MSP430x5xx Family

User's Guide

Literature Number: SLAU208F
June 2008–Revised March 2010

Contents

Preface **17**

1 System Resets, Interrupts, and Operating Modes, System Control Module (SYS) **19**

1.1 System Control Module (SYS) Introduction 20
1.2 System Reset and Initialization 20
1.2.1 Device Initial Conditions After System Reset 22
1.3 Interrupts 22
1.3.1 (Non)Maskable Interrupts (NMIs) 22
1.3.2 SNMI Timing 24
1.3.3 Maskable Interrupts 25
1.3.4 Interrupt Processing 25
1.3.5 Interrupt Nesting 26
1.3.6 Interrupt Vectors 26
1.3.7 SYS Interrupt Vector Generators 27
1.4 Operating Modes 28
1.4.1 Entering and Exiting Low-Power Modes LPM0 Through LPM4 31
1.4.2 Entering and Exiting Low-Power Modes LPMx.5 31
1.4.3 Extended Time in Low-Power Modes 32
1.5 Principles for Low-Power Applications 33
1.6 Connection of Unused Pins 33
1.7 Reset pin ($\overline{\textbf{RST}}$/NMI) Configuration 33
1.8 Configuring JTAG pins 33
1.9 Boot Code 34
1.10 Bootstrap Loader (BSL) 34
1.11 Memory Map – Uses and Abilities 35
1.11.1 Vacant Memory Space 35
1.11.2 JTAG Lock Mechanism via the Electronic Fuse 35
1.12 JTAG Mailbox (JMB) System 36
1.12.1 JMB Configuration 36
1.12.2 JMBOUT0 and JMBOUT1 Outgoing Mailbox 36
1.12.3 JMBIN0 and JMBIN1 Incoming Mailbox 36
1.12.4 JMB NMI Usage 36
1.13 Device Descriptor Table 37
1.13.1 Identifying Device Type 38
1.13.2 TLV Descriptors 39
1.13.3 Peripheral discovery descriptor 39
1.13.4 Calibration Values 42
1.14 Special Function Registers (SFRs) 45
1.15 SYS Configuration Registers 49

2 Power Management Module and Supply Voltage Supervisor **57**

2.1 Power Management Module (PMM) Introduction 58
2.2 PMM Operation 60
2.2.1 V_{CORE} and the Regulator 60
2.2.2 Supply Voltage Supervisor and Monitor 60
2.2.3 Supply Voltage Supervisor and Monitor - Power-Up 65
2.2.4 Increasing V_{CORE} to Support Higher MCLK Frequencies 65

2.2.5 Decreasing V_{CORE} for Power Optimization ... 66
2.2.6 LPM3.5, LPM4.5 ... 67
2.2.7 Brownout Reset (BOR), Software BOR, Software POR ... 67
2.2.8 SVS/SVM Performance Modes and Wakeup Times ... 67
2.2.9 PMM Interrupts ... 69
2.2.10 Port I/O Control ... 69
2.2.11 Supply Voltage Monitor Output (SVMOUT, Optional) ... 69
2.3 PMM Registers ... 70

3 Unified Clock System (UCS) ... 77
3.1 Unified Clock System (UCS) Introduction ... 78
3.2 UCS Operation ... 80
3.2.1 UCS Module Features for Low-Power Applications ... 80
3.2.2 Internal Very-Low-Power Low-Frequency Oscillator (VLO) ... 80
3.2.3 Internal Trimmed Low-Frequency Reference Oscillator (REFO) ... 81
3.2.4 XT1 Oscillator ... 81
3.2.5 XT2 Oscillator ... 82
3.2.6 Digitally-Controlled Oscillator (DCO) ... 83
3.2.7 Frequency Locked Loop (FLL) ... 83
3.2.8 DCO Modulator ... 84
3.2.9 Disabling FLL Hardware and Modulator ... 85
3.2.10 FLL Operation From Low-Power Modes ... 85
3.2.11 Operation From Low-Power Modes, Requested by Peripheral Modules ... 85
3.2.12 UCS Module Fail-Safe Operation ... 86
3.2.13 Synchronization of Clock Signals ... 89
3.3 Module Oscillator (MODOSC) ... 90
3.3.1 MODOSC Operation ... 90
3.4 UCS Module Registers ... 91

4 CPUX ... 101
4.1 MSP430X CPU (CPUX) Introduction ... 102
4.2 Interrupts ... 104
4.3 CPU Registers ... 105
4.3.1 Program Counter (PC) ... 105
4.3.2 Stack Pointer (SP) ... 105
4.3.3 Status Register (SR) ... 107
4.3.4 Constant Generator Registers (CG1 and CG2) ... 108
4.3.5 General-Purpose Registers (R4 –R15) ... 109
4.4 Addressing Modes ... 111
4.4.1 Register Mode ... 112
4.4.2 Indexed Mode ... 113
4.4.3 Symbolic Mode ... 117
4.4.4 Absolute Mode ... 122
4.4.5 Indirect Register Mode ... 124
4.4.6 Indirect Autoincrement Mode ... 125
4.4.7 Immediate Mode ... 126
4.5 MSP430 and MSP430X Instructions ... 128
4.5.1 MSP430 Instructions ... 128
4.5.2 MSP430X Extended Instructions ... 133
4.6 Instruction Set Description ... 144
4.6.1 Extended Instruction Binary Descriptions ... 145
4.6.2 MSP430 Instructions ... 147
4.6.3 Extended Instructions ... 198
4.6.4 Address Instructions ... 239

5 Flash Memory Controller ... 255

5.1 Flash Memory Introduction 256
5.2 Flash Memory Segmentation 257
5.2.1 Segment A 258
5.3 Flash Memory Operation 259
5.3.1 Erasing Flash Memory 259
5.3.2 Writing Flash Memory 263
5.3.3 Flash Memory Access During Write or Erase 270
5.3.4 Checking Flash memory 271
5.3.5 Configuring and Accessing the Flash Memory Controller 271
5.3.6 Flash Memory Controller Interrupts 271
5.3.7 Programming Flash Memory Devices 272
5.4 Flash Memory Registers 273

6 RAM Controller 277
6.1 Ram Controller (RAMCTL) Introduction 278
6.2 RAMCTL Operation 278
6.3 RAMCTL Module Registers 279

7 DMA Controller 281
7.1 Direct Memory Access (DMA) Introduction 282
7.2 DMA Operation 284
7.2.1 DMA Addressing Modes 284
7.2.2 DMA Transfer Modes 284
7.2.3 Initiating DMA Transfers 290
7.2.4 Halting Executing Instructions for DMA Transfers 290
7.2.5 Stopping DMA Transfers 291
7.2.6 DMA Channel Priorities 291
7.2.7 DMA Transfer Cycle Time 292
7.2.8 Using DMA With System Interrupts 292
7.2.9 DMA Controller Interrupts 292
7.2.10 Using the USCI_B I²C Module With the DMA Controller 294
7.2.11 Using ADC12 With the DMA Controller 294
7.2.12 Using DAC12 With the DMA Controller 294
7.3 DMA Registers 295
7.3.1 DMA Control 0 Register (DMACTL0) 296
7.3.2 DMA Control 1 Register (DMACTL1) 296
7.3.3 DMA Control 2 Register (DMACTL2) 297
7.3.4 DMA Control 3 Register (DMACTL3) 297
7.3.5 DMA Control 4 Register (DMACTL4) 298
7.3.6 DMA Channel x Control Register (DMAxCTL) 299
7.3.7 DMA Source Address Register (DMAxSA) 301
7.3.8 DMA Destination Address Register (DMAxDA) 301
7.3.9 DMA Size Address Register (DMAxSZ) 302
7.3.10 DMA Interrupt Vector Register (DMAIV) 302

8 Digital I/O 303
8.1 Digital I/O Introduction 304
8.2 Digital I/O Operation 305
8.2.1 Input Registers PxIN 305
8.2.2 Output Registers PxOUT 305
8.2.3 Direction Registers PxDIR 305
8.2.4 Pullup/Pulldown Resistor Enable Registers PxREN 305
8.2.5 Output Drive Strength Registers PxDS 306
8.2.6 Function Select Registers PxSEL 306
8.2.7 P1 and P2 Interrupts, Port Interrupts 306

8.2.8 Configuring Unused Port Pins 308
8.3 I/O Configuration and LPMx.5 Low-Power Modes 308
8.4 Digital I/O Registers 310

9 Port Mapping Controller 321
9.1 Port Mapping Controller Introduction 322
9.2 Port Mapping Controller Operation 322
9.2.1 Access 322
9.2.2 Mapping 322
9.3 Port Mapping Controller Registers 323

10 CRC Module 327
10.1 Cyclic Redundancy Check (CRC) Module Introduction 328
10.2 CRC Checksum Generation 329
10.2.1 CRC Implementation 329
10.2.2 Assembler Examples 330
10.3 CRC Module Registers 332

11 Watchdog Timer (WDT_A) 335
11.1 WDT_A Introduction 336
11.2 WDT_A Operation 338
11.2.1 Watchdog Timer Counter (WDTCNT) 338
11.2.2 Watchdog Mode 338
11.2.3 Interval Timer Mode 338
11.2.4 Watchdog Timer Interrupts 338
11.2.5 Clock Fail-Safe Feature 339
11.2.6 Operation in Low-Power Modes 339
11.2.7 Software Examples 339
11.3 WDT_A Registers 340

12 Timer_A 341
12.1 Timer_A Introduction 342
12.2 Timer_A Operation 343
12.2.1 16-Bit Timer Counter 343
12.2.2 Starting the Timer 344
12.2.3 Timer Mode Control 344
12.2.4 Capture/Compare Blocks 348
12.2.5 Output Unit 350
12.2.6 Timer_A Interrupts 354
12.3 Timer_A Registers 356

13 Timer_B 361
13.1 Timer_B Introduction 362
13.1.1 Similarities and Differences From Timer_A 362
13.2 Timer_B Operation 364
13.2.1 16-Bit Timer Counter 364
13.2.2 Starting the Timer 364
13.2.3 Timer Mode Control 364
13.2.4 Capture/Compare Blocks 368
13.2.5 Output Unit 371
13.2.6 Timer_B Interrupts 375
13.3 Timer_B Registers 377

14 Real-Time Clock (RTC_A) 383
14.1 RTC_A Introduction 384
14.2 RTC_A Operation 386
14.2.1 Counter Mode 386
14.2.2 Calendar Mode 386

14.2.3 Real-Time Clock Interrupts 388
14.2.4 Real-Time Clock Calibration 390
14.3 Real-Time Clock Registers 391
15 32-Bit Hardware Multiplier (MPY32) 403
15.1 32-Bit Hardware Multiplier (MPY32) Introduction 404
15.2 MPY32 Operation 405
15.2.1 Operand Registers 406
15.2.2 Result Registers 407
15.2.3 Software Examples 408
15.2.4 Fractional Numbers 409
15.2.5 Putting It All Together 413
15.2.6 Indirect Addressing of Result Registers 416
15.2.7 Using Interrupts 416
15.2.8 Using DMA 417
15.3 MPY32 Registers 418
16 REF 423
16.1 REF Introduction 423
16.2 Principle of Operation 425
16.2.1 Low-Power Operation 425
16.2.2 REFCTL 425
16.2.3 Reference System Requests 426
16.3 REF Registers 429
17 ADC12_A 431
17.1 ADC12_A Introduction 432
17.2 ADC12_A Operation 434
17.2.1 12-Bit ADC Core 434
17.2.2 ADC12_A Inputs and Multiplexer 434
17.2.3 Voltage Reference Generator 435
17.2.4 Auto Power Down 435
17.2.5 Sample and Conversion Timing 436
17.2.6 Conversion Memory 438
17.2.7 ADC12_A Conversion Modes 438
17.2.8 Using the Integrated Temperature Sensor 444
17.2.9 ADC12_A Grounding and Noise Considerations 445
17.2.10 ADC12_A Interrupts 446
17.3 ADC12_A Registers 448
18 Comp_B 457
18.1 Comp_B Introduction 458
18.2 Comp_B Operation 459
18.2.1 Comparator 459
18.2.2 Analog Input Switches 459
18.2.3 Port Logic 459
18.2.4 Input Short Switch 459
18.2.5 Output Filter 460
18.2.6 Reference Voltage Generator 461
18.2.7 Comp_B, Port Disable Register CBPD 462
18.2.8 Comp_B Interrupts 462
18.2.9 Comp_B Used to Measure Resistive Elements 462
18.3 Comp_B Registers 464
19 Universal Serial Communication Interface – UART Mode 469
19.1 Universal Serial Communication Interface (USCI) Overview 470
19.2 USCI Introduction – UART Mode 471

19.3 USCI Operation – UART Mode 473
19.3.1 USCI Initialization and Reset 473
19.3.2 Character Format 473
19.3.3 Asynchronous Communication Format 473
19.3.4 Automatic Baud-Rate Detection 476
19.3.5 IrDA Encoding and Decoding 477
19.3.6 Automatic Error Detection 478
19.3.7 USCI Receive Enable 479
19.3.8 USCI Transmit Enable 479
19.3.9 UART Baud-Rate Generation 480
19.3.10 Setting a Baud Rate 482
19.3.11 Transmit Bit Timing 482
19.3.12 Receive Bit Timing 483
19.3.13 Typical Baud Rates and Errors 484
19.3.14 Using the USCI Module in UART Mode With Low-Power Modes 487
19.3.15 USCI Interrupts 487
19.4 USCI Registers – UART Mode 489

20 Universal Serial Communication Interface – SPI Mode 497
20.1 Universal Serial Communication Interface (USCI) Overview 498
20.2 USCI Introduction – SPI Mode 499
20.3 USCI Operation – SPI Mode 501
20.3.1 USCI Initialization and Reset 501
20.3.2 Character Format 501
20.3.3 Master Mode 502
20.3.4 Slave Mode 503
20.3.5 SPI Enable 503
20.3.6 Serial Clock Control 504
20.3.7 Using the SPI Mode With Low-Power Modes 504
20.3.8 SPI Interrupts 505
20.4 USCI Registers – SPI Mode 506

21 Universal Serial Communication Interface – I²C Mode 511
21.1 Universal Serial Communication Interface (USCI) Overview 512
21.2 USCI Introduction – I²C Mode 513
21.3 USCI Operation – I²C Mode 514
21.3.1 USCI Initialization and Reset 515
21.3.2 I²C Serial Data 515
21.3.3 I²C Addressing Modes 517
21.3.4 I²C Module Operating Modes 518
21.3.5 I²C Clock Generation and Synchronization 529
21.3.6 Using the USCI Module in I²C Mode With Low-Power Modes 530
21.3.7 USCI Interrupts in I²C Mode 530
21.4 USCI Registers– I²C Mode 533

22 USB Module 541
22.1 USB Introduction 542
22.2 USB Operation 544
22.2.1 USB Transceiver (PHY) 544
22.2.2 USB Power System 545
22.2.3 USB Phase-Locked Loop (PLL) 547
22.2.4 USB Controller Engine 550
22.2.5 USB Vector Interrupts 553
22.2.6 Power Consumption 554
22.2.7 Suspend and Resume 554
22.3 USB Transfers 555

		22.3.1 Control Transfers	555
		22.3.2 Interrupt Transfers	558
		22.3.3 Bulk Transfers	559
	22.4	Registers	561
		22.4.1 USB Configuration Registers	561
		22.4.2 USB Control Registers	567
		22.4.3 USB Buffer Registers and Memory	574
23	**Embedded Emulation Module (EEM)**		**581**
	23.1	Embedded Emulation Module (EEM) Introduction	582
	23.2	EEM Building Blocks	584
		23.2.1 Triggers	584
		23.2.2 Trigger Sequencer	584
		23.2.3 State Storage (Internal Trace Buffer)	584
		23.2.4 Cycle Counter	584
		23.2.5 Clock Control	584
	23.3	EEM Configurations	585

List of Figures

1-1. BOR/POR/PUC Reset Circuit 21
1-2. Interrupt Priority 22
1-3. NMIs With Reentrance Protection 24
1-4. Interrupt Processing 25
1-5. Return From Interrupt 26
1-6. Operation Modes 29
1-7. Devices Descriptor Table 38
2-1. System Frequency and Supply/Core Voltages - See Device Specific Datasheet 58
2-2. PMM Block Diagram 59
2-3. High-Side and Low-Side Voltage Failure and Resulting PMM Actions 62
2-4. High-Side SVS and SVM 63
2-5. Low-Side SVS and SVM 64
2-6. PMM Action at Device Power-Up 65
2-7. Changing V_{CORE} and SVM_L and SVS_L Levels 66
3-1. UCS Block Diagram 79
3-2. Modulator Patterns 84
3-3. Module Request Clock System 86
3-4. Oscillator Fault Logic 88
3-5. Switch MCLK from DCOCLK to XT1CLK 89
4-1. MSP430X CPU Block Diagram 103
4-2. PC Storage on the Stack for Interrupts 104
4-3. Program Counter 105
4-4. PC Storage on the Stack for CALLA 105
4-5. Stack Pointer 106
4-6. Stack Usage 106
4-7. PUSHX.A Format on the Stack 106
4-8. PUSH SP, POP SP Sequence 106
4-9. SR Bits 107
4-10. Register-Byte/Byte-Register Operation 109
4-11. Register-Word Operation 109
4-12. Word-Register Operation 110
4-13. Register – Address-Word Operation 110
4-14. Address-Word – Register Operation 111
4-15. Indexed Mode in Lower 64 KB 113
4-16. Indexed Mode in Upper Memory 114
4-17. Overflow and Underflow for Indexed Mode 115
4-18. Example for Indexed Mode 116
4-19. Symbolic Mode Running in Lower 64 KB 118
4-20. Symbolic Mode Running in Upper Memory 119
4-21. Overflow and Underflow for Symbolic Mode 120
4-22. MSP430 Double-Operand Instruction Format 128
4-23. MSP430 Single-Operand Instructions 129
4-24. Format of Conditional Jump Instructions 130
4-25. Extension Word for Register Modes 133
4-26. Extension Word for Non-Register Modes 133
4-27. Example for Extended Register/Register Instruction 134
4-28. Example for Extended Immediate/Indexed Instruction 135

4-29. Extended Format I Instruction Formats 136
4-30. 20-Bit Addresses in Memory 136
4-31. Extended Format II Instruction Format 137
4-32. PUSHM/POPM Instruction Format 138
4-33. RRCM, RRAM, RRUM, and RLAM Instruction Format 138
4-34. BRA Instruction Format 138
4-35. CALLA Instruction Format 138
4-36. Decrement Overlap 163
4-37. Stack After a RET Instruction 182
4-38. Destination Operand—Arithmetic Shift Left 184
4-39. Destination Operand—Carry Left Shift 185
4-40. Rotate Right Arithmetically RRA.B and RRA.W 186
4-41. Rotate Right Through Carry RRC.B and RRC.W 187
4-42. Swap Bytes in Memory 194
4-43. Swap Bytes in a Register 194
4-44. Rotate Left Arithmetically—RLAM[.W] and RLAM.A 221
4-45. Destination Operand-Arithmetic Shift Left 222
4-46. Destination Operand-Carry Left Shift 223
4-47. Rotate Right Arithmetically RRAM[.W] and RRAM.A 224
4-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode 226
4-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode 226
4-50. Rotate Right Through Carry RRCM[.W] and RRCM.A 227
4-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode 229
4-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode 229
4-53. Rotate Right Unsigned RRUM[.W] and RRUM.A 230
4-54. Rotate Right Unsigned RRUX(.B,.A) – Register Mode 231
4-55. Swap Bytes SWPBX.A Register Mode 235
4-56. Swap Bytes SWPBX.A In Memory 235
4-57. Swap Bytes SWPBX[.W] Register Mode 236
4-58. Swap Bytes SWPBX[.W] In Memory 236
4-59. Sign Extend SXTX.A 237
4-60. Sign Extend SXTX[.W] 237
5-1. Flash Memory Module Block Diagram 256
5-2. 256-KB Flash Memory Segments Example 257
5-3. Erase Cycle Timing 260
5-4. Erase Cycle From Flash 261
5-5. Erase Cycle From RAM 262
5-6. Byte/Word/Long-Word Write Timing 263
5-7. Initiating a Byte/Word Write From Flash 264
5-8. Initiating a Byte/Word Write From RAM 265
5-9. Initiating Long-Word Write From Flash 266
5-10. Initiating Long-Word Write from RAM 267
5-11. Block-Write Cycle Timing 268
5-12. Block Write Flow 269
5-13. User-Developed Programming Solution 272
7-1. DMA Controller Block Diagram 283
7-2. DMA Addressing Modes 284
7-3. DMA Single Transfer State Diagram 286
7-4. DMA Block Transfer State Diagram 287

7-5. DMA Burst-Block Transfer State Diagram 289
10-1. LFSR Implementation of CRC-CCITT Standard, Bit 0 is the MSB of the Result 328
10-2. Implementation of CRC-CCITT using the CRCDI and CRCINIRES registers 330
11-1. Watchdog Timer Block Diagram 337
12-1. Timer_A Block Diagram 343
12-2. Up Mode 345
12-3. Up Mode Flag Setting 345
12-4. Continuous Mode 345
12-5. Continuous Mode Flag Setting 346
12-6. Continuous Mode Time Intervals 346
12-7. Up/Down Mode 347
12-8. Up/Down Mode Flag Setting 347
12-9. Output Unit in Up/Down Mode 348
12-10. Capture Signal (SCS = 1) 348
12-11. Capture Cycle 349
12-12. Output Example – Timer in Up Mode 351
12-13. Output Example – Timer in Continuous Mode 352
12-14. Output Example – Timer in Up/Down Mode 353
12-15. Capture/Compare TAxCCR0 Interrupt Flag 354
13-1. Timer_B Block Diagram 363
13-2. Up Mode 365
13-3. Up Mode Flag Setting 365
13-4. Continuous Mode 366
13-5. Continuous Mode Flag Setting 366
13-6. Continuous Mode Time Intervals 366
13-7. Up/Down Mode 367
13-8. Up/Down Mode Flag Setting 367
13-9. Output Unit in Up/Down Mode 368
13-10. Capture Signal (SCS = 1) 369
13-11. Capture Cycle 369
13-12. Output Example – Timer in Up Mode 372
13-13. Output Example – Timer in Continuous Mode 373
13-14. Output Example – Timer in Up/Down Mode 374
13-15. Capture/Compare TBxCCR0 Interrupt Flag 375
14-1. RTC_A 385
15-1. MPY32 Block Diagram 404
15-2. Q15 Format Representation 409
15-3. Q14 Format Representation 410
15-4. Saturation Flow Chart 412
15-5. Multiplication Flow Chart 414
16-1. REF Block Diagram 424
17-1. ADC12_A Block Diagram 433
17-2. Analog Multiplexer 434
17-3. Extended Sample Mode 436
17-4. Pulse Sample Mode 437
17-5. Analog Input Equivalent Circuit 437
17-6. Single-Channel Single-Conversion Mode 439
17-7. Sequence-of-Channels Mode 440
17-8. Repeat-Single-Channel Mode 441

17-9. Repeat-Sequence-of-Channels Mode 442
17-10. Typical Temperature Sensor Transfer Function 444
17-11. ADC12_A Grounding and Noise Considerations 445
18-1. Comp_B Block Diagram 458
18-2. Comp_B Sample-And-Hold 460
18-3. RC-Filter Response at the Output of the Comparator 461
18-4. Reference Generator Block Diagram 461
18-5. Transfer Characteristic and Power Dissipation in a CMOS Inverter/Buffer 462
18-6. Temperature Measurement System 462
18-7. Timing for Temperature Measurement Systems 463
19-1. USCI_Ax Block Diagram – UART Mode (UCSYNC = 0) 472
19-2. Character Format 473
19-3. Idle-Line Format 474
19-4. Address-Bit Multiprocessor Format 475
19-5. Auto Baud-Rate Detection – Break/Synch Sequence 476
19-6. Auto Baud-Rate Detection – Synch Field 476
19-7. UART vs IrDA Data Format 477
19-8. Glitch Suppression, USCI Receive Not Started 479
19-9. Glitch Suppression, USCI Activated 479
19-10. BITCLK Baud-Rate Timing With UCOS16 = 0 480
19-11. Receive Error 483
20-1. USCI Block Diagram – SPI Mode 500
20-2. USCI Master and External Slave 502
20-3. USCI Slave and External Master 503
20-4. USCI SPI Timing With UCMSB = 1 504
21-1. USCI Block Diagram – I^2C Mode 514
21-2. I^2C Bus Connection Diagram 515
21-3. I^2C Module Data Transfer 516
21-4. Bit Transfer on I^2C Bus 516
21-5. I^2C Module 7-Bit Addressing Format 517
21-6. I^2C Module 10-Bit Addressing Format 517
21-7. I^2C Module Addressing Format With Repeated START Condition 517
21-8. I^2C Time-Line Legend 518
21-9. I^2C Slave Transmitter Mode 519
21-10. I^2C Slave Receiver Mode 521
21-11. I^2C Slave 10-Bit Addressing Mode 522
21-12. I^2C Master Transmitter Mode 524
21-13. I^2C Master Receiver Mode 526
21-14. I^2C Master 10-Bit Addressing Mode 527
21-15. Arbitration Procedure Between Two Master Transmitters 528
21-16. Synchronization of Two I^2C Clock Generators During Arbitration 529
22-1. USB Block Diagram 543
22-2. USB Power System 545
22-3. USB Power Up/Down Profile 546
22-4. Powering Entire MSP430 From VBUS 546
22-5. USB-PLL Analog Block Diagram 547
22-6. Data Buffers and Descriptors 550
22-7. USB Timer and Time Stamp Generation 552
23-1. Large Implementation of EEM 583

List of Tables

1-1. Interrupt Sources, Flags, and Vectors 26
1-2. Connection of Unused Pins 33
1-3. Tag Values 39
1-4. Peripheral Discovery Descriptor 39
1-5. Values for Memory Entry 40
1-6. Values for Peripheral Entry 41
1-7. Peripheral IDs 41
1-8. Sample Peripheral Discovery Descriptor 42
1-9. SFR Base Address 45
1-10. Special Function Registers 45
1-11. SYS Base Address 49
1-12. SYS Configuration Registers 49
2-1. SVS/SVM Thresholds 60
2-2. Recommended SVS_L Settings 61
2-3. Recommended SVS_H Settings 61
2-4. Available SVS_H, SVS_M Settings Versus VCORE Settings 61
2-5. SVS_L Performance Control Modes 68
2-6. SVM_L Performance Control Modes 68
2-7. SVS_H Performance Control Modes 68
2-8. SVM_H Performance Control Modes 69
2-9. PMM Registers 70
3-1. Clock Request System and Power Modes 86
3-2. Unified Clock System Registers 91
4-1. SR Bit Description 107
4-2. Values of Constant Generators CG1, CG2 108
4-3. Source/Destination Addressing 111
4-4. MSP430 Double-Operand Instructions 129
4-5. MSP430 Single-Operand Instructions 129
4-6. Conditional Jump Instructions 130
4-7. Emulated Instructions 130
4-8. Interrupt, Return, and Reset Cycles and Length 131
4-9. MSP430 Format II Instruction Cycles and Length 131
4-10. MSP430 Format I Instructions Cycles and Length 132
4-11. Description of the Extension Word Bits for Register Mode 133
4-12. Description of Extension Word Bits for Non-Register Modes 134
4-13. Extended Double-Operand Instructions 135
4-14. Extended Single-Operand Instructions 137
4-15. Extended Emulated Instructions 139
4-16. Address Instructions, Operate on 20-Bit Register Data 140
4-17. MSP430X Format II Instruction Cycles and Length 141
4-18. MSP430X Format I Instruction Cycles and Length 142
4-19. Address Instruction Cycles and Length 143
4-20. Instruction Map of MSP430X 144
5-1. Erase Modes 259
5-2. Write Modes 263
5-3. Flash Access While Flash is Busy (BUSY = 1) 270
5-4. Flash Controller Registers 273

6-1. RAMCTL Module Register 279
7-1. DMA Transfer Modes 285
7-2. DMA Trigger Operation 291
7-3. Maximum Single-Transfer DMA Cycle Time 292
7-4. DMA Registers 295
8-1. I/O Configuration 305
8-2. Digital I/O Registers 310
9-1. Examples for Port Mapping Mnemonics and Functions 323
9-2. Port Mapping Control Registers 324
9-3. Port Mapping Registers for Port Px – Byte Access 324
9-4. Port Mapping Registers for Port Px – Word Access 324
10-1. CRC Module Registers 332
11-1. Watchdog Timer Registers 340
12-1. Timer Modes 344
12-2. Output Modes 350
12-3. Timer_A Registers 356
13-1. Timer Modes 365
13-2. TBxCLn Load Events 370
13-3. Compare Latch Operating Modes 370
13-4. Output Modes 371
13-5. Timer_B Registers 377
14-1. Real-Time Clock Registers 391
15-1. Result Availability (MPYFRAC = 0, MPYSAT = 0) 405
15-2. OP1 Registers 407
15-3. OP2 Registers 407
15-4. SUMEXT and MPYC Contents 408
15-5. Result Availability in Fractional Mode (MPYFRAC = 1, MPYSAT = 0) 410
15-6. Result Availability in Saturation Mode (MPYSAT = 1) 411
15-7. MPY32 Registers 418
15-8. Alternative Registers 420
16-1. REF Control of Reference System (REFMSTR = 1) (Default) 426
16-2. Table 2. ADC Control of Reference System (REFMSTR = 0) 426
16-3. REF Registers 429
17-1. ADC12_A Conversion Result Formats 438
17-2. Conversion Mode Summary 438
17-3. ADC12_A Registers 448
18-1. Comp_B Registers 464
19-1. Receive Error Conditions 478
19-2. BITCLK Modulation Pattern 480
19-3. BITCLK16 Modulation Pattern 481
19-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0 484
19-5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1 486
19-6. USCI_Ax Registers 489
20-1. UCxSTE Operation 501
20-2. USCI_Ax Registers 506
20-3. USCI_Bx Registers 506
21-1. I²C State Change Interrupt Flags 531
21-2. USCI_Bx Registers 533
22-1. USB-PLL Pre-Scale Divider 548

22-2. Register Settings to Generate 48 MHz Using Common Crystals 548
22-3. USB Buffer Memory Map 551
22-4. USB Interrupt Vector Generation 553
22-5. USB Configuration Registers 561
22-6. USB Control Registers 567
22-7. USB Buffer Memory 574
22-8. USB Buffer Descriptor Registers 574
23-1. 5xx EEM Configurations 585

Read This First

About This Manual

This manual describes the modules and peripherals of the MSP430x5xx family of devices. Each description presents the module or peripheral in a general sense. Not all features and functions of all modules or peripherals may be present on all devices. In addition, modules or peripherals may differ in their exact implementation between device families, or may not be fully implemented on an individual device or device family.

Pin functions, internal signal connections, and operational parameters differ from device to device. The user should consult the device-specific data sheet for these details.

Related Documentation From Texas Instruments

For related documentation see the web site http://www.ti.com/msp430.

FCC Warning

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct this interference.

Notational Conventions

Program examples, are shown in a special typeface.

Glossary

ACLK	Auxiliary Clock
ADC	Analog-to-Digital Converter
BOR	Brown-Out Reset; see System Resets, Interrupts, and Operating Modes
BSL	Bootstrap Loader; see www.ti.com/msp430 for application reports
CPU	Central Processing Unit See RISC 16-Bit CPU
DAC	Digital-to-Analog Converter
DCO	Digitally Controlled Oscillator; see FLL+ Module
dst	Destination; see RISC 16-Bit CPU
FLL	Frequency Locked Loop; see FLL+ Module
GIE Modes	General Interrupt Enable; see System Resets Interrupts and Operating
INT(N/2)	Integer portion of N/2
I/O	Input/Output; see Digital I/O
ISR	Interrupt Service Routine
LSB	Least-Significant Bit
LSD	Least-Significant Digit
LPM	Low-Power Mode; see System Resets Interrupts and Operating Modes; also named PM for Power Mode

MAB	Memory Address Bus
MCLK	Master Clock
MDB	Memory Data Bus
MSB	Most-Significant Bit
MSD	Most-Significant Digit
NMI	(Non)-Maskable Interrupt; see System Resets Interrupts and Operating Modes; also split to UNMI and SNMI
PC	Program Counter; see RISC 16-Bit CPU
PM	Power Mode See; system Resets Interrupts and Operating Modes
POR	Power-On Reset; see System Resets Interrupts and Operating Modes
PUC	Power-Up Clear; see System Resets Interrupts and Operating Modes
RAM	Random Access Memory
SCG	System Clock Generator; see System Resets Interrupts and Operating Modes
SFR	Special Function Register; see System Resets, Interrupts, and Operating Modes
SMCLK	Sub-System Master Clock
SNMI	System NMI; see System Resets, Interrupts, and Operating Modes
SP	Stack Pointer; see RISC 16-Bit CPU
SR	Status Register; see RISC 16-Bit CPU
src	Source; see RISC 16-Bit CPU
TOS	Top of stack; see RISC 16-Bit CPU
UNMI	User NMI; see System Resets, Interrupts, and Operating Modes
WDT	Watchdog Timer; see Watchdog Timer
z16	16 bit address space

Register Bit Conventions

Each register is shown with a key indicating the accessibility of the each individual bit, and the initial condition:

Register Bit Accessibility and Initial Condition

Key	Bit Accessibility
rw	Read/write
r	Read only
r0	Read as 0
r1	Read as 1
w	Write only
w0	Write as 0
w1	Write as 1
(w)	No register bit implemented; writing a 1 results in a pulse. The register bit is always read as 0.
h0	Cleared by hardware
h1	Set by hardware
-0,-1	Condition after PUC
-(0),-(1)	Condition after POR
-[0],-[1]	Condition after BOR
-{0},-{1}	Condition after Brownout

Chapter 1

SLAU208F–June 2008–Revised March 2010

System Resets, Interrupts, and Operating Modes, System Control Module (SYS)

The system control module (SYS) is available on all devices. The following list shows the basic feature set of SYS.

- Brownout reset/power on reset (BOR/POR) handling
- Power up clear (PUC) handling
- (Non)maskable interrupt (SNMI/UNMI) event source selection and management
- Address decoding
- Providing an user data-exchange mechanism via the JTAG mailbox (JMB)
- Bootstrap loader (BSL) entry mechanism
- Configuration management (device descriptors)
- Providing interrupt vector generators for reset and NMIs

Topic **Page**

1.1	System Control Module (SYS) Introduction	20
1.2	System Reset and Initialization	20
1.3	Interrupts	22
1.4	Operating Modes	28
1.5	Principles for Low-Power Applications	33
1.6	Connection of Unused Pins	33
1.7	Reset pin ($\overline{RST}$/NMI) Configuration	33
1.8	Configuring JTAG pins	33
1.9	Boot Code	34
1.10	Bootstrap Loader (BSL)	34
1.11	Memory Map – Uses and Abilities	35
1.12	JTAG Mailbox (JMB) System	36
1.13	Device Descriptor Table	37
1.14	Special Function Registers (SFRs)	45
1.15	SYS Configuration Registers	49

1.1 System Control Module (SYS) Introduction

SYS is responsible for the interaction between various modules throughout the system. The functions that SYS provides for are not inherent to the modules themselves. Address decoding, bus arbitration, interrupt event consolidation, and reset generation are some examples of the many functions that SYS provides.

