I/O | TTL | SSI frame enable (input for an SSI slave device and output for an SSI master device). |
| 21 | PA4 | I/O | TTL | GPIO port A bit 4. |
| | SSIRx | I | TTL | SSI receive data input. |
| 22 | PA5 | I/O | TTL | GPIO port A bit 5. |
| | SSITx | O | TTL | SSI transmit data output. |
| 23 | VDD | - | Power | Positive supply for logic and I/O pins. |
| 24 | GND | - | Power | Ground reference for logic and I/O pins. |
| 25 | PD0 | I/O | TTL | GPIO port D bit 0. |
| | PWM0 | O | TTL | Pulse width modulator channel 0 output. |
| 26 | PD1 | I/O | TTL | GPIO port D bit 1. |
| | PWM1 | O | TTL | Pulse width modulator channel 1 output. |
| 27 | PD2 | I/O | TTL | GPIO port D bit 2. |
| | U1Rx | I | TTL | UART1 receive data input. |
| 28 | PD3 | I/O | TTL | GPIO port D bit 3. |
| | U1Tx | O | TTL | UART1 transmit data output. |
| 29 | PB0 | I/O | TTL | GPIO port B bit 0. |
| | PWM2 | O | TTL | Pulse width modulator channel 2 output. |
| 30 | PB1 | I/O | TTL | GPIO port B bit 1. |
| | PWM3 | O | TTL | Pulse width modulator channel 3 output. |
| 31 | GND | - | Power | Ground reference for logic and I/O pins. |
| 32 | VDD | - | Power | Positive supply for logic and I/O pins. |
| 33 | PB2 | I/O | TTL | GPIO port B bit 2. |
| | I2CSCL | I/O | OD | $I^2C$ serial clock. |

**Table 18-1. Signals by Pin Number (Sheet 3 of 4)**

| Pin Number | Pin Name | Pin Type | Buffer Type | Description |
|---|---|---|---|---|
| 34 | PB3 | I/O | TTL | GPIO port B bit 3. |
| | I2CSDA | I/O | OD | $I^2C$ serial data. |
| 35 | PE0 | I/O | TTL | GPIO port E bit 0. |
| 36 | PE1 | I/O | TTL | GPIO port E bit 1. |
| 37 | PC3 | I/O | TTL | GPIO port C bit 3. |
| | TDO | O | TTL | JTAG scan test data output. |
| | SWO | O | TTL | Serial-wire output. |
| 38 | PC2 | I/O | TTL | GPIO port C bit 2. |
| | TDI | I | TTL | JTAG scan test data input. |
| 39 | PC1 | I/O | TTL | GPIO port C bit 1. |
| | TMS | I | TTL | JTAG scan test mode select input. |
| | SWDIO | I/O | TTL | Serial-wire debug input/output. |
| 40 | PC0 | I/O | TTL | GPIO port C bit 0. |
| | TCK | I | TTL | JTAG scan test clock reference input. |
| | SWCLK | I | TTL | Serial wire clock reference input. |
| 41 | PB7 | I/O | TTL | GPIO port B bit 7. |
| | $\overline{\text{TRST}}$ | I | TTL | JTAG scan test reset input. |
| 42 | PB6 | I/O | TTL | GPIO port B bit 6. |
| | C0+ | I | Analog | Analog comparator 0 positive-reference input. |
| 43 | PB5 | I/O | TTL | GPIO port B bit 5. |
| | CCP5 | I/O | TTL | Timer 2 capture input, compare output, or PWM output channel 5. |
| 44 | PB4 | I/O | TTL | GPIO port B bit 4. |
| | C0– | I | Analog | Analog comparator 0 negative-reference input. |
| 45 | PD4 | I/O | TTL | GPIO port D bit 4. |
| | CCP0 | I/O | TTL | Timer 0 capture input, compare output, or PWM output channel 0. |
| 46 | PD5 | I/O | TTL | GPIO port D bit 5. |
| | CCP2 | I/O | TTL | Timer 1 capture input, compare output, or PWM output channel 2. |
| 47 | PD6 | I/O | TTL | GPIO port D bit 6. |
| | Fault | I | TTL | PWM fault detect input. |

**Table 18-1. Signals by Pin Number (Sheet 4 of 4)**

| Pin Number | Pin Name | Pin Type | Buffer Type | Description |
|---|---|---|---|---|
| 48 | PD7 | I/O | TTL | GPIO port D bit 7. |
| | C0o | O | TTL | Analog comparator 0 output. |

**Table 18-2. Signals by Signal Name (Sheet 1 of 3)**

| Pin Name | Pin Number | Pin Type | Buffer Type | Description |
|---|---|---|---|---|
| ADC0 | 1 | I | Analog | Analog-to-digital converter input 0. |
| ADC1 | 2 | I | Analog | Analog-to-digital converter input 1. |
| ADC2 | 3 | I | Analog | Analog-to-digital converter input 2. |
| ADC3 | 4 | I | Analog | Analog-to-digital converter input 3. |
| C0+ | 42 | I | Analog | Analog comparator 0 positive-reference input. |
| C0– | 44 | I | Analog | Analog comparator 0 negative-reference input. |
| C0o | 48 | O | TTL | Analog comparator 0 output. |
| CCP0 | 45 | I/O | TTL | Timer 0 capture input, compare output, or PWM output channel 0. |
| CCP1 | 13 | I/O | TTL | Timer 0 capture input, compare output, or PWM output channel 1. |
| CCP2 | 46 | I/O | TTL | Timer 1 capture input, compare output, or PWM output channel 2. |
| CCP3 | 12 | I/O | TTL | Timer 1 capture input, compare output, or PWM output channel 3. |
| CCP4 | 11 | I/O | TTL | Timer 2 capture input, compare output, or PWM output channel 4. |
| CCP5 | 43 | I/O | TTL | Timer 2 capture input, compare output, or PWM output channel 5. |
| Fault | 47 | I | TTL | PWM fault detect input. |
| GND | 8 | - | Power | Ground reference for logic and I/O pins. |
| GND | 16 | - | Power | Ground reference for logic and I/O pins. |
| GND | 24 | - | Power | Ground reference for logic and I/O pins. |
| GND | 31 | - | Power | Ground reference for logic and I/O pins. |
| I2CSCL | 33 | I/O | OD | $I^2C$ serial clock. |
| I2CSDA | 34 | I/O | OD | $I^2C$ serial data. |
| LDO | 6 | - | Power | The low drop-out regulator output voltage. This pin requires an external capacitor between the pin and GND of 1 µF or greater. |
| OSC0 | 9 | I | Analog | Oscillator crystal input or an external clock reference input. |
| OSC1 | 10 | O | Analog | Oscillator crystal output. |
| PA0 | 17 | I/O | TTL | GPIO port A bit 0. |