1.2 System Reset and Initialization

The system reset circuitry is shown in Figure 1-1 and sources a brownout reset (BOR), a power on reset (POR), and a power up clear (PUC). Different events trigger these reset signals and different initial conditions exist depending on which signal was generated.

A BOR is a device reset. A BOR is only generated by the following events:

- Powering up the device
- A low signal on $\overline{\text{RST}}$/NMI pin when configured in the reset mode
- A wakeup event from LPMx.5 (LPM3.5 or LPM4.5) modes
- A software BOR event

A POR is always generated when a BOR is generated, but a BOR is not generated by a POR. The following events trigger a POR:

- A BOR signal
- A SVS_H and/or SVS_M low condition when enabled (see the *PMM* chapter for details)
- A SVS_L and/or SVS_L low condition when enabled (see the *PMM* chapter for details)
- A software POR event

A PUC is always generated when a POR is generated, but a POR is not generated by a PUC. The following events trigger a PUC:

- A POR signal
- Watchdog timer expiration when watchdog mode only (see the *WDT_A* chapter for details)
- Watchdog timer password violation (see the *WDT_A* chapter for details)
- A Flash memory password violation (see the *Flash Memory Controller* chapter for details)
- Power Management Module password violation (see the *PMM* chapter for details)
- Fetch from peripheral area

NOTE: The number and type of resets available may vary from device to device. See the device-specific data sheet for all reset sources available.

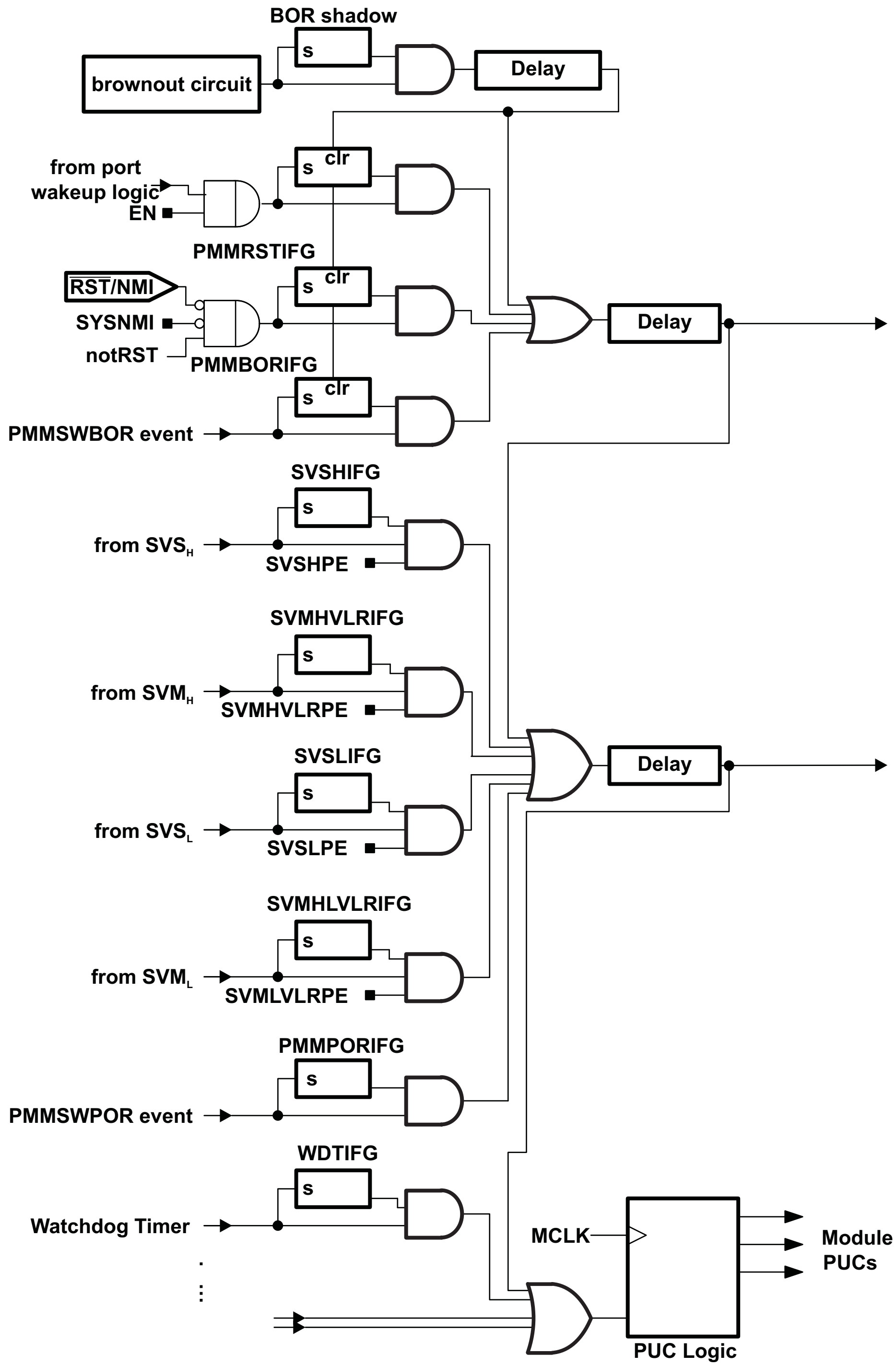


Figure 1-1. BOR/POR/PUC Reset Circuit

1.2.1 Device Initial Conditions After System Reset

After a BOR, the initial device conditions are:

- The RST/NMI pin is configured in the reset mode. See Section 1.7 on configuring the RST/NMI pin.
- I/O pins are switched to input mode as described in the *Digital I/O* chapter.
- Other peripheral modules and registers are initialized as described in their respective chapters in this manual.
- Status register (SR) is reset.
- The watchdog timer powers up active in watchdog mode.
- Program counter (PC) is loaded with the boot code address and boot code execution begins at that address. See Section 1.9 for more information regarding the boot code. Upon completion of the boot code, the PC is loaded with the address contained at the SYSRSTIV reset location (0FFFEh).

After a system reset, user software must initialize the device for the application requirements. The following must occur:

- Initialize the stack pointer (SP), typically to the top of RAM.
- Initialize the watchdog to the requirements of the application.
- Configure peripheral modules to the requirements of the application.

1.3 Interrupts

The interrupt priorities are fixed and defined by the arrangement of the modules in the connection chain as shown in Figure 1-2. Interrupt priorities determine what interrupt is taken when more than one interrupt is pending simultaneously.

There are three types of interrupts:

- System reset
- (Non)maskable
- Maskable

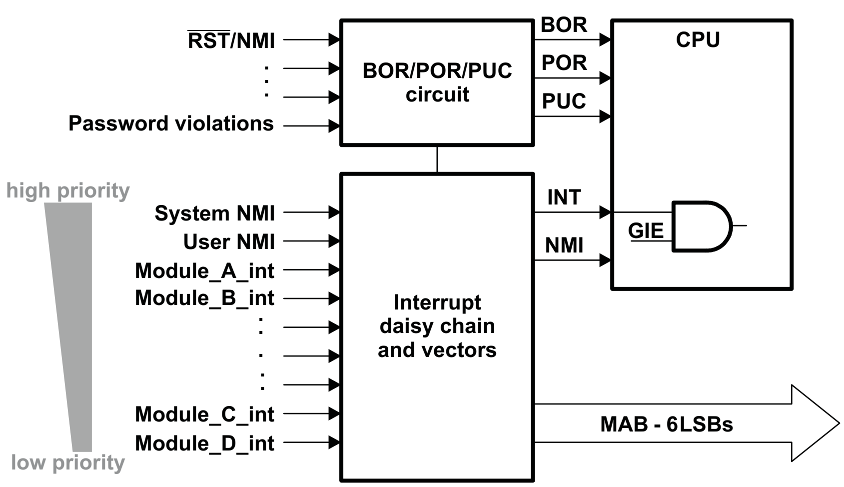


Figure 1-2. Interrupt Priority

NOTE: The types of Interrupt sources available and their respective priorities can change from device to device. Please see the device specific data sheet for all interrupt sources and their priorities.

1.3.1 (Non)Maskable Interrupts (NMIs)

In general, NMIs are not masked by the general interrupt enable (GIE) bit. The family supports two levels of NMIs — system NMI (SNMI) and user NMI (UNMI). The NMI sources are enabled by individual interrupt enable bits. When an NMI interrupt is accepted, other NMIs of that level are automatically disabled to prevent nesting of consecutive NMIs of the same level. Program execution begins at the address stored in the NMI vector as shown in Table 1-1. To allow software backward compatibility to users of earlier MSP430 families, the software may, but does not need to, reenable NMI sources. The block diagram for NMI sources is shown in Figure 1-3.

A UNMI interrupt can be generated by following sources:

- An edge on the $\overline{RST}$/NMI pin when configured in NMI mode
- An oscillator fault occurs
- An access violation to the flash memory

A SNMI interrupt can be generated by following sources:

- Power Management Module (PMM) SVM_L/SVM_H supply voltage fault
- PMM high/low side delay expiration
- Vacant memory access
- JTAG mailbox (JMB) event

NOTE: The number and types of NMI sources may vary from device to device. See the device-specific data sheet for all NMI sources available.

1.3.2 SNMI Timing

Consecutive SNMIs that occur at a higher rate than they can be handled (interrupt storm) allow the main program to execute one instruction after the SNMI handler is finished with a RETI instruction, before the SNMI handler is executed again. Consecutive SNMIs are not interrupted by UNMIs in this case. This avoids a blocking behavior on high SNMI rates.

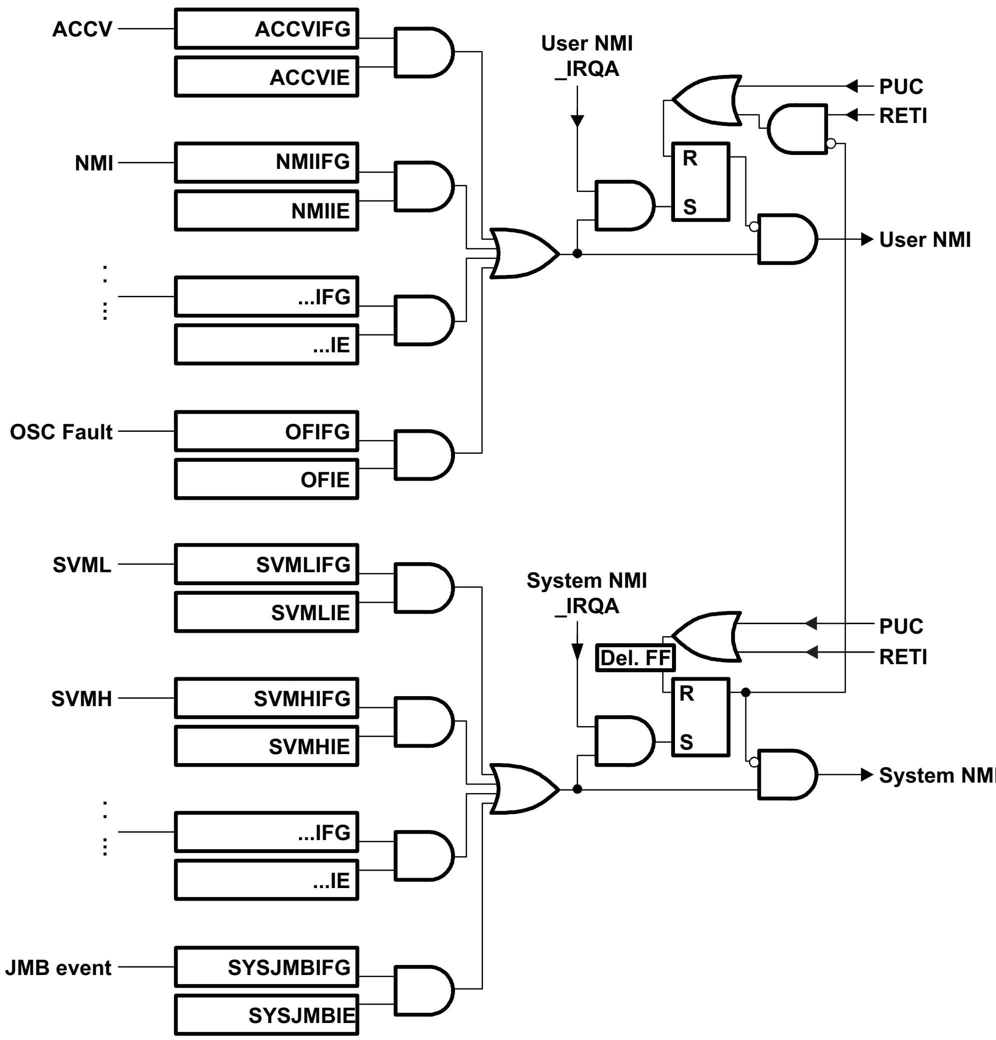


Figure 1-3. NMIs With Reentrance Protection

1.3.3 Maskable Interrupts

Maskable interrupts are caused by peripherals with interrupt capability. Each maskable interrupt source can be disabled individually by an interrupt enable bit, or all maskable interrupts can be disabled by the general interrupt enable (GIE) bit in the status register (SR).

Each individual peripheral interrupt is discussed in its respective module chapter in this manual.

1.3.4 Interrupt Processing

When an interrupt is requested from a peripheral and the peripheral interrupt enable bit and GIE bit are set, the interrupt service routine is requested. Only the individual enable bit must be set for (non)-maskable interrupts (NMI) to be requested.

1.3.4.1 Interrupt Acceptance

The interrupt latency is six cycles, starting with the acceptance of an interrupt request, and lasting until the start of execution of the first instruction of the interrupt service routine, as shown in Figure 1-4. The interrupt logic executes the following:

1. Any currently executing instruction is completed.
2. The PC, which points to the next instruction, is pushed onto the stack.
3. The SR is pushed onto the stack.
4. The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
5. The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.
6. The SR is cleared. This terminates any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.
7. The content of the interrupt vector is loaded into the PC; the program continues with the interrupt service routine at that address.

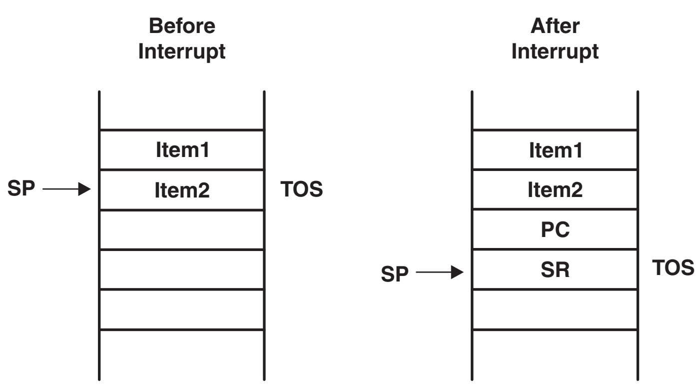


Figure 1-4. Interrupt Processing

1.3.4.2 Return From Interrupt

The interrupt handling routine terminates with the instruction:

```
RETI //return from an interrupt service routine
```

The return from the interrupt takes five cycles to execute the following actions and is illustrated in Figure 1-5.

1. The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings used during the interrupt service routine.
2. The PC pops from the stack and begins execution at the point where it was interrupted.

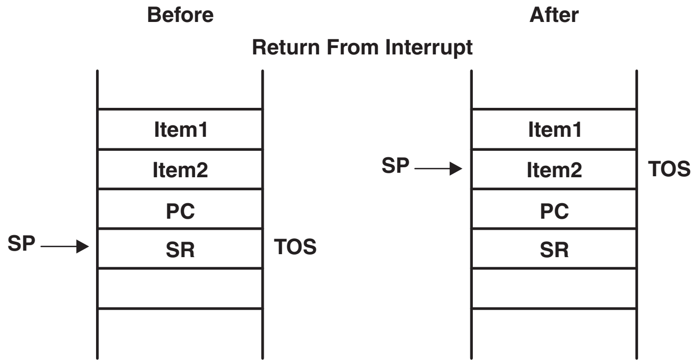


Figure 1-5. Return From Interrupt

1.3.5 Interrupt Nesting

Interrupt nesting is enabled if the GIE bit is set inside an interrupt service routine. When interrupt nesting is enabled, any interrupt occurring during an interrupt service routine interrupts the routine, regardless of the interrupt priorities.

1.3.6 Interrupt Vectors

The interrupt vectors are located in the address range 0FFFFh to 0FF80h, for a maximum of 64 interrupt sources. A vector is programmed by the user and points to the start location of the corresponding interrupt service routine. Table 1-1 is an example of the interrupt vectors available. See the device-specific data sheet for the complete interrupt vector list.

Table 1-1. Interrupt Sources, Flags, and Vectors

Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
Reset: power up, external reset watchdog, flash password	... WDTIFG KEYV	... Reset	... 0FFFEh	... Highest
System NMI: PMM		(Non)maskable	0FFFCh	...
User NMI: NMI, oscillator fault, flash memory access violation	... NMIIFG OFIFG ACCVIFG	... (Non)maskable (Non)maskable (Non)maskable	... 0FFFAh	... ...
Device specific			0FFF8h	...
...			...	...
Watchdog timer	WDTIFG	Maskable	...	...
...			...	...
Device specific			...	...
Reserved		Maskable	...	Lowest

Some interrupt enable bits, and interrupt flags, as well as, control bits for the $\overline{RST}$/NMI pin are located in the special function registers (SFR). The SFR are located in the peripheral address range and are byte and word accessible. See the device-specific data sheet for the SFR configuration.

1.3.6.1 Alternate Interrupt Vectors

It is possible to use the RAM as an alternate location for the interrupt vector locations. Setting the SYSRIVECT bit in SYSCTL causes the interrupt vectors to be remapped to the top of RAM. Once set, any interrupt will vector to the alternate locations now residing in RAM. Because SYSRIVECT is automatically cleared on a BOR, it is critical that the reset vector at location 0FFFEh still be available and handled properly in firmware.

1.3.7 SYS Interrupt Vector Generators

SYS collects all system NMI (SNMI) sources, user NMI (UNMI) sources, and BOR/POR/PUC (reset) sources of all the other modules. They are combined into three interrupt vectors. The interrupt vector registers SYSRSTIV, SYSSNIV, SYSUNIV are used to determine which flags requested an interrupt or a reset. The interrupt with the highest priority of a group, when enabled, generates a number in the corresponding SYSRSTIV, SYSSNIV, SYSUNIV register. This number can be directly added to the program counter, causing a branch to the appropriate portion of the interrupt service routine. Disabled interrupts do not affect the SYSRSTIV, SYSSNIV, SYSUNIV values. Reading SYSRSTIV, SYSSNIV, SYSUNIV register automatically resets the highest pending interrupt flag of that register. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. Writing to the SYSRSTIV, SYSSNIV, SYSUNIV register automatically resets all pending interrupt flags of the group.

1.3.7.1 SYSSNIV Software Example

The following software example shows the recommended use of SYSSNIV. The SYSSNIV value is added to the PC to automatically jump to the appropriate routine. For SYSRSTIV and SYSUNIV, a similar software approach can be used. The following is an example for a generic device. Vectors can change in priority for a given device. The device specific data sheet should be referenced for the vector locations. All vectors should be coded symbolically to allow for easy portability of code.

```
SNI_ISR:    ADD      &SYSSNIV,PC ; Add offset to jump table
         RETI              ; Vector 0: No interrupt
         JMP      SVML_ISR       ; Vector 2: SVMLIFG
         JMP      SVMH_ISR       ; Vector 4: SVMHIFG
         JMP      DLYL_ISR       ; Vector 6: SVSMLDLYIFG
         JMP      DLYH_ISR       ; Vector 8: SVSMHDLYIFG
         JMP      VMA_ISR        ; Vector 10: VMAIFG
         JMP      JMBI_ISR       ; Vector 12: JMBINIFG
JMBO_ISR:                           ; Vector 14: JMBOUTIFG
        ...                         ; Task_E starts here
        RETI                        ; Return
SVML_ISR:                           ; Vector 2
        ...                         ; Task_2 starts here
        RETI                        ; Return
SVMH_ISR:                           ; Vector 4
        ...                   ; Task_4 starts here
        RETI                        ; Return
DLYL_ISR:                           ; Vector 6
        ...                         ; Task_6 starts here
        RETI                        ; Return
DLYH_ISR:                           ; Vector 8
        ...                         ; Task_8 starts here
        RETI                        ; Return
VMA_ISR:                               ; Vector A
       ...                          ; Task_A starts here
          RETI                      ; Return
JMBI_ISR:                           ; Vector C
```

```
        ...                       ; Task_C starts here
        RETI                      ; Return
```

1.3.7.2 SYSBERRIV Bus Error Interrupt Vector Generator

Some devices, for example those that contain the USB module, include an additional system interrupt vector generator, SYSBERRIV. In general, any type of system related bus error or timeout error is associated with a user NMI event. Upon this event, the SYSUNIV will contain an offset value corresponding to a bus error event (BUSIFG). This offset can be added to the PC to automatically jump to the appropriate NMI routine. Similarly, SYSBERRIV will also contain an offset value corresponding to which specific event caused the bus error event. The offset value in SYSBERRIV can be added inside the NMI routine to automatically jump to the appropriate routine. In this way, the SYSBERRIV can be thought of as an extension to the user NMI vectors.

1.4 Operating Modes

The MSP430 family is designed for ultralow-power applications and uses different operating modes shown in Figure 1-6.

The operating modes take into account three different needs:

- Ultralow power
- Speed and data throughput
- Minimization of individual peripheral current consumption

The low-power modes LPM0 through LPM4 are configured with the CPUOFF, OSCOFF, SCG0, and SCG1 bits in the SR. The advantage of including the CPUOFF, OSCOFF, SCG0, and SCG1 mode-control bits in the SR is that the present operating mode is saved onto the stack during an interrupt service routine. Program flow returns to the previous operating mode if the saved SR value is not altered during the interrupt service routine. Program flow can be returned to a different operating mode by manipulating the saved SR value on the stack inside of the interrupt service routine. When setting any of the mode-control bits, the selected operating mode takes effect immediately. Peripherals operating with any disabled clock are disabled until the clock becomes active. Peripherals may also be disabled with their individual control register settings. All I/O port pins and RAM/registers are unchanged. Wakeup from LPM0 through LPM4 is possible through all enabled interrupts.

When LPMx.5 (LPM3.5 or LPM4.5) is entered, the voltage regulator of the Power Management Module (PMM) is disabled. All RAM and register contents are lost, as well as I/O configuration. Wakeup from LPM4.5 is possible via a power sequence, a $\overline{\text{RST}}$ event, or from specific I/O. Wakeup from LPM3.5 is possible via a power sequence, a $\overline{\text{RST}}$ event, RTC event, or from specific I/O.

NOTE: LPM3.5 and LPM4.5 low power modes are not available on all devices. Please refer to the device specific data sheet to see which LPMx.5 power modes are available.

NOTE: The TEST/SWBTCK pin is used for interfacing to the development tools via Spy-Bi-Wire and JTAG. When the TEST/SBWTCK pin is high, wakeup times from LPM2, LPM3, and LPM4 may be different compared to when TEST/SBWTCK is low. Pay careful attention to the real-time behavior when exiting from LPM2, LPM3, and LPM4 with the device connected to a development tool (e.g. - MSP-FETU430IF). Please see the *Power Management Module* chapter for futher details.

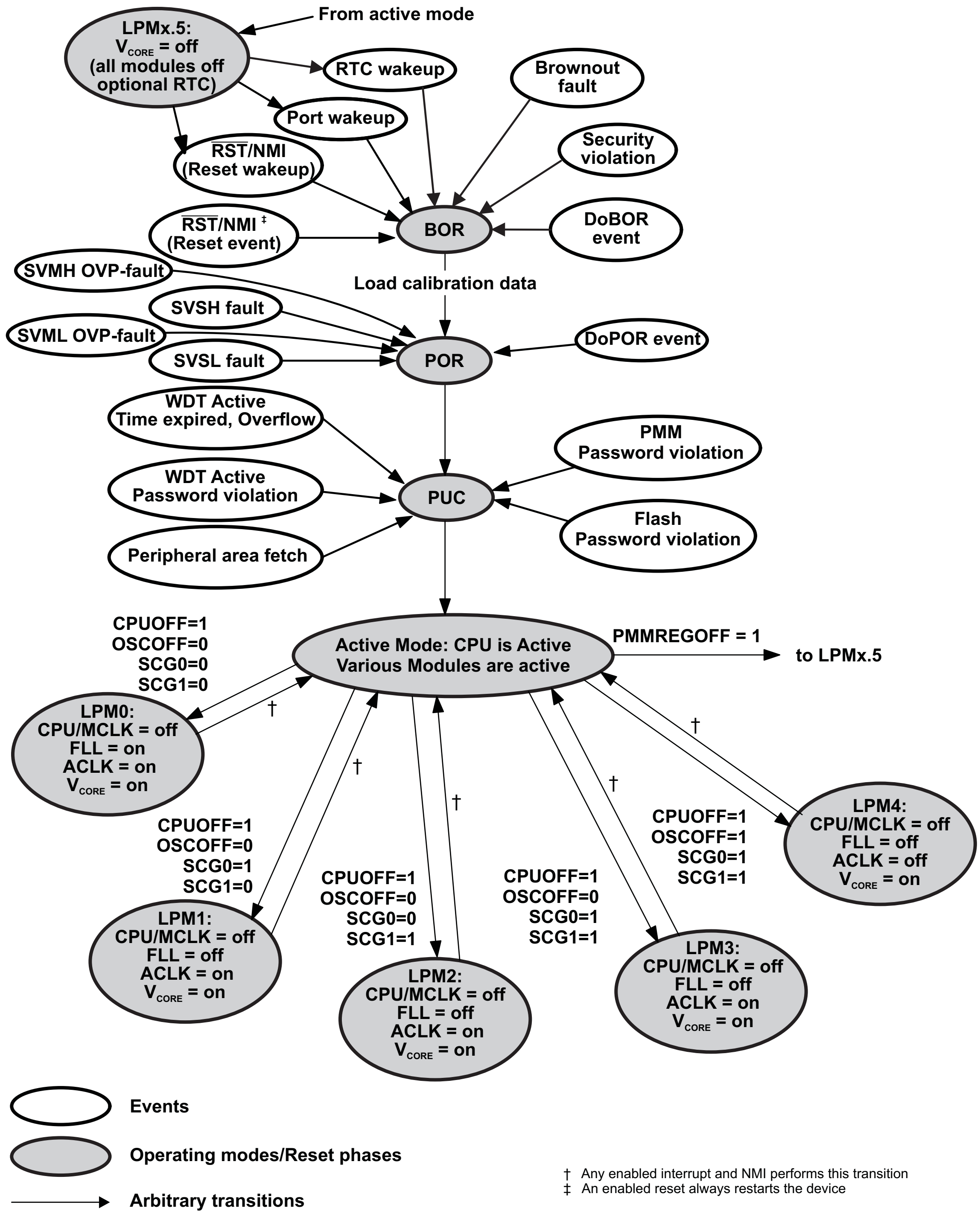


Figure 1-6. Operation Modes

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status [(1)]
0	0	0	0	Active	CPU, MCLK are active.
					ACLK is active. SMCLK optionally active (SMCLKOFF = 0).
					DCO is enabled if sources ACLK, MCLK, or SMCLK (SMCLKOFF = 0).
					DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0).
					FLL is enabled if DCO is enabled.
0	0	0	1	LPM0	CPU, MCLK are disabled.
					ACLK is active. SMCLK optionally active (SMCLKOFF = 0).
					DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0).
					DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0).
					FLL is enabled if DCO is enabled.
0	1	0	1	LPM1	CPU, MCLK are disabled.
					ACLK is active. SMCLK optionally active (SMCLKOFF = 0).
					DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0).
					DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0).
					FLL is disabled.
1	0	0	1	LPM2	CPU, MCLK are disabled.
					ACLK is active. SMCLK is disabled.
					DCO is enabled if sources ACLK.
					FLL is disabled.
1	1	0	1	LPM3	CPU, MCLK are disabled.
					ACLK is active. SMCLK is disabled.
					DCO is enabled if sources ACLK.
					FLL is disabled.
1	1	1	1	LPM4	CPU and all clocks are disabled.
1	1	1	1	LPM3.5[(2)]	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, RTC operation is possible when configured properly. Please refer to the *RTC* module for further details.
1	1	1	1	LPM4.5[(2)]	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, all clock sources are disabled i.e. no RTC operation is possible.

(1) The low power modes and hence the system clocks can be affected by the clock request system. Please refer to the *Unified Clock System* chapter for details.

(2) LPM3.5 and LPM4.5 modes are not available on all devices. Please refer to the device specific data sheet for availability.

1.4.1 Entering and Exiting Low-Power Modes LPM0 Through LPM4

An enabled interrupt event wakes the device from low-power operating modes LPM0 through LPM4. The program flow for exiting LPM0 through LPM4 is:

- Enter interrupt service routine
  - The PC and SR are stored on the stack.
  - The CPUOFF, SCG1, and OSCOFF bits are automatically reset.
- Options for returning from the interrupt service routine
  - The original SR is popped from the stack, restoring the previous operating mode.
  - The SR bits stored on the stack can be modified within the interrupt service routine returning to a different operating mode when the RETI instruction is executed.

```
; Enter LPM0 Example
   BIS   #GIE+CPUOFF,SR                    ; Enter LPM0
;  ...                                     ; Program stops here
;
; Exit LPM0 Interrupt Service Routine
   BIC   #CPUOFF,0(SP)                     ; Exit LPM0 on RETI
   RETI
```

```
; Enter LPM3 Example
   BIS   #GIE+CPUOFF+SCG1+SCG0,SR          ; Enter LPM3
;  ...                                     ; Program stops here
;
; Exit LPM3 Interrupt Service Routine
   BIC   #CPUOFF+SCG1+SCG0,0(SP)           ; Exit LPM3 on RETI
   RETI
```

```
; Enter LPM4 Example
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR   ; Enter LPM4
;  ...                                     ; Program stops here
;
; Exit LPM4 Interrupt Service Routine
   BIC   #CPUOFF+OSCOFF+SCG1+SCG0,0(SP)    ; Exit LPM4 on RETI
   RETI
```

1.4.2 Entering and Exiting Low-Power Modes LPMx.5

LPMx.5 entry and exit is handled differently than the other low power modes. LPMx.5, when used properly, gives the lowest power consumption available on a device. To achieve this, entry to LPMx.5 disables the LDO of the PMM module, removing the supply voltage from the core of the device. Since the supply voltage is removed from the core, all register contents, as well as, SRAM contents are lost. Exit from LPMx.5 causes a BOR event, which forces a complete reset of the system. Therefore, it is the application's responsibility to properly reconfigure the device upon exit from LPMx.5.

The wakeup time from LPMx.5 is significantly longer than the wakeup time from the other power modes (please see the device specific data sheet). This is primarily due to the facts that after exit from LPMx.5, time is required for the core voltage supply to be regenerated, as well as, boot code execution to complete before the application code can begin. Therefore, the usage of LPMx.5 is restricted to very low duty cycle events.

There are two LPMx.5 power modes, LPM3.5 and LPM4.5. Not all of these are available on all devices. Please refer to the device specific data sheet to see which LPMx.5 power modes are available. LPM4.5 allows for the lowest power consumption available. No clock sources are active during LPM4.5. LPM3.5 is similar to LPM4.5, but has the additional capability of having a RTC mode available. In addition to the wakeup events possible in LPM4.5, RTC wakeup events are also possible in LPM3.5.

The program flow for entering LPMx.5 is:

- Configure I/O appropriately. See the *Digital I/O* chapter for complete details on configuring I/O for LPMx.5.
  - Set all ports to general purpose I/O. Configure each port to ensure no floating inputs based on the application requirements.
  - If wakeup from I/O is desired, configure input ports with interrupt capability appropriately.
- If LPM3.5, is available, and desired, enable RTC operation. In addition, configure any RTC interrupts, if desired for LPM3.5 wakeup event. See the *RTC* chapter for complete details.
- Enter LPMx.5. The following code example shows how to enter LPMx.5 mode. See the *Power Management Module and Supply Voltage Supervisor* chapter for further details.

```
; Enter LPM5 Example
   MOV.B #PMMPW_H, &PMMCTL0_H                ; Open PMM registers for write
   BIS.B #PMMREGOFF, &PMMCTL0_L              ;
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR     ; Enter LPM5 when PMMREGOFF is set.
```

Exit from LPMx.5 is possible with a $\overline{\text{RST}}$ event, a power on cycle, or via specific I/O. Any exit from LPMx.5 will cause a BOR. Program execution will continue at the location stored in the system reset vector location 0FFFEh after execution of the boot code. The PMMLPM5IFG bit inside the PMM module will be set indicating that the device was in LPMx.5 prior to the wakeup event. Additionally, SYSRSTIV = 08h which can be used to generate an efficient reset handler routine. During LPMx.5, all I/O pin conditions are automatically locked to the current state. Upon exit from LPMx.5, the I/O pin conditions remain locked until the application unlocks them. See the *Digital I/O* chapter for complete details. If LPM3.5 was in effect, RTC operation will continue uninterrupted upon wakeup.The program flow for exiting LPMx.5 is:

- Enter system reset service routine
  - Reconfigure system as required for the application.
  - Reconfigure I/O as required for the application.

1.4.3 Extended Time in Low-Power Modes

The temperature coefficient of the DCO should be considered when the DCO is disabled for extended low-power mode periods. If the temperature changes significantly, the DCO frequency at wakeup may be significantly different from when the low-power mode was entered and may be out of the specified operating range. To avoid this, the DCO can be set to it lowest value before entering the low-power mode for extended periods of time where temperature can change.

```
; Enter LPM4 Example with lowest DCO Setting
   BIC   #SCG0, SR                             ; Disable FLL
   MOV   #0100h, &UCSCTL0                      ; Set DCO tap to first tap, clear
modulation.
   BIC   #DCORSEL2+DCORSEL1+DCORSEL0,&UCSCTL1 ; Lowest DCORSEL
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR      ; Enter LPM4
;  ...                                         ; Program stops
;

; Interrupt Service Routine
   BIC   #CPUOFF+OSCOFF+SCG1+SCG0,0(SR)       ; Exit LPM4 on RETI
   RETI
```

1.5 Principles for Low-Power Applications

Often, the most important factor for reducing power consumption is using the device clock system to maximize the time in LPM3 or LPM4 modes whenever possible.

- Use interrupts to wake the processor and control program flow.
- Peripherals should be switched on only when needed.
- Use low-power integrated peripheral modules in place of software driven functions. For example, Timer_A and Timer_B can automatically generate PWM and capture external timing with no CPU resources.
- Calculated branching and fast table look-ups should be used in place of flag polling and long software calculations.
- Avoid frequent subroutine and function calls due to overhead.
- For longer software routines, single-cycle CPU registers should be used.

If the application has low duty cycle, slow response time events, maximizing time in LPMx.5 can further reduce power consumption significantly.

1.6 Connection of Unused Pins

The correct termination of all unused pins is listed in Table 1-2.

Table 1-2. Connection of Unused Pins

Pin	Potential	Comment
AV_{CC}	DV_{CC}	
AV_{SS}	DV_{SS}	
Px.0 to Px.7	Open	Switched to port function, output direction (PxDIR.n = 1)
$\overline{RST}$/NMI	DV_{CC} or V_{CC}	47-kΩ pullup or internal pullup selected with 10-nF (2.2 nF[(1)]) pulldown[(1)]
TDO/TDI/TMS/TCK	Open	
TEST	Open	

(1) The pulldown capacitor should not exceed 2.2 nF when using devices with Spy-Bi-Wire interface in Spy-Bi-Wire mode or in 4-wire JTAG mode with TI tools like FET interfaces or GANG programmers.

1.7 Reset pin ($\overline{RST}$/NMI) Configuration

The reset pin can be configured as a reset function (default) or as an NMI function via the Special Function Register (SFR), SFRRPCR. Setting SYSNMI causes the $\overline{RST}$/NMI pin to be configured as an external NMI source. The external NMI is edge sensitive and its edge is selectable by SYSNMIIES. Setting the NMIIE enables the interrupt of the external NMI. Upon an external NMI event, the NMIIFG will be set.

The $\overline{RST}$/NMI pin can have either a pull-up or pull-down present or not. SYSRSTUP selects either pull-up or pull-down and SYSRSTRE will cause the pull-up or pull-down to be enabled or not. If the $\overline{RST}$/NMI pin is unused, it is required to have either the internal pull-up selected and enabled or an external resistor connected to the $\overline{RST}$/NMI pin as shown in Table 1-2

NOTE: All devices except the 543x (non-A devices) have the internal pull-up enabled. In this case, no external pull-up resistor is required.

1.8 Configuring JTAG pins

The JTAG pins are shared with general purpose I/O pins. There are several ways that the JTAG pins can be selected for four wire JTAG mode via software. Normally, upon a BOR, SYSJTAGPIN is cleared. With

SYSJTAGPIN cleared, the JTAG are configured as general purpse I/O. Please refer to the *Digital I/O* chapter for details on controlling the JTAG pins as general purpose I/O. If SYSJTAG = 1, the JTAG pins are configured to four wire JTAG mode and will remain in this mode until another BOR condition occurs. Therefore, SYSJTAGPIN is a write only once function. Clearing it by software is not possible, and the device will not change from four wire JTAG mode to general purpose I/O.

1.9 Boot Code

The boot code is always executed after a BOR. The boot code loads factory stored calibration values of the oscillator and reference voltages. In addition, it checks for a BSL entry sequence, as well as, checks for the presence of a user defined boot strap loader (BSL).

1.10 Bootstrap Loader (BSL)

The BSL is software that is executed after start-up when a certain BSL entry condition is applied. The BSL enables the user to communicate with the embedded memory in the microcontroller during the prototyping phase, final production, and in service. All memory mapped resources, the programmable memory (flash memory), the data memory (RAM), and the peripherals, can be modified by the BSL as required. The user can define his own BSL code for flash-based devices and protect it against erasure and unintentional or unauthorized access.

A basic BSL program is provided by TI. This supports the commonly used UART protocol with RS232 interfacing, allowing flexible use of both hardware and software. To use the BSL, a specific BSL entry sequence must be applied to specific device pins. The correct entry sequence will cause SYSBSLIND to be set. An added sequence of commands initiates the desired function. A boot-loading session can be exited by continuing operation at a defined user program address or by applying the standared reset sequence. Access to the device memory via the BSL is protected against misuse by a user-defined password. For more details, see the *MSP430 Memory Programming User's Guide* (SLAU265) at www.ti.com/msp430.

The amount of BSL memory that is available is device specific. The BSL memory size is organized into segments and can be set using the SYSBSLSIZE bits. Please refer to the device specific data sheet for the number and size of the segments available. It is possible to assign a small amount of RAM to the allocated BSL memory. Setting SYSBSLR allocates the lowest 16 bytes of RAM for the BSL. When the BSL memory is protected, access to these RAM locations is only possible from within the protected BSL memory segments.

It may be desirable in some BSL applications to only allow changing of the Power Management Module settings from the protected BSL segments. This is possible with the SYSPMMPE bit. Normally, this bit is cleared and allows access of the PMM control registers from any memory location. Setting SYSPMMPE, allows access to the PMM control registers only from the protected BSL memory. Once set, SYSPMMPE can only be cleared by a BOR event.

1.11 Memory Map – Uses and Abilities

This memory map represents the MSP430F5438 device. Though the address ranges differs from device to device, overall behavior remains the same.

Address Range	Name and Usage	Properties: Segment erase by user possible	Bank erase by user possible	Mass erase by user possible	Always able to access PMM registers from[(1)]; Mass erase by user possible	Protectable for read/write accesses	Generates PUC on fetch access	Can generate NMI on read/write/fetch
00000h-00FFFh	Peripherals with gaps							
00000h-000FFh	Reserved for system extension							
00100h-00FEFh	Peripherals						x	
00FF0h-00FF3h	Descriptor type[(2)]						x	
00FF4h-00FF7h	Start address of descriptor structure						x	
01000h-011FFh	BSL 0	x				x		
01200h-013FFh	BSL 1	x				x		
01400h-015FFh	BSL 2	x				x		
01600h-017FFh	BSL 3	x			x	x		
017FCh-017FFh	BSL Signature Location							
01800h-0187Fh	Info D	x						
01880h-018FFh	Info C	x						
01900h-0197Fh	Info B	x						
01980h-019FFh	Info A	x						
01A00h-01A7Fh	Device Descriptor Table						x	
01C00h-05BFFh	RAM 16 KB							
05B80-05BFFh	Alternate Interrupt Vectors							
05C00h-0FFFFh	Program	x	x[(1)]	x				
0FF80h-0FFFFh	Interrupt Vectors							
10000h-45BFFh	Program	x	x	x				
45C00h-FFFFFh	Vacant							x[(3)]

(1) Access rights are separately programmable for SYS and PMM.
(2) Fixed ID for all MSP430 devices. See Section 1.13.1 for further details.
(3) On vacant memory space, the value 03FFFh is driven on the data bus.

1.11.1 Vacant Memory Space

Vacant memory is non-existent memory space. Accesses to vacant memory space generate a system (non)maskable interrupt (SNMI) when enabled (VMAIE = 1). Reads from vacant memory results in the value 3FFFh. In the case of a fetch, this is taken as JMP $. Fetch accesses from vacant peripheral space result in a PUC. After the boot code is executed, it behaves like vacant memory space and also causes an NMI on access.

1.11.2 JTAG Lock Mechanism via the Electronic Fuse

A device can be protected from unauthorized access by disabling the JTAG and SBW interface. This is achieved by programming the electronic fuse. Programming the electronic fuse, completely disables the debug and access capabilities associated with the JTAG and SpyBiWire interface and is not reversible. The JTAG is locked by programming a certain signature into the devices' flash memory at dedicated addresses. The JTAG security lock key resides at the end of the bootstrap loader (BSL) memory at addresses 17FCh through 17FFh. Anything other than 0h or FFFFFFFFh programmed to these addresses locks the JTAG interface irreversibly.

All of the 5xx MSP430 devices come with a preprogrammed BSL (TI-BSL) code which by default protects itself from unintended erase and write access. This is done by setting SYSBSLPE in the SYSBSLC register. Since the JTAG security lock key resides in the BSL memory address range, appropriate action must be taken to unprotect the BSL memory area before programming the protection key. For more details on the electronic fuse see the *MSP430 Memory Programming User's Guide* (SLAU265) at www.ti.com/msp430.

Some JTAG commands are still possible after the device is secured, including the BYPASS command (see IEEE1149-2001 Standard) and the JMB_EXCHANGE command which allows access to the JTAG Mailbox System (see Table 7-2 for details).

1.12 JTAG Mailbox (JMB) System

The SYS module provides the capability to exchange user data via the regular JTAG test/debug interface. The idea behind the JMB is to have a direct interface to the CPU during debugging, programming, and test that is identical for all '430 devices of this family and uses only few or no user application resources. The JTAG interface was chosen because it is available on all '430 devices and is a dedicated resource for debugging, programming and test.

Applications of the JMB are:

- Providing entry password for device lock/unlock protection
- Run-time data exchange (RTDX)

1.12.1 JMB Configuration

The JMB supports two transfer modes - 16-bit and 32-bit. Setting JMBMODE enables 32-bit transfer mode. Clearing JMBMODE enables 16-bit transfer mode.

1.12.2 JMBOUT0 and JMBOUT1 Outgoing Mailbox

Two 16-bit registers are available for outgoing messages to the JTAG port. JMBOUT0 is only used when using 16-bit transfer mode (JMBMODE = 0). JMBOUT1 is used in addition to JMBOUT0 when using 32-bit transfer mode (JMBMODE = 1). When the application wishes to send a message to the JTAG port, it writes data to JMBOUT0 for 16-bit mode, or JMBOUT0 and JMBOUT1 for 32-bit mode.

JMBOUT0FG and JMBOUT1FG are read only flags that indicate the status of JMBOUT0 and JMBOUT1, respectively. When JMBOUT0FG is set, JMBOUT0 has been read by the JTAG port and is ready to receive new data. When JMBOUT0FG is reset, the JMBOUT0 is not ready to receive new data. JMBOUT1FG behaves similarly.

1.12.3 JMBIN0 and JMBIN1 Incoming Mailbox

Two 16-bit registers are available for incoming messages from the JTAG port. Only JMBIN0 is used when in 16-bit transfer mode (JMBMODE = 0). JMBIN1 is used in addition to JMBIN0 when using 32-bit transfer mode (JMBMODE = 1). When the JTAG port wishes to send a message to the application, it writes data to JMBIN0 for 16-bit mode, or JMBIN0 and JMBIN1 for 32-bit mode.

JMBIN0FG and JMBIN1FG are flags that indicate the status of JMBIN0 and JMBIN1, respectively. When JMBIN0FG is set, JMBIN0 has data that is available for reading. When JMBIN0FG is reset, no new data is available in JMBIN0. JMBIN1FG behaves similarly.

JMBIN0FG and JMBIN1FG can be configured to clear automatically by clearing JMBCLR0OFF and JMBCLR1OFF, respectively. Otherwise, these flags must be cleared by software.

1.12.4 JMB NMI Usage

The JMB handshake mechanism can be configured to use interrupts to avoid unnecessary polling if desired. In 16-bit mode, JMBOUTIFG is set when JMBOUT0 has been read by the JTAG port and is ready to receive data. In 32-bit mode, JMBOUTIFG is set when both JMBOUT0 and JMBOUT1 has been read by the JTAG port and are ready to receive data. If JMBOUTIE is set, these events cause a system NMI. In 16-bit mode, JMBOUTIFG is cleared automatically when data is written to JMBOUT0. In 32-bit mode, JMBOUTIFG Is cleared automatically when data is written to both JMBOUT0 and JMBOUT1. In addition, the JMBOUTIFG can be cleared when reading SYSSNIV. Clearing JMBOUTIE disables the NMI interrupt.

In 16-bit mode, JMBINIFG is set when JMBIN0 is available for reading. In 32-bit mode, JMBINIFG is set when both JMBIN0 and JMBIN1 are available for reading. If JMBOUTIE is set, these events cause a system NMI. In 16-bit mode, JMBINIFG is cleared automatically when JMBIN0 is read. In 32-bit mode, JMBINIFG Is cleared automatically when both JMBIN0 and JMBIN1 are read. In addition, the JMBINIFG can be cleared when reading SYSSNIV. Clearing JMBINIE disables the NMI interrupt.

1.13 Device Descriptor Table

Each device provides a data structure in memory that allows an unambiguous identification of the device, as well as, a more detailed description of the available modules on a given device. SYS provides this information and can be used by device-adaptive SW tools and libraries to clearly identify a particular device and all modules and capabilities contained within it. The validity of the device descriptor can be verified by cyclic redundancy check (CRC).Figure 1-7 shows the logical order and structure of the device descriptor table. The complete device descriptor table and its contents can be found in the device specific data sheet.

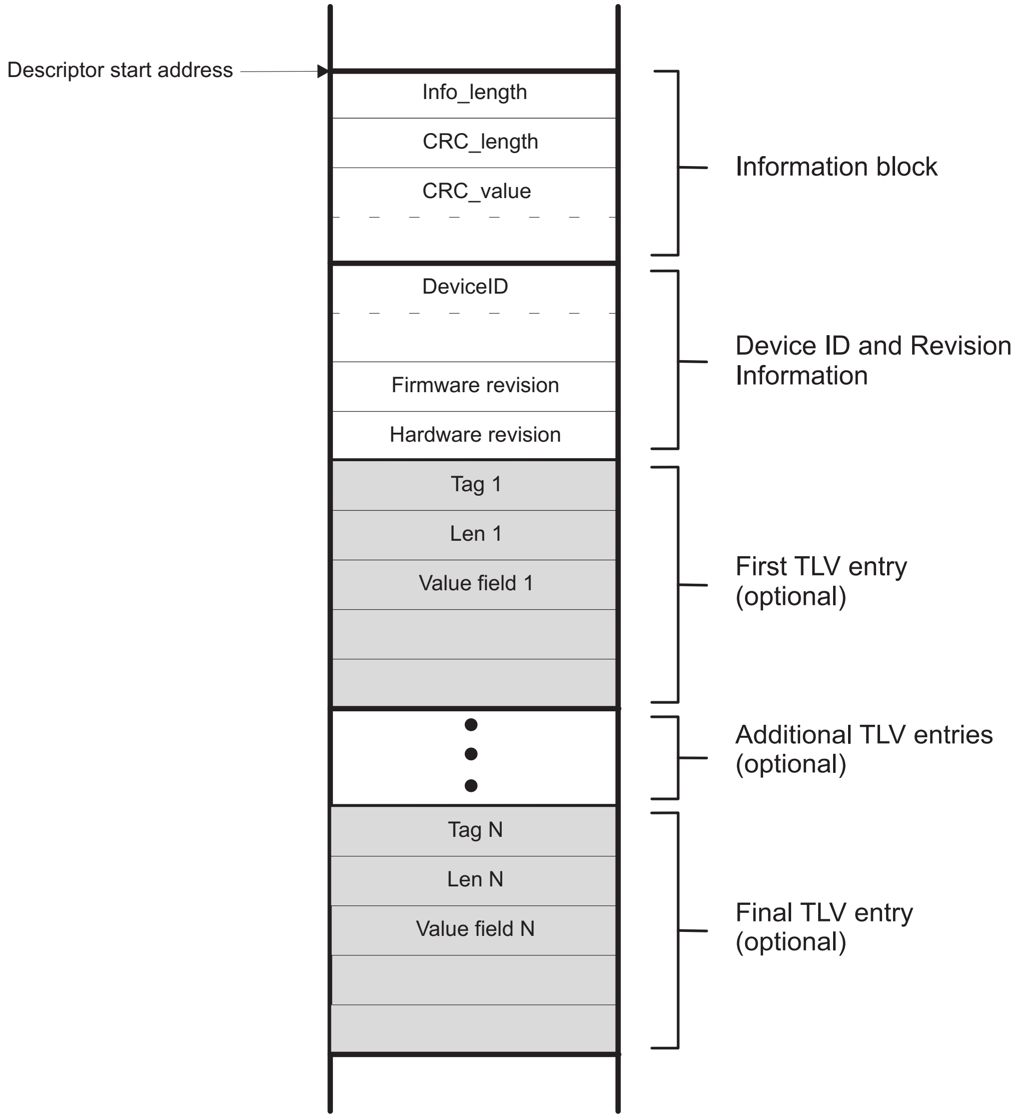


Figure 1-7. Devices Descriptor Table

1.13.1 Identifying Device Type

The value read at address location 00FF0h identifies the family branch of the device. All values starting with 80h indicate a hierarchical structure consisting of the information block and a TLV tag-length-value (TLV) structure containing the various descriptors. Any other value than 80h read at address location 00FF0h indicates the device is of an older family and contains a flat descriptor beginning at location 0FF0h. The information block, shown in Figure 1-7 contains the the device ID, die revisions, firmware revisions, and other manufacturer and tool related information. The descriptors contains information about the available peripherals, their subtypes and addresses and provides the information required to build adaptive HW drivers for operating systems.

The length of the descriptors represented by Info_length is computed as follows:

$$\text{Length} = 2^{\text{Info\_length}} \text{ in 32-bit words} \quad (1)$$

For example, if Info_length = 5, then the length of the descriptors equals 128 bytes.

1.13.2 TLV Descriptors

The TLV descriptors follow the information block. Because the information block is always a fixed length, the start location of the TLV descriptors is fixed for a given device family. For the MSP430x5xx family, this location is 01A08h. See the device-specific data sheet for the complete TLV structure and what descriptors are available.

The TLV descriptors are unique to their respective TLV block and are always follwed by the descriptor desciptor block length in

Each TLV descriptor contains a tag field which identifies the descriptor type. Table 1-3 shows the currently supported tags.

Table 1-3. Tag Values

Short Name	Value	Description
LDTAG	01h	Legacy descriptor (1xx, 2xx, 4xx families)
PDTAG	02h	Peripheral discovery descriptor
Reserved	03h	Future usage
Reserved	04h	Future usage
BLANK	05h	Blank descriptor
Reserved	06h	Future usage
ADCCAL	11h	ADC calibration
REFCAL	12h	REF calibration
Reserved	13h - FDh	Future usage
TAGEXT	FEh	Tag extender

Each tag field is unique to its respective descriptor and is always follwed by a length field. The length field is one byte if the tag value is 01h through 0FDh and represents the length of the descriptor in bytes. If the tag value equals 0FEh (TAGEXT), the next byte extends the tag values, and the following two bytes represent the length of the descriptor in bytes. In this way, a user can search through the TLV descriptor table for a particular tag value, using a routine similar to below written in pseudo code:

```
// Identifiy the descriptor ID (d_ID_value) for the TLV descriptor of interest:
descriptor_address = TLV_START address;

while ( value at descriptor_address != d_ID_value && descriptor_address != TLV_TAGEND &&
descriptor_address < TLV_END)
{
  // Point to next descriptor
  descriptor_address = descriptor_address + (length of the current TLV block) + 2;
}

if (value at descriptor_address == d_ID_value) {
  // Appropriate TLV decriptor has been found!
  Return length of descriptor & descriptor_address as the location of the TLV descriptor
} else {
  // No TLV descriptor found with a matching d_ID_value
  Return a failing condition
}
```

1.13.3 Peripheral discovery descriptor

This descriptor type can describe concatenated or distributed memory or peripheral mappings, as well as, the number of interrupt vectors and their order. The peripheral discovery discriptor has tag value 02h (PDTAG). Table 1-4 shows the structure of the peripheral discovery descriptor.

Table 1-4. Peripheral Discovery Descriptor

Element	Size (bytes)	Comments
memory entry 1	2	Optional
memory entry 2	2	Optional
...	2	Optional

Table 1-4. Peripheral Discovery Descriptor (continued)

Element	Size (bytes)	Comments
delimiter (00h)	1	Mandatory
peripheral count	1	Mandatory
peripheral entry 1	2	Optional
peripheral entry 2	2	Optional
...	2	Optional
Interrupt priority N-3	1	Optional
Interrupt priority N-4	1	Optional
...	1	Optional
delimiter (00h)	1	Mandatory

The structures for a memory entry and peripheral entry are shown below. A memory entry consists of two bytes (one word). Table 1-5 shows the individual bit fields of a memory entry word and their respective meanings. Similarly, a peripheral entry consists of two bytes (one word). Table 1-6 shows the individual bit fields of a peripheral entry word and their respective meanings.

Table 1-5. Values for Memory Entry

Bit fields				
[15:13]	[12:9]	[8]	[7]	[6:0]
Memory type	Size	More	Unit Size	Address value
000: None	0000: 0 B	0: End Entry	0: 0200h	0000000
001: RAM	0001: 128 B	1: More Entries	0: 010000h	0000001
010: EEPROM	0010: 256 B			0000010
011: Reserved	0011: 512 B			0000011
100: FLASH	0100: 1 KB			0000100
101: ROM	0101: 2KB			0000101
110: MemType appended	0110: 4 KB			0000110
111: Undefined	0111: 8 KB			0000111
	1000: 16 KB			0001000
	1001: 32 KB			0001001
	1010: 64 KB			0001010
	1011: 128 KB			0001011
	1100: 256 KB			0001100
	1101: 512 KB			...
	1110: Size appended			...
	1111: Undefined			1111111

Table 1-6. Values for Peripheral Entry

Bit fields		
[15:8]	[7]	[6:0]
Peripheral ID (PID) [(1)]	**UnitSize**	**AdrVal**
Any PID	0: 010h	0000000
Any PID	1: 0800h	0000001
Any PID		0000010
Any PID		0000011
Any PID		0000100
Any PID		0000101
Any PID		...
Any PID		...
Any PID		1111111

(1) The Peripheral IDs are listed in Table 1-7. This is not a complete list, but shown as an example.

Table 1-7. Peripheral IDs[(1)]

Peripheral or Module	PID
No Module	00h
WDT	01h
SFR	02h
UCS	03h
SYS	04h
PMM	05h
Flash Controller	08h
CRC16	09h
Port 1, 2	51h
Port 3, 4	52h
Port 5, 6	53h
Port 7, 8	54h
Port 9, 10	55h
Port J	5Fh
Timer A0	81h
Timer A1	82h
Special info appended	FEh
Undefined module	FFh

(1) This table is not a complete list of all peripheral IDs available on a device, but is shown here for illustrative purposes only.

Table 1-8 shows a simple example for a peripheral discovery descriptor of a hypothetical device:

Table 1-8. Sample Peripheral Discovery Descriptor

Hex	Binary	Entry type	Description
030h, 0Eh	001_1000_ 0_0_0001110	memory	RAM 16 KB; Start address = 01C00h (0Eh * 0200h)[1]
09Bh, 02Eh	100_1011_0_0_0101110	memory	FLASH 128 KB Start address = 05C00h (2Eh * 0200h)
00h	0000_0000_0000_0000	delimiter	No more memory entries
0Fh	0000_1111	peripheral count	Peripheral count = 15
02h, 10h	00000010_0_0010000	peripheral	SFR at address = 0100h (10h * 10h)
01h, 01h	00000001_0_0000001	peripheral	WDT at address = 0110h (0100h + 10h)
05h, 01h	00000101_0_0000001	peripheral	PMM at address = 0120h (0110h + 10h)
03h, 01h	00000011_0_0000001	peripheral	UCS at address = 0130h (0120h + 10h)
08h, 01h	00001000_0_0000001	peripheral	FLCTL at address = 0140h (0130h + 10h)
09h, 01h	00001001_0_0000001	peripheral	CRC16 at address = 0150h (0140h + 10h)
04h, 01h	00000100_0_0000001	peripheral	SYS at address = 0160h (0150h + 10h)
51h, 0Ah	01010001_0_0001010	peripheral	Port 1, 2 at address = 0200h (0160h + 10h * 10h)
52h, 02h	01010010_0_0000010	peripheral	Port 3, 4 at address = 0220h (0200h + 02h * 10h)
53h, 02h	01010011_0_0000010	peripheral	Port 5, 6 at address = 0240h (0220h + 02h * 10h)
54h, 02h	01010100_0_0000010	peripheral	Port 7, 8 at address = 0260h (0240h + 02h * 10h)
55h, 02h	01010101_0_0000010	peripheral	Port 9, 10 at address = 0280h (0260h + 02h * 10h)
5Fh, 0Ah	01011111_0_0001010	peripheral	Port J at address = 0320h (0280h + 0Ah * 10h)
81h, 02h	10000001_0_0000010	peripheral	Timer A0 at address = 0340h (0320h + 02h * 10h)
82h, 04h	10000010_0_0000100	peripheral	Timer A1 at address = 0380h (0340h + 04h * 10h)
–			No appended entries
			SYSRSTIV @0FFFEh (implied)
			SYSSNIV @0FFFC (implied)
			SYSUNIV @ 0FFFA (implied)
81h	1000_0001	interrupt	TA0 CCR0 @ 0FFF8
81h	1000_0001	interrupt	TA0 CCR1, CCR1, TA0IFG@ 0FFF6
51h	0101_0001	interrupt	Port 1 @ 0FFF4
82h	1000_0010	interrupt	TA1CCR0 @ 0FFF2
51h	0101_0001	interrupt	Port 2 @ 0FFF0
81h	1000_0010	interrupt	TA1 CCR1, CCR1, TA1IFG@ 0FFEE
00h	0000_0000	delimiter	No more interrupt entries

(1) In this example, the memory type is RAM (bits[15:13] = 001), the size is 16KB (bits[12:9] = 1000), and the starting address is 01C00h. The starting address is computed by taking the size field indicated by bit[7] (in this case 0200h) and multiplying it by the address value (bits[6:0] = 0001110. In this case, we have 0200h * 00Eh = 01C00h.

NOTE: The interrupt ordering has some implied rules:

- For timers, CCR0 interrupt has higher priority over all other CCRn interrupts.
- For communication ports, RX has higher priority over TX
- For port pairs, Port 1 has higher priority over Port 2, Port 3 has higher priority over Port 4, etc.

1.13.4 Calibration Values

The TLV structure contains calibraton values that can be used to improve the measurement capability of various functions. The calibration values available on a given device are shown in the TLV structure of the device-specific data sheet.

1.13.4.1 REF Calibration

The calibration data for the REF module consists of three words, one word for each reference voltage available (1.5, 2.0, and 2.5 V). The reference voltages are measured at room temperature. The measured values are normalized by 1.5/2.0/2.5V before being stored into the TLV structure, as shown below:

$$CAL\_ADC\_15VREF\_FACTOR = \frac{V_{REF+}}{1.5V} \times 2^{15}$$

$$CAL\_ADC\_20VREF\_FACTOR = \frac{V_{REF+}}{2.0V} \times 2^{15}$$

$$CAL\_ADC\_25VREF\_FACTOR = \frac{V_{REF+}}{2.5V} \times 2^{15} \tag{2}$$

In this way, a conversion result is corrected by multiplying it with the CAL_15VREF_FACTOR (or CAL_20VREF_FACTOR, CAL_25VREF_FACTOR) and dividing the result by 2^{15}as shown below for each of the respective reference voltages:

$$ADC(corrected) = ADC(raw) \times CAL\_ADC15VREF\_FACTOR \times \frac{1}{2^{15}}$$

$$ADC(corrected) = ADC(raw) \times CAL\_ADC20VREF\_FACTOR \times \frac{1}{2^{15}}$$

$$ADC(corrected) = ADC(raw) \times CAL\_ADC25VREF\_FACTOR \times \frac{1}{2^{15}} \tag{3}$$

In the following example, the integrated 1.5V reference voltage is used during a conversion.

- Conversion result: 0x0100 = 256 decimal
- Reference voltage calibration factor (CAL_15VREF_FACTOR) : 0x7BBB

The following steps show how the ADC conversion result can be corrected:

- Multiply the conversion result by 2 (this step simplifies the final division): 0x0100 x 0x0002 = 0x0200
- Multiply the result by CAL_15VREF_FACTOR: 0x200 x 0x7FEE = 0x00F7_7600
- Divide the result by 2^{16}: 0x00F7_7600 / 0x0001_0000 = 0x0000_00F7 = 247 decimal

1.13.4.2 ADC Offset and Gain Calibration

The offset of the ADC is determined and stored as a twos-complement number in the TLV structure. The offset error correction is done by adding the CAL_ADC_OFFSET to the conversion result.

$$ADC(offset\_corrected) = ADC(raw) + CAL\_ADC\_OFFSET \tag{4}$$

The gain of the ADC12 is calculated by the following equation:

$$CAL\_ADC\_GAIN\_FACTOR = \frac{1}{GAIN} \times 2^{15} \tag{5}$$

The conversion result is gain corrected by multiplying it with the CAL_ADC_GAIN_FACTOR and dividing the result by 2^{15}:

$$ADC(gain\_corrected) = ADC(raw) \times CAL\_ADC\_GAIN\_FACTOR \times \frac{1}{2^{15}} \tag{6}$$

If both gain and offset are corrected, the gain correction is done first:

$$ADC(gain\_corrected) = ADC(raw) \times CAL\_ADC\_GAIN\_FACTOR \times \frac{1}{2^{15}}$$

$$ADC(final) = ADC(gain\_corrected) + CAL\_ADC\_OFFSET \tag{7}$$

1.13.4.3 Temperature Sensor Calibration

The temperature sensor is calibrated using the internal voltage references. Each reference voltage (1.5/2.0/2.5V) contains a measured value for two temperatures, 30 °C±3 °C and 85 °C ±3 °C and are stored in the TLV structure. The characteristic equation of the temperature sensor voltage, in mV is:

$$V_{SENSE} = TC_{SENSOR} \times Temp + V_{SENSOR} \tag{8}$$

The temperature coefficient, TC_{SENSOR}in mV/°C, represents the slope of the equation. V_{SENSOR}, in mV, represents the y-intercept of the equation. Temp, in °C, is the temperature of interest.

The temperature (Temp, °C) can be computed as follows for each of the reference voltages used in the ADC measurement:

$$Temp = (ADC(raw) - CAL\_ADC\_15T30) \times \left(\frac{85-30}{CAL\_ADC\_15T85 - CAL\_ADC\_15T30} \right) + 30$$

$$Temp = (ADC(raw) - CAL\_ADC\_20T30) \times \left(\frac{85-30}{CAL\_ADC\_20T85 - CAL\_ADC\_20T30} \right) + 30$$

$$Temp = (ADC(raw) - CAL\_ADC\_25T30) \times \left(\frac{85-30}{CAL\_ADC\_25T85 - CAL\_ADC\_25T30} \right) + 30 \tag{9}$$

1.14 Special Function Registers (SFRs)

The SFRs are listed in Table 1-10. The base address for the SFRs is listed in Table 1-9. Many of the bits inside the SFRs are described in other chapters throughout the Users Guide. These bits will be marked with a note and a reference. Please refer to the specific chapter of the respective module for details.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 1-9. SFR Base Address

Module	Base Address
SFR	00100h

Table 1-10. Special Function Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
	SFRIE1	Read/write	Word	00h	0000h
Interrupt Enable	SFRIE1_L (IE1)	Read/write	Byte	00h	00h
	SFRIE1_H (IE2)	Read/write	Byte	01h	00h
	SFRIFG1	Read/write	Word	02h	0082h
Interrupt Flag	SFRIFG1_L (IFG1)	Read/write	Byte	02h	82h
	SFRIFG1_H (IFG2)	Read/write	Byte	03h	00h
	SFRRPCR	Read/write	Word	04h	0000h
Reset Pin Control	SFRRPCR_L	Read/write	Byte	04h	00h
	SFRRPCR_H	Read/write	Byte	05h	00h

Interrupt Enable Register (SFRIE1)

15	14	13	12	11	10	9	8
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
JMBOUTIE	**JMBINIE**	**ACCVIE** [1]	**NMIIE**	**VMAIE**	**Reserved**	**OFIE** [2]	**WDTIE** [3]
rw-0	rw-0	rw-0	rw-0	rw-0	r0	rw-0	rw-0

Reserved	Bits 15-8	Reserved. Reads back 0.
JMBOUTIE	Bit 7	JTAG mailbox output interrupt enable flag 0 Interrupts disabled 1 Interrupts enabled
JMBINIE	Bit 6	JTAG mailbox input interrupt enable flag 0 Interrupts disabled 1 Interrupts enabled
ACCVIE	Bit 5	Flash controller access violation interrupt enable flag 0 Interrupts disabled 1 Interrupts enabled
NMIIE	Bit 4	NMI pin interrupt enable flag 0 Interrupts disabled 1 Interrupts enabled
VMAIE	Bit 3	Vacant memory access interrupt enable flag 0 Interrupts disabled 1 Interrupts enabled
Reserved	Bit 2	Reserved. Reads back 0.
OFIE	Bit 1	Oscillator fault interrupt enable flag 0 Interrupts disabled 1 Interrupts enabled
WDTIE	Bit 0	Watchdog timer interrupt enable. This bit enables the WDTIFG interrupt for interval timer mode. It is not necessary to set this bit for watchdog mode. Because other bits in ~IE1 may be used for other modules, it is recommended to set or clear this bit using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instruction 0 Interrupts disabled 1 Interrupts enabled

(1) Refer to the *Flash Memory Controller* chapter for details.
(2) Refer to the *Unified Clock System* chapter for details.
(3) Refer to the *Watchdog Timer* chapter for details.

Interrupt Flag Register (SFRIFG1)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
JMBOUTIFG	**JMBINIFG**	**Reserved**	**NMIIFG**	**VMAIFG**	**Reserved**	**OFIFG** [1]	**WDTIFG** [2]
rw-(1)	rw-(0)	r0	rw-0	rw-0	r0	rw-(1)	rw-0

Reserved	Bits 15–8	Reserved. Reads back 0.
JMBOUTIFG	Bit 7	JTAG mailbox output interrupt flag 0 No interrupt pending. When in 16-bit mode (JMBMODE = 0), this bit is cleared automatically when JMBO0 has been written with a new message to the JTAG module by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both JMBO0 and JMBO1 have been written with new messages to the JTAG module by the CPU. This bit is also cleared when the associated vector in SYSUNIV has been read. 1 Interrupt pending, JMBO registers are ready for new messages. In 16-bit mode (JMBMODE = 0), JMBO0 has been received by the JTAG module and is ready for a new message from the CPU. In 32-bit mode (JMBMODE = 1) , JMBO0 and JMBO1 have been received by the JTAG moudle and are ready for new messages from the CPU.
JMBINIFG	Bit 6	JTAG mailbox input interrupt flag 0 No interrupt pending. When in 16-bit mode (JMBMODE = 0), this bit is cleared automatically when JMBI0 is read by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both JMBI0 and JMBI1 have been read by the CPU. This bit is also cleared when the associated vector in SYSUNIV has been read 1 Interrupt pending, a message is waiting in the JMBIN registers. In 16-bit mode (JMBMODE = 0) when JMBI0 has been written by the JTAG module. In 32-bit mode (JMBMODE = 1) when JMBI0 and JMBI1 have been written by the JTAG module.
Reserved	Bit 5	Reserved. Reads back 0.
NMIIFG	Bit 4	NMI pin interrupt flag 0 No interrupt pending 1 Interrupt pending
VMAIFG	Bit 3	Vacant memory access interrupt flag 0 No interrupt pending 1 Interrupt pending
Reserved	Bit 2	Reserved. Reads back 0.
OFIFG	Bit 1	Oscillator fault interrupt flag 0 No interrupt pending 1 Interrupt pending
WDTIFG	Bit 0	Watchdog timer interrupt flag. In watchdog mode, WDTIFG remains set until reset by software. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or can be reset by software. Because other bits in ~IFG1 may be used for other modules, it is recommended to set or clear WDTIFG by using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions. 0 No interrupt pending 1 Interrupt pending

(1) Refer to the *Unified Clock System* chapter for details.
(2) Refer to the *Watchdog Timer* chapter for details.