**Table 18-2. Signals by Signal Name (Sheet 2 of 3)**

| Pin Name | Pin Number | Pin Type | Buffer Type | Description |
|---|---|---|---|---|
| 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. |
| PB0 | 29 | I/O | TTL | GPIO port B bit 0. |
| PB1 | 30 | I/O | TTL | GPIO port B bit 1. |
| PB2 | 33 | I/O | TTL | GPIO port B bit 2. |
| PB3 | 34 | I/O | TTL | GPIO port B bit 3. |
| PB4 | 44 | I/O | TTL | GPIO port B bit 4. |
| PB5 | 43 | I/O | TTL | GPIO port B bit 5. |
| PB6 | 42 | I/O | TTL | GPIO port B bit 6. |
| PB7 | 41 | I/O | TTL | GPIO port B bit 7. |
| PC0 | 40 | I/O | TTL | GPIO port C bit 0. |
| PC1 | 39 | I/O | TTL | GPIO port C bit 1. |
| PC2 | 38 | I/O | TTL | GPIO port C bit 2. |
| PC3 | 37 | I/O | TTL | GPIO port C bit 3. |
| PC4 | 14 | I/O | TTL | GPIO port C bit 4. |
| PC5 | 13 | I/O | TTL | GPIO port C bit 5. |
| PC6 | 12 | I/O | TTL | GPIO port C bit 6. |
| PC7 | 11 | I/O | TTL | GPIO port C bit 7. |
| PD0 | 25 | I/O | TTL | GPIO port D bit 0. |
| PD1 | 26 | I/O | TTL | GPIO port D bit 1. |
| PD2 | 27 | I/O | TTL | GPIO port D bit 2. |
| PD3 | 28 | I/O | TTL | GPIO port D bit 3. |
| PD4 | 45 | I/O | TTL | GPIO port D bit 4. |
| PD5 | 46 | I/O | TTL | GPIO port D bit 5. |
| PD6 | 47 | I/O | TTL | GPIO port D bit 6. |
| PD7 | 48 | I/O | TTL | GPIO port D bit 7. |
| PE0 | 35 | I/O | TTL | GPIO port E bit 0. |

**Table 18-2. Signals by Signal Name (Sheet 3 of 3)**

| Pin Name | Pin Number | Pin Type | Buffer Type | Description |
|---|---|---|---|---|
| PE1 | 36 | I/O | TTL | GPIO port E bit 1. |
| PWM0 | 25 | O | TTL | Pulse width modulator channel 0 output. |
| PWM1 | 26 | O | TTL | Pulse width modulator channel 1 output. |
| PWM2 | 29 | O | TTL | Pulse width modulator channel 2 output. |
| PWM3 | 30 | O | TTL | Pulse width modulator channel 3 output. |
| $\overline{\text{RST}}$ | 5 | I | TTL | System reset input. |
| SSIClk | 19 | I/O | TTL | SSI clock reference (input when in slave mode and output in master mode). |
| SSIFss | 20 | I/O | TTL | SSI frame enable (input for an SSI slave device and output for an SSI master device). |
| SSIRx | 21 | I | TTL | SSI receive data input. |
| SSITx | 22 | O | TTL | SSI transmit data output. |
| SWCLK | 40 | I | TTL | Serial wire clock reference input. |
| SWDIO | 39 | I/O | TTL | Serial-wire debug input/output. |
| SWO | 37 | O | TTL | Serial-wire output. |
| TCK | 40 | I | TTL | JTAG scan test clock reference input. |
| TDI | 38 | I | TTL | JTAG scan test data input. |
| TDO | 37 | O | TTL | JTAG scan test data output. |
| TMS | 39 | I | TTL | JTAG scan test mode select input. |
| $\overline{\text{TRST}}$ | 41 | I | TTL | JTAG scan test reset input. |
| U0Rx | 17 | I | TTL | UART0 receive data input. |
| U0Tx | 18 | O | TTL | UART0 transmit data output. |
| U1Rx | 27 | I | TTL | UART1 receive data input. |
| U1Tx | 28 | O | TTL | UART1 transmit data output. |
| VDD | 7 | - | Power | Positive supply for logic and I/O pins. |
| VDD | 15 | - | Power | Positive supply for logic and I/O pins. |
| VDD | 23 | - | Power | Positive supply for logic and I/O pins. |
| VDD | 32 | - | Power | Positive supply for logic and I/O pins. |

**Table 18-3. Signals by Function, Except for GPIO (Sheet 1 of 2)**

| Function | Pin Name | Pin Number | Pin Type | Buffer Type | Description |
|---|---|---|---|---|---|
| ADC | ADC0 | 1 | I | Analog | Analog-to-digital converter input 0. |
| | ADC1 | 2 | I | Analog | Analog-to-digital converter input 1. |
| | ADC2 | 3 | I | Analog | Analog-to-digital converter input 2. |
| | ADC3 | 4 | I | Analog | Analog-to-digital converter input 3. |
| Analog Comparators | C0+ | 42 | I | Analog | Analog comparator 0 positive-reference input. |
| | C0– | 44 | I | Analog | Analog comparator 0 negative-reference input. |
| | C0o | 48 | O | TTL | Analog comparator 0 output. |
| General-Purpose Timers | CCP0 | 45 | I/O | TTL | Timer 0 capture input, compare output, or PWM output channel 0. |
| | CCP1 | 13 | I/O | TTL | Timer 0 capture input, compare output, or PWM output channel 1. |
| | CCP2 | 46 | I/O | TTL | Timer 1 capture input, compare output, or PWM output channel 2. |
| | CCP3 | 12 | I/O | TTL | Timer 1 capture input, compare output, or PWM output channel 3. |
| | CCP4 | 11 | I/O | TTL | Timer 2 capture input, compare output, or PWM output channel 4. |
| | CCP5 | 43 | I/O | TTL | Timer 2 capture input, compare output, or PWM output channel 5. |
| I2C | I2CSCL | 33 | I/O | OD | $I^2C$ serial clock. |
| | I2CSDA | 34 | I/O | OD | $I^2C$ serial data. |
| JTAG/SWD/SWO | SWCLK | 40 | I | TTL | Serial-wire clock reference input. |
| | SWDIO | 39 | I/O | TTL | Serial-wire debug input/output. |
| | SWO | 37 | O | TTL | Serial-wire output. |
| | TCK | 40 | I | TTL | JTAG scan test clock reference input. |
| | TDI | 38 | I | TTL | JTAG scan test data input. |
| | TDO | 37 | O | TTL | JTAG scan test data output. |
| | TMS | 39 | I | TTL | JTAG scan test mode select input. |
| | $\overline{\text{TRST}}$ | 41 | I | TTL | JTAG scan test reset input. |

**Table 18-3. Signals by Function, Except for GPIO (Sheet 2 of 2)**

| Function | Pin Name | Pin Number | Pin Type | Buffer Type | Description |
|---|---|---|---|---|---|
| Power | GND | 8 | - | Power | Ground reference for logic and I/O pins. |
| | GND | 16 | - | Power | Ground reference for logic and I/O pins. |
| | GND | 24 | - | Power | Ground reference for logic and I/O pins. |
| | GND | 31 | - | Power | Ground reference for logic and I/O pins. |
| | LDO | 6 | - | Power | The low drop-out regulator output voltage. This pin requires an external capacitor between the pin and GND of 1 µF or greater. |
| | VDD | 7 | - | Power | Positive supply for logic and I/O pins. |
| | VDD | 15 | - | Power | Positive supply for logic and I/O pins. |
| | VDD | 23 | - | Power | Positive supply for logic and I/O pins. |
| | VDD | 32 | - | Power | Positive supply for logic and I/O pins. |
| PWM | Fault | 47 | I | TTL | PWM fault detect input. |
| | PWM0 | 25 | O | TTL | Pulse width modulator channel 0 output. |
| | PWM1 | 26 | O | TTL | Pulse width modulator channel 1 output. |
| | PWM2 | 29 | O | TTL | Pulse width modulator channel 2 output. |
| | PWM3 | 30 | O | TTL | Pulse width modulator channel 3 output. |
| SSI | SSIClk | 19 | I/O | TTL | SSI clock reference (input when in slave mode and output in master mode). |
| | SSIFss | 20 | I/O | TTL | SSI frame enable (input for an SSI slave device and output for an SSI master device). |
| | SSIRx | 21 | I | TTL | SSI receive data input. |
| | SSITx | 22 | O | TTL | SSI transmit data output. |
| System Control & Clocks | OSC0 | 9 | I | Analog | Oscillator crystal input or an external clock reference input. |
| | OSC1 | 10 | O | Analog | Oscillator crystal output. |
| | $\overline{\text{RST}}$ | 5 | I | TTL | System reset input. |
| UART | U0Rx | 17 | I | TTL | UART0 receive data input. |
| | U0Tx | 18 | O | TTL | UART0 transmit data output. |
| | U1Rx | 27 | I | TTL | UART1 receive data input. |
| | U1Tx | 28 | O | TTL | UART1 transmit data output. |

**Table 18-4. GPIO Pins and Alternate Functions (Sheet 1 of 2)**

| GPIO Pin | Pin Number | Multiplexed Function | Multiplexed Function |
|---|---|---|---|
| PA0 | 17 | U0Rx | |
| PA1 | 18 | U0Tx | |
| PA2 | 19 | SSIClk | |
| PA3 | 20 | SSIFss | |
| PA4 | 21 | SSIRx | |
| PA5 | 22 | SSITx | |
| PB0 | 29 | PWM2 | |
| PB1 | 30 | PWM3 | |
| PB2 | 33 | I2CSCL | |
| PB3 | 34 | I2CSDA | |
| PB4 | 44 | C0- | |
| PB5 | 43 | CCP5 | |
| PB6 | 42 | C0+ | |
| PB7 | 41 | $\overline{\text{TRST}}$ | |
| PC0 | 40 | TCK | SWCLK |
| PC1 | 39 | TMS | SWDIO |
| PC2 | 38 | TDI | |
| PC3 | 37 | TDO | SWO |
| PC4 | 14 | | |
| PC5 | 13 | CCP1 | |
| PC6 | 12 | CCP3 | |
| PC7 | 11 | CCP4 | |
| PD0 | 25 | PWM0 | |
| PD1 | 26 | PWM1 | |
| PD2 | 27 | U1Rx | |
| PD3 | 28 | U1Tx | |
| PD4 | 45 | CCP0 | |
| PD5 | 46 | CCP2 | |
| PD6 | 47 | Fault | |
| PD7 | 48 | C0o | |

**Table 18-4. GPIO Pins and Alternate Functions (Sheet 2 of 2)**

| GPIO Pin | Pin Number | Multiplexed Function | Multiplexed Function |
|---|---|---|---|
| PE0 | 35 | | |
| PE1 | 36 | | |

# 19 Operating Characteristics

**Table 19-1. Temperature Characteristics**

| Characteristic | Symbol | Value | Unit |
|---|---|---|---|
| Operating temperature range[a] | $T_A$ | -40 to +85 for industrial | °C |

a. Maximum storage temperature is 150°C.

**Table 19-2. Thermal Characteristics**

| Characteristic | Symbol | Value | Unit |
|---|---|---|---|
| Thermal resistance (junction to ambient)[a] | $\theta_{JA}$ | 76 | °C/W |
| Average junction temperature[b] | $T_J$ | $T_A + (P_{AVG} \bullet \theta_{JA})$ | °C |
| Maximum junction temperature | $T_{JMAX}$ | pending[c] | °C |

a. Junction to ambient thermal resistance $\theta_{JA}$ numbers are determined by a package simulator.
b. Power dissipation is a function of temperature.
c. Pending characterization completion.

# 20 Electrical Characteristics

## 20.1 DC Characteristics

### 20.1.1 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 20-1. Maximum Ratings**

| Characteristic[a] | Symbol | Value | Unit |
|---|---|---|---|
| Supply voltage range ($V_{DD}$) | $V_{DD}$ | 0.0 to +3.6 | V |
| Input voltage | $V_{IN}$ | -0.3 to 5.5 | V |
| Maximum current for pins, excluding pins operating as GPIOs | I | 100 | mA |
| Maximum current for GPIO pins | I | 100 | mA |

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 $V_{DD}$).

### 20.1.2 Recommended DC Operating Conditions

**Table 20-2. Recommended DC Operating Conditions**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{DD}$ | Supply voltage | 3.0 | 3.3 | 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 |
| $V_{SIH}$ | High-level input voltage for Schottky inputs | 0.8 * $V_{DD}$ | - | $V_{DD}$ | V |
| $V_{SIL}$ | Low-level input voltage for Schottky inputs | 0 | - | 0.2 * $V_{DD}$ | V |
| $V_{OH}$ | High-level output voltage | 2.4 | - | - | V |
| $V_{OL}$ | Low-level output voltage | - | - | 0.4 | V |

**Table 20-2. Recommended DC Operating Conditions (Continued)**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $I_{OH}$ | 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 |

## 20.1.3 On-Chip Low Drop-Out (LDO) Regulator Characteristics

**Table 20-3. LDO Regulator Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{LDOOUT}$ | Programmable internal (logic) power supply output value | 2.25 | - | 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 | - | µF |

### 20.1.4 Power Specifications

The power measurements specified in Table 20-4 are run on the core processor using SRAM with the following specifications:

- $V_{DD}$=3.3 V
- LDO=2.5
- Temperature=25°C
- System Clock=25 MHz (with PLL)
- Code `while(1){}` executed from SRAM with no active peripherals

**Table 20-4. Power Specifications**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $I_{DD_RUN}$ | Run mode | - | 40[a] | pending[a] | mA |
| $I_{DD_SLEEP}$ | Sleep mode | - | pending[a] | pending[a] | µA |
| $I_{DD_DEEPSLEEP}$ | Deep-Sleep mode | - | pending[a] | pending[a] | µA |

a. Pending characterization completion.