Reset Pin Control Register (SFRRPCR)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**SYSRSTRE**[(1)]	**SYSRSTUP**[(1)]	**SYSNMIIES**	**SYSNMI**
r0	r0	r0	r0	rw-1	rw-1	rw-0	rw-0

Reserved	Bits 15-4	Reserved. Reads back 0.
SYSRSTRE[(1)]	Bit 3	Reset pin resistor enable 0 Pullup/pulldown resistor at the $\overline{\text{RST}}$/NMI pin is disabled. 1 Pullup/pulldown resistor at the $\overline{\text{RST}}$/NMI pin is enabled.
SYSRSTUP[(1)]	Bit 2	Reset resistor pin pullup/pulldown 0 Pulldown is selected. 1 Pullup is selected.
SYSNMIIES	Bit 1	NMI edge select. This bit selects the interrupt edge for the NMI when SYSNMI = 1. Modifying this bit can trigger an NMI. Modify this bit when SYSNMI = 0 to avoid triggering an accidental NMI. 0 NMI on rising edge 1 NMI on falling edge
SYSNMI	Bit 0	NMI select. This bit selects the function for the RST/NMI pin. 0 Reset function 1 NMI function

(1) All devices except the MSP430F5438 (non-A) default to pullup enabled on the reset pin.
(1) All devices except the MSP430F5438 (non-A) default to pullup enabled on the reset pin.

1.15 SYS Configuration Registers

The SYS configuration registers are listed in Table 1-11 and the base address is listed in Table 1-11. A detailed description of each register and its bits is also provided. Each register starts at a word boundary. Both, word or byte data can be written to the SYS configuration registers.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 1-11. SYS Base Address

Module	Base address
SYS	00180h

Table 1-12. SYS Configuration Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
System Control	SYSCTL	Read/write	Word	00h	0000h
	SYSCTL_L	Read/write	Byte	00h	00h
	SYSCTL_H	Read/write	Byte	01h	00h
Bootstrap Loader Configuration	SYSBSLC	Read/write	Word	02h	0003h
	SYSBSLC_L	Read/write	Byte	02h	03h
	SYSBSLC_H	Read/write	Byte	03h	00h
JTAG Mailbox Control	SYSJMBC	Read/write	Word	06h	0000h
	SYSJMBC_L	Read/write	Byte	06h	00h
	SYSJMBC_H	Read/write	Byte	07h	00h
JTAG Mailbox Input 0	SYSJMBI0	Read/write	Word	08h	0000h
	SYSJMBI0_L	Read/write	Byte	08h	00h
	SYSJMBI0_H	Read/write	Byte	09h	00h
JTAG Mailbox Input 1	SYSJMBI1	Read/write	Word	0Ah	0000h
	SYSJMBI1_L	Read/write	Byte	0Ah	00h
	SYSJMBI1_H	Read/write	Byte	0Bh	00h
JTAG Mailbox Output 0	SYSJMBO0	Read/write	Word	0Ch	0000h
	SYSJMBO0_L	Read/write	Byte	0Ch	00h
	SYSJMBO0_H	Read/write	Byte	0Dh	00h
JTAG Mailbox Output 1	SYSJMBO1	Read/write	Word	0Eh	0000h
	SYSJMBO1_L	Read/write	Byte	0Eh	00h
	SYSJMBO1_H	Read/write	Byte	0Fh	00h
Bus Error Vector Generator	SYSBERRIV	Read	Word	18h	0000h
User NMI Vector Generator	SYSUNIV	Read	Word	1Ah	0000h
System NMI Vector Generator	SYSSNIV	Read	Word	1Ch	0000h
Reset Vector Generator	SYSRSTIV	Read	Word	1Eh	0002h

SYS Control Register (SYSCTL)

15	14	13	12	11	10	9	8
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**SYSJTAGPIN**	**SYSBSLIND**	**Reserved**	**SYSPMMPE**	**Reserved**	**SYSRIVECT**
r0	r0	rw-[0]	r-0	r0	rw-[0]	r0	rw-[0]

Reserved Bits 15-6 Reserved. Reads back 0.

SYSJTAGPIN Bit 5 Dedicated JTAG pins enable. Setting this bit disables the shared functionality of the JTAG pins and permanently enables the JTAG function. This bit can only be set once. Once it is set it remains set until a BOR occurs.

- 0 Shared JTAG pins (JTAG mode selectable via SBW sequence)
- 1 Dedicated JTAG pins (explicit 4-wire JTAG mode selection)

SYSBSLIND Bit 4 BSL entry indication. This bit indicates a BSL entry sequence detected on the Spy-Bi-Wire pins.

- 0 No BSL entry sequence detected
- 1 BSL entry sequence detected

Reserved Bit 3 Reserved. Reads back 0.

SYSPMMPE Bit 2 PMM access protect. This controls the accessibility of the PMM control registers. Once set to 1, it only can be cleared by a BOR.

- 0 Access from anywhere in memory
- 1 Access only from the protected BSL segments

Reserved Bit 1 Reserved. Reads back 0.

SYSRIVECT Bit 0 RAM-based interrupt vectors

- 0 Interrupt vectors generated with end address TOP of lower 64k flash FFFFh
- 1 Interrupt vectors generated with end address TOP of RAM

Bootstrap Loader Configuration Register (SYSBSLC)

15	14	13	12	11	10	9	8
SYSBSLPE	**SYSBSLOFF**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
rw-[0]	rw-[0]	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**SYSBSLR**	**SYSBSLSIZE**	
r0	r0	r0	r0	r0	rw-[0]	rw-[1]	rw-[1]

SYSBSLPE Bit 15 Bootstrap loader memory protection enable for the size covered in SYSBSLSIZE. By default, this bit is cleared by hardware with a BOR event (as indicated above), however the boot code that checks for an available BSL may set this bit via software in order to protect the BSL. Since devices normally come with a TI BSL preprogrammed and protected, the boot code will set this bit.

- 0 Area not protected. Read, program, and erase of BSL memory is possible.
- 1 Area protected

SYSBSLOFF Bit 14 Bootstrap loader memory disable for the size covered in SYSBSLSIZE

- 0 BSL memory is addressed when this area is read.
- 1 BSL memory behaves like vacant memory. Reads will cause 3FFFh to be read. Fetches will cause JMP $ to be executed.

Reserved Bits 13-3 Reserved. Reads back 0.

SYSBSLR Bit 2 RAM assigned to BSL

- 0 No RAM assigned to BSL area
- 1 Lowest 16 bytes of RAM assigned to BSL

SYSBSLSIZE Bits 1-0 Bootstrap loader size. Defines the space and size of flash memory that is reserved for the BSL.

- 00 Size: BSL segment 3.
- 01 Size: BSL segments 2 and 3.
- 10 Size: BSL segements 1, 2, and 3.
- 11 Size: BSL segments 1, 2, 3, and 4.

JTAG Mailbox Control Register (SYSJMBC)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
JMBCLR1OFF	**JMBCLR0OFF**	**Reserved**	**JMBM0DE**	**JMBOUT1FG**	**JMBOUT0FG**	**JMBIN1FG**	**JMBIN0FG**
rw-(0)	rw-(0)	r0	rw-0	r-(1)	r-(1)	rw-(0)	rw-(0)

Reserved	Bits 15-8	Reserved. Reads back 0.
JMBCLR1OFF	Bit 7	Incoming JTAG Mailbox 1 flag auto-clear disable 0 JMBIN1FG cleared on read of JMB1IN register 1 JMBIN1FG cleared by SW
JMBCLR0OFF	Bit 6	Incoming JTAG Mailbox 0 flag auto-clear disable 0 JMBIN0FG cleared on read of JMB0IN register 1 JMBIN0FG cleared by SW
Reserved	Bit 5	Reserved. Reads back 0.
JMBMODE	Bit 4	This bit defines the operation mode of JMB for JMBI0/1 and JMBO0/1. Before switching this bit, pad and flush out any partial content to avoid data drops. 0 16-bit transfers using JMBO0 and JMBI0 only 1 32-bit transfers using JMBO0/1 and JMBI0/1
JMBOUT1FG	Bit 3	Outgoing JTAG Mailbox 1 flag. This bit is cleared automatically when a message is written to the upper byte of JMBO1 or as word access (by the CPU, DMA,...) and is set after the message was read via JTAG. 0 JMBO1 is not ready to receive new data. 1 JMBO1 is ready to receive new data.
JMBOUT0FG	Bit 2	Outgoing JTAG Mailbox 0 flag. This bit is cleared automatically when a message is written to the upper byte of JMBO0 or as word access (by the CPU, DMA,...) and is set after the message was read via JTAG. 0 JMBO0 is not ready to receive new data. 1 JMBO0 is ready to receive new data.
JMBIN1FG	Bit 1	Incoming JTAG Mailbox 1 flag. This bit is set when a new message (provided via JTAG) is available in JMBI1. This flag is cleared automatically on read of JMBI1 when JMBCLR1OFF = 0 (auto clear mode). On JMBCLR1OFF = 1, JMBIN1FG needs to be cleared by SW. 0 JMBI1 has no new data. 1 JMBI1 has new data available.
JMBIN0FG	Bit 0	Incoming JTAG Mailbox 0 flag. This bit is set when a new message (provided via JTAG) is available in JMBI0. This flag is cleared automatically on read of JMBI0 when JMBCLR0OFF = 0 (auto clear mode). On JMBCLR0OFF = 1, JMBIN0FG needs to be cleared by SW. 0 JMBI1 has no new data. 1 JMBI1 has new data available.

JTAG Mailbox Input 0 Register (SYSJMBI0)
JTAG Mailbox Input 1 Register (SYSJMBI1)

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

7	6	5	4	3	2	1	0
MSGL0							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

MSGHI	Bits 15-8	JTAG mailbox incoming message high byte
MSGLO	Bits 7-0	JTAG mailbox incoming message low byte

JTAG Mailbox Output 0 Register (SYSJMBO0)
JTAG Mailbox Output 1 Register (SYSJMBO1)

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

7	6	5	4	3	2	1	0
MSGL0							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

MSGHI	Bits 15-8	JTAG mailbox outgoing message high byte
MSGLO	Bits 7-0	JTAG mailbox outgoing message low byte

User NMI Vector Register (SYSUNIV)

15 7	14 6	13 5	12 4	11 3	10 2	9 1	8 0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**SYSUNVEC**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSUNIV Bits 15-0 User NMI vector. Generates a value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending user NMI flags.

Value	**Interrupt Type**
0000h	No interrupt pending
0002h	NMIIFG interrupt pending (highest priority)
0004h	OFIFG interrupt pending
0006h	ACCVIFG interrupt pending
0008h	Reserved for future extensions

NOTE: Additional events for more complex devices will be appended to this table; sources that are removed will reduce the length of this table. The vectors are expected to be accessed symbolic only with the corresponding include file of the device in use.

System NMI Vector Register (SYSSNIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**SYSSNVEC**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSSNIV Bits 15-0 System NMI vector. Generates a value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending system NMI flags.

Value	Interrupt Type
0000h	No interrupt pending
0002h	SVMLIFG interrupt pending (highest priority)
0004h	SVMHIFG interrupt pending
0006h	SVSMLDLYIFG interrupt pending
0008h	SVSMHDLYIFG interrupt pending
000Ah	VMAIFG interrupt pending
000Ch	JMBINIFG interrupt pending
000Eh	JMBOUTIFG interrupt pending
0010h	SVMLVLRIFG interrupt pending
0012h	SVMHVLRIFG interrupt pending
0014h	Reserved for future extensions

NOTE: Additional events for more complex devices will be appended to this table; sources that are removed will reduce the length of this table. The vectors are expected to be accessed symbolic only with the corresponding include file of the used device.

Reset Interrupt Vector Register (SYSRSTIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**SYSRSTVEC**					**0**
r0	r0	r-0	r-0	r-0	r-0	r-1	r0

SYSRSTIV Bits 15-0 Reset interrupt vector. Generates a value that can be used as address offset for fast interrupt service routine handling to identify the last cause of a reset (BOR, POR, PUC) . Writing to this register clears all pending reset source flags.

Value	Interrupt Type
0000h	No interrupt pending
0002h	Brownout (BOR) (highest priority)
0004h	$\overline{\text{RST}}$/NMI (BOR)
0006h	PMMSWBOR (BOR)
0008h	Wakeup from LPMx.5 (BOR)
000Ah	Security violation (BOR)
000Ch	SVSL (POR)
000Eh	SVSH (POR)
0010h	SVML_OVP (POR)
0012h	SVMH_OVP (POR)
0014h	PMMSWPOR (POR)
0016h	WDT time out (PUC)
0018h	WDT password violation (PUC)
001Ah	Flash password violation (PUC)
001Ch	PLL unlock (PUC)
001Eh	PERF peripheral/configuration area fetch (PUC)
0020h	PMM password violation (PUC)
0022h-003Eh	Reserved for future extensions

NOTE: Additional events for more complex devices will be appended to this table; sources that are removed will reduce the length of this table. The vectors are expected to be accessed symbolic only with the corresponding include file of the used device.

System Bus Error Interrupt Vector Register (SYSBERRIV)

15 7	14 6	13 5	12 4	11 3	10 2	9 1	8 0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**SYSBERRIV**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSBERRIV Bits 15-0 System bus error interrupt vector. Generates a value that can be used as an address offset for fast interrupt service routine handling. Writing to this register clears all pending flags.

Value	Interrupt Type
0000h	No interrupt pending
0002h	USB module timed out. Wait state time out of 8 clock cycles. 16 clock cycles only on the 'F552x, 'F551x devices.
0004h	Reserved for future extensions
0006h	Reserved for future extensions
0008h	Reserved for future extensions

NOTE: Additional events for more complex devices will be appended to this table; sources that are removed will reduce the length of this table. The vectors are expected to be accessed symbolic only with the corresponding include file of the used device.

Power Management Module and Supply Voltage Supervisor

This chapter describes the operation of the Power Management Module (PMM) and Supply Voltage Supervisor (SVS).

Topic **Page**

2.1	**Power Management Module (PMM) Introduction**	**58**
2.2	**PMM Operation**	**60**
2.3	**PMM Registers**	**70**

2.1 Power Management Module (PMM) Introduction

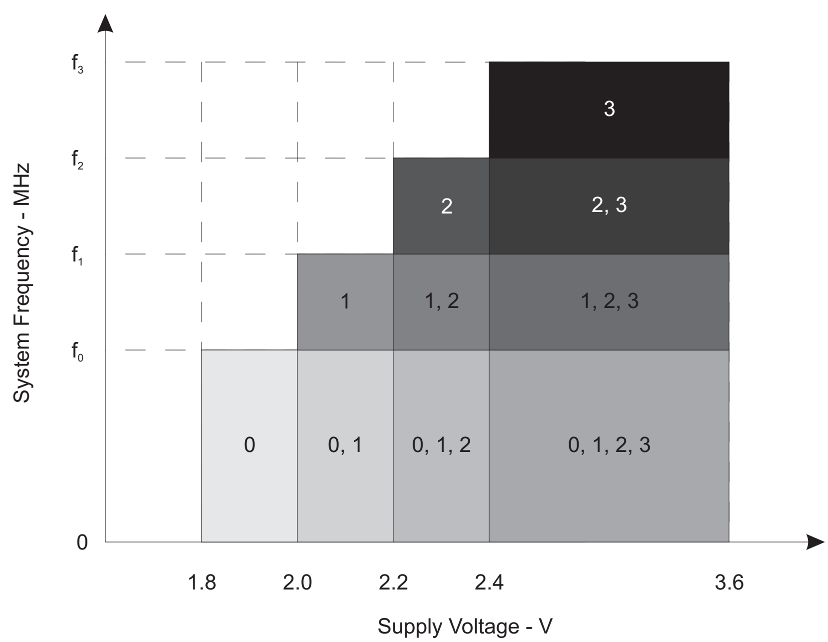
PMM features include:

- Wide supply voltage (DV_{CC}) range: 1.8 V to 3.6 V
- Generation of voltage for the device core (V_{CORE}) with up to four programmable levels
- Supply voltage supervisor (SVS) for DV_{CC} and V_{CORE} with programmable threshold levels
- Supply voltage monitor (SVM) for DV_{CC} and V_{CORE} with programmable threshold levels
- Brownout reset (BOR)
- Software accessible power-fail indicators
- I/O protection during power-fail condition
- Software selectable supervisor or monitor state output (optional)

The PMM manages all functions related to the power supply and its supervision for the device. Its primary functions are first to generate a supply voltage for the core logic, and second, provide several mechanisms for the supervision and monitoring of both the voltage applied to the device (DV_{CC}) and the voltage generated for the core (V_{CORE}).

The PMM uses an integrated low-dropout voltage regulator (LDO) to produce a secondary core voltage (V_{CORE}) from the primary one applied to the device (DV_{CC}). In general, V_{CORE} supplies the CPU, memories (flash/RAM), and the digital modules, while DV_{CC} supplies the I/Os and all analog modules (including the oscillators). The V_{CORE} output is maintained using a dedicated voltage reference. V_{CORE} is programmable up to four steps, to provide only as much power as is needed for the speed that has been selected for the CPU. This enhances power efficiency of the system. The input or primary side of the regulator is referred to in this chapter as its high side. The output or secondary side is referred to in this chapter as its low side.

The required minimum voltage for the core depends on the selected MCLK rate. Figure 2-1 shows the relationship between the system frequency for a given core voltage setting, as well as the minimum required voltage applied to the device. Figure 2-1 only serves as an example, and the device-specific data sheet should be referenced to determine which core voltage levels are supported and what level of system frequency performance is possible.



The numbers within the fields denote the supported PMMCOREVx settings.

Figure 2-1. System Frequency and Supply/Core Voltages - See Device Specific Datasheet

The PMM module provides a means for DV_{CC} and V_{CORE} to be supervised and monitored. Both of these functions detect when a voltage falls under a specific threshold. In general, the difference is that supervision results in a power-on reset (POR) event, while monitoring results in the generation of an interrupt flag that software may then handle. As such, DV_{CC} is supervised and monitored by the high-side supervisor (SVS_H) and high-side monitor (SVM_H), respectively. V_{CORE} is supervised and monitored by the low-side supervisor (SVS_L) and low-side monitor (SVM_L), respectively. Thus, there are four separate supervision/monitoring modules that can be active at any given time. The thresholds enforced by these modules are derived from the same voltage reference used by the regulator to generate V_{CORE}.

In addition to the SVS_H / SVM_H / SVS_L / SVM_L modules, V_{CORE} is further monitored by the brownout reset (BOR) circuit. As DV_{CC} ramps up from 0 V at power up, the BOR keeps the device in reset until V_{CORE} is at a sufficient level for operation at the default MCLK rate and for the SVS_H/SVS_L mechanisms to be activated. During operation, the BOR also generates a reset if V_{CORE} falls below a preset threshold. BOR can be used to provide an even lower-power means of monitoring the supply rail if the flexibility of the SVS_L is not required.

The block diagram of the PMM is shown in Figure 2-2.

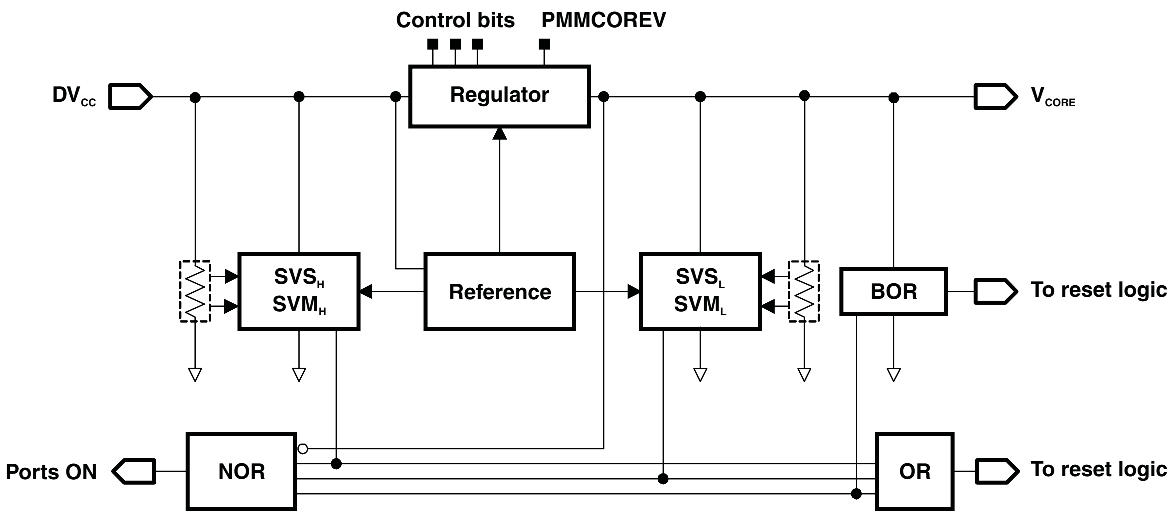


Figure 2-2. PMM Block Diagram

2.2 PMM Operation

2.2.1 V_{CORE} and the Regulator

DV_{CC} can be powered from a wide input voltage range, but the core logic of the device must be kept at a voltage lower than what this range allows. For this reason, a regulator has been integrated into the PMM. The regulator derives the necessary core voltage (V_{CORE}) from DV_{CC}.

Higher MCLK speeds require higher levels of V_{CORE}. Higher levels of V_{CORE} consume more power, and so the core voltage has been made programmable in up to four steps to allow it to provide only as much power as is required for a given MCLK setting. The level is controlled by the PMMCOREV bits. Note that the default setting, the lowest value of PMMCOREV, enables operation of MCLK over a very wide frequency range. As such, no PMM changes are required for many applications. See the device-specific data sheet for performance characteristics and core step levels supported.

Before increasing MCLK to a higher speed, it is necessary for software to ensure that the V_{CORE} level is sufficiently high for the chosen frequency. Failure to do so may force the CPU to attempt operation without sufficient power, which can cause unpredictable results. See Section 2.2.4 for more information on the appropriate procedure to raise V_{CORE} for higher MCLK frequencies.

The regulator supports two different load settings to optimize power. The high-current mode is required when:

- The CPU is in active, LPM0, or LPM1 modes
- A clock source greater than 32 kHz is used to drive any module
- An interrupt is executed

Otherwise, the low-current mode is used. The hardware controls the load settings automatically, according to the criteria above.

2.2.2 Supply Voltage Supervisor and Monitor

The high-side supervisor and monitor (SVS_H and SVM_H) and the low-side supervisor and monitor (SVS_L and SVM_L) oversee DV_{CC} and V_{CORE}, respectively. By default, all these modules are active, but each can be disabled using the corresponding enable bit (SVSHE/SVMHE/SVSLE/SVMLE), resulting in some power savings.

2.2.2.1 SVS/SVM Thresholds

The voltage thresholds enforced by the SVS/SVM modules are selectable. Table 2-1 shows the SVS/SVM threshold registers, the voltage threshold they control, and the number of threshold options.

Table 2-1. SVS/SVM Thresholds

Register	Description	Threshold	Available Steps
SVSHRVL	SVS_H reset voltage level	SVS_{H_IT-}	4
SVSMHRRL	SVS_H/SVM_H reset release voltage level	SVS_{H_IT+}, SVM_H	8
SVSLRVL	SVS_L reset voltage level	SVS_{L_IT-}	4
SVSMLRRL	SVS_L/SVM_L reset release voltage level	SVS_{L_IT+}, SVM_L	4

Recommended SVS_L Settings

For each of the core voltages, there are two supply voltage supervisor levels available. The SVSLRVL bits define the voltage level of VCORE below which the reset is activated. The SVSMLRRL bits define the voltage level of VCORE at which the reset is released. Although various settings can be chosen, there is one set of SVSLRVL and SVSMLRRL settings that is well suited for each core voltage selected by PMMCOREV. By default, an SVS_L event will always generate a POR (SVSLPE = 1) and it is recommended to always configure SVSLPE = 1 for reliable device startup. The most commonly used and recommended settings are shown in Table 2-2.

Table 2-2. Recommended SVS_L Settings

PMMCOREV[1:0]	DVCC, (Volts)	SVSLRVL[1:0] Sets SVS_{L_IT-} level	SVSMLRRL[2:0] Sets SVS_{L_IT+} and SVM_L levels
00	≥ 1.8	00	000
01	≥ 2.0	01	001
10	≥ 2.2	10	010
11	≥ 2.4	11	011

Recommended SVS_H Settings

For the high side supply, there are two supply voltage supervisor levels available. The SVSMHRRL bits define the voltage level of DVCC at which the reset is released. The SVSHRVL register defines the voltage level of DVCC below which the reset is turned on. These settings should be selected according to the minimum voltages required for device operation in a given application, as well as system power supply characteristics. See the device-specific data sheet for threshold values corresponding to the settings shown here. Although various settings are available, the most common are based on the maximum frequency required, which will in turn, determine the minimum DVCC level supervised. By default, an SVS_H event will always generate a POR (SVSHPE = 1) and it is recommended to always configure SVSHPE = 1 for reliable device startup. The most commonly used and recommended settings are shown in Table 2-3 .

Table 2-3. Recommended SVS_H Settings

f_{SYS} max in MHz	DVCC in V	SVSHRVL[1:0] Sets SVS_{H_IT-} level	SVSMHRRL[2:0] Sets SVS_{H_IT+} and SVM_H levels	PMMCOREV[1:0]
8	>1.8	00	000	00
12	>2.0	01	001	01
20	>2.2	10	010	10
25	>2.4	11	011	11

The available voltage threshold settings of SVS_H and SVM_H are dependent on the voltage level setting of VCORE. Table 2-4 summarizes all the possible settings available. All other settings not listed are invalid and should not be used.

Table 2-4. Available SVS_H, SVS_M Settings Versus VCORE Settings

PMMCOREV[1:0]	SVSHRVL[1:0] Sets SVS_{H_IT-} level	SVSMHRRL[2:0] Sets SVS_{H_IT+} and SVM_H levels
00	00 through 11	000 through 011
01	00 through 11	000 through 100
10	00 through 11	000 through 101
11	00 through 11	000 through 111

The behavior of the SVS/SVM according to these thresholds is best portrayed graphically. Figure 2-3 shows how the supervisors and monitors respond to various supply failure conditions.

As Figure 2-3 shows, there is hysteresis built into the supervision thresholds, such that the thresholds in force depend on whether the voltage rail is going up or down. There is no hysteresis in the monitoring thresholds.

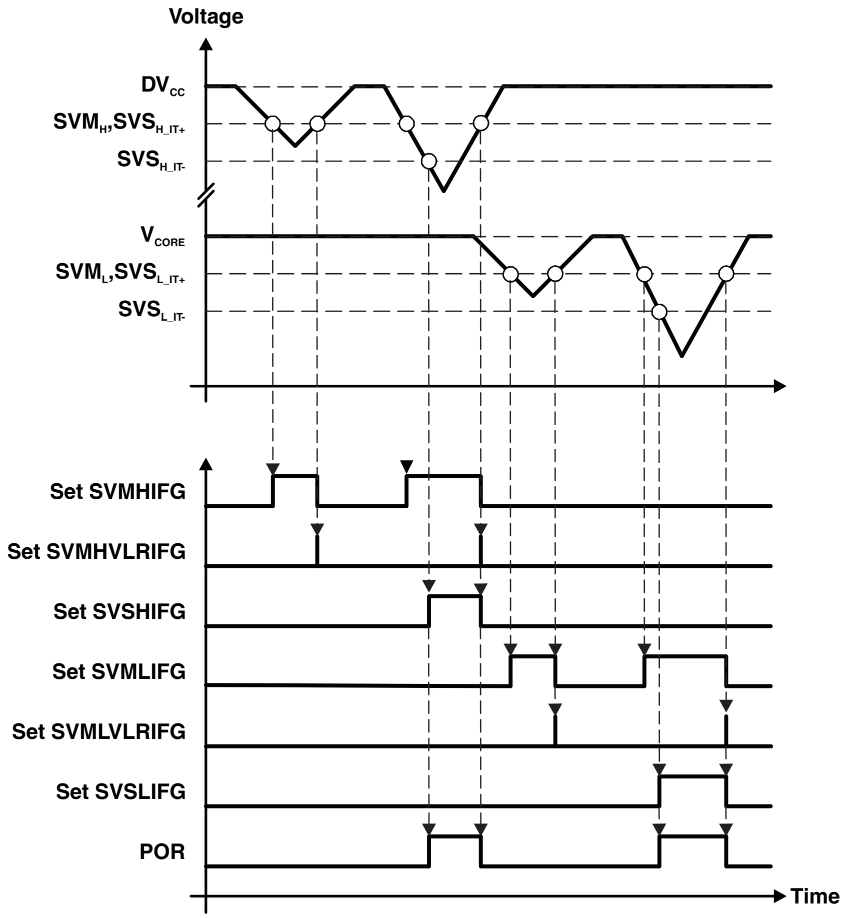


Figure 2-3. High-Side and Low-Side Voltage Failure and Resulting PMM Actions

2.2.2.2 High Side Supervisor/Monitor (SVS_H/SVM_H)

The SVS_H and SVM_H modules are enabled by default. They can be disabled by clearing the SVSHE and SVMHE bits, respectively. Their block diagrams are shown in Figure 2-4.

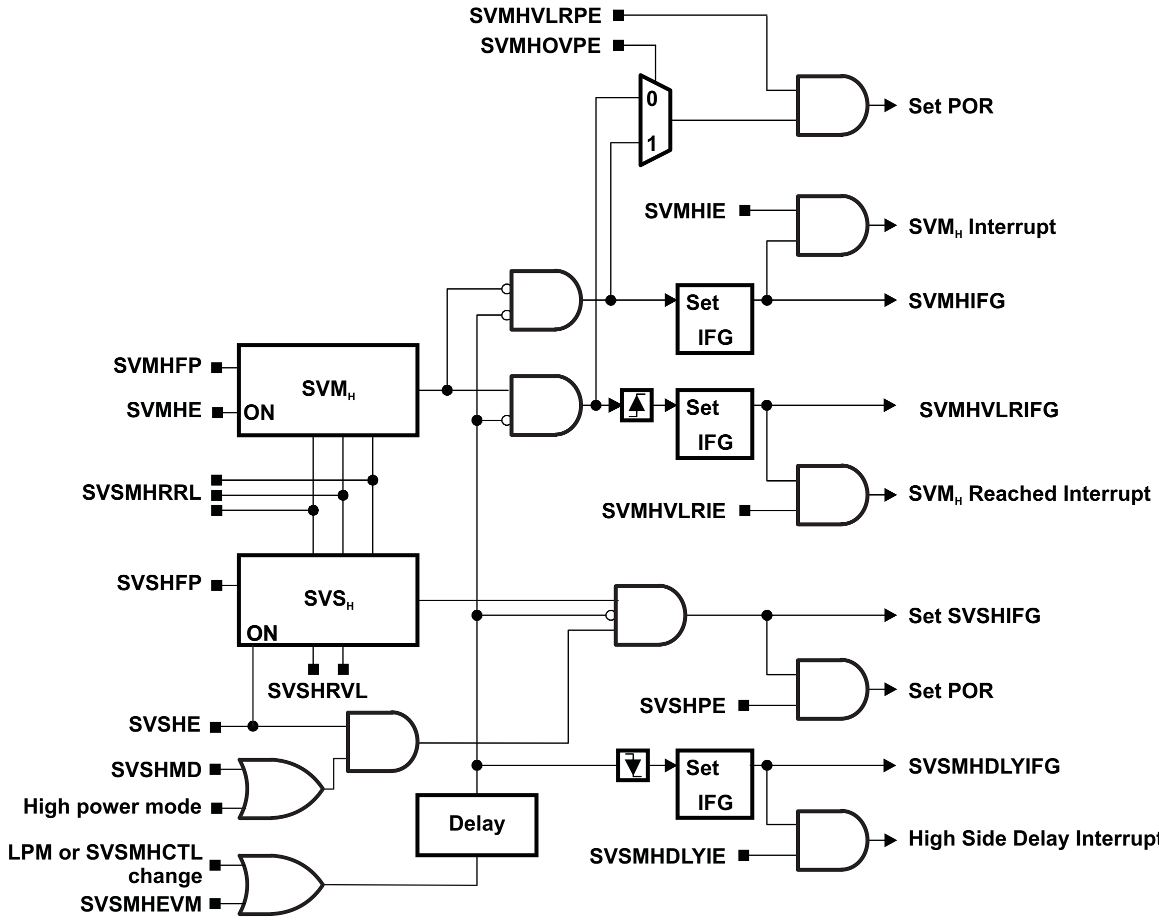


Figure 2-4. High-Side SVS and SVM

If DV_{CC} falls below the SVS_H level, SVSHIFG (SVS_H interrupt flag) is set. If DV_{CC} remains below the SVS_H level and software attempts to clear SVSHIFG, it is immediately set again by hardware. If the SVSHPE (SVS_H POR enable) bit is set when SVSHIFG gets set, a POR is generated.

If DV_{CC} falls below the SVM_H level, SVMHIFG (SVM_H interrupt flag) is set. If DV_{CC} remains below the SVM_H level and software attempts to clear SVMHIFG, it is immediately set again by hardware. If the SVMHIE (SVM_H interrupt enable) bit is set when SVMHIFG gets set, an interrupt is generated. If a POR is desired when SVMHIFG is set, the SVM_H can be configured to do so by setting the SVMHVLRPE (SVM_H voltage level reached POR enable) bit while SVMHOVPE bit is cleared.

If DV_{CC} rises above the SVM_H level, the SVMHVLRIFG (SVM_H voltage level reached) interrupt flag is set. If SVMHVLRIE (SVM_H voltage level reached interrupt enable) is set when this occurs, an interrupt is also generated.

The SVM_H module can also be used for overvoltage detection. This is accomplished by setting the SVMHOVPE (SVM_H overvoltage POR enable) bit, in addition to setting SVMHVLRPE. Under these conditions, if DV_{CC} exceeds safe device operation, a POR is generated.

The SVS_H/SVM_H modules have configurable performance modes for power-saving operation. (See Section 2.2.8 for more information.) If these SVS_H/SVM_H power modes are modified, or if a voltage level is modified, a delay element masks the interrupts and POR sources until the SVS_H/SVM_H circuits have settled. When SVSMHDLYST (delay status) reads zero, the delay has expired. In addition, the SVSMHDLYIFG (SVS_H/SVM_H delay expired) interrupt flag is set. If the SVSMHDLYIE (SVS_H /SVM_H delay expired interrupt enable) is set when this occurs, an interrupt is also generated.

In case of power-fail conditions, setting SVSHMD will cause the SVS_H interrupt flag to be set in LPM2, LPM3, and LPM4. If SVSHMD is not set, the SVS_H interrupt flag will not be set in LPM2, LPM3, and LPM4. In addition, all SVS_H and SVM_H events can be masked by setting SVSMHEVM. For most applications, SVSMHEVM should be cleared.

All the interrupt flags of SVS_H /SVM_H remain set until cleared by a BOR or by software.

2.2.2.3 Low-Side Supervisor/Monitor (SVS_L/SVM_L)

The SVS_L and SVM_L modules are enabled by default. They can be disabled by clearing SVSLE and SVMLE bits, respectively. Their block diagrams are shown in Figure 2-5.

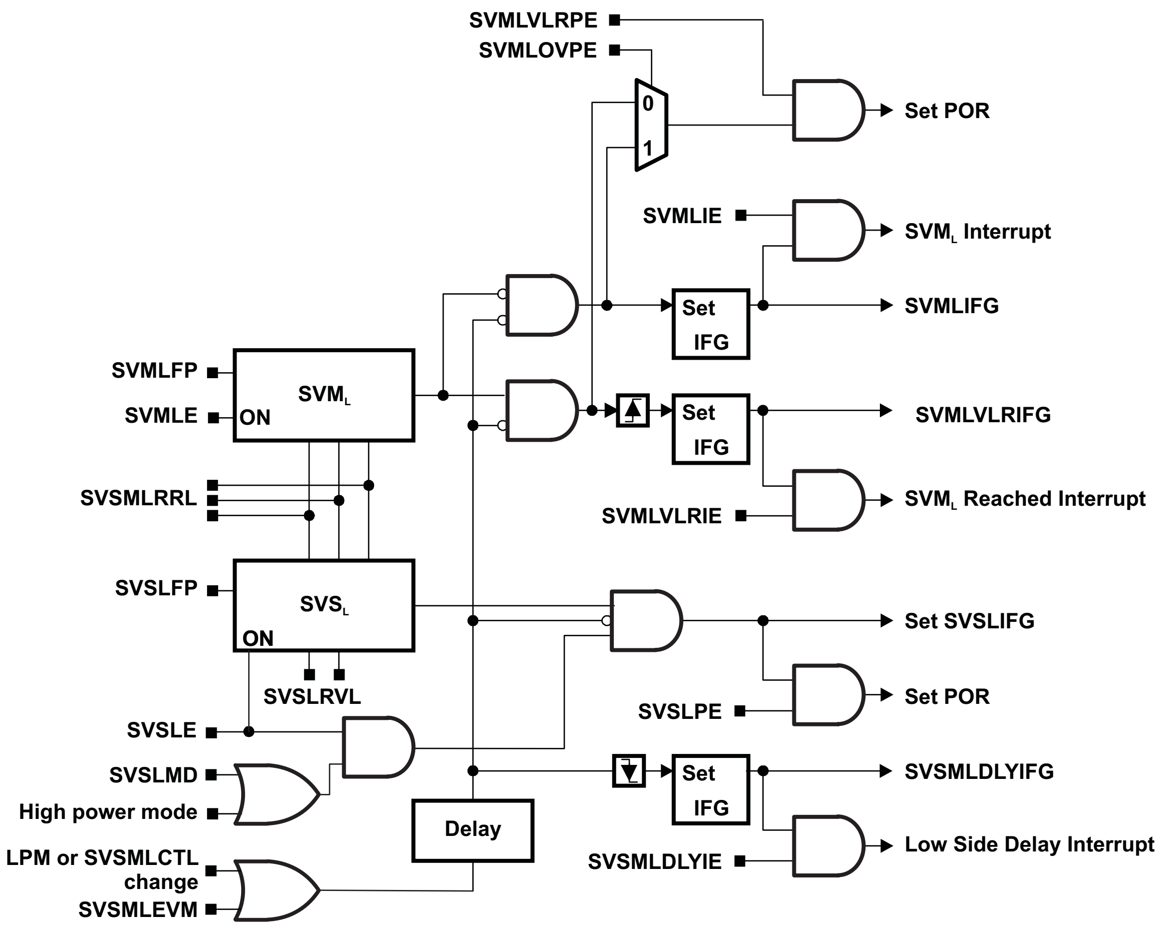


Figure 2-5. Low-Side SVS and SVM

If V_{CORE} falls below the SVS_L level, SVSLIFG (SVS_L interrupt flag) is set. If V_{CORE} remains below the SVS_L level and software attempts to clear SVSLIFG, it is immediately set again by hardware. If the SVSLPE (SVS_L POR enable) bit is set when SVSLIFG gets set, a POR is generated.

If V_{CORE} falls below the SVM_L level, SVMLIFG (SVM_L interrupt flag) is set. If V_{CORE} remains below the SVM_L level and software attempts to clear SVMLIFG, it is immediately set again by hardware. If the SVMLIE (SVM_L interrupt enable) bit is set when SVMLIFG gets set, an interrupt is generated. If a POR is desired when SVMLIFG is set, the SVM_L can be configured to do so by setting the SVMLVLRPE (SVM_L voltage level reached POR enable) bit while SVMLOVPE bit is cleared.

If V_{CORE} rises above the SVM_L level, the SVMLVLRIFG (SVM_L voltage level reached) interrupt flag is set. If SVMLVLRIE (SVM_L voltage level reached interrupt enable) is set when this occurs, an interrupt is also generated.

The SVM_L module can also be used for overvoltage detection. This is accomplished by setting the SVMLOVPE (SVM_L overvoltage POR enable) bit, in addition to setting SVMLVLRPE. Under these conditions, if V_{CORE} exceeds safe device operation, a POR is generated.

The SVS_L/SVM_L modules have configurable performance modes for power-saving operation. (See Section 2.2.8 for more information.) If these SVS_L/SVM_L power modes are modified, or if a voltage level is modified, a delay element masks the interrupts and POR sources until the SVS_L/SVM_L circuits have settled. When SVSMLDLYST (delay status) reads zero, the delay has expired. In addition, the SVSMLDLYIFG (SVS_L/SVM_L delay expired) interrupt flag is set. If the SVSMLDLYIE (SVS_L /SVM_L delay expired interrupt enable) is set when this occurs, an interrupt is also generated.

In case of power-fail conditions, setting SVSLMD will cause the SVS_L interrupt flag to be set in LPM2, LPM3, and LPM4. If SVSLMD is not set, the SVS_L interrupt flag will not be set in LPM2, LPM3, and LPM4. In addition, all SVS_L and SVM_L events can be masked by setting SVSMLEVM. For most applications, SVSMLEVM should be cleared.

All the interrupt flags of SVS_L /SVM_L remain set until cleared by a BOR or by software.

2.2.3 Supply Voltage Supervisor and Monitor - Power-Up

When the device is powering up, the SVS_H and SVS_L functions are enabled by default. Initially, DV_{CC} is low, and therefore the PMM holds the device in POR reset. Once both the SVS_H and SVS_L levels are met, the reset is released. Figure 2-6 shows this process.

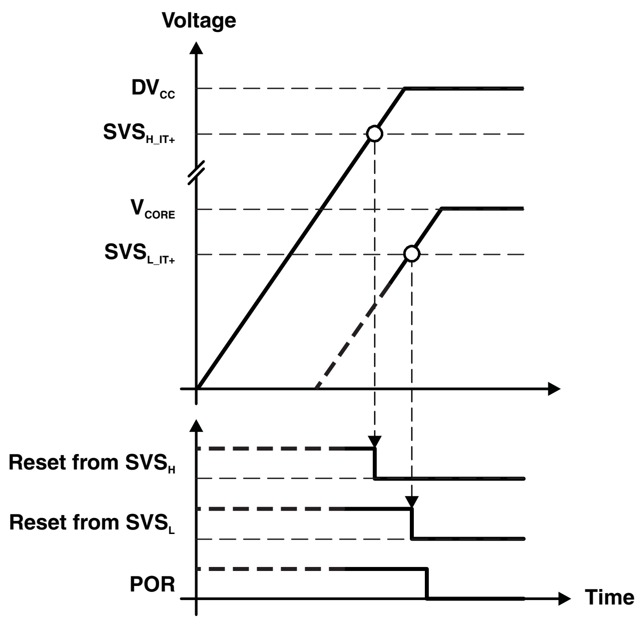


Figure 2-6. PMM Action at Device Power-Up

After this point, both voltage domains are supervised and monitored while the respective modules are enabled.

2.2.4 Increasing V_{CORE} to Support Higher MCLK Frequencies

With a reset, V_{CORE} and all the PMM thresholds, default to their lowest possible levels. These default settings allow a wide range of MCLK operation, and in many applications no change to these levels is required. However, if the application requires the performance provided by higher MCLK frequencies, software should ensure that V_{CORE} has been raised to a sufficient voltage level before changing MCLK, since failing to supply sufficient voltage to the CPU could produce unpredictable results. For a given device, minimum V_{CORE} levels required for maximum MCLK frequencies have been established (See the device data sheet for specific values).

After setting PMMCOREV to increase V_{CORE}, there is a time delay until the new voltage has been established. Software must not raise MCLK until the necessary core voltage has settled. SVM_L can be used to verify that V_{CORE} has met the required minimum value, prior to increasing MCLK. Figure 2-7 shows this procedure graphically.

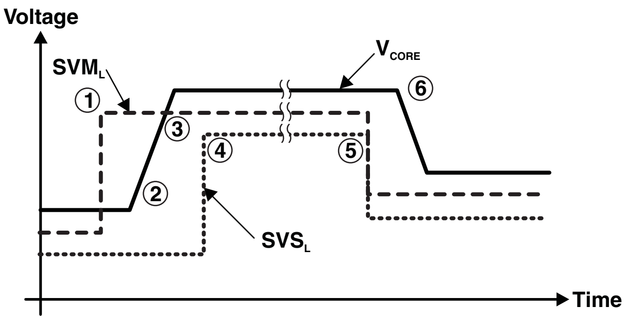


Figure 2-7. Changing V_{CORE} and SVM_L and SVS_L Levels

It is critical that the V_{CORE} level be increased by only one level at a time. The following steps 1 through 4 show the procedure to increase V_{CORE} by one level. This sequence is repeated to change the V_{CORE} level until the targeted level is obtained:

- Step 1: Program the SVM_H and SVS_H to the next level to ensure DV_{CC} is high enough for the next V_{CORE} level. Program the SVM_L to the next level and wait for (SVSMLDLYIFG) to be set.
- Step 2: Program PMMCOREV to the next V_{CORE} level.
- Step 3: Wait for the voltage level reached (SVMLVLRIFG) flag.
- Step 4: Program the SVS_L to the next level.

As a reference, the following is a C code example for increasing V_{CORE}. The sample libraries provide routines for increasing and decreasing the V_{CORE} and should be utilized whenever possible.

```
; C Code example for increasing core voltage.
; Note: Change core voltage one level at a time.


void SetVCoreUp (unsigned int level)
{
 // Open PMM registers for write access
 PMMCTL0_H = 0xA5;
 // Set SVS/SVM high side new level
 SVSMHCTL = SVSHE + SVSHRVL0 * level + SVMHE + SVSMHRRL0 * level;
 // Set SVM low side to new level
 SVSMLCTL = SVSLE + SVMLE + SVSMLRRL0 * level;
 // Wait till SVM is settled
 while ((PMMIFG & SVSMLDLYIFG) == 0);
 // Clear already set flags
 PMMIFG &= ~(SVMLVLRIFG + SVMLIFG);
 // Set VCore to new level
 PMMCTL0_L = PMMCOREV0 * level;
 // Wait till new level reached
 if ((PMMIFG & SVMLIFG))
   while ((PMMIFG & SVMLVLRIFG) == 0);
 // Set SVS/SVM low side to new level
 SVSMLCTL = SVSLE + SVSLRVL0 * level + SVMLE + SVSMLRRL0 * level;
 // Lock PMM registers for write access
 PMMCTL0_H = 0x00;
}
```

2.2.5 Decreasing V_{CORE} for Power Optimization

The risk posed by increasing MCLK frequency does not exist when decreasing MCLK from the current

V_{CORE} or higher settings, because higher V_{CORE} levels can still support MCLK frequencies below the ones for which they were intended. However, significant power efficiency gains can be made by operating V_{CORE} at the lowest value required for a given MCLK frequency. It is critical that the V_{CORE} level be decreased by only one level at a time. The following steps show the procedure to decrease V_{CORE} by one level. This sequence is repeated to change the V_{CORE} level until the targeted level is obtained:

Steps 5 through 6 show the procedure to decrease V_{CORE}:

- Step 5: Program the SVM_L and SVS_L to the new level and wait for (SVSMLDLYIFG) to be set.
- Step 6: Program PMMCOREV to the new V_{CORE} level. Wait for the voltage level reached (SVMLVLRIFG) interrupt.

It is critical when lowering the V_{CORE} setting that the maximum MCLK frequency for the new V_{CORE} setting is not violated (see the device-specific data sheet).

2.2.6 LPM3.5, LPM4.5

LPM3.5 and LMP4.5 are additional low-power modes in which the regulator of the PMM is completely disabled, providing additional power savings. Not all devices support all LPMx.5 modes, so refer to the device specific datasheet. Because there is no power supplied to VCORE during LPMx.5, the CPU and all digital modules including RAM are unpowered. This essentially disables the entire device and, as a result, the contents of the registers and RAM are lost. Any essential values should be stored to flash prior to entering LPMx.5. PMMREGOFF bit is used to disable the regulator. See the SYS module for complete descriptions and proper usages of LMPx.5.

SInce the regulator of the PMM is disabled upon entering LPMx.5, all I/O register configurations are lost. Because the I/O register configurations are lost, the configuration of I/O pins must be handled differently to ensure that all pins in the application behave in a controlled manner upon entering and exiting LPMx.5. Properly setting the I/O pins is critical to achieving the lowest possible power consumption in LPMx.5, as well as preventing any possible uncontrolled input or output I/O state in the application. The application has complete control of the I/O pin conditions preventing the possibility of unwanted spurious activity upon entry and exit from LPMx.5. The I/O pin state is held and locked based on the settings prior to LPMx.5 entry. Upon entry into LPMx.5, LOCKLPM5 residing in PM5CTL0 of the PMM module, is set automatically. Please note that only the pin condition is retained. All other port configuration register settings are lost. Please refer to the Digital I/O module for further details.

2.2.7 Brownout Reset (BOR), Software BOR, Software POR

The primary function of the brownout reset (BOR) circuit occurs when the device is powering up. It is functional very early in the power-up ramp, generating a POR that initializes the system. It also functions when no SVS is enabled and a brownout condition occurs. It sustains this reset until the input power is sufficient for the logic, for proper reset of the system.

In an application, it may be desired to cause a BOR via software. Setting PMMSWBOR will cause a software driven BOR. PMMBORIFG will be set accordingly. Please note that a BOR also initiates a POR and PUC. PMMBORIFG can be cleared by software or by reading SYSRSTIV. Similarly, it is possible to cause a POR via software by setting PMMSWPOR. PMMPORIFG will be set accordingly. A POR will also initiate a PUC. PMMPORIFG can be cleared by software or by reading SYSRSTIV. Both PMMSWBOR and PMMSWPOR are self clearing. Please refer to the SYS module for complete descriptions of BOR, POR, and PUC resets.

2.2.8 SVS/SVM Performance Modes and Wakeup Times

The supervisors/monitors can function in one of two power modes: normal and full performance. The difference is a tradeoff in response time versus the power consumed; full-performance mode has a faster response time but consumes considerably more power than normal mode. Full-performance mode might be considered in applications in which the decoupling of the external power supply cannot adequately prevent fast spikes on DV_{CC} from occurring, or when the application has a particular intolerance to failure. In such cases, full-performance mode provides an additional layer of protection.

There are two ways to control the performance mode: manual and automatic. In manual mode, the normal/full-performance selection is the same for every operational mode except LPMx.5 (the SVS/SVM are always disabled in LPMx.5). In this case, the normal/full-performance selection is made with the SVSHFP/SVMHFP/SVSLFP/SVMLFP bits, for their respective modules.

In automatic mode, hardware changes the normal/full-performance selection depending on the operational mode in effect. In automatic mode, the SVSHFP/SVMHFP/SVSLFP/SVMLFP select one of two automatic control schemes.

The selection of automatic or manual mode is by setting the SVSMHACE/SVSMLACE bits, which apply to the high-side and low-side, respectively. Table 2-5 and Table 2-6 show the selection of performance modes for SVS_L and SVM_L.

The wakeup time of the device from low power modes is also effected by the settings of the SVS_L and SVM_L performance modes. Table 2-7 and Table 2-8 show the selection of pefromance modes for SVS_H and SVM_H. The wakeup from low modes is not effected by the settings of the SVS_H and SVM_H performance modes.All wakeups from LPMx.5 (LPM3.5 or LPM4.5), are defined by the datasheet parametric, $t_{WAKE-UP-LPM5}$, regardless of the performance modes for SVS_L or SVM_L since these are disabled in LPMx.5.

Table 2-5. SVS_L Performance Control Modes

SVSLE	SVSLMD	SVSLFP	AM, LPM0, LPM1 SVS_L state	Manual mode SVSMLACE = 0	Automatic mode SVSMLACE = 1	Wakeup time LPM2, LPM3, LPM4
				LPM2, LPM3, LPM4 SVS_L state	LPM2, LPM3, LPM4 SVS_L state	
0	x	x	Off	Off	Off	$t_{WAKE-UP-FAST}$
1	0	0	Normal	Off	Off	$t_{WAKE-UP-SLOW}$
1	0	1	Full performance	Off	Off	$t_{WAKE-UP-FAST}$
1	1	0	Normal	Normal	Off	$t_{WAKE-UP-SLOW}$
1	1	1	Full performance	Full performance	Normal	$t_{WAKE-UP-FAST}$

Table 2-6. SVM_L Performance Control Modes

SVMLE	SVMLFP	AM, LPM0, LPM1 SVS_L state	Manual mode SVSMLACE = 0	Automatic mode SVSMLACE = 1	Wakeup time LPM2, LPM3, LPM4
			LPM2, LPM3, LPM4 SVS_L state	LPM2, LPM3, LPM4 SVS_L state	
0	x	Off	Off	Off	$t_{WAKE-UP-FAST}$
1	0	Normal	Normal	Off	$t_{WAKE-UP-SLOW}$
1	1	Full performance	Full performance	Normal	$t_{WAKE-UP-FAST}$

Table 2-7. SVS_H Performance Control Modes

SVSHE	SVSHMD	SVSHFP	AM, LPM0, LPM1 SVS_H state	Manual mode SVSMHACE = 0	Automatic mode SVSMHACE = 1
				LPM2, LPM3, LPM4 SVS_H state	LPM2, LPM3, LPM4 SVS_H state
0	x	x	Off	Off	Off
1	0	0	Normal	Off	Off
1	0	1	Full performance	Off	Off
1	1	0	Normal	Normal	Off
1	1	1	Full performance	Full performance	Normal

Table 2-8. SVM_H Performance Control Modes

SVMHE	SVMHFP	AM, LPM0, LPM1 SVS_H state	Manual mode SVSMHACE = 0	Automatic mode SVSMHACE = 1
			LPM2, LPM3, LPM4 SVS_H state	LPM2, LPM3, LPM4 SVS_H state
0	x	Off	Off	Off
1	0	Normal	Normal	Off
1	1	Full performance	Full performance	Normal

2.2.8.1 Wakeup Times in Debug Mode

The TEST/SWBTCK pin is used for interfacing to the development tools via Spy-Bi-Wire and JTAG. When the TEST/SBWTCK pin is high, wakeup times from LPM2, LPM3, and LPM4 may be different compared to when TEST/SBWTCK is low. When the TEST/SBWTCK pin is high, all delays associated with the SVS_L and SVM_L settings have no effect and the device will wakeup within $t_{WAKE\text{-}UP\text{-}FAST}$. Pay careful attention to the real-time behavior when exiting from LPM2, LPM3, and LPM4 with the device connected to a development tool (e.g. - MSP-FETU430IF).

2.2.9 PMM Interrupts

Interrupt flags generated by the PMM are routed to the system NMI interrupt vector generator register, SYSSNIV. When the PMM causes a reset, a value is generated in the system reset interrupt vector generator register, SYSRSTIV, corresponding to the source of the reset. These registers are defined within the SYS module. More information on the relationship between the PMM and SYS modules is available in the SYS chapter.

2.2.10 Port I/O Control

The PMM provides a means of ensuring that I/O pins cannot behave in uncontrolled fashion during an undervoltage event. During these times, outputs are disabled, both normal drive and the weak pullup/pulldown function. If the CPU is functioning normally, and then an undervoltage event occurs, any pin configured as an input has its PxIN register value locked in at the point the event occurs, until voltage is restored. During the undervoltage event, external voltage changes on the pin are not registered internally. This helps prevent erratic behavior from occurring.

2.2.11 Supply Voltage Monitor Output (SVMOUT, Optional)

The state of SVMLIFG, SVMLVLRIFG, SVMHIFG, and SVMLVLRIFG can be monitored on the external SVMOUT pin. Each of these interrupt flags can be enabled (SVMLOE, SVMLVLROE, SVMHOE, SVMLVLROE) to generate an output signal. The polarity of the output is selected by the SVMOUTPOL bit. If SVMOUTPOL is set, the output is set to 1 if an enabled interrupt flag is set.

2.3 PMM Registers

The PMM registers are listed in Table 2-9. The base address of the PMM module can be found in the device-specific data sheet. The address offset of each PMM register is given in Table 2-9. The password, PMMPW, defined in the PMMCTL0 register controls access to all PMM, SVS, and SVM registers. Once the correct password is written, the write access is enabled. The write access is disabled by writing a wrong password in byte mode to the PMMCTL0 upper byte. Word accesses to PMMCTL0 with a wrong password triggers a PUC. A write access to a register other than PMMCTL0 while write access is not enabled causes a PUC.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 2-9. PMM Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
PMM control register 0	PMMCTL0	Read/write	Word	00h	9600h
	PMMCTL0_L	Read/write	Byte	00h	00h
	PMMCTL0_H	Read/write	Byte	01h	96h
PMM control register 1	PMMCTL1	Read/write	Word	02h	0000h
	PMMCTL1_L	Read/write	Byte	02h	00h
	PMMCTL1_H	Read/write	Byte	03h	00h
SVS and SVM high side control register	SVSMHCTL	Read/write	Word	04h	4400h
	SVSMHCTL_L	Read/write	Byte	04h	00h
	SVSMHCTL_H	Read/write	Byte	05h	44h
SVS and SVM low side control register	SVSMLCTL	Read/write	Word	06h	4400h
	SVSMLCTL_L	Read/write	Byte	06h	00h
	SVSMLCTL_H	Read/write	Byte	07h	44h
SVSIN and SVMOUT control register (optional)	SVSMIO	Read/write	Word	08h	0020h
	SVSMIO_L	Read/write	Byte	08h	20h
	SVSMIO_H	Read/write	Byte	09h	00h
PMM interrupt flag register	PMMIFG	Read/write	Word	0Ah	0000h
	PMMIFG_L	Read/write	Byte	0Ah	00h
	PMMIFG_H	Read/write	Byte	0Bh	00h
PMM interrupt enable register	PMMRIE	Read/write	Word	0Eh	0000h
	PMMRIE_L	Read/write	Byte	0Eh	00h
	PMMRIE_H	Read/write	Byte	0Fh	00h
Power mode 5 control register 0	PM5CTL0	Read/write	Word	10h	0000h
	PM5CTL0_L	Read/write	Byte	10h	00h
	PM5CTL0_H	Read/write	Byte	11h	00h

Power Management Module Control Register 0 (PMMCTL0)

15	14	13	12	11	10	9	8
PMMPW, Read as 96h, Must be written as A5h							
rw-1	rw-0	rw-0	rw-1	rw-0	rw-1	rw-1	rw-0

7	6	5	4	3	2	1	0
Reserved	**Reserved**		**PMMREGOFF**	**PMMSWPOR**	**PMMSWBOR**	**PMMCOREV**	
rw-0	r-0	r-0	rw-0	rw-0	rw-0	rw-[0]	rw-[0]

PMMPW	Bits 15-8	PMM password. Always read as 096h. Must be written with 0A5h or a PUC is generated.
Reserved	Bit 7	Reserved. Must always be written with 0.
Reserved	Bits 6-5	Reserved. Always read 0.
PMMREGOFF	Bit 4	Regulator off (see SYS chapter for further details)
PMMSWPOR	Bit 3	Software power-on reset. Setting this bit to 1 triggers a POR. This bit is self clearing.
PMMSWBOR	Bit 2	Software brownout reset. Setting this bit to 1 triggers a BOR. This bit is self clearing.
PMMCOREV	Bits 1-0	Core voltage (see the device-specific data sheet for supported levels and corresponding voltages) 00 V_{CORE} level 0 01 V_{CORE} level 1 10 V_{CORE} level 2 11 V_{CORE} level 3

Power Management Module Control Register 1 (PMMCTL1)

15	14	13	12	11	10	9	8
Reserved							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

7	6	5	4	3	2	1	0
Reserved		**Reserved**		**Reserved**		**Reserved**	**Reserved**
r-0	r-0	rw-[0]	rw-[0]	r-0	r-0	rw-0	rw-0

Reserved	Bits 15-6	Reserved. Always read 0.
Reserved	Bits 5-4	Reserved. Must always be written with 0.
Reserved	Bits 3-2	Reserved. Always read 0.
Reserved	Bit 1	Reserved. Must always be written with 0.
Reserved	Bit 0	Reserved. Must always be written with 0.

Supply Voltage Supervisor and Monitor High-Side Control Register (SVSMHCTL)

15	14	13	12	11	10	9	8
SVMHFP	**SVMHE**	**Reserved**	**SVMHOVPE**	**SVSHFP**	**SVSHE**	**SVSHRVL**	
rw-[0]	rw-1	r-0	rw-[0]	rw-[0]	rw-1	rw-[0]	rw-[0]

7	6	5	4	3	2	1	0
SVSMHACE	**SVSMHEVM**	**Reserved**	**SVSHMD**	**SVSMHDLYST**	**SVSMHRRL**		
rw-[0]	rw-0	r-0	rw-0	r-0	rw-[0]	rw-[0]	rw-[0]

SVMHFP Bit 15 SVM high-side full-performance mode. If this bit is set, the SVM_H operates in full-performance mode.
- 0 Normal mode. See the device-specific data sheet for response times.
- 1 Full-performance mode. See the device-specific data sheet for response times.

SVMHE Bit 14 SVM high-side enable. If this bit is set, the SVM_H is enabled.

Reserved Bit 13 Reserved. Always read 0.

SVMHOVPE Bit 12 SVM high-side overvoltage enable. If this bit is set, the SVM_H overvoltage detection is enabled. If SVMHVLRPE is also set, a POR occurs on an overvoltage condition.

SVSHFP Bit 11 SVS high-side full-performance mode. If this bit is set, the SVS_H operates in full-performance mode.
- 0 Normal mode. See the device-specific data sheet for response times.
- 1 Full-performance mode. See the device-specific data sheet for response times.

SVSHE Bit 10 SVS high-side enable. If this bit is set, the SVS_H is enabled.

SVSHRVL Bits 9-8 SVS high-side reset voltage level. If DV_{CC} falls short of the SVS_H voltage level selected by SVSHRVL, a reset is triggered (if SVSHPE = 1). The voltage levels are defined in the device-specific data sheet.

SVSMHACE Bit 7 SVS and SVM high-side automatic control enable. If this bit is set, the low-power mode of the SVS_H and SVM_H circuits is under hardware control.

SVSMHEVM Bit 6 SVS and SVM high-side event mask. If this bit is set, the SVS_H and SVM_H events are masked.
- 0 No events are masked.
- 1 All events are masked.

Reserved Bit 5 Reserved. Always read 0.

SVSHMD Bit 4 SVS high-side mode. If this bit is set, the SVS_H interrupt flag is set in LPM2, LPM3, and LPM4 in case of power-fail conditions. If this bit is not set, the SVS_H interrupt is not set in LPM2, LPM3, and LPM4.

SVSMHDLYST Bit 3 SVS and SVM high-side delay status. If this bit is set, the SVS_H and SVM_H events are masked for some delay time. The delay time depends on the power mode of the SVS_H and SVM_H. If SVMHFP = 1 and SVSHFP = 1 i.e. full-performance mode the delay is shorter. See the device-specific data sheet for details. The bit is cleared by hardware if the delay has expired.

SVSMHRRL Bits 2-0 SVS and SVM high-side reset release voltage level. These bits define the reset release voltage level of the SVS_H. It is also used for the SVM_H to define the voltage reached level. The voltage levels are defined in the device-specific data sheet.

Supply Voltage Supervisor and Monitor Low-Side Control Register (SVSMLCTL)

15	14	13	12	11	10	9	8
SVMLFP	SVMLE	Reserved	SVMLOVPE	SVSLFP	SVSLE	SVSLRVL	
rw-[0]	rw-1	r-0	rw-[0]	rw-[0]	rw-1	rw-[0]	rw-[0]

7	6	5	4	3	2	1	0
SVSMLACE	SVSMLEVM	Reserved	SVSLMD	SVSMLDLYST	SVSMLRRL		
rw-[0]	rw-0	r-0	rw-0	r-0	rw-[0]	rw-[0]	rw-[0]

SVMLFP	Bit 15	SVM low-side full-performance mode. If this bit is set, the SVM_L operates in full-performance mode. 0 Normal mode. See the device-specific data sheet for response times. 1 Full-performance mode. See the device-specific data sheet for response times.
SVMLE	Bit 14	SVM low-side enable. If this bit is set, the SVM_L is enabled.
Reserved	Bit 13	Reserved. Always read 0.
SVMLOVPE	Bit 12	SVM low-side overvoltage enable. If this bit is set, the SVM_L overvoltage detection is enabled.
SVSLFP	Bit 11	SVS low-side full-performance mode. If this bit is set, the SVS_L operates in full-performance mode. 0 Normal mode. See the device-specific data sheet for response times. 1 Full-performance mode. See the device-specific data sheet for response times.
SVSLE	Bit 10	SVS low-side enable. If this bit is set, the SVS_L is enabled.
SVSLRVL	Bits 9-8	SVS low-side reset voltage level. If V_{CORE} falls short of the SVS_L voltage level selected by SVSLRVL, a reset is triggered (if SVSLPE = 1).
SVSMLACE	Bit 7	SVS and SVM low-side automatic control enable. If this bit is set, the low-power mode of the SVS_L and SVM_L circuits is under hardware control.
SVSMLEVM	Bit 6	SVS and SVM low-side event mask. If this bit is set, the SVS_L and SVM_L events are masked. 0 No events are masked. 1 All events are masked.
Reserved	Bit 5	Reserved. Always read 0.
SVSLMD	Bit 4	SVS low-side mode. If this bit is set, the SVS_L interrupt flag is set in LPM2, LPM3 and LPM4 in case of power-fail conditions. If this bit is not set, the SVS_L interrupt is not set in LPM2, LPM3, and LPM4.
SVSMLDLYST	Bit 3	SVS and SVM low-side delay status. If this bit is set, the SVS_L and SVM_L events are masked for some delay time. The delay time depends on the power mode of the SVS_L and SVM_L. If SVMLFP = 1 and SVSLFP = 1 i.e. full-performance mode, it is shorter. The bit is cleared by hardware if the delay has expired.
SVSMLRRL	Bits 2-0	SVS and SVM low-side reset release voltage level. These bits define the reset release voltage level of the SVS_L. It is also used for the SVM_L to define the voltage reached level.

SVSIN and SVMOUT Control Register (SVSMIO)

15	14	13	12	11	10	9	8
Reserved			SVMHVLROE	SVMHOE	Reserved		
r-0	r-0	r-0	rw-[0]	rw-[0]	r-0	r-0	r-0

7	6	5	4	3	2	1	0
Reserved		SVMOUTPOL	SVMLVLROE	SVMLOE	Reserved		
r-0	r-0	rw-[1]	rw-[0]	rw-[0]	r-0	r-0	r-0

Reserved	Bits 15-13	Reserved. Always read 0.
SVMHVLROE	Bit 12	SVM high-side voltage level reached output enable. If this bit is set, the SVMHVLRIFG bit is output to the device SVMOUT pin. The device-specific port logic has to be configured accordingly.
SVMHOE	Bit 11	SVM high-side output enable. If this bit is set, the SVMHIFG bit is output to the device SVMOUT pin. The device-specific port logic has to be configured accordingly.
Reserved	Bits 10-6	Reserved. Always read 0.
SVMOUTPOL	Bit 5	SVMOUT pin polarity. If this bit is set, SVMOUT is active high. An error condition is signaled by a 1 at SVMOUT. If SVMOUTPOL is cleared, the error condition is signaled by a 0 at the SVMOUT pin.
SVMLVLROE	Bit 4	SVM low-side voltage level reached output enable. If this bit is set, the SVMLVLRIFG bit is output to the device SVMOUT pin. The device-specific port logic has to be configured accordingly.
SVMLOE	Bit 3	SVM low-side output enable. If this bit is set, the SVMLIFG bit is output to the device SVMOUT pin. The device-specific port logic has to be configured accordingly.
Reserved	Bits 2-0	Reserved. Always read 0.

Power Management Module Interrupt Flag Register (PMMIFG)

15	14	13	12	11	10	9	8
PMMLPM5IFG	**Reserved**	**SVSLIFG**[1]	**SVSHIFG**[1]	**Reserved**	**PMMPORIFG**	**PMMRSTIFG**	**PMMBORIFG**
rw-[0]	r-0	rw-[0]	rw-[0]	r-0	rw-[0]	rw-[0]	rw-[0]

7	6	5	4	3	2	1	0
Reserved	**SVMHVLRIFG**[1]	**SVMHIFG**	**SVSMHDLYIFG**	**Reserved**	**SVMLVLRIFG**[1]	**SVMLIFG**	**SVSMLDLYIFG**
r-0	rw-[0]	rw-[0]	rw-0	r-0	rw-[0]	rw-[0]	rw-0

[1] After power up, the reset value depends on the power sequence.