### 20.1.5 Flash Memory Characteristics

**Table 20-5. Flash Memory Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $PE_{CYC}$ | Number of guaranteed program/erase cycles[a] before failure | 10,000 | - | - | 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 | 200 | - | - | ms |

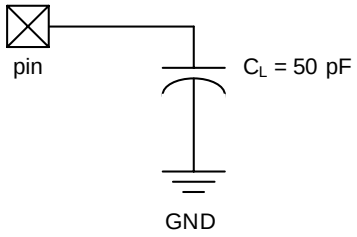
a. A program/erase cycle is defined as switching the bits from 1-> 0 -> 1.

# 20.2 AC Characteristics

## 20.2.1 Load Conditions

Unless otherwise specified, the following conditions are true for all timing measurements. Timing measurements are for 4-mA drive strength.

**Figure 20-1. Load Conditions**




## 20.2.2 Clocks

**Table 20-6. Phase Locked Loop (PLL) Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $f_{REF_CRYSTAL}$ | Crystal reference[a] | 3.579545 | - | 8.192 | MHz |
| $f_{REF_EXT}$ | External clock reference[a] | 3.579545 | - | 8.192 | MHz |
| $f_{PLL}$ | PLL frequency[b] | - | 200 | - | 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 (see page 80).
b. PLL frequency is automatically calculated by the hardware based on the XTAL field of the **RCC** register.

**Table 20-7. Clock Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $f_{IOSC}$ | Internal oscillator frequency | 7 | 15 | 22 | MHz |
| $f_{MOSC}$ | Main oscillator frequency | 1 | - | 8 | MHz |
| $t_{MOSC_PER}$ | Main oscillator period | 125 | - | 1000 | ns |
| $f_{REF_CRYSTAL_BYPASS}$ | Crystal reference using the main oscillator (PLL in BYPASS mode)[a] | 1 | - | 8 | MHz |
| $f_{REF_EXT_BYPASS}$ | External clock reference (PLL in BYPASS mode)[a] | 0 | - | 25 | MHz |
| $f_{SYSTEM_CLOCK}$ | System clock | 0 | - | 25 | MHz |

a. The ADC module cannot be used when the PLL is in Bypass mode (BYPASS set to 1 in the **RCC** register).

### 20.2.3 Temperature Sensor

**Table 20-8. Temperature Sensor Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{TSO}$ | Output voltage | 0.3 | - | 2.7 | V |
| $t_{TSERR}$ | Output voltage temperature accuracy | - | - | ± 3.5 | °C |
| $t_{TSNL}$ | Output temperature nonlinearity | - | - | ± 1 | °C |

### 20.2.4 Analog-to-Digital Converter

**Table 20-9. ADC Characteristics**

| 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 | V |
| | Maximum differential, full-scale analog input voltage | - | - | 1.5 | V |
| | Minimum differential, full-scale analog input voltage | - | - | -1.5 | V |
| $C_{ADCIN}$ | Equivalent input capacitance | - | 1 | - | pF |
| N | Resolution | - | 10 | - | bits |
| $f_{ADC}$ | ADC internal clock frequency | 3.5 | 4 | 4.5 | MHz |
| $t_{ADCCONV}$ | Conversion time | - | - | 16 | $t_{ADC}$ cycles[a] |
| $f_{ADCCONV}$ | Conversion rate | 219 | 250 | 281 | k samples/s |
| INL | Integral nonlinearity | - | - | ±1 | LSB |
| DNL | Differential nonlinearity | - | - | ±1 | LSB |
| OFF | Offset | - | - | +2 | LSB |
| GAIN | Gain | - | - | ±2 | LSB |

a. $t_{ADC} = 1/f_{ADC\ clock}$

### 20.2.5 Analog Comparator

**Table 20-10. Analog Comparator Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{OS}$ | Input offset voltage | - | ± 10 | ± 25 | mV |
| $V_{CM}$ | Input common mode voltage range | 0 | - | $V_{DD}$-1.5 | V |
| $C_{MRR}$ | Common mode rejection ratio | 50 | - | - | dB |
| $T_{RT}$ | Response time | - | - | 1 | µs |
| $T_{MC}$ | Comparator mode change to Output Valid | - | - | 10 | µs |

**Table 20-11. Analog Comparator Voltage Reference Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $R_{HR}$ | Resolution high range | - | $V_{DD}$/32 | - | LSB |
| $R_{LR}$ | Resolution low range | - | $V_{DD}$/24 | - | LSB |
| $A_{HR}$ | Absolute accuracy high range | - | - | ± 1/2 | LSB |
| $A_{LR}$ | Absolute accuracy low range | - | - | ± 1/4 | LSB |

### 20.2.6 $I^2C$

**Table 20-12. $I^2C$ 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 |

**Table 20-12. I²C Characteristics (Continued)**

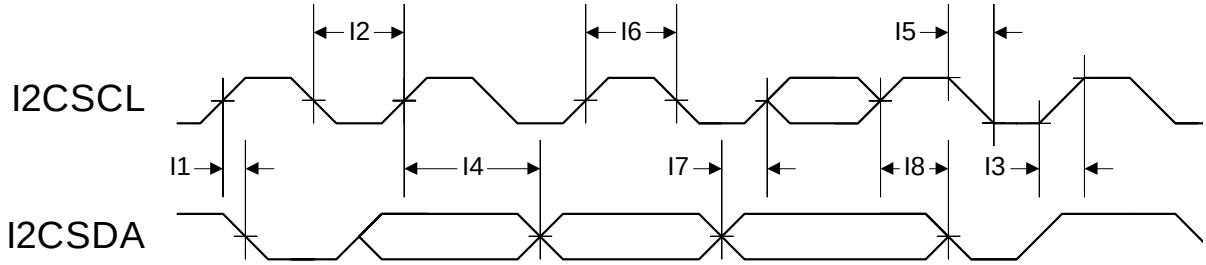
| Parameter No. | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| I8[a] | $t_{SCSR}$ | Start condition setup time (for repeated start condition only) | 36 | - | - | system clocks |
| I9[a] | $t_{SCS}$ | Stop condition setup time | 24 | - | - | system clocks |

a. Values depend on the value programmed into the TPR bit in the **I²C Master Timer Period (I2CMTPR)** register (see page 320); a TPR programmed for the maximum I2CSCL frequency (TPR=0x2) results in a minimum output timing as shown in the table above. The I²C 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 20-2. I²C Timing**



## 20.2.7 Synchronous Serial Interface (SSI)

**Table 20-13. 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 | - | 1/2 | - | $t_{CLK_PER}$ |
| S3 | $t_{CLK_LOW}$ | SSIClk low time | - | 1/2 | - | $t_{CLK_PER}$ |
| S4 | $t_{CLKRF}$ | SSIClk rise/fall time | - | 7.4 | 26 | ns |
| S5 | $t_{DMD}$ | Data from master valid delay time | 0 | - | 20 | ns |
| S6 | $t_{DMS}$ | Data from master setup time | 20 | - | - | ns |
| S7 | $t_{DMH}$ | Data from master hold time | 40 | - | - | ns |
| S8 | $t_{DSS}$ | Data from slave setup time | 20 | - | - | ns |
| S9 | $t_{DSH}$ | Data from slave hold time | 40 | - | - | ns |

**Figure 20-3. SSI Timing for TI Frame Format (FRF=01), Single Transfer Timing Measurement**

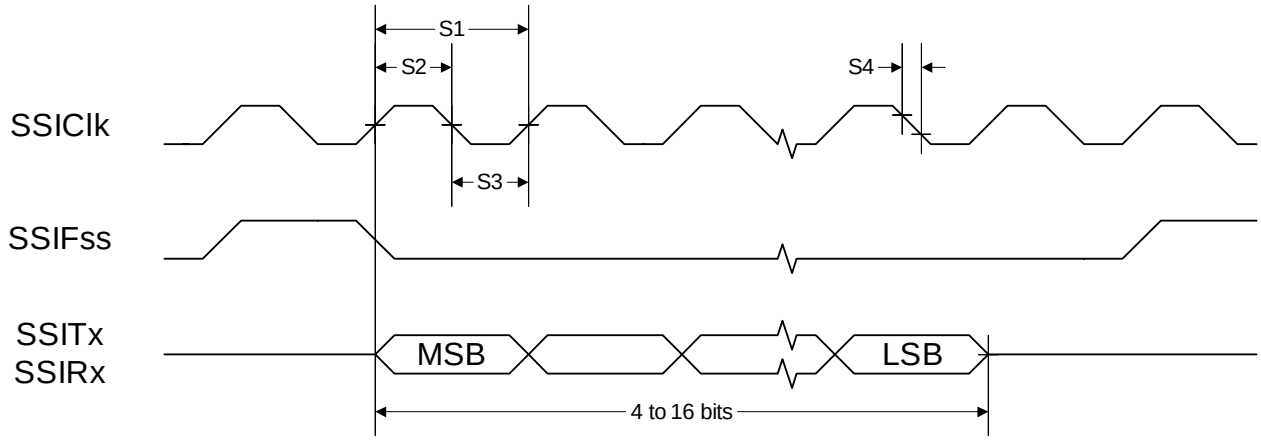


**Figure 20-4. SSI Timing for MICROWIRE Frame Format (FRF=10), Single Transfer**

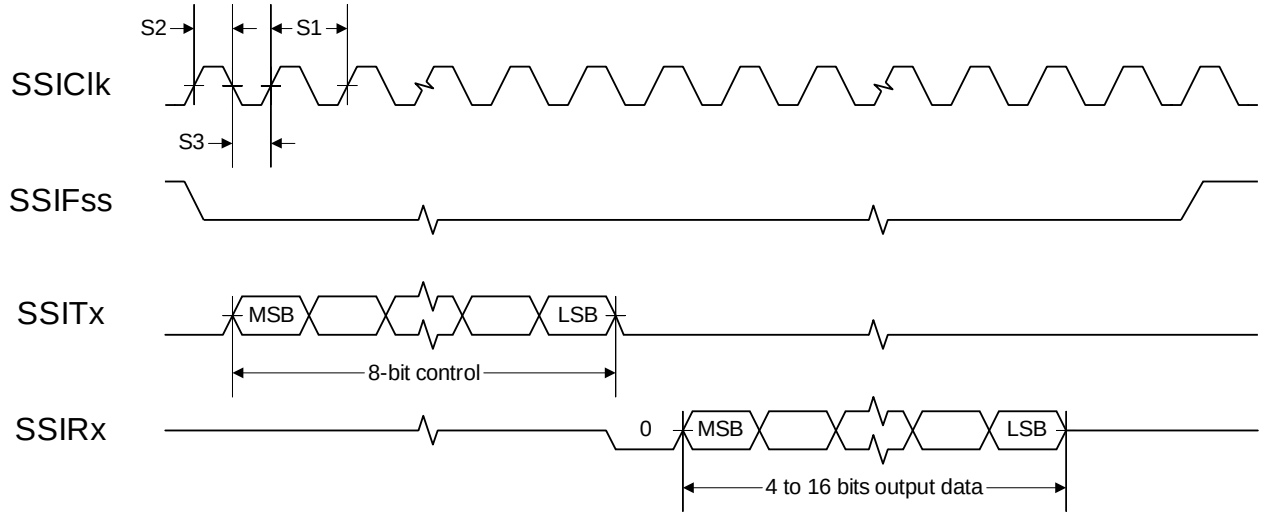
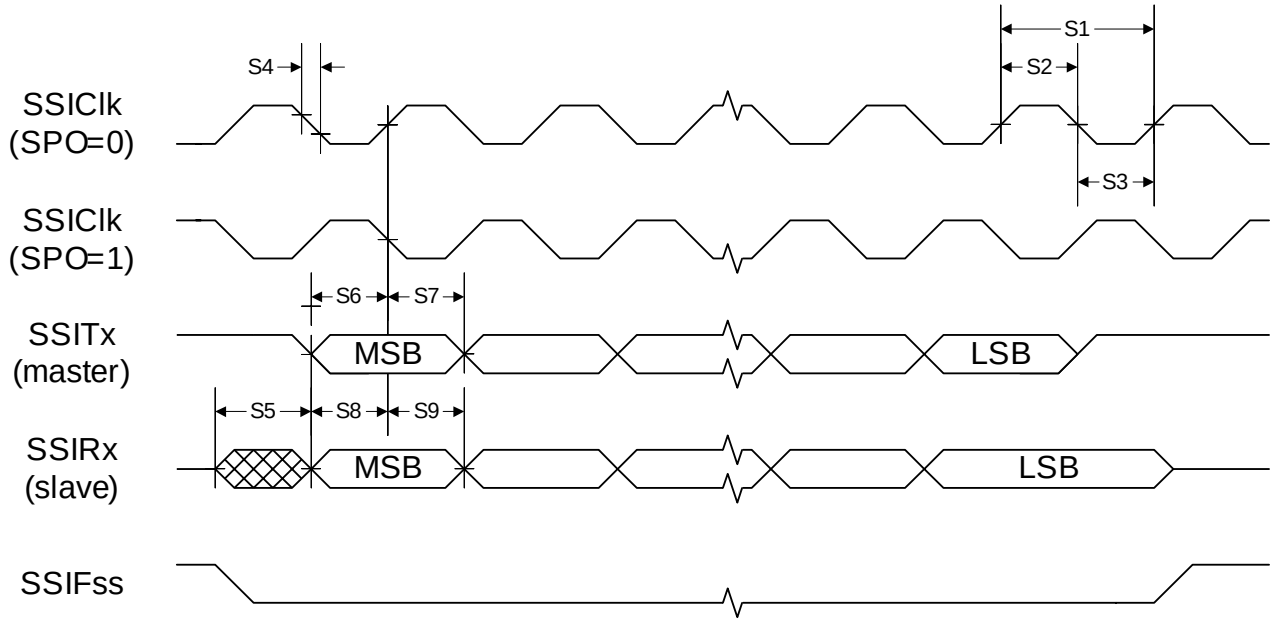