PMMLPM5IFG	Bit 15	LPMx.5 flag. This bit is set if the system was in LPMx.5 before. The bit is cleared by software or by reading the reset vector word. A power failure on the DV_{CC} domain clears the bit. 0 No interrupt pending 1 Interrupt pending
Reserved	Bit 14	Reserved. Always read 0.
SVSLIFG	Bit 13	SVS low-side interrupt flag. The bit is cleared by software or by reading the reset vector word. 0 No interrupt pending 1 Interrupt pending
SVSHIFG	Bit 12	SVS high-side interrupt flag. The bit is cleared by software or by reading the reset vector word. 0 No interrupt pending 1 Interrupt pending
Reserved	Bit 11	Reserved. Always read 0.
PMMPORIFG	Bit 10	PMM software power-on reset interrupt flag. This interrupt flag is set if a software POR is triggered. The bit is cleared by software or by reading the reset vector word, SYSRSTIV. 0 No interrupt pending 1 Interrupt pending
PMMRSTIFG	Bit 9	PMM reset pin interrupt flag. This interrupt flag is set if the $\overline{\text{RST}}$/NMI pin is the reset source. The bit is cleared by software or by reading the reset vector word. 0 No interrupt pending 1 Interrupt pending
PMMBORIFG	Bit 8	PMM software brownout reset interrupt flag. This interrupt flag is set if a software BOR (PMMSWBOR) is triggered. The bit is cleared by software or by reading the reset vector word, SYSRSTIV. 0 No interrupt pending 1 Interrupt pending
Reserved	Bit 7	Reserved. Always read 0.
SVMHVLRIFG	Bit 6	SVM high-side voltage level reached interrupt flag. The bit is cleared by software or by reading the reset vector (SVSHPE = 1) word or by reading the interrupt vector (SVSHPE = 0) word. 0 No interrupt pending 1 Interrupt pending
SVMHIFG	Bit 5	SVM high-side interrupt flag. The bit is cleared by software. 0 No interrupt pending 1 Interrupt pending
SVSMHDLYIFG	Bit 4	SVS and SVM high-side delay expired interrupt flag. This interrupt flag is set if the delay element expired. The bit is cleared by software or by reading the interrupt vector word. 0 No interrupt pending 1 Interrupt pending
Reserved	Bit 3	Reserved. Always read 0.
SVMLVLRIFG	Bit 2	SVM low-side voltage level reached interrupt flag. The bit is cleared by software or by reading the reset vector (SVSLPE = 1) word or by reading the interrupt vector (SVSLPE = 0) word. 0 No interrupt pending 1 Interrupt pending

(continued)

SVMLIFG	Bit 1	SVM low-side interrupt flag. The bit is cleared by software. 0 No interrupt pending 1 Interrupt pending
SVSMLDLYIFG	Bit 0	SVS and SVM low-side delay expired interrupt flag. This interrupt flag is set if the delay element expired. The bit is cleared by software or by reading the interrupt vector word. 0 No interrupt pending 1 Interrupt pending

Power Management Module Reset and Interrupt Enable Register (PMMRIE)

15	14	13	12	11	10	9	8
Reserved		SVMHVLRPE	SVSHPE	Reserved		SVMLVLRPE	SVSLPE
r-0	r-0	rw-[0]	rw-[1]	r-0	r-0	rw-[0]	rw-[1]

7	6	5	4	3	2	1	0
Reserved	SVMHVLRIE	SVMHIE	SVSMHDLYIE	Reserved	SVMLVLRIE	SVMLIE	SVSMLDLYIE
r-0	rw-0	rw-0	rw-0	r-0	rw-0	rw-0	rw-0

Reserved	Bits 15-14	Reserved. Always read 0.
SVMHVLRPE	Bit 13	SVM high-side voltage level reached power-on reset enable. If this bit is set, exceeding the SVM_H voltage level triggers a POR.
SVSHPE	Bit 12	SVS high-side power-on reset enable. If this bit is set, falling below the SVS_H voltage level triggers a POR.
Reserved	Bits 11-10	Reserved. Always read 0.
SVMLVLRPE	Bit 9	SVM low-side voltage level reached power-on reset enable. If this bit is set, exceeding the SVM_L voltage level triggers a POR.
SVSLPE	Bit 8	SVS low-side power-on reset enable. If this bit is set, falling below the SVS_L voltage level triggers a POR.
Reserved	Bit 7	Reserved. Always read 0.
SVMHVLRIE	Bit 6	SVM high-side reset voltage level interrupt enable
SVMHIE	Bit 5	SVM high-side interrupt enable. This bit is cleared by software or if the interrupt vector word is read.
SVSMHDLYIE	Bit 4	SVS and SVM high-side delay expired interrupt enable
Reserved	Bit 3	Reserved. Always read 0.
SVMLVLRIE	Bit 2	SVM low-side reset voltage level interrupt enable
SVMLIE	Bit 1	SVM low-side interrupt enable. This bit is cleared by software or if the interrupt vector word is read.
SVSMLDLYIE	Bit 0	SVS and SVM low-side delay expired interrupt enable

Power Mode 5 Control Register 0 (PM5CTL0)

15	14	13	12	11	10	9	8
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	LOCKLPM5
r0	r0	r0	r0	r0	r0	r0	rw-[0]

Reserved	Bits 15-1	Reserved. Always read as zero.
LOCKLPM5	Bit 0	Lock I/O pin configuration upon entry/exit to/from LPMx.5. Once power is applied to the device, this bit, once set, can only be cleared by the user or via another power cycle. 0 I/O pin configuration is not locked and defaults to its reset condition. 1 I/O pin configuration remains locked. Pin state is held during LPMx.5 entry and exit.

Chapter 3

SLAU208F–June 2008–Revised March 2010

Unified Clock System (UCS)

The Unified Clock System (UCS) module provides the various clocks for a device. This chapter describes the operation of the UCS module, which is implemented in all devices.

Topic **Page**

3.1	**Unified Clock System (UCS) Introduction**	**78**
3.2	**UCS Operation**	**80**
3.3	**Module Oscillator (MODOSC)**	**90**
3.4	**UCS Module Registers**	**91**

3.1 Unified Clock System (UCS) Introduction

The UCS module supports low system cost and ultralow power consumption. Using three internal clock signals, the user can select the best balance of performance and low power consumption. The UCS module can be configured to operate without any external components, with one or two external crystals, or with resonators, under full software control.

The UCS module includes up to five clock sources:

- XT1CLK: Low-frequency/high-frequency oscillator that can be used either with low-frequency 32768 Hz watch crystals, standard crystals, resonators, or external clock sources in the 4 MHz to 32 MHz range. XT1CLK can be used as a clock reference into the FLL. Some devices only support the low frequency oscillator for XT1CLK. See the device-specific data sheet for supported functions.
- VLOCLK: Internal very low power, low frequency oscillator with 10 kHz typical frequency
- REFOCLK: Internal, trimmed, low-frequency oscillator with 32768 Hz typical frequency, with the ability to be used as a clock reference into the FLL
- DCOCLK: Internal digitally-controlled oscillator (DCO) that can be stabilized by the FLL
- XT2CLK: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 4 MHz to 32 MHz range. XT1CLK can be used as a clock reference into the FLL.

Three clock signals are available from the UCS module:

- ACLK: Auxiliary clock. The ACLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. DCOCLKDIV is the DCOCLK frequency divided by 1, 2, 4, 8, 16, or 32 within the FLL block. ACLK can be divided by 1, 2, 4, 8, 16, or 32. ACLK/n is ACLK divided by 1, 2, 4, 8, 16, or 32 and is available externally at a pin. ACLK is software selectable by individual peripheral modules.
- MCLK: Master clock. MCLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. DCOCLKDIV is the DCOCLK frequency divided by 1, 2, 4, 8, 16, or 32 within the FLL block. MCLK can be divided by 1, 2, 4, 8, 16, or 32. MCLK is used by the CPU and system.
- SMCLK: Subsystem master clock. SMCLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. DCOCLKDIV is the DCOCLK frequency divided by 1, 2, 4, 8, 16, or 32 within the FLL block. SMCLK can be divided by 1, 2, 4, 8, 16, or 32. SMCLK is software selectable by individual peripheral modules.

The block diagram of the UCS module is shown in Figure 3-1.

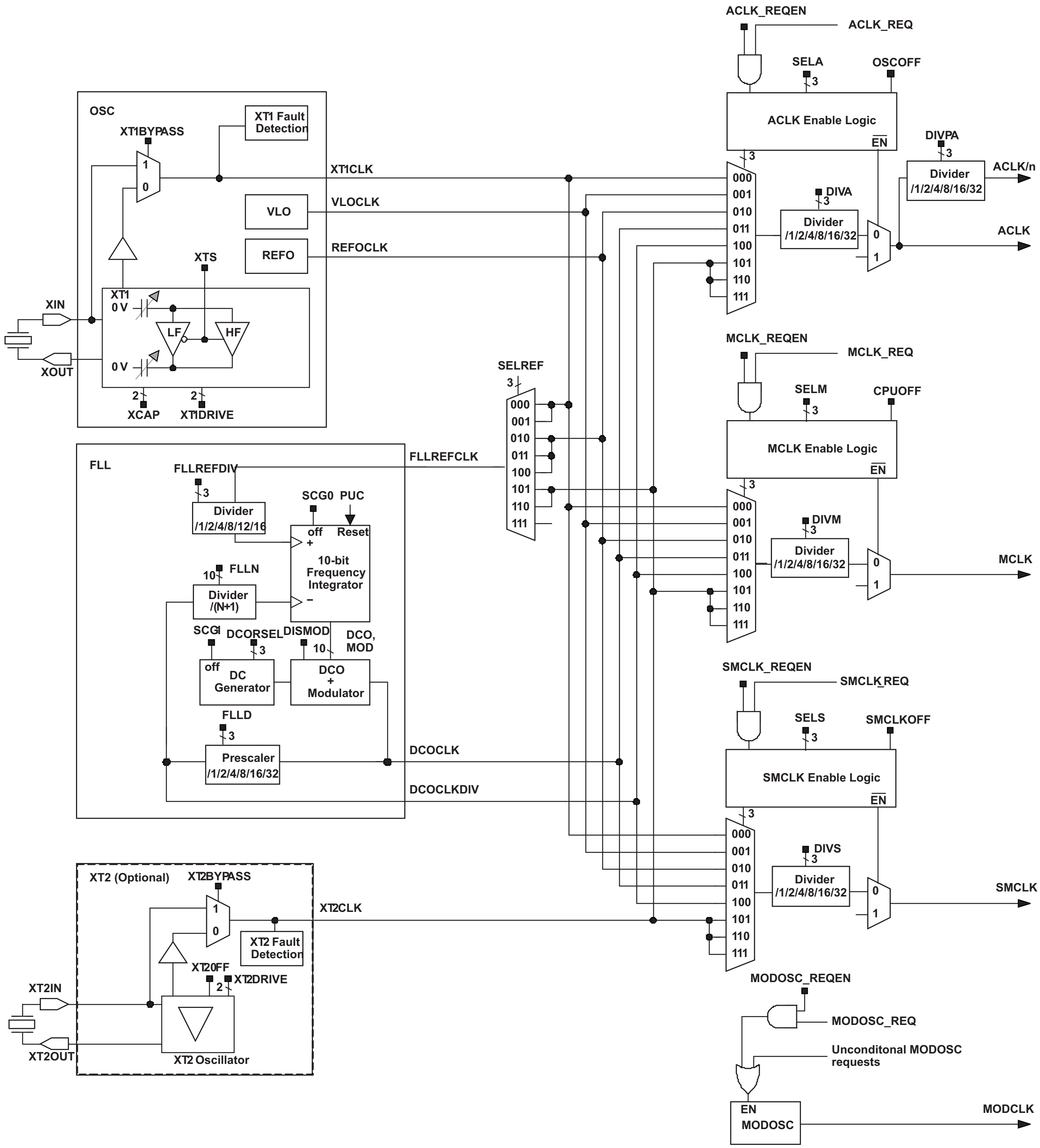


Figure 3-1. UCS Block Diagram

3.2 UCS Operation

After a PUC, the UCS module default configuration is:

- XT1 in LF mode is selected as the oscillator source for XT1CLK. XT1CLK is selected for ACLK.
- DCOCLKDIV is selected for MCLK.
- DCOCLKDIV is selected for SMCLK.
- FLL operation is enabled and XT1CLK is selected as the FLL reference clock, FLLREFCLK.
- XIN and XOUT pins are set to general-purpose I/Os and XT1 remains disabled until the I/O ports are configured for XT1 operation.
- When available, XT2IN and XT2OUT pins are set to general-purpose I/Os and XT2 is disabled.

As previously stated, FLL operation with XT1 is selected by default, but XT1 is disabled. The crystal pins (XIN, XOUT) are shared with general-purpose I/Os. To enable XT1, the PSEL bits associated with the crystal pins must be set. When a 32,768 Hz crystal is used for XT1CLK, the fault control logic immediately causes ACLK to be sourced by the REFOCLK, because XT1 is not stable immediately (see Section 3.2.12). Once crystal startup is obtained and settled, the FLL stabilizes MCLK and SMCLK to 1.048576 MHz and f_{DCO} = 2.097152 MHz.

Status register control bits (SCG0, SCG1, OSCOFF, and CPUOFF) configure the MSP430 operating modes and enable or disable portions of the UCS module (see System Resets, Interrupts, and Operating Modes chapter). Registers UCSCTL0 through UCSCTL8, configure the UCS module.

The UCS module can be configured or reconfigured by software at any time during program execution.

3.2.1 UCS Module Features for Low-Power Applications

Conflicting requirements typically exist in battery-powered applications:

- Low clock frequency for energy conservation and time keeping
- High clock frequency for fast response times and fast burst processing capabilities
- Clock stability over operating temperature and supply voltage
- Low-cost applications with less-constrained clock accuracy requirements

The UCS module addresses these conflicting requirements by allowing the user to select from the three available clock signals: ACLK, MCLK, and SMCLK.

All three available clock signals can be sourced via any of the available clock sources (XT1CLK, VLOCLK, REFOCLK, DCOCLK, DCOCLKDIV, or XT2CLK), giving complete flexibility in the system clock configuration. A flexible clock distribution and divider system is provided to fine tune the individual clock requirements.

3.2.2 Internal Very-Low-Power Low-Frequency Oscillator (VLO)

The internal VLO provides a typical frequency of 10 kHz (see device-specific data sheet for parameters) without requiring a crystal. The VLO provides for a low-cost ultralow-power clock source for applications that do not require an accurate time base.

The VLO is enabled when it is used to source ACLK, MCLK, or SMCLK (SELA = {1} or SELM = {1} or SELS = {1}).

3.2.3 Internal Trimmed Low-Frequency Reference Oscillator (REFO)

The internal trimmed low-frequency REFO can be used for cost-sensitive applications where a crystal is not required or desired. REFO is internally trimmed to 32.768 kHz typical and provides for a stable reference frequency that can be used as FLLREFCLK. REFO, combined with the FLL, provides for a flexible range of system clock settings without the need for a crystal. REFO consumes no power when not being used.

REFO is enabled under any of the following conditions:

- REFO is a source for ACLK (SELA = {2}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- REFO is a source for MCLK (SELM = {2}) and in active mode (AM) (CPUOFF = 0)
- REFO is a source for SMCLK (SELS = {2}) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- REFO is a source for FLLREFCLK (SELREF = {2}) and the DCO is a source for ACLK (SELA = {3,4}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- REFO is a source for FLLREFCLK (SELREF = {2}) and the DCO is a source for MCLK (SELM = {3,4}) and in active mode (AM) (CPUOFF = 0)
- REFO is a source for FLLREFCLK (SELREF = {2}) and the DCO is a source for SMCLK (SELS = {3,4}) and in active mode (AM) through LPM1 (SMCLKOFF = 0)

NOTE: REFO Enable for MSP430F543x, MSP430F541x devices

REFO is enabled under any of the following conditions:

- REFO is a source for ACLK (SELA = {2}), MCLK (SELM = {2}), or SMCLK (SELS = {2}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- REFO is a source for FLLREFCLK (SELREF = {2}) and the DCO is a source for ACLK, MCLK, or SMCLK (SELA = {3,4}), MCLK (SELM = {3,4}), or SMCLK (SELS = {3,4}) and in active mode (AM) through LPM3 (OSCOFF = 0)

3.2.4 XT1 Oscillator

The XT1 oscillator supports ultralow-current consumption using a 32,768 Hz watch crystal in low-frequency (LF) mode (XTS = 0) . A watch crystal connects to XIN and XOUT without any other external components. The software-selectable XCAP bits configure the internally provided load capacitance for the XT1 crystal in LF mode. This capacitance can be selected as 2 pF, 6 pF, 9 pF, or 12 pF (typical). Additional external capacitors can be added if necessary.

On some devices, the XT1 oscillator also supports high-speed crystals or resonators when in high-frequency (HF) mode (XTS = 1). The high-speed crystal or resonator connects to XIN and XOUT and requires external capacitors on both terminals. These capacitors should be sized according to the crystal or resonator specifications.

The drive settings of XT1 in LF mode can be increased with the XT1DRIVE bits. At power up, the XT1 starts with the highest drive settings for fast, reliable startup. If needed, user software can reduce the drive strength to further reduce power. In HF mode, different crystal or resonator ranges are supported by choosing the proper XT1DRIVE settings .

XT1 may be used with an external clock signal on the XIN pin in either LF or HF mode by setting XT1BYPASS. When used with an external signal, the external frequency must meet the data sheet parameters for the chosen mode. XT1 is powered down when used in bypass mode.

The XT1 pins are shared with general-purpose I/O ports. At power up, the default operation is XT1, LF mode of operation . However, XT1 remains disabled until the ports shared with XT1 are configured for XT1 operation. The configuration of the shared I/O is determined by the PSEL bit associated with XIN and the XT1BYPASS bit. Setting the PSEL bit causes the XIN and XOUT ports to be configured for XT1 operation. If XT1BYPASS is also set, XT1 is configured for bypass mode of operation, and the oscillator associated with XT1 is powered down. In bypass mode of operation, XIN can accept an external clock input signal and XOUT is configured as a general-purpose I/O. The PSEL bit associated with XOUT is a don't care.

If the PSEL bit associated with XIN is cleared, both XIN and XOUT ports are configured as general-purpose I/Os, and XT1 is disabled.

XT1 is enabled under any of the following conditions:

- XT1 is a source for ACLK (SELA = {0}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT1 is a source for MCLK (SELM = {0}) and in active mode (AM) (CPUOFF = 0)
- XT1 is a source for SMCLK (SELS = {0}) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- XT1 is a source for FLLREFCLK (SELREF = {0}) and the DCO is a source for ACLK (SELA = {3,4}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT1 is a source for FLLREFCLK (SELREF = {0}) and the DCO is a source for MCLK (SELM = {3,4}) and in active mode (AM) (CPUOFF = 0)
- XT1 is a source for FLLREFCLK (SELREF = {0}) and the DCO is a source for SMCLK (SELS = {3,4}) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- XT1OFF = 0. XT1 enabled in active mode (AM) through LPM4.

NOTE: XT1 Enable for MSP430F543x, MSP430F541x devices

XT1 is enabled under any of the following conditions:

- XT1 is a source for ACLK, MCLK, or SMCLK (SELA = {0}), MCLK (SELM = {0}), or SMCLK (SELS = {0}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT1 is a source for FLLREFCLK (SELREF = {0}) and the DCO is a source for ACLK, MCLK, or SMCLK (SELA = {3,4}), MCLK (SELM = {3,4}), or SMCLK (SELS = {3,4}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT1OFF = 0. XT1 enabled in active mode (AM) through LPM4.

3.2.5 XT2 Oscillator

Some devices have a second crystal oscillator, XT2. XT2 sources XT2CLK, and its characteristics are identical to XT1 in HF mode. The XT2DRIVE bits select the frequency range of operation of XT2.

XT2 may be used with external clock signals on the XT2IN pin by setting XT2BYPASS. When used with an external signal, the external frequency must meet the data-sheet parameters for XT2. XT2 is powered down when used in bypass mode.

The XT2 pins are shared with general-purpose I/O ports. At power up, the default operation is XT2. However, XT2 remains disabled until the ports shared with XT2 are configured for XT2 operation. The configuration of the shared I/O is determined by the PSEL bit associated with XT2IN and the XT2BYPASS bit. Setting the PSEL bit causes the XT2IN and XT2OUT ports to be configured for XT2 operation. If XT2BYPASS is also set, XT2 is configured for bypass mode of operation, and the oscillator associated with XT2 is powered down. In bypass mode of operation, XT2IN can accept an external clock input signal and XT2OUT is configured as a general-purpose I/O. The PSEL bit associated with XT2OUT is a don't care.

If the PSEL bit associated with XT2IN is cleared, both XT2IN and XT2OUT ports are configured as general-purpose I/Os, and XT2 is disabled.

XT2 is enabled under any of the following conditions:

- XT2 is a source for ACLK (SELA = {5,6,7}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT2 is a source for MCLK (SELM = {5,6,7}) and in active mode (AM) (CPUOFF = 0)
- XT2 is a source for SMCLK (SELS = {5,6,7}) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- XT2 is a source for FLLREFCLK (SELREF = {5,6}) and the DCO is a source for ACLK (SELA = {3,4}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT2 is a source for FLLREFCLK (SELREF = {5,6}) and the DCO is a source for MCLK (SELM = {3,4}) and in active mode (AM) (CPUOFF = 0)
- XT2 is a source for FLLREFCLK (SELREF = {5,6}) and the DCO is a source for SMCLK (SELS = {3,4}) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- XT2OFF = 0. XT2 enabled in active mode (AM) through LPM4.

NOTE: XT2 Enable for MSP430F543x, MSP430F541x devices

XT2 is enabled under any of the following conditions:

- XT2 is a source for ACLK, MCLK, or SMCLK (SELA = {5,6,7}), MCLK (SELM = {5,6,7}), or SMCLK (SELS = {5,6,7}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT2 is a source for FLLREFCLK (SELREF = {5,6,7}) and the DCO is a source for ACLK, MCLK, or SMCLK (SELA = {3,4}), MCLK (SELM = {3,4}), or SMCLK (SELS = {3,4}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT2OFF = 0. XT1 enabled in active mode (AM) through LPM4.

3.2.6 Digitally-Controlled Oscillator (DCO)

The DCO is an integrated digitally controlled oscillator. The DCO frequency can be adjusted by software using the DCORSEL, DCO, and MOD bits. The DCO frequency can be optionally stabilized by the FLL to a multiple frequency of FLLREFCLK/n. The FLL can accept different reference sources selectable via the SELREF bits. Reference sources include XT1CLK, REFOCLK, or XT2CLK (if available) . The value of n is defined by the FLLREFDIV bits (n = 1, 2, 4, 8, 12, or 16). The default is n = 1. There may be scenarios, where FLL operation is not required or desired, therefore no FLLREFCLK is necessary. This can be accomplished by setting SELREF = {7}.

NOTE: For the 'F543x and 'F541x non-A versions only: Setting SELREF = {7} sets XT2CLK as the FLL reference clock.

The FLLD bits configure the FLL prescaler divider value D to 1, 2, 4, 8, 16, or 32. By default, D = 2, and MCLK and SMCLK are sourced from DCOCLKDIV, providing a clock frequency DCOCLK/2.

The divider (N + 1) and the divider value D define the DCOCLK and DCOCLKDIV frequencies, where N > 0. Writing N = 0 causes the divider to be set to 2.

$f_{DCOCLK} = D \times (N + 1) \times (f_{FLLREFCLK} \div n)$

$f_{DCOCLKDIV} = (N + 1) \times (f_{FLLREFCLK} \div n)$

3.2.6.1 Adjusting DCO Frequency

By default, FLL operation is enabled. FLL operation can be disabled by setting SCG0 or SCG1. Once disabled, the DCO continues to operate at the current settings defined in UCSCTL0 and UCSCTL1. The DCO frequency can be adjusted manually if desired. Otherwise, the DCO frequency is stabilized by the FLL operation.

After a PUC, DCORSEL = {2} and DCO = {0}. MCLK and SMCLK are sourced from DCOCLKDIV. Because the CPU executes code from MCLK, which is sourced from the fast-starting DCO, code execution begins from PUC in less than 5 µs.

The frequency of DCOCLK is set by the following functions:

- The three DCORSEL bits select one of eight nominal frequency ranges for the DCO. These ranges are defined for an individual device in the device-specific data sheet.
- The five DCO bits divide the DCO range selected by the DCORSEL bits into 32 frequency steps, separated by approximately 8%.
- The five MOD bits switch between the frequency selected by the DCO bits and the next-higher frequency set by {DCO + 1}. When DCO = {31}, the MOD bits have no effect, because the DCO is already at the highest setting for the selected DCORSEL range.

3.2.7 Frequency Locked Loop (FLL)

The FLL continuously counts up or down a frequency integrator. The output of the frequency integrator that drives the DCO can be read in UCSCTL0, UCSCTL1 (bits MOD and DCO). The count is adjusted +1 with the frequency $f_{FLLREFCLK}$/n (n = 1, 2, 4, 8, 12, or 16) or –1 with the frequency f_{DCOCLK}/[D × (N+1)].

NOTE: Reading MOD and DCO bits

The integrator is updated via the DCOCLK, which may differ in frequency of operation of MCLK. It is possible that immediate reads of a previously written value are not visible to the user since the update to the integrator has not occurred. This is normal. Once the integrator is updated at the next successive DCOCLK, the correct value can be read.

In addition, since the MCLK can be asynchronous to the integrator updates, reading the values may be cause a corrupted value to be read under this condition. In this case, a majority vote method should be performed.

Five of the integrator bits (UCSCTL0 bits 12 to 8) set the DCO frequency tap. Thirty-two taps are implemented for the DCO, and each is approximately 8% higher than the previous. The modulator mixes two adjacent DCO frequencies to produce fractional taps.

For a given DCO bias range setting, time must be allowed for the DCO to settle on the proper tap for normal operation. (n × 32) $f_{FLLREFCLK}$ cycles are required between taps requiring a worst case of (n × 32 × 32) $f_{FLLREFCLK}$ cycles for the DCO to settle. The value n is defined by the FLLREFDIV bits (n = 1, 2, 4, 8, 12, or 16).

3.2.8 DCO Modulator

The modulator mixes two DCO frequencies, f_{DCO} and f_{DCO}+1 to produce an intermediate effective frequency between f_{DCO} and f_{DCO}+1 and spread the clock energy, reducing electromagnetic interference (EMI). The modulator mixes f_{DCO} and f_{DCO}+1 for 32 DCOCLK clock cycles and is configured with the MOD bits. When MOD = {0}, the modulator is off.

The modulator mixing formula is:

$$t = (32 - MOD) \times t_{DCO} + MOD \times t_{DCO+1}$$

Figure 3-2 shows the modulator operation.

When FLL operation is enabled, the modulator settings and DCO are controlled by the FLL hardware. If FLL operation is not desired, the modulator settings and DCO control can be configured with software.

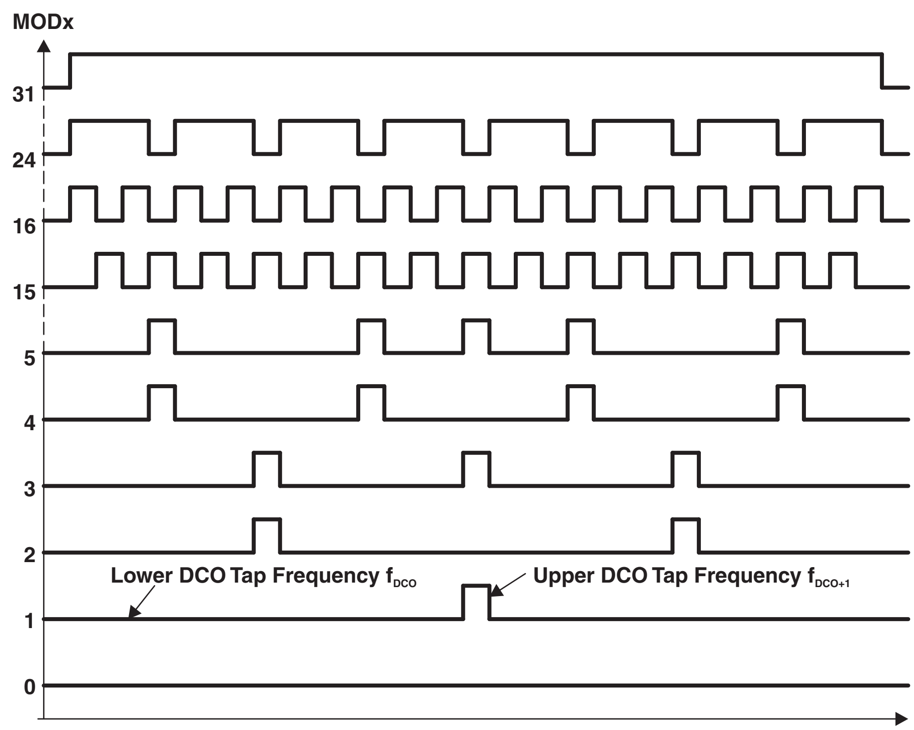


Figure 3-2. Modulator Patterns

3.2.9 Disabling FLL Hardware and Modulator

The FLL is disabled when the status register bits SCG0 or SCG1 are set. When the FLL is disabled, the DCO runs at the previously selected tap and DCOCLK is not automatically stabilized.

The DCO modulator is disabled when DISMOD is set. When the DCO modulator is disabled, the DCOCLK is adjusted to the DCO tap selected by the DCO bits.

NOTE: DCO operation without FLL

When the FLL operation is disabled, the DCO continues to operate at the current settings. Because it is not stabilized by the FLL, temperature and voltage variations influence the frequency of operation. See the device-specific data sheet for voltage and temperature coefficients to ensure reliable operation.

3.2.10 FLL Operation From Low-Power Modes

An interrupt service request clears SCG1, CPUOFF, and OSCOFF if set, but does not clear SCG0. This means that for FLL operation from within an interrupt service routine entered from LPM1, 2, 3, or 4, the FLL remains disabled and the DCO operates at the previous setting as defined in UCSCTL0 and UCSCTL1. SCG0 can be cleared by user software if FLL operation is required.

3.2.11 Operation From Low-Power Modes, Requested by Peripheral Modules

A peripheral module requests its clock sources automatically from the UCS module if required for its proper operation, regardless of the current mode of operation, as shown in Figure 3-3.

A peripheral module asserts one of three possible clock request signals based on its control bits: ACLK_REQ, MCLK_REQ, or SMCLK_REQ. These request signals are based on the configuration and clock selection of the respective module. For example, if a timer selects ACLK as its clock source and the timer is enabled, the timer generates an ACLK_REQ signal to the UCS system. The UCS, in turn, enables ACLK regardless of the LPM settings.

Any clock request from a peripheral module causes its respective clock off signal to be overridden, but does not change the setting of clock off control bit. For example, a peripheral module may require ACLK that is currently disabled by the OSCOFF bit (OSCOFF = 1). The module can request ACLK by generating an ACLK_REQ. This causes the OSCOFF bit to have no effect, thereby allowing ACLK to be available to the requesting peripheral module. The OSCOFF bit remains at its current setting (OSCOFF = 1).

If the requested source is not active, the software NMI handler must take care of the required actions. For the previous example, if ACLK was sourced by XT1 and XT1 was not enabled, an oscillator fault condition will occur and the software must handle the event. The watchdog, due to its security requirement, actively selects the VLOCLK source if the originally selected clock source is not available.

Due to the clock request feature, care must be taken in the application when entering low power modes to save power. Although the device enters the selected low-power mode, a clock request may exhibit more current consumption than the specified values in the data sheet.

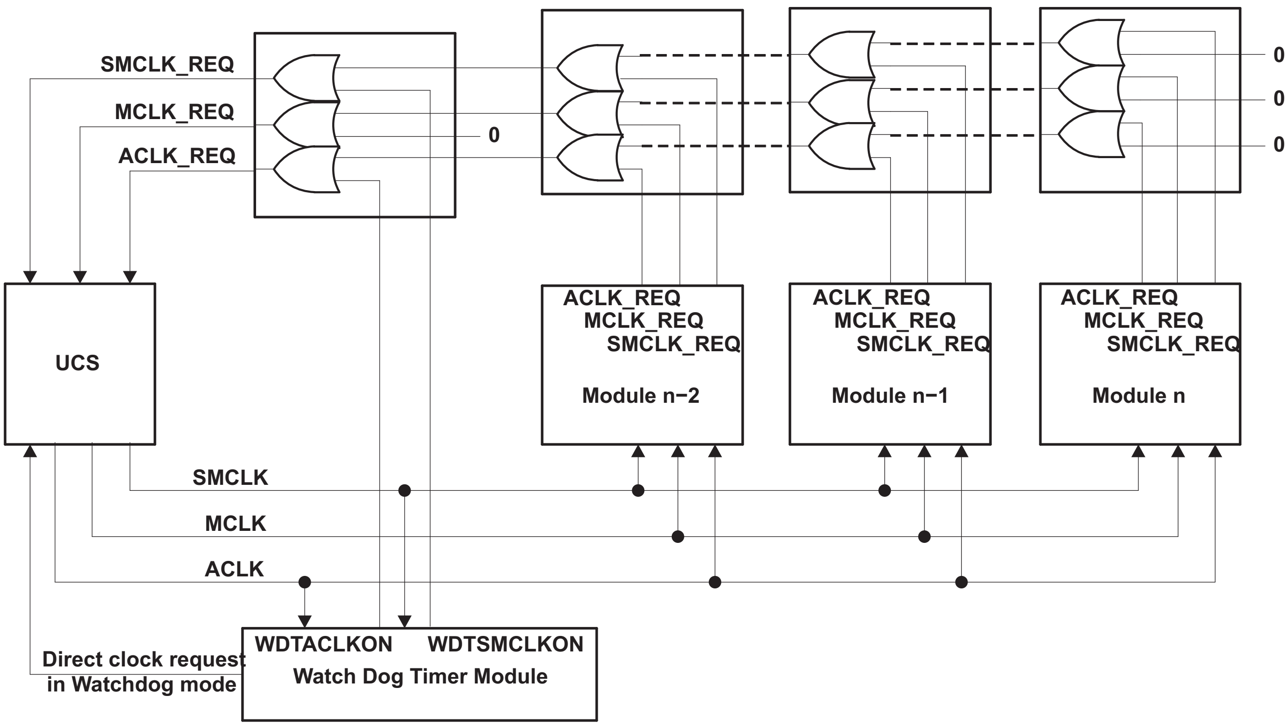


Figure 3-3. Module Request Clock System

By default, the clock request logic is enabled. The clock request logic can be disabled by clearing ACLKREQEN, MCLKREQEN, or SMCLKREQEN, for each respective system clock. When ACLKREQEN or MCLKREQEN bits are set, or active, the clock is available to the system and will prevent entry into a low power mode until all modules requesting the clock are disabled. When ACLKREQEN or MCLKREQEN bits are cleared, or disabled, the clock is always halted as defined by the low power modes. The SMCLKREQEN logic behaves similarly, but is also influenced by the SMCLKOFF bit in the UCSCTL6 register.Table 3-1 shows the relationship between the system clocks and the low power modes in conjunction with the clock request logic.

Table 3-1. Clock Request System and Power Modes

Mode	ACLK		MCLK		SMCLK			
					SMCLKOFF = 0		SMCLKOFF = 1	
	ACLKREQEN = 0	ACLKREQEN = 1	MCLKREQEN = 0	MCLKREQEN = 1	SMCLKREQEN = 0	SMCLKREQEN = 1	SMCLKREQEN = 0	SMCLKREQEN = 1
AM	Active	Active	Active	Active	Active	Active	Disabled	Active
LPM0	Active	Active	Disabled	Active	Active	Active	Disabled	Active
LPM1	Active	Active	Disabled	Active	Active	Active	Disabled	Active
LPM2	Active	Active	Disabled	Active	Disabled	Active	Disabled	Active
LPM3	Active	Active	Disabled	Active	Disabled	Active	Disabled	Active
LPM4	Disabled	Active	Disabled	Active	Disabled	Active	Disabled	Active
LPM3.5 (1)	Disabled	Active	Disabled	Active	Disabled	Active	Disabled	Active
LPM4.5 (1)	Disabled	Active	Disabled	Active	Disabled	Active	Disabled	Active

(1) Any clock request prior to entry into LPM3.5 or LPM4.5, will cause the respective system clock to remain active. In these cases, LPM3.5 or LPM4.5 mode is not entered.