**Figure 20-5. SSI Timing for SPI Frame Format (FRF=00), with SPH=1**



## 20.2.8 JTAG and Boundary Scan

**Table 20-14. 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 | - | ½ $t_{TCK}$ | - | ns |
| J4 | $t_{TCK_HIGH}$ | TCK clock High time | - | ½ $t_{TCK}$ | - | 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 |
| J14 | $t_{TRST}$ | $\overline{\text{TRST}}$ assertion time | 100 | - | - | ns |
| J15 | $t_{TRST_SU}$ | $\overline{\text{TRST}}$ setup time to TCK rise | 10 | - | - | ns |

**Figure 20-6. JTAG Test Clock Input Timing**

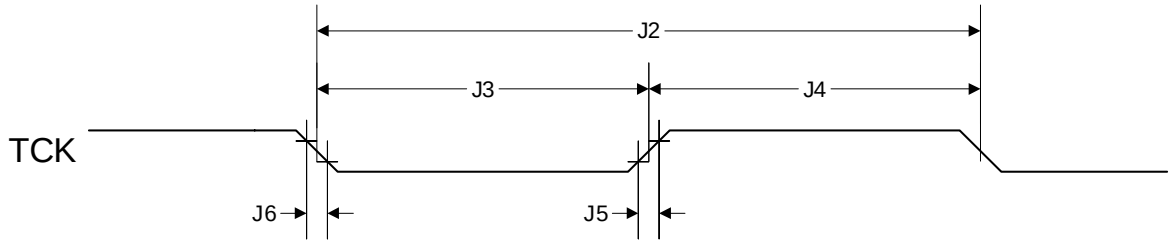



**Figure 20-7. JTAG Test Access Port (TAP) Timing**

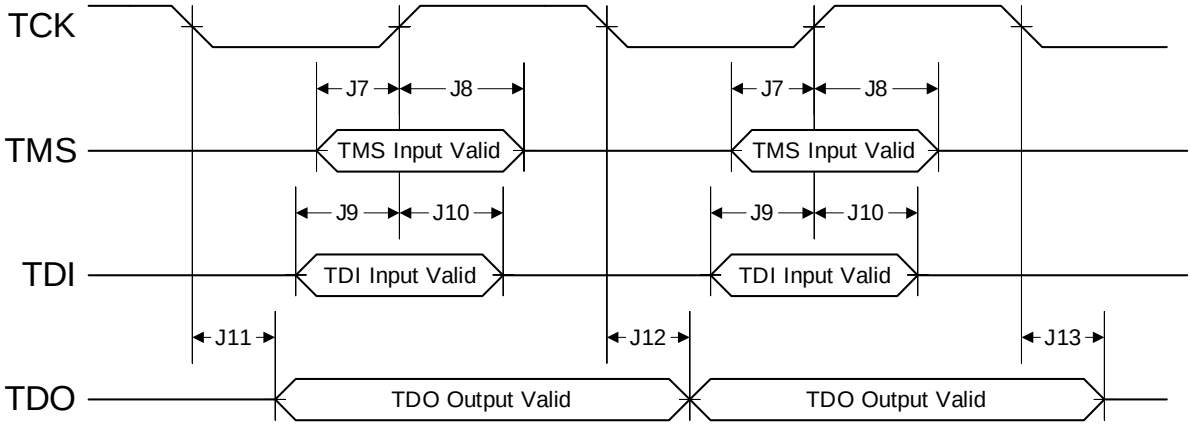
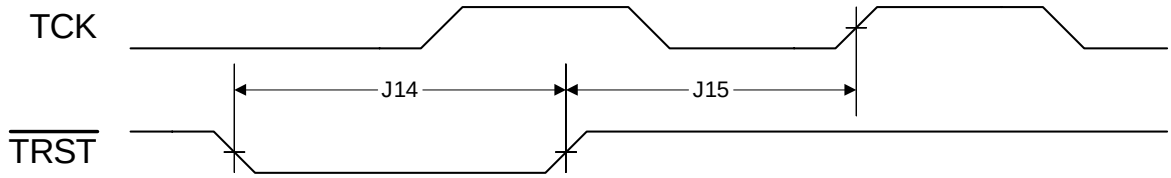



**Figure 20-8. JTAG $\overline{\text{TRST}}$ Timing**



## 20.2.9 General-Purpose I/O

**Table 20-15. GPIO Characteristics[a]**

| Parameter | Parameter Name | Condition | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| $t_{GPIOR}$ | GPO 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 |
| $t_{GPIOF}$ | GPO 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 |

a. All GPIOs are 5 V-tolerant.

## 20.2.10 Reset

**Table 20-16. 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 | 15 | - | 30 | ms |
| R6 | $T_{IRBOR}$ | Internal reset timeout after BOR[a] | 2.5 | - | 20 | µs |
| R7 | $T_{IRHWR}$ | Internal reset timeout after hardware reset ($\overline{RST}$ pin) | 15 | - | 30 | 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 | $T_{IRLDOR}$ | Internal reset timeout after LDO reset[a] | 2.5 | - | 20 | µs |
| R11 | $T_{VDDRISE}$ | Supply voltage ($V_{DD}$) rise time (0V-3.3V) | | | 100 | ms |

a. 20 * $t_{MOSC_PER}$

**Figure 20-9. External Reset Timing ($\overline{RST}$)**

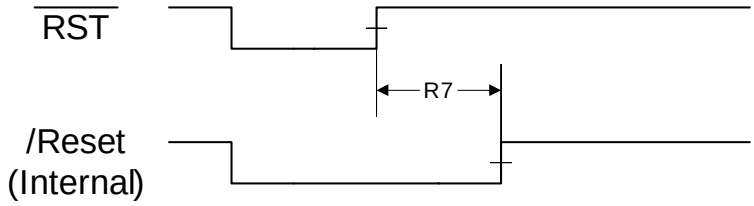



**Figure 20-10. Power-On Reset Timing**

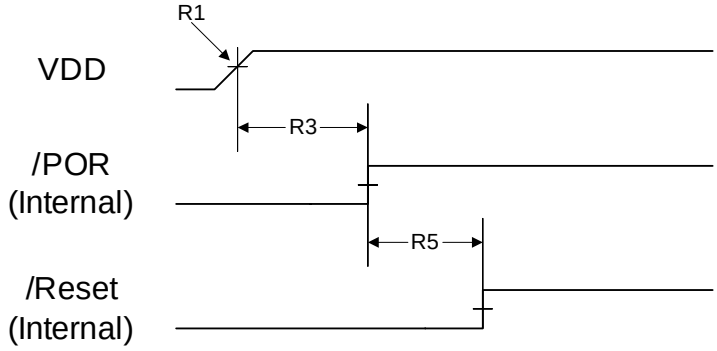



**Figure 20-11. Brown-Out Reset Timing**

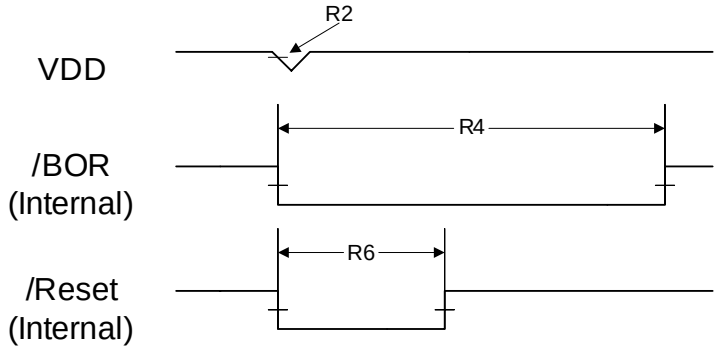



**Figure 20-12. Software Reset Timing**

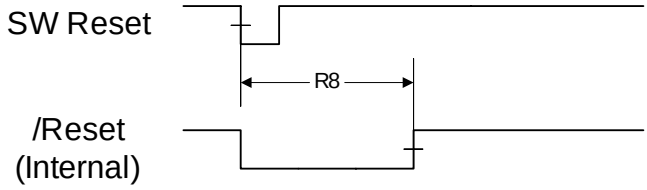


**Figure 20-13. Watchdog Reset Timing**

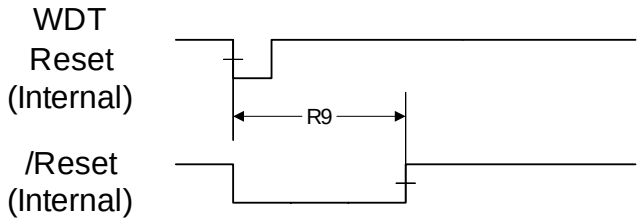
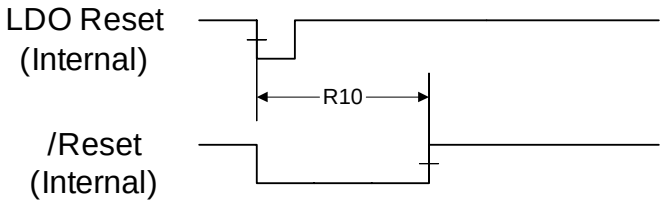
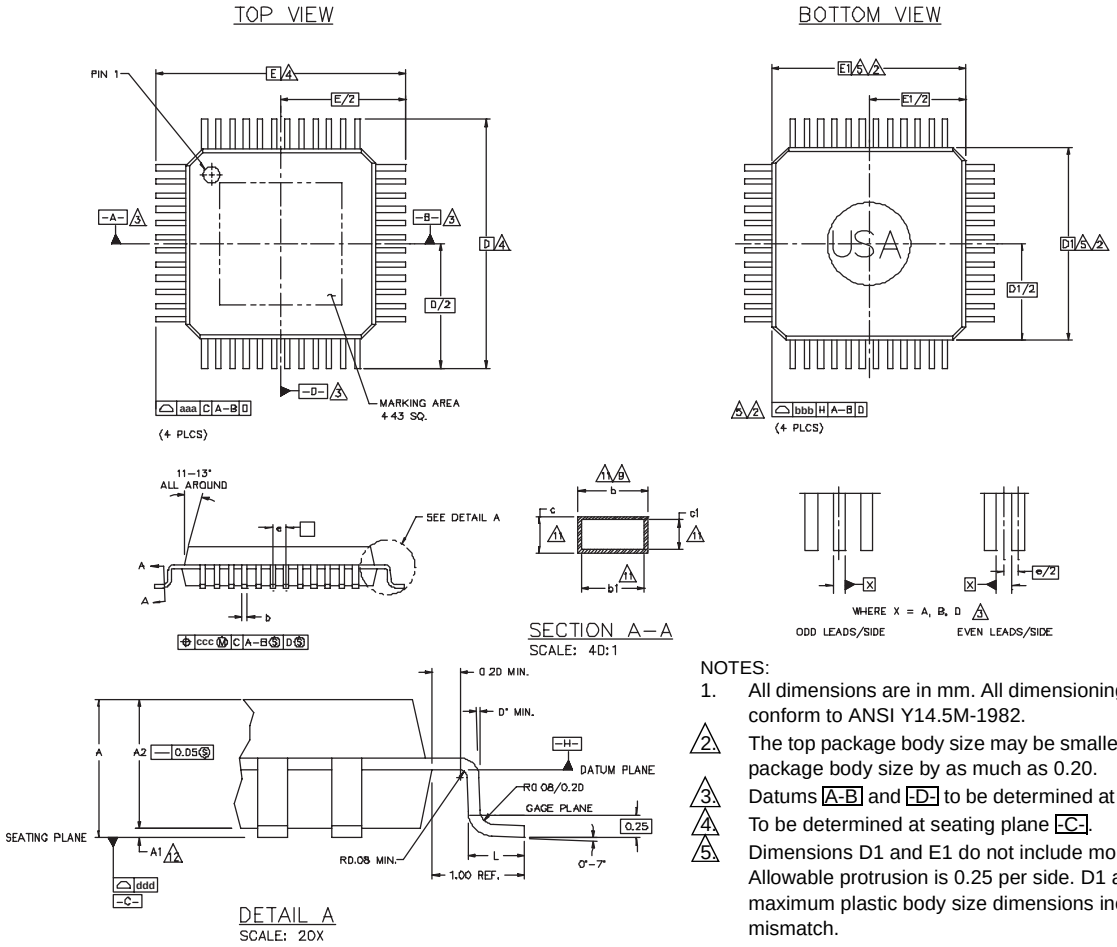



**Figure 20-14. LDO Reset Timing**



# 21 Package Information

**Figure 21-1. 48-Pin LQFP Package**




| SYMBOL | LEAD COUNT; FOOT PRINT | | | NOTE |
|---|---|---|---|---|
| | 48, 2.0 FP | | | |
| | MIN | NOM | MAX | |
| A | === | === | 1.60 | |
| A1 | 0.05 | === | 0.15 | |
| A2 | 1.35 | 1.40 | 1.45 | |
| D | 8.00 BSC | | | |
| D1 | 7.00 BSC | | | |
| E | 9.00 BSC | | | |
| E1 | 7.00 BSC | | | |
| L | 0.45 | 0.80 | 0.75 | |
| | 0.50 BSC | | | |
| b | 0.17 | 0.22 | 0.27 | |
| b1 | 0.17 | 0.20 | 0.23 | |
| c | 0.09 | === | 0.20 | |
| c1 | 0.09 | === | 0.16 | |
| | Tolerances of form and position | | | |
| aaa | 0.20 | | | |
| bbb | 0.20 | | | |
| ccc | 0.08 | | | |
| ddd | 0.08 | | | |