3.2.12 UCS Module Fail-Safe Operation

The UCS module incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault for XT1, DCO, and XT2 as shown in Figure 3-4. The available fault conditions are:

- Low-frequency oscillator fault (XT1LFOFFG) for XT1 in LF mode
- High-frequency oscillator fault (XT1HFOFFG) for XT1 in HF mode
- High-frequency oscillator fault (XT2OFFG) for XT2
- DCO fault flag (DCOFFG) for the DCO

The crystal oscillator fault bits XT1LFOFFG, XT1HFOFFG, and XT2OFFG are set if the corresponding crystal oscillator is turned on and not operating properly. Once set, the fault bits remain set until reset in software, regardless if the fault condition no longer exists. If the user clears the fault bits and the fault condition still exists, the fault bits are automatically set, otherwise they remain cleared.

When using XT1 operation in LF mode as the reference source into the FLL (SELREF = {0}), a crystal fault automatically causes the FLL reference source, FLLREFCLK, to be sourced by the REFO. XT1LFOFFG is set. When using XT1 operation in HF mode as the reference source into the FLL, a crystal fault causes no FLLREFCLK signal to be generated and the FLL continues to count down to zero in an attempt to lock FLLREFCLK and DCOCLK/[D × (N + 1)]. The DCO tap moves to the lowest position (DCO are cleared) and the DCOFFG is set. DCOFFG is also set if the N-multiplier value is set too high for the selected DCO frequency range, resulting in the DCO tap moving to the highest position (UCSCTL0.12 to UCSCTL0.8 are set). The DCOFFG remains set until cleared by the user. If the user clears the DCOFFG and the fault condition remains, it is automatically set, otherwise it remains cleared. XT1HFOFFG is set.

When using XT2 as the reference source into the FLL, a crystal fault causes no FLLREFCLK signal to be generated, and the FLL continues to count down to zero in an attempt to lock FLLREFCLK and DCOCLK/[D × (N + 1)]. The DCO tap moves to the lowest position (DCO are cleared) and the DCOFFG is set. DCOFFG is also set if the N-multiplier value is set too high for the selected DCO frequency range, resulting in the DCO tap moving to the highest position (UCSCTL0.12 to UCSCTL0.8 are set). The DCOFFG remains set until cleared by the user. If the user clears the DCOFFG and the fault condition remains, it is automatically set, otherwise it remains cleared. XT2OFFG is set.

The OFIFG oscillator-fault interrupt flag is set and latched at POR or when any oscillator fault (XT1LFOFFG, XT1HFOFFG, XT2OFFG, or DCOFFG) is detected. When OFIFG is set and OFIE is set, the OFIFG requests an NMI. When the interrupt is granted, the OFIE is not reset automatically as it is in previous MSP430 families. It is no longer required to reset the OFIE. NMI entry/exit circuitry removes this requirement. The OFIFG flag must be cleared by software. The source of the fault can be identified by checking the individual fault bits.

If a fault is detected for the oscillator sourcing MCLK, MCLK is automatically switched to the DCO for its clock source (DCOCLKDIV) for all clock sources except XT1 LF mode. If MCLK is sourced from XT1 in LF mode, an oscillator fault causes MCLK to be automatically switched to the REFO for its clock source (REFOCLK). This does not change the SELM bit settings. This condition must be handled by user software.

If a fault is detected for the oscillator sourcing SMCLK, SMCLK is automatically switched to the DCO for its clock source (DCOCLKDIV) for all clock sources except XT1 LF mode. If SMCLK is sourced from XT1 in LF mode, an oscillator fault causes SMCLK to be automatically switched to the REFO for its clock source (REFOCLK). This does not change the SELS bit settings. This condition must be handled by user software.

If a fault is detected for the oscillator sourcing ACLK, ACLK is automatically switched to the DCO for its clock source (DCOCLKDIV) for all clock sources except XT1 LF mode. If ACLK is sourced from XT1 in LF mode, an oscillator fault causes ACLK to be automatically switched to the REFO for its clock source (REFOCLK). This does not change the SELA bit settings. This condition must be handled by user software.

NOTE: DCO active during oscillator fault

DCOCLKDIV is active even at the lowest DCO tap. The clock signal is available for the CPU to execute code and service an NMI during an oscillator fault.

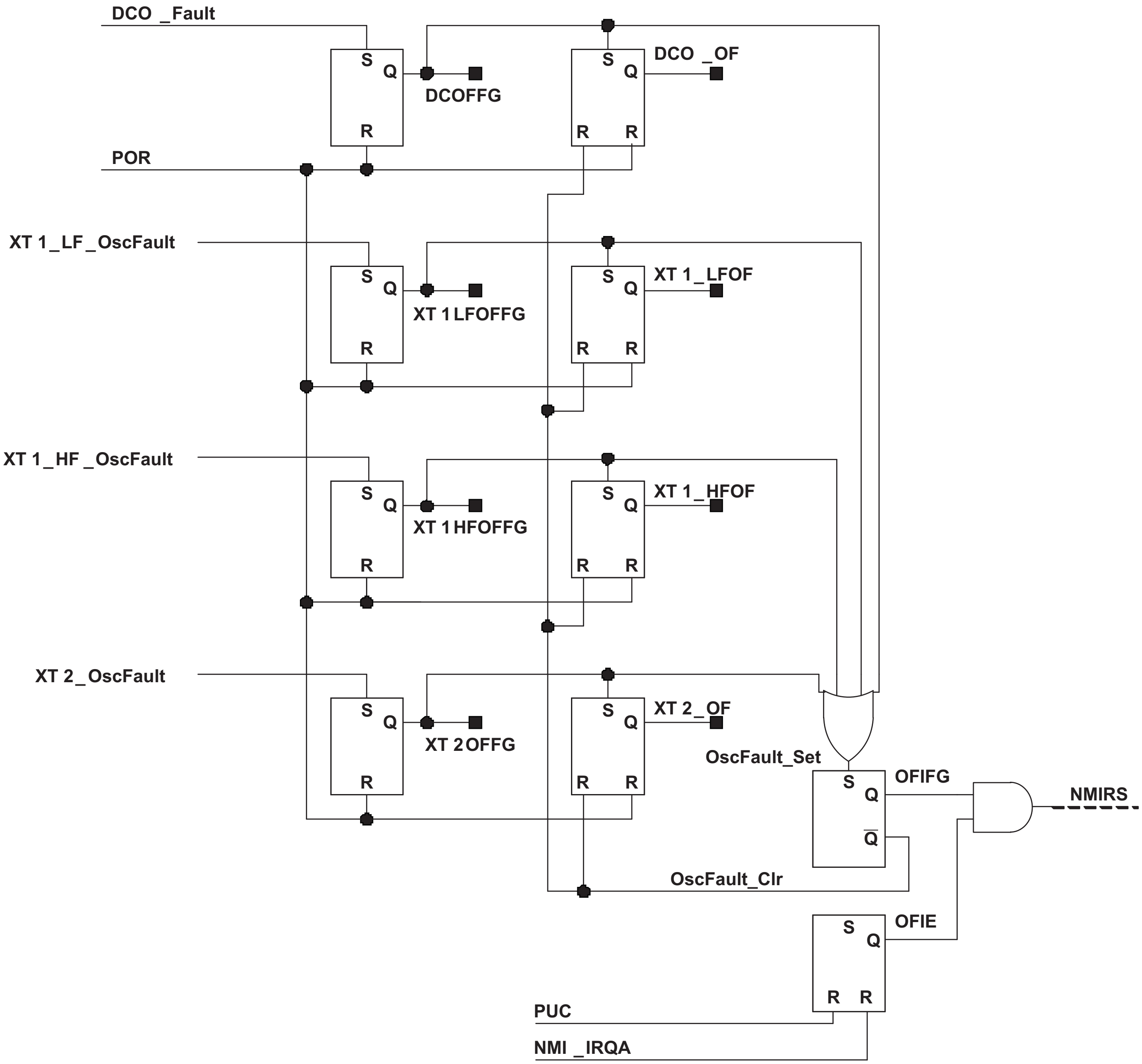


Figure 3-4. Oscillator Fault Logic

NOTE: Fault conditions

DCO_Fault: DCOFFG is set if DCO bits in UCSCTL0 register value equals {0} or {31}.

XT1_LF_OscFault: This signal is set after the XT1 (LF mode) oscillator has stopped operation and cleared after operation resumes. The fault condition causes XT1LFOFFG to be set and remain set. If the user clears XT1LFOFFG and the fault condition still exists, XT1LFOFFG remains set.

XT1_HF_OscFault: This signal is set after the XT1 (HF mode) oscillator has stopped operation and cleared after operation resumes. The fault condition causes XT1HFOFFG to be set and remain set. If the user clears XT1HFOFFG and the fault condition still exists, XT1HFOFFG remains set.

XT2_OscFault: This signal is set after the XT2 oscillator has stopped operation and cleared after operation resumes. The fault condition causes XT2OFFG to be set and remain set. If the user clears XT2OFFG and the fault condition still exists, XT2OFFG remains set.

NOTE: Fault logic

Please note that as long as a fault condition still exists, the OFIFG remains set. The application must take special care when clearing the OFIFG signal. If no fault condition remains when the OFIFG signal is cleared, the clock logic switches back to the original user settings prior to the fault condition.

NOTE: Fault logic counters

Each crystal oscillator circuit has hardware counters. These counters are reset each time a fault condition occurs on its respective oscillator, causing the fault flag to be set. The counters will begin to count after the fault condition is removed. Once the maximum count is reached, the fault flag is removed.

In XT1 LF mode, the maximum count is 8192. In XT1 HF mode (and XT2 when available), the maximum count is 1024. In bypass modes, regardless of LF or HF settings, the maximum count is 8192.

3.2.13 Synchronization of Clock Signals

When switching MCLK or SMCLK from one clock source to the another, the switch is synchronized to avoid critical race conditions as shown in Figure 3-5:

- The current clock cycle continues until the next rising edge.
- The clock remains high until the next rising edge of the new clock.
- The new clock source is selected and continues with a full high period.

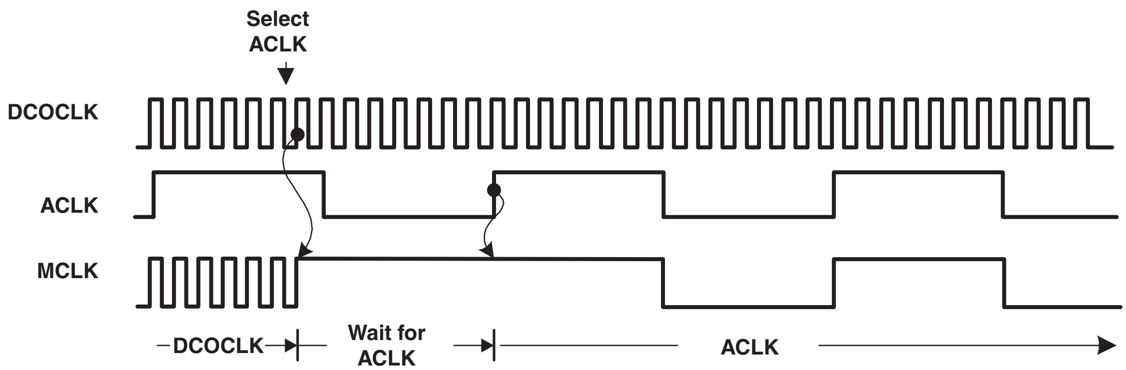


Figure 3-5. Switch MCLK from DCOCLK to XT1CLK

3.3 Module Oscillator (MODOSC)

The UCS module also supports an internal oscillator, MODOSC, that is used by the flash memory controller module and, optionally, by other modules in the system. The MODOSC sources MODCLK.

3.3.1 MODOSC Operation

To conserve power, MODOSC is powered down when not needed and enabled only when required. When the MODOSC source is required, the respective module requests it. MODOSC is enabled based on unconditional and conditional requests. Setting MODOSCREQEN enables conditional requests. Unconditional requests are always enabled. It is not necessary to set MODOSCREQEN for modules that utilize unconditional requests; e.g., flash controller, ADC12_A.

The flash memory controller only requires MODCLK when performing write or erase operations. When performing such operations, the flash memory controller issues an unconditional request for the MODOSC source. Upon doing so, the MODOSC source is enabled, if not already enabled from other modules' previous requests.

The ADC12_A may optionally use MODOSC as a clock source for its conversion clock. The user chooses the ADC12OSC as the conversion clock source. During a conversion, the ADC12_A module issues an unconditional request for the ADC12OSC clock source. Upon doing so, the MODOSC source is enabled, if not already enabled from other modules' previous requests.

3.4 UCS Module Registers

The UCS module registers are listed in Table 3-2. The base address can be found in the device-specific data sheet. The address offset is listed in Table 3-2.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 3-2. Unified Clock System Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Unified Clock System Control 0	UCSCTL0	Read/write	Word	00h	0000h
	UCSCTL0_L	Read/write	Byte	00h	00h
	UCSCTL0_H	Read/write	Byte	01h	00h
Unified Clock System Control 1	UCSCTL1	Read/write	Word	02h	0020h
	UCSCTL1_L	Read/write	Byte	02h	20h
	UCSCTL1_H	Read/write	Byte	03h	00h
Unified Clock System Control 2	UCSCTL2	Read/write	Word	04h	101Fh
	UCSCTL2_L	Read/write	Byte	04h	1Fh
	UCSCTL2_H	Read/write	Byte	05h	10h
Unified Clock System Control 3	UCSCTL3	Read/write	Word	06h	0000h
	UCSCTL3_L	Read/write	Byte	06h	00h
	UCSCTL3_H	Read/write	Byte	07h	00h
Unified Clock System Control 4	UCSCTL4	Read/write	Word	08h	0044h
	UCSCTL4_L	Read/write	Byte	08h	44h
	UCSCTL4_H	Read/write	Byte	09h	00h
Unified Clock System Control 5	UCSCTL5	Read/write	Word	0Ah	0000h
	UCSCTL5_L	Read/write	Byte	0Ah	00h
	UCSCTL5_H	Read/write	Byte	0Bh	00h
Unified Clock System Control 6	UCSCTL6	Read/write	Word	0Ch	C1CDh
	UCSCTL6_L	Read/write	Byte	0Ch	CDh
	UCSCTL6_H	Read/write	Byte	0Dh	C1h
Unified Clock System Control 7	UCSCTL7	Read/write	Word	0Eh	0703h
	UCSCTL7_L	Read/write	Byte	0Eh	03h
	UCSCTL7_H	Read/write	Byte	0Fh	07h
Unified Clock System Control 8	UCSCTL8	Read/write	Word	10h	0707h
	UCSCTL8_L	Read/write	Byte	10h	07h
	UCSCTL8_H	Read/write	Byte	11h	07h

Unified Clock System Control 0 Register (UCSCTL0)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved			**DCO**				
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
MOD					**Reserved**		
rw-0	rw-0	rw-0	rw-0	rw-0	r0	r0	r0

Reserved	Bits 15-13	Reserved. Reads back as 0.
DCO	Bits 12-8	DCO tap selection. These bits select the DCO tap and are modified automatically during FLL operation.
MOD	Bits 7-3	Modulation bit counter. These bits select the modulation pattern. All MOD bits are modified automatically during FLL operation. The DCO register value is incremented when the modulation bit counter rolls over from 31 to 0. If the modulation bit counter decrements from 0 to the maximum count, the DCO register value is also decremented.
Reserved	Bits 2-0	Reserved. Reads back as 0.

Unified Clock System Control 1 Register (UCSCTL1)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	**DCORSEL**			**Reserved**		**Reserved**	**DISMOD**
r0	rw-0	rw-1	rw-0	r0	r0	rw-0	rw-0

Reserved	Bits 15-8	Reserved. Reads back as 0.
Reserved	Bit 7	Reserved. Reads back as 0.
DCORSEL	Bits 6-4	DCO frequency range select. These bits select the DCO frequency range of operation.
Reserved	Bits 3-2	Reserved. Reads back as 0.
Reserved	Bit 1	Reserved. Reads back as 0.
DISMOD	Bit 0	Modulation. This bit enables/disables the modulation. 0 Modulation enabled 1 Modulation disabled

Unified Clock System Control 2 Register (UCSCTL2)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	**FLLD**			**Reserved**		**FLLN**	
r0	rw-0	rw-0	rw-1	r0	r0	rw-0	rw-0

7	6	5	4	3	2	1	0
FLLN							
rw-0	rw-0	rw-0	rw-1	rw-1	rw-1	rw-1	rw-1

Reserved Bit 15 Reserved. Reads back as 0.

FLLD Bits 14-12 FLL loop divider. These bits divide f_{DCOCLK} in the FLL feedback loop. This results in an additional multiplier for the multiplier bits. See also multiplier bits.

- 000 $f_{DCOCLK}/1$
- 001 $f_{DCOCLK}/2$
- 010 $f_{DCOCLK}/4$
- 011 $f_{DCOCLK}/8$
- 100 $f_{DCOCLK}/16$
- 101 $f_{DCOCLK}/32$
- 110 Reserved for future use. Defaults to $f_{DCOCLK}/32$.
- 111 Reserved for future use. Defaults to $f_{DCOCLK}/32$.

Reserved Bits 11-10 Reserved. Reads back as 0.

FLLN Bits 9-0 Multiplier bits. These bits set the multiplier value N of the DCO. N must be greater than 0. Writing zero to FLLN causes N to be set to 1.

Unified Clock System Control 3 Register (UCSCTL3)

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	SELREF			Reserved	FLLREFDIV		
r0	rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0

Reserved Bits 15-8 Reserved. Reads back as 0.

Reserved Bit 7 Reserved. Reads back as 0.

SELREF Bits 6-4 FLL reference select. These bits select the FLL reference clock source.

- 000 XT1CLK
- 001 Reserved for future use. Defaults to XT1CLK.
- 010 REFOCLK
- 011 Reserved for future use. Defaults to REFOCLK.
- 100 Reserved for future use. Defaults to REFOCLK.
- 101 XT2CLK when available, otherwise REFOCLK.
- 110 Reserved for future use. XT2CLK when available, otherwise REFOCLK.
- 111 No selection. For the 'F543x and 'F541x non-A versions only, this defaults to XT2CLK. Reserved for future use. XT2CLK when available, otherwise REFOCLK.

Reserved Bit 3 Reserved. Reads back as 0.

FLLREFDIV Bits 2-0 FLL reference divider. These bits define the divide factor for $f_{FLLREFCLK}$. The divided frequency is used as the FLL reference frequency.

- 000 $f_{FLLREFCLK}/1$
- 001 $f_{FLLREFCLK}/2$
- 010 $f_{FLLREFCLK}/4$
- 011 $f_{FLLREFCLK}/8$
- 100 $f_{FLLREFCLK}/12$
- 101 $f_{FLLREFCLK}/16$
- 110 Reserved for future use. Defaults to $f_{FLLREFCLK}/16$.
- 111 Reserved for future use. Defaults to $f_{FLLREFCLK}/16$.

Unified Clock System Control 4 Register (UCSCTL4)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved					SELA		
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved	SELS			Reserved	SELM		
r0	rw-1	rw-0	rw-0	r0	rw-1	rw-0	rw-0

Reserved	Bits 15-11	Reserved. Reads back as 0.
SELA	Bits 10-8	Selects the ACLK source 000 XT1CLK 001 VLOCLK 010 REFOCLK 011 DCOCLK 100 DCOCLKDIV 101 XT2CLK when available, otherwise DCOCLKDIV 110 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV. 111 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.
Reserved	Bit 7	Reserved. Reads back as 0.
SELS	Bits 6-4	Selects the SMCLK source 000 XT1CLK 001 VLOCLK 010 REFOCLK 011 DCOCLK 100 DCOCLKDIV 101 XT2CLK when available, otherwise DCOCLKDIV 110 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV. 111 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.
Reserved	Bit 3	Reserved. Reads back as 0.
SELM	Bits 2-0	Selects the MCLK source 000 XT1CLK 001 VLOCLK 010 REFOCLK 011 DCOCLK 100 DCOCLKDIV 101 XT2CLK when available, otherwise DCOCLKDIV 110 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV. 111 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.

Unified Clock System Control 5 Register (UCSCTL5)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	**DIVPA**			**Reserved**	**DIVA**		
r0	rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved	**DIVS**			**Reserved**	**DIVM**		
r0	rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0

Reserved Bit 15 Reserved. Reads back as 0.

DIVPA Bits 14-12 ACLK source divider available at external pin. Divides the frequency of ACLK and presents it to an external pin.

- 000 $f_{ACLK}/1$
- 001 $f_{ACLK}/2$
- 010 $f_{ACLK}/4$
- 011 $f_{ACLK}/8$
- 100 $f_{ACLK}/16$
- 101 $f_{ACLK}/32$
- 110 Reserved for future use. Defaults to $f_{ACLK}/32$.
- 111 Reserved for future use. Defaults to $f_{ACLK}/32$.

Reserved Bit 11 Reserved. Reads back as 0.

DIVA Bits 10-8 ACLK source divider. Divides the frequency of the ACLK clock source.

- 000 $f_{ACLK}/1$
- 001 $f_{ACLK}/2$
- 010 $f_{ACLK}/4$
- 011 $f_{ACLK}/8$
- 100 $f_{ACLK}/16$
- 101 $f_{ACLK}/32$
- 110 Reserved for future use. Defaults to $f_{ACLK}/32$.
- 111 Reserved for future use. Defaults to $f_{ACLK}/32$.

Reserved Bit 7 Reserved. Reads back as 0.

DIVS Bits 6-4 SMCLK source divider

- 000 $f_{SMCLK}/1$
- 001 $f_{SMCLK}/2$
- 010 $f_{SMCLK}/4$
- 011 $f_{SMCLK}/8$
- 100 $f_{SMCLK}/16$
- 101 $f_{SMCLK}/32$
- 110 Reserved for future use. Defaults to $f_{SMCLK}/32$.
- 111 Reserved for future use. Defaults to $f_{SMCLK}/32$.

Reserved Bit 3 Reserved. Reads back as 0.

DIVM Bits 2-0 MCLK source divider

- 000 $f_{MCLK}/1$
- 001 $f_{MCLK}/2$
- 010 $f_{MCLK}/4$
- 011 $f_{MCLK}/8$
- 100 $f_{MCLK}/16$
- 101 $f_{MCLK}/32$
- 110 Reserved for future use. Defaults to $f_{MCLK}/32$.
- 111 Reserved for future use. Defaults to $f_{MCLK}/32$.

Unified Clock System Control 6 Register (UCSCTL6)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
XT2DRIVE		**Reserved**	**XT2BYPASS**	**Reserved**			**XT2OFF**
rw-1	rw-1	r0	rw-0	r0	r0	r0	rw-1

7	6	5	4	3	2	1	0
XT1DRIVE		**XTS**	**XT1BYPASS**	**XCAP**		**SMCLKOFF**	**XT1OFF**
rw-1	rw-1	rw-0	rw-0	rw-1	rw-1	rw-0	rw-1

XT2DRIVE Bits 15-14 The XT2 oscillator current can be adjusted to its drive needs. Initially, it starts with the highest supply current for reliable and quick startup. If needed, user software can reduce the drive strength.

- 00 Lowest current consumption. XT2 oscillator operating range is 4 MHz to 8 MHz.
- 01 Increased drive strength XT2 oscillator. XT2 oscillator operating range is 8 MHz to 16 MHz.
- 10 Increased drive capability XT2 oscillator. XT2 oscillator operating range is 16 MHz to 24 MHz.
- 11 Maximum drive capability and maximum current consumption for both XT2 oscillator. XT2 oscillator operating range is 24 MHz to 32 MHz.

Reserved Bit 13 Reserved. Reads back as 0.

XT2BYPASS Bit 12 XT2 bypass select

- 0 XT2 sourced internally
- 1 XT2 sourced externally from pin

Reserved Bits 11-9 Reserved. Reads back as 0.

XT2OFF Bit 8 Turns off the XT2 oscillator

- 0 XT2 is on if XT2 is selected via the port selection and XT2 is not in bypass mode of operation.
- 1 XT2 is off if it is not used as a source for ACLK, MCLK, or SMCLK or is not used as a reference source required for FLL operation.

XT1DRIVE Bits 7-6 The XT1 oscillator current can be adjusted to its drive needs. Initially, it starts with the highest supply current for reliable and quick startup. If needed, user software can reduce the drive strength.

- 00 Lowest current consumption for XT1 LF mode. XT1 oscillator operating range in HF mode is 4 MHz to 8 MHz.
- 01 Increased drive strength for XT1 LF mode. XT1 oscillator operating range in HF mode is 8 MHz to 16 MHz.
- 10 Increased drive capability for XT1 LF mode. XT1 oscillator operating range in HF mode is 16 MHz to 24 MHz.
- 11 Maximum drive capability and maximum current consumption for XT1 LF mode. XT1 oscillator operating range in HF mode is 24 MHz to 32 MHz.

XTS Bit 5 XT1 mode select

- 0 Low-frequency mode. XCAP bits define the capacitance at the XIN and XOUT pins.
- 1 High-frequency mode. XCAP bits are not used.

XT1BYPASS Bit 4 XT1 bypass select

- 0 XT1 sourced internally
- 1 XT1 sourced externally from pin

XCAP Bits 3-2 Oscillator capacitor selection. These bits select the capacitors applied to the LF crystal or resonator in the LF mode (XTS = 0). The effective capacitance (seen by the crystal) is C_{eff} X (C_{XIN} + 2 pF)/2. It is assumed that $C_{XIN} = C_{XOUT}$ and that a parasitic capacitance of 2 pF is added by the package and the printed circuit board. For details about the typical internal and the effective capacitors, refer to the device-specific data sheet.

SMCLKOFF Bit 1 SMCLK off. This bit turns off the SMCLK.

- 0 SMCLK on
- 1 SMCLK off

XT1OFF Bit 0 XT1 off. This bit turns off the XT1.

- 0 XT1 is on if XT1 is selected via the port selection and XT1 is not in bypass mode of operation.
- 1 XT1 is off if it is not used as a source for ACLK, MCLK, or SMCLK or is not used as a reference source required for FLL operation.

Unified Clock System Control 7 Register (UCSCTL7)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved		**Reserved**	**Reserved**	**Reserved**		**Reserved**	
r0	r0	rw-0	rw-(0)	rw-(1)	rw-(1)	r-1	r-1

7	6	5	4	3	2	1	0
Reserved			**Reserved**	**XT2OFFG** [(1)]	**XT1HFOFFG** [(1)]	**XT1LFOFFG**	**DCOFFG**
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(1)	rw-(1)

Reserved	Bits 15-14	Reserved. Reads back as 0.
Reserved	Bit 13	Reserved. This bit must always be written with 0.
Reserved	Bit 12	Reserved. This bit must always be written with 0.
Reserved	Bits 11-10	Reserved. The states of these bits should be ignored.
Reserved	Bits 9-8	Reserved. The states of these bits should be ignored.
Reserved	Bits 7-5	Reserved. Reads back as 0.
Reserved	Bit 4	Reserved. The state of this bit should be ignored.
XT2OFFG[(1)]	Bit 3	XT2 oscillator fault flag. If this bit is set, the OFIFG flag is also set. XT2OFFG is set if a XT2 fault condition exists. XT2OFFG can be cleared via software. If the XT2 fault condition still remains, XT2OFFG is set. 0 No fault condition occurred after the last reset. 1 XT2 fault. An XT2 fault occurred after the last reset.
XT1HFOFFG [(1)]	Bit 2	XT1 oscillator fault flag (HF mode). If this bit is set, the OFIFG flag is also set. XT1HFOFFG is set if a XT1 fault condition exists. XT1HFOFFG can be cleared via software. If the XT1 fault condition still remains, XT1HFOFFG is set. 0 No fault condition occurred after the last reset. 1 XT1 fault. An XT1 fault occurred after the last reset.
XT1LFOFFG	Bit 1	XT1 oscillator fault flag (LF mode). If this bit is set, the OFIFG flag is also set. XT1LFOFFG is set if a XT1 fault condition exists. XT1LFOFFG can be cleared via software. If the XT1 fault condition still remains, XT1LFOFFG is set. 0 No fault condition occurred after the last reset. 1 XT1 fault (LF mode). A XT1 fault occurred after the last reset.
DCOFFG	Bit 0	DCO fault flag. If this bit is set, the OFIFG flag is also set. The DCOFFG bit is set if DCO = {0} or DCO = {31}. DCOOFFG can be cleared via software. If the DCO fault condition still remains, DCOOFFG is set. 0 No fault condition occurred after the last reset. 1 DCO fault. A DCO fault occurred after the last reset.

(1) Not available on all devices. When not available, this bit is reserved.
(1) Not available on all devices. When not available, this bit is reserved.

Unified Clock System Control 8 Register (UCSCTL8)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved					**Reserved**		
r0	r0	r0	r0	r0	rw-(1)	rw-(1)	rw-(1)

7	6	5	4	3	2	1	0
Reserved			**Reserved**	**MODOSC REQEN**	**SMCLKREQEN**	**MCLKREQEN**	**ACLKREQEN**
r0	r0	r0	rw-(0)	rw-(0)	rw-(1)	rw-(1)	rw-(1)

Reserved	Bits 15-11	Reserved. Reads back as 0.
Reserved	Bits 10-8	Reserved. Must always be written as 1.
Reserved	Bits 7-5	Reserved. Reads back as 0.
Reserved	Bit 4	Reserved. Must always be written as 0.
MODOSCREQEN	Bit 3	MODOSC clock request enable. Setting this enables conditional module requests for MODOSC. 0 MODOSC conditional requests are disabled. 1 MODOSC conditional requests are enabled.
SMCLKREQEN	Bit 2	SMCLK clock request enable. Setting this enables conditional module requests for SMCLK 0 SMCLK conditional requests are disabled. 1 SMCLK conditional requests are enabled.
MCLKREQEN	Bit 1	MCLK clock request enable. Setting this enables conditional module requests for MCLK 0 MCLK conditional requests are disabled. 1 MCLK conditional requests are enabled.
ACLKREQEN	Bit 0	ACLK clock request enable. Setting this enables conditional module requests for ACLK 0 ACLK conditional requests are disabled. 1 ACLK conditional requests are enabled.

CPUX

This chapter describes the extended MSP430X 16-bit RISC CPU (CPUX) with 1-MB memory access, its addressing modes, and instruction set.

NOTE: The MSP430X CPU implemented on MSP430F5xx devices has, in some cases, slightly different cycle counts from the MSP430X CPU implemented on the 2xx and 4xx families.

Topic **Page**

4.1	**MSP430X CPU (CPUX) Introduction**	**102**
4.2	**Interrupts**	**104**
4.3	**CPU Registers**	**105**
4.4	**Addressing Modes**	**111**
4.5	**MSP430 and MSP430X Instructions**	**128**
4.6	**Instruction Set Description**	**144**

4.1 MSP430X CPU (CPUX) Introduction

The MSP430X CPU incorporates features specifically designed for modern programming techniques, such as calculated branching, table processing, and the use of high-level languages such as C. The MSP430X CPU can address a 1-MB address range without paging. The MSP430X CPU is completely backwards compatible with the MSP430 CPU.

The MSP430X CPU features include:

- RISC architecture
- Orthogonal architecture
- Full register access including program counter (PC), status register (SR), and stack pointer (SP)
- Single-cycle register operations
- Large register file reduces fetches to memory.
- 20-bit address bus allows direct access and branching throughout the entire memory range without paging.
- 16-bit data bus allows direct manipulation of word-wide arguments.
- Constant generator provides the six most often used immediate values and reduces code size.
- Direct memory-to-memory transfers without intermediate register holding
- Byte, word, and 20-bit address-word addressing

The block diagram of the MSP430X CPU is shown in Figure 4-1.

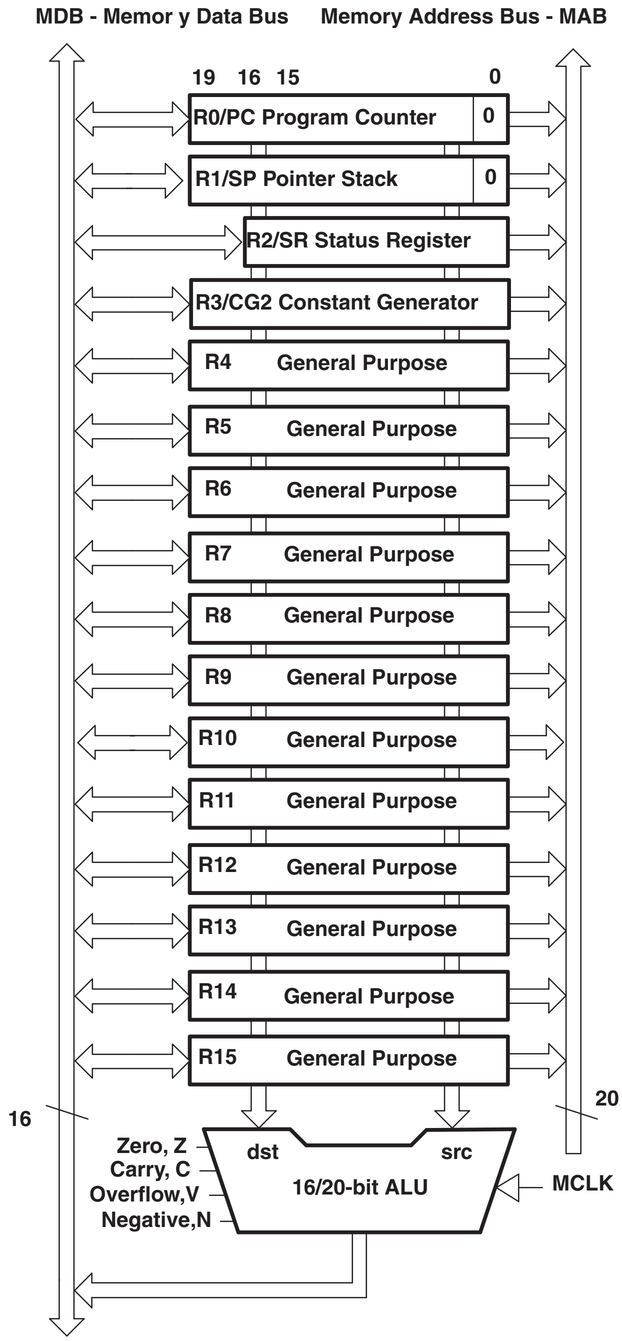


Figure 4-1. MSP430X CPU Block Diagram

4.2 Interrupts

The MSP430X has the following interrupt structure:

- Vectored interrupts with no polling necessary
- Interrupt vectors are located downward from address 0FFFEh.