NOTES:

1. All dimensions are in mm. All dimensioning and tolerancing conform to ANSI Y14.5M-1982.
2. The top package body size may be smaller than the bottom package body size by as much as 0.20.
3. Datums A-B and -D- to be determined at datum plane -H-.
4. To be determined at seating plane -C-.
5. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 per side. D1 and E1 are maximum plastic body size dimensions including mold mismatch.
6. Surface finish of the package is #24-27 Charmille (1.6-2.3µm$R_0$) Pin 1 and ejector pin may be less than 0.1µm$R_0$.
7. Dambar removal protrusion does not exceed 0.08. Intrusion does not exceed 0.03.
8. Burr does not exceed 0.08 in any direction.
9. Dimension b does not include Dambar protrusion. Allowable Dambar protrusion shall not cause the lead width to exceed the maximum b dimension by more than 0.08. Dambar cannot be located on the lower radius or the foot. Minimum space between protrusion and adjacent lead is 0.07 for 0.40 and 0.50 pitch package.
10. Corner radius of plastic body does not exceed 0.20.
11. These dimensions apply to the flat section of the lead between 0.10 and 0.25 from the lead tip.
12. A1 is defined as the distance from the seating plane to the lowest point of the package body.
13. Finish of leads is tin plated.
14. All specifications and dimensions are subjected to IPAC'S manufacturing process flow and materials.
15. The packages described in the drawing conform to JEDEC M5-026A. Where discrepancies between the JEDEC and IPAC documents exist, this drawing will take the precedence.

# Appendix A. Serial Flash Loader

The Stellaris serial flash loader is used to download code to the flash memory of a device without the use of a debug interface. The serial flash loader uses a simple packet interface to provide synchronous communication with the device. The flash loader runs off the crystal and does not enable the PLL, so its speed is determined by the crystal used. The two serial interfaces that can be used are the UART0 and SSI interfaces. For simplicity, both the data format and communication protocol are identical for both serial interfaces.

## A.1 Interfaces

Once communication with the flash loader is established via one of the serial interfaces, that interface is used until the flash loader is reset or new code takes over. For example, once you start communicating using the SSI port, communications with the flash loader via the UART are disabled until the device is reset.

### A.1.1 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 flash 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 crystal frequency of the board that is running the serial flash loader. This is actually the same as the hardware limitation for the maximum baud rate for any UART on a Stellaris device.

In order to determine the baud rate, the serial flash loader needs to determine the relationship between its own crystal frequency and the baud rate. This is enough information for the flash 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 flash loader two bytes that are both 0x55. This generates a series of pulses to the flash 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 flash 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 flash loader acknowledges that it has received a synchronization pattern correctly. For example, the time to wait for data back from the flash 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 0.35ms.

### A.1.2 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 the section on SSI formats for more details on 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 crystal frequency of the board running the flash loader. Since the host device is the master, the SSI on the flash loader device does not need to determine the clock as it is provided directly by the host.

## A.2 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.2.1 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.2.2 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 commands that interact with the flash.

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.2.3 Receiving Packets

The flash loader sends a packet of data in the same format that it receives a packet. The flash 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 flash loader. Once the device communicating with the flash 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 flash 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 flash loader, as the flash 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 flash loader.

## A.3 Commands

The next section defines the list of commands that can be sent to the flash 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.3.1 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 flash loader.

### A.3.2 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 flash loader knows that the data has been read.

```
Byte[0] = 0x03
Byte[1] = checksum(Byte[2])
Byte[2] = COMMAND_GET_STATUS
```

### A.3.3 COMMAND_DOWNLOAD (0x21)

This command is sent to the flash 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 flash 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.3.4 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. The caller should limit transfers of data to a maximum 8 bytes of packet data to allow the flash to program successfully and not overflow input buffers of the serial interfaces. 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 flash loader sends a NAK to this command, the flash loader does not increment the current address to allow retransmission of the previous 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.3.5 COMMAND_RUN (0x22)

This command is used to tell the flash 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 flash 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.3.6 COMMAND_RESET (0x25)

This command is used to tell the flash loader device to reset. This is useful when downloading a new image that overwrote the flash loader and wants to start from a full 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 flash loader if a critical error occurs and the host device wants to restart communication with the flash loader.

```
Byte[0] = 3
Byte[1] = checksum(Byte[2])
Byte[2] = COMMAND_RESET
```

The flash loader responds with an ACK signal back to the host device before actually executing the software reset to the device running the flash loader. This allows the host to know that the command was received successfully and the part will be reset.

# Ordering and Contact Information

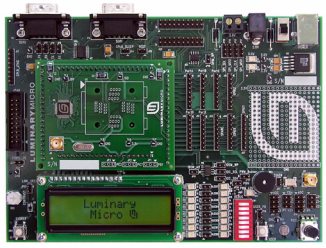
## Ordering Information

| Order Number | Features | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Flash (KB) | SRAM (KB) | GPIOs[a] | Timers[b] | ADC | | UART(s) | SSI | $I^2C$ | Analog Comparator(s) | PWM[c] | | QEI | Operating Temperature[d] | Package[e] | Speed (Clock Frequency in MHz) |
| | | | | | Samples Per Second | # of 10-Bit Channels | | | | | PWM Pins | CCP Pins | | | | |
| LM3S316-IQN25<br>LM3S316-IQN25(T)[f] | 16 | 4 | 3 to 32 | 3 | 250K | 4 | 2 | √ | √ | 1 | 4 | 6 | - | I | QN | 25 |

a. Minimum is number of pins dedicated to GPIO; additional pins are available if certain peripherals are not used. See data sheet for details.
b. One timer available as RTC.
c. PWM motion control functionality can be achieved through dedicated motion control hardware (using the PWM pins) or through the motion control features of the general-purpose timers (using the CCP pins). See data sheet for details.
d. I=Industrial (–40 to 85°C).
e. QN=48-pin RoHS-compliant PQFP.
f. T=Tape and Reel.

## Development Kit

The Luminary Micro Stellaris™ Family Development Kit provides the hardware and software tools that engineers need to begin development quickly. Ask your Luminary Micro distributor for part number DK-LM3S811. See the Luminary Micro website for the latest tools available.




***Tools to begin development quickly***

## Company Information

Luminary Micro, Inc. designs, markets, and sells ARM Cortex-M3 based microcontrollers for use in embedded applications within the industrial, commercial, and consumer markets. Luminary Micro is ARM's lead partner in the implementation of the Cortex-M3 core. Please contact us if you are interested in obtaining further information about our company or our products.

Luminary Micro, Inc.
2499 South Capital of Texas Hwy, Suite A-100
Austin, TX 78746
Main: +1-512-279-8800
Fax: +1-512-279-8879
http://www.luminarymicro.com
sales@luminarymicro.com

# Support Information

For support on Luminary Micro products, contact:

support@luminarymicro.com
+1-512-279-8800, ext. 3