The interrupt vectors contain 16-bit addresses that point into the lower 64-KB memory. This means all interrupt handlers must start in the lower 64-KB memory.

During an interrupt, the program counter (PC) and the status register (SR) are pushed onto the stack as shown in Figure 4-2. The MSP430X architecture stores the complete 20-bit PC value efficiently by appending the PC bits 19:16 to the stored SR value automatically on the stack. When the RETI instruction is executed, the full 20-bit PC is restored making return from interrupt to any address in the memory range possible.

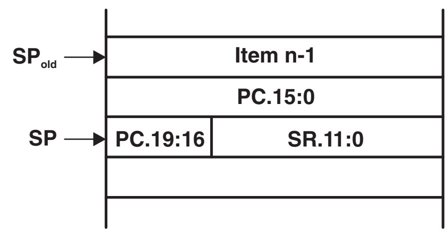


Figure 4-2. PC Storage on the Stack for Interrupts

4.3 CPU Registers

The CPU incorporates 16 registers (R0 through R15). Registers R0, R1, R2, and R3 have dedicated functions. Registers R4 through R15 are working registers for general use.

4.3.1 Program Counter (PC)

The 20-bit PC (PC/R0) points to the next instruction to be executed. Each instruction uses an even number of bytes (2, 4, 6, or 8 bytes), and the PC is incremented accordingly. Instruction accesses are performed on word boundaries, and the PC is aligned to even addresses. Figure 4-3 shows the PC.

Figure 4-3. Program Counter

The PC can be addressed with all instructions and addressing modes. A few examples:

```
MOV.W  #LABEL,PC  ; Branch to address LABEL (lower 64 KB)

MOVA   #LABEL,PC  ; Branch to address LABEL (1MB memory)

MOV.W  LABEL,PC   ; Branch to address in word LABEL
                  ; (lower 64 KB)

MOV.W  @R14,PC    ; Branch indirect to address in
                  ; R14 (lower 64 KB)

ADDA   #4,PC      ; Skip two words (1 MB memory)
```

The BR and CALL instructions reset the upper four PC bits to 0. Only addresses in the lower 64-KB address range can be reached with the BR or CALL instruction. When branching or calling, addresses beyond the lower 64-KB range can only be reached using the BRA or CALLA instructions. Also, any instruction to directly modify the PC does so according to the used addressing mode. For example, `MOV.W #value,PC` clears the upper four bits of the PC, because it is a .W instruction.

The PC is automatically stored on the stack with CALL (or CALLA) instructions and during an interrupt service routine. Figure 4-4 shows the storage of the PC with the return address after a CALLA instruction. A CALL instruction stores only bits 15:0 of the PC.

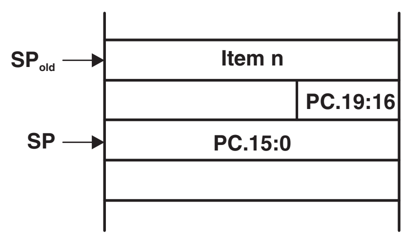


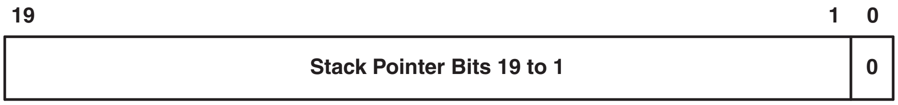
Figure 4-4. PC Storage on the Stack for CALLA

The RETA instruction restores bits 19:0 of the PC and adds 4 to the stack pointer (SP). The RET instruction restores bits 15:0 to the PC and adds 2 to the SP.

4.3.2 Stack Pointer (SP)

The 20-bit SP (SP/R1) is used by the CPU to store the return addresses of subroutine calls and interrupts. It uses a predecrement, postincrement scheme. In addition, the SP can be used by software with all instructions and addressing modes. Figure 4-5 shows the SP. The SP is initialized into RAM by the user, and is always aligned to even addresses.

Figure 4-6 shows the stack usage. Figure 4-7 shows the stack usage when 20-bit address words are pushed.




```
MOV.W   2(SP),R6      ; Copy Item I2 to R6
MOV.W   R7,0(SP)      ; Overwrite TOS with R7
PUSH    #0123h        ; Put 0123h on stack
POP     R8            ; R8 = 0123h
```

Figure 4-5. Stack Pointer

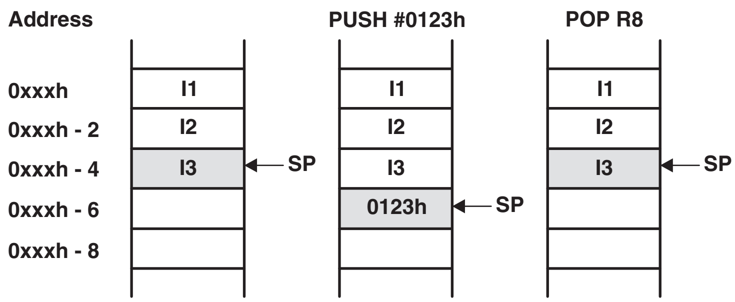


Figure 4-6. Stack Usage

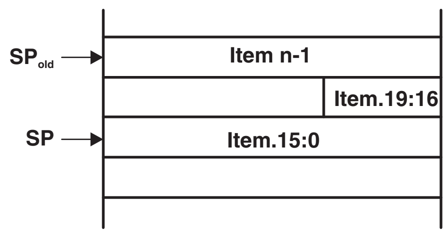


Figure 4-7. PUSHX.A Format on the Stack

The special cases of using the SP as an argument to the PUSH and POP instructions are described and shown in Figure 4-8.

The stack pointer is changed after a PUSH SP instruction.

The stack pointer is not changed after a POP SP instruction. The POP SP instruction places SP1 into the stack pointer SP (SP2 = SP1)

Figure 4-8. PUSH SP, POP SP Sequence

4.3.3 Status Register (SR)

The 16-bit SR (SR/R2), used as a source or destination register, can only be used in register mode addressed with word instructions. The remaining combinations of addressing modes are used to support the constant generator. Figure 4-9 shows the SR bits. Do not write 20-bit values to the SR. Unpredictable operation can result.

15–9	8	7							0
Reserved	V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C
rw-0									

Figure 4-9. SR Bits

Table 4-1 describes the SR bits.

Table 4-1. SR Bit Description

Bit	Description	
Reserved	Reserved	
V	Overflow. This bit is set when the result of an arithmetic operation overflows the signed-variable range.	
	`ADD(.B), ADDX(.B,.A), ADDC(.B), ADDCX(.B.A), ADDA`	Set when: positive + positive = negative negative + negative = positive otherwise reset
	`SUB(.B), SUBX(.B,.A), SUBC(.B),SUBCX(.B,.A), SUBA, CMP(.B), CMPX(.B,.A), CMPA`	Set when: positive – negative = negative negative – positive = positive otherwise reset
SCG1	System clock generator 1. This bit, when set, turns off the DCO dc generator if DCOCLK is not used for MCLK or SMCLK.	
SCG0	System clock generator 0. This bit, when set, turns off the FLL+ loop control.	
OSCOFF	Oscillator off. This bit, when set, turns off the LFXT1 crystal oscillator when LFXT1CLK is not used for MCLK or SMCLK.	
CPUOFF	CPU off. This bit, when set, turns off the CPU.	
GIE	General interrupt enable. This bit, when set, enables maskable interrupts. When reset, all maskable interrupts are disabled.	
N	Negative. This bit is set when the result of an operation is negative and cleared when the result is positive.	
Z	Zero. This bit is set when the result of an operation is 0 and cleared when the result is not 0.	
C	Carry. This bit is set when the result of an operation produced a carry and cleared when no carry occurred.	

NOTE: Bit manipulations of the SR should be done via the following instructions: *MOV*, *BIS*, and *BIC*.

4.3.4 Constant Generator Registers (CG1 and CG2)

Six commonly-used constants are generated with the constant generator registers R2 (CG1) and R3 (CG2), without requiring an additional 16-bit word of program code. The constants are selected with the source register addressing modes (As), as described in Table 4-2.

Table 4-2. Values of Constant Generators CG1, CG2

Register	As	Constant	Remarks
R2	00	–	Register mode
R2	01	(0)	Absolute address mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	FFh, FFFFh, FFFFFh	–1, word processing

The constant generator advantages are:

- No special instructions required
- No additional code word for the six constants
- No code memory access required to retrieve the constant

The assembler uses the constant generator automatically if one of the six constants is used as an immediate source operand. Registers R2 and R3, used in the constant mode, cannot be addressed explicitly; they act as source-only registers.

4.3.4.1 Constant Generator – Expanded Instruction Set

The RISC instruction set of the MSP430 has only 27 instructions. However, the constant generator allows the MSP430 assembler to support 24 additional emulated instructions. For example, the single-operand instruction:

```
CLR dst
```

is emulated by the double-operand instruction with the same length:

```
MOV R3,dst
```

where the #0 is replaced by the assembler, and R3 is used with As = 00.

```
INC dst
```

is replaced by:

```
ADD 0(R3),dst
```

4.3.5 General-Purpose Registers (R4 –R15)

The 12 CPU registers (R4 to R15) contain 8-bit, 16-bit, or 20-bit values. Any byte-write to a CPU register clears bits 19:8. Any word-write to a register clears bits 19:16. The only exception is the SXT instruction. The SXT instruction extends the sign through the complete 20-bit register.

The following figures show the handling of byte, word, and address-word data. Note the reset of the leading most significant bits (MSBs) if a register is the destination of a byte or word instruction.

Figure 4-10 shows byte handling (8-bit data, .B suffix). The handling is shown for a source register and a destination memory byte and for a source memory byte and a destination register.

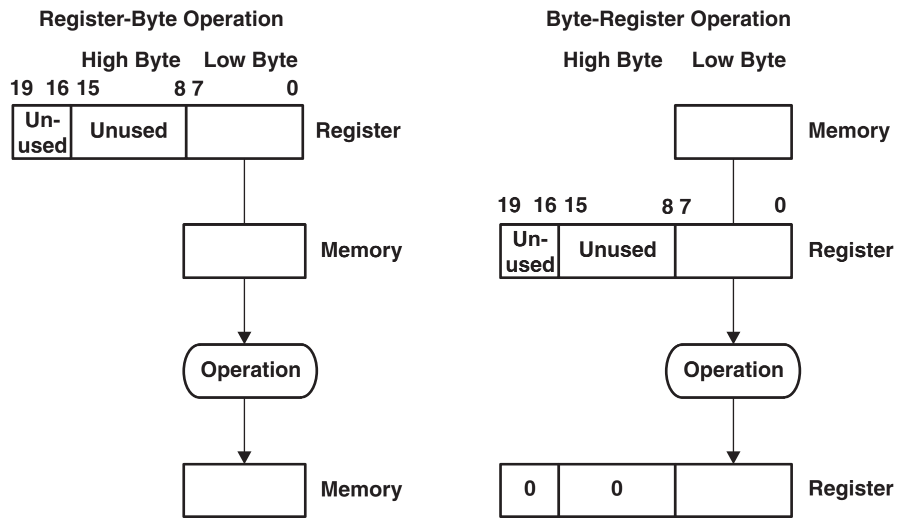


Figure 4-10. Register-Byte/Byte-Register Operation

Figure 4-11 and Figure 4-12 show 16-bit word handling (.W suffix). The handling is shown for a source register and a destination memory word and for a source memory word and a destination register.

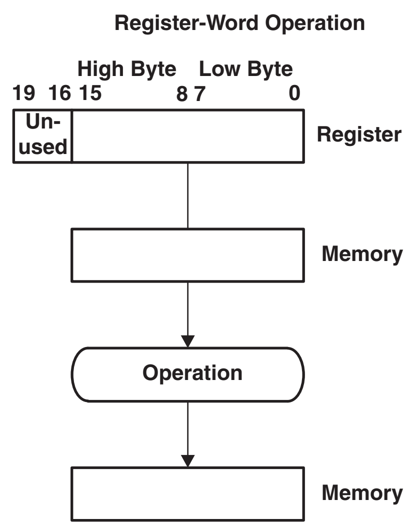


Figure 4-11. Register-Word Operation

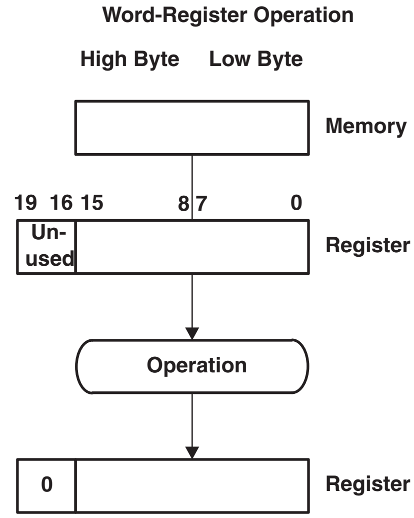


Figure 4-12. Word-Register Operation

Figure 4-13 and Figure 4-14 show 20-bit address-word handling (.A suffix). The handling is shown for a source register and a destination memory address-word and for a source memory address-word and a destination register.

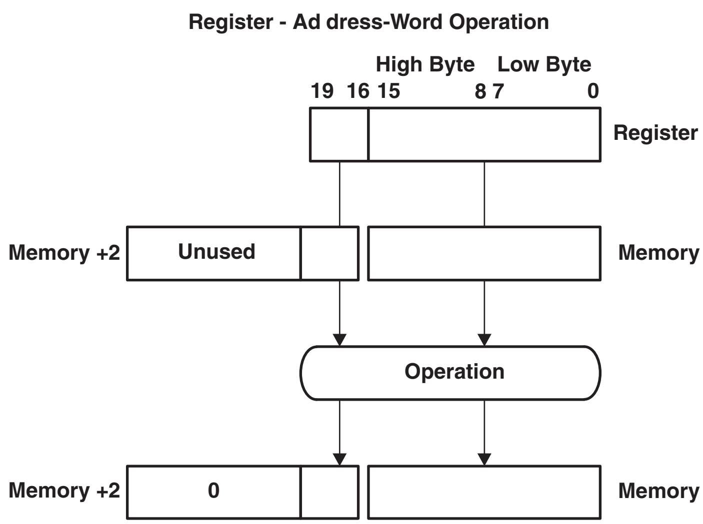


Figure 4-13. Register – Address-Word Operation

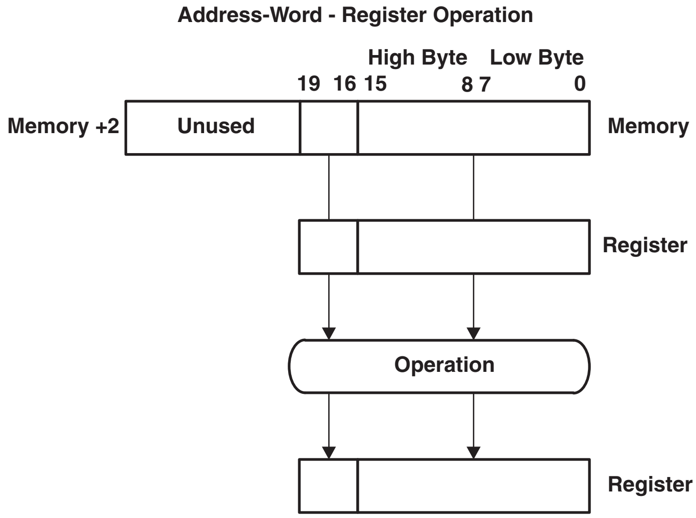


Figure 4-14. Address-Word – Register Operation

4.4 Addressing Modes

Seven addressing modes for the source operand and four addressing modes for the destination operand use 16-bit or 20-bit addresses (see Table 4-3). The MSP430 and MSP430X instructions are usable throughout the entire 1-MB memory range.

Table 4-3. Source/Destination Addressing

As/Ad	Addressing Mode	Syntax	Description
00/0	Register	Rn	Register contents are operand.
01/1	Indexed	X(Rn)	(Rn + X) points to the operand. X is stored in the next word, or stored in combination of the preceding extension word and the next word.
01/1	Symbolic	ADDR	(PC + X) points to the operand. X is stored in the next word, or stored in combination of the preceding extension word and the next word. Indexed mode X(PC) is used.
01/1	Absolute	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word, or stored in combination of the preceding extension word and the next word. Indexed mode X(SR) is used.
10/–	Indirect Register	@Rn	Rn is used as a pointer to the operand.
11/–	Indirect Autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions. by 2 for .W instructions, and by 4 for .A instructions.
11/–	Immediate	#N	N is stored in the next word, or stored in combination of the preceding extension word and the next word. Indirect autoincrement mode @PC+ is used.

The seven addressing modes are explained in detail in the following sections. Most of the examples show the same addressing mode for the source and destination, but any valid combination of source and destination addressing modes is possible in an instruction.

NOTE: Use of Labels EDE, TONI, TOM, and LEO

Throughout MSP430 documentation, EDE, TONI, TOM, and LEO are used as generic labels. They are only labels and have no special meaning.

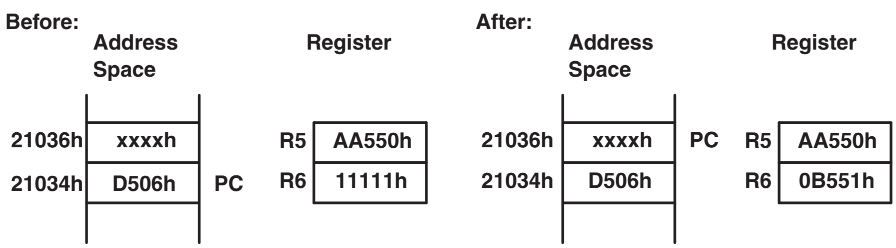
4.4.1 Register Mode

Operation:	The operand is the 8-, 16-, or 20-bit content of the used CPU register.
Length:	One, two, or three words
Comment:	Valid for source and destination
Byte operation:	Byte operation reads only the eight least significant bits (LSBs) of the source register Rsrc and writes the result to the eight LSBs of the destination register Rdst. The bits Rdst.19:8 are cleared. The register Rsrc is not modified.
Word operation:	Word operation reads the 16 LSBs of the source register Rsrc and writes the result to the 16 LSBs of the destination register Rdst. The bits Rdst.19:16 are cleared. The register Rsrc is not modified.
Address-word operation:	Address-word operation reads the 20 bits of the source register Rsrc and writes the result to the 20 bits of the destination register Rdst. The register Rsrc is not modified
SXT exception:	The SXT instruction is the only exception for register operation. The sign of the low byte in bit 7 is extended to the bits Rdst.19:8.

Example:

```
BIS.W R5,R6 ;
```

This instruction logically ORs the 16-bit data contained in R5 with the 16-bit contents of R6. R6.19:16 is cleared.



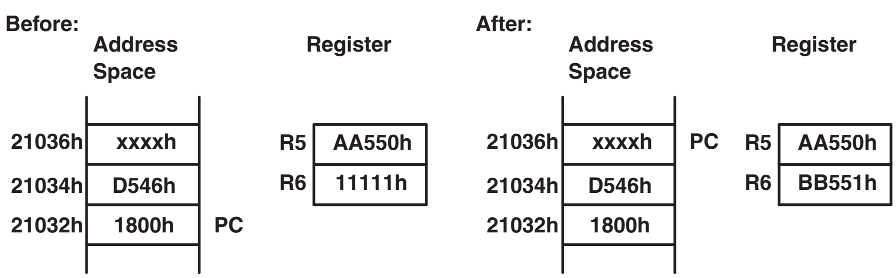
A550h.or.1111h = B551h

Example:

```
BISX.A R5,R6 ;
```

This instruction logically ORs the 20-bit data contained in R5 with the 20-bit contents of R6.

The extension word contains the A/L bit for 20-bit data. The instruction word uses byte mode with bits A/L:B/W = 01. The result of the instruction is:



AA550h.or.11111h = BB551h

4.4.2 Indexed Mode

The Indexed mode calculates the address of the operand by adding the signed index to a CPU register. The Indexed mode has three addressing possibilities:

- Indexed mode in lower 64-KB memory
- MSP430 instruction with Indexed mode addressing memory above the lower 64-KB memory
- MSP430X instruction with Indexed mode

4.4.2.1 Indexed Mode in Lower 64-KB Memory

If the CPU register Rn points to an address in the lower 64 KB of the memory range, the calculated memory address bits 19:16 are cleared after the addition of the CPU register Rn and the signed 16-bit index. This means the calculated memory address is always located in the lower 64 KB and does not overflow or underflow out of the lower 64-KB memory space. The RAM and the peripheral registers can be accessed this way and existing MSP430 software is usable without modifications as shown in Figure 4-15.

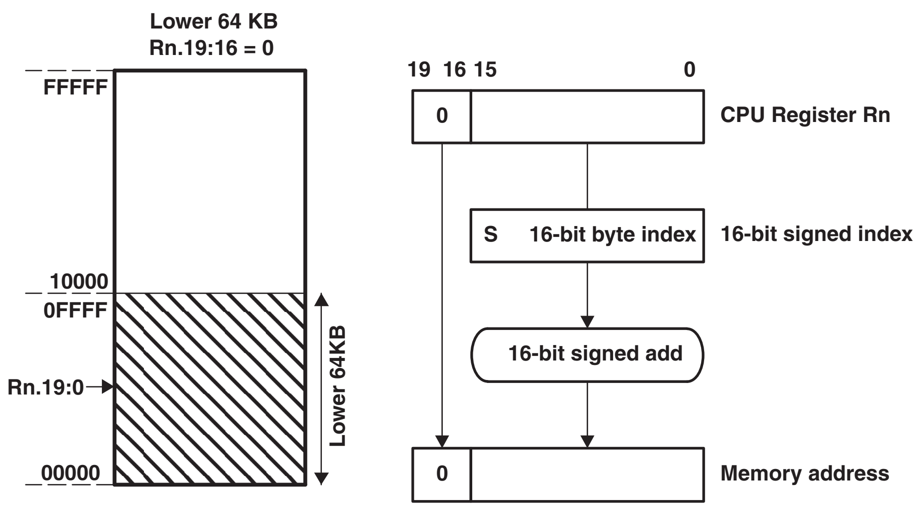
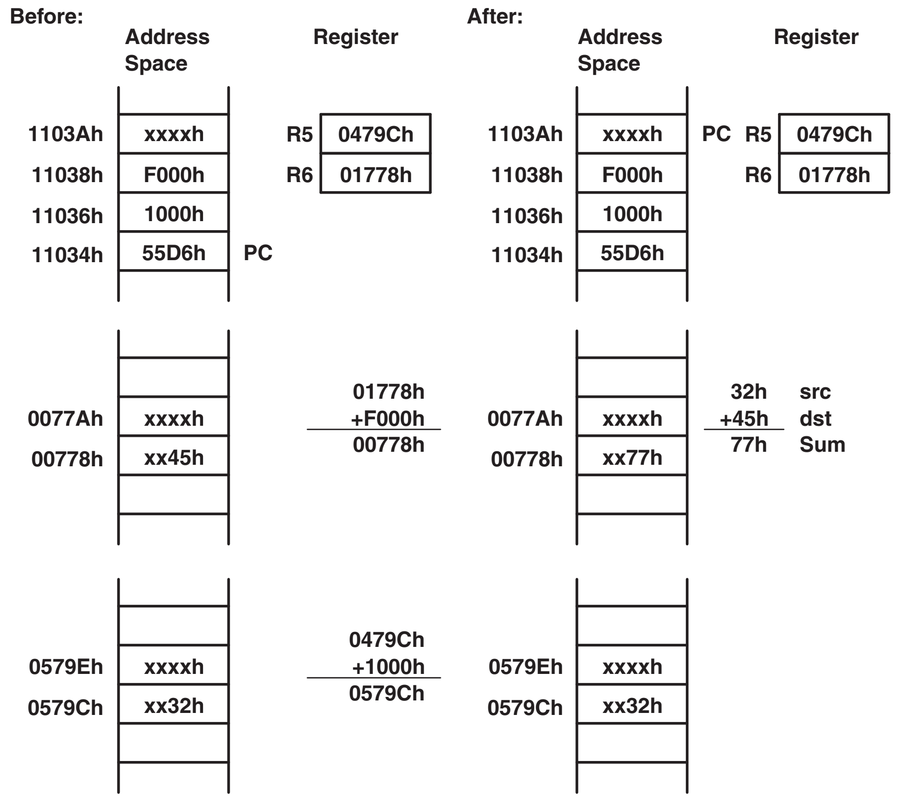


Figure 4-15. Indexed Mode in Lower 64 KB

Length:	Two or three words
Operation:	The signed 16-bit index is located in the next word after the instruction and is added to the CPU register Rn. The resulting bits 19:16 are cleared giving a truncated 16-bit memory address, which points to an operand address in the range 00000h to 0FFFFh. The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the register index and inserts it.
Example:	`ADD.B 1000h(R5),0F000h(R6);`
	This instruction adds the 8-bit data contained in source byte 1000h(R5) and the destination byte 0F000h(R6) and places the result into the destination byte. Source and destination bytes are both located in the lower 64 KB due to the cleared bits 19:16 of registers R5 and R6.
Source:	The byte pointed to by R5 + 1000h results in address 0479Ch + 1000h = 0579Ch after truncation to a 16-bit address.
Destination:	The byte pointed to by R6 + F000h results in address 01778h + F000h = 00778h after truncation to a 16-bit address.



4.4.2.2 MSP430 Instruction With Indexed Mode in Upper Memory

If the CPU register Rn points to an address above the lower 64-KB memory, the Rn bits 19:16 are used for the address calculation of the operand. The operand may be located in memory in the range Rn ±32 KB, because the index, X, is a signed 16-bit value. In this case, the address of the operand can overflow or underflow into the lower 64-KB memory space (see Figure 4-16 and Figure 4-17).

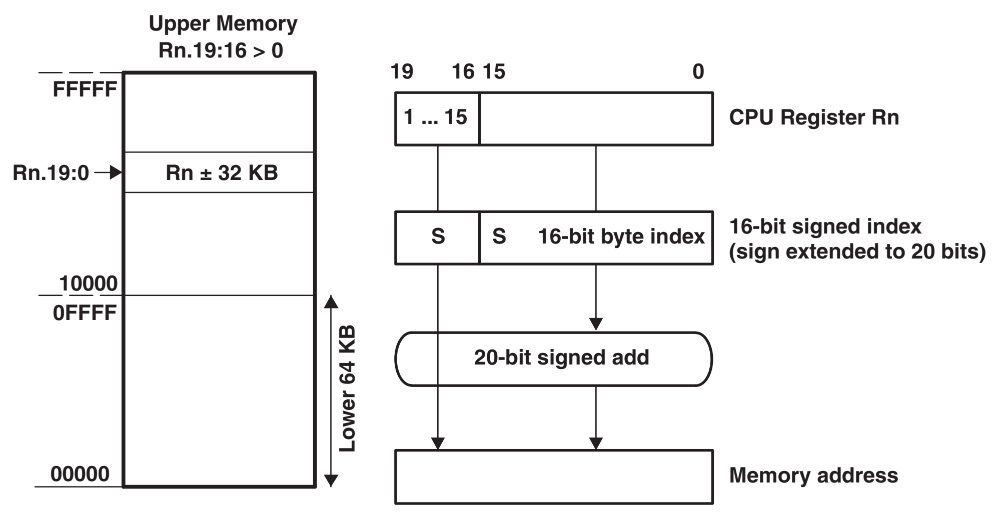


Figure 4-16. Indexed Mode in Upper Memory

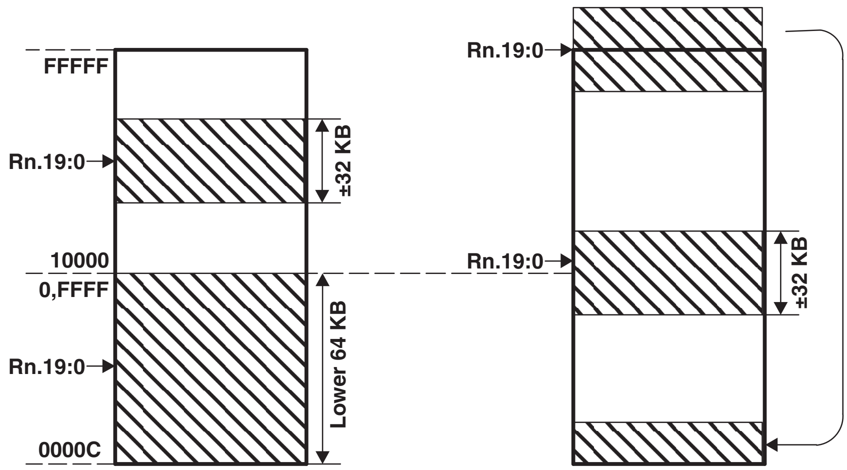


Figure 4-17. Overflow and Underflow for Indexed Mode

Length:	Two or three words
Operation:	The sign-extended 16-bit index in the next word after the instruction is added to the 20 bits of the CPU register Rn. This delivers a 20-bit address, which points to an address in the range 0 to FFFFFh. The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the register index and inserts it.
Example:	`ADD.W 8346h(R5),2100h(R6) ;`
	This instruction adds the 16-bit data contained in the source and the destination addresses and places the 16-bit result into the destination. Source and destination operand can be located in the entire address range.
Source:	The word pointed to by R5 + 8346h. The negative index 8346h is sign extended, which results in address 23456h + F8346h = 1B79Ch.
Destination:	The word pointed to by R6 + 2100h results in address 15678h + 2100h = 17778h.

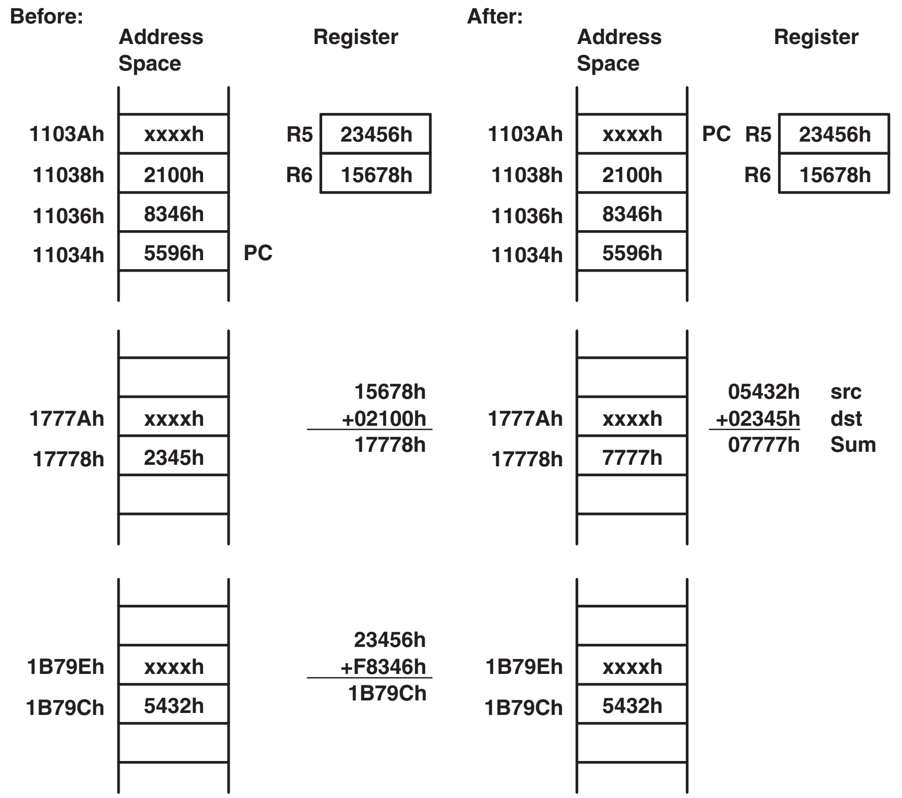


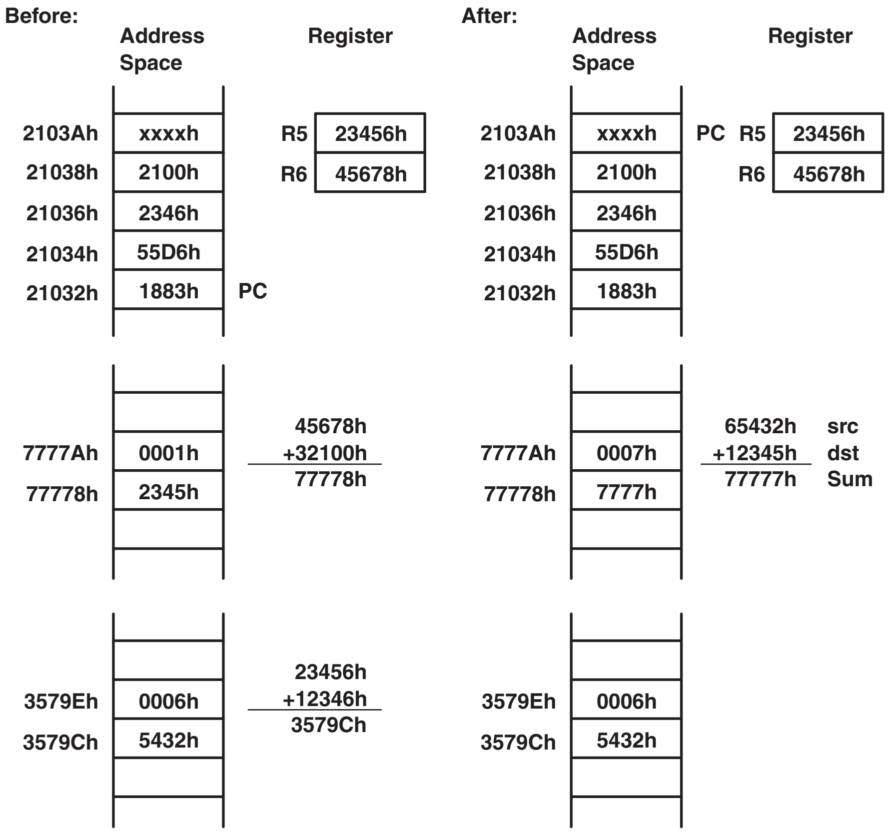
Figure 4-18. Example for Indexed Mode

4.4.2.3 MSP430X Instruction With Indexed Mode

When using an MSP430X instruction with Indexed mode, the operand can be located anywhere in the range of Rn + 19 bits.

Length:	Three or four words
Operation:	The operand address is the sum of the 20-bit CPU register content and the 20-bit index. The 4 MSBs of the index are contained in the extension word; the 16 LSBs are contained in the word following the instruction. The CPU register is not modified
Comment:	Valid for source and destination. The assembler calculates the register index and inserts it.
Example:	`ADDX.A 12346h(R5),32100h(R6) ;`
	This instruction adds the 20-bit data contained in the source and the destination addresses and places the result into the destination.
Source:	Two words pointed to by R5 + 12346h which results in address 23456h + 12346h = 3579Ch.
Destination:	Two words pointed to by R6 + 32100h which results in address 45678h + 32100h = 77778h.

The extension word contains the MSBs of the source index and of the destination index and the A/L bit for 20-bit data. The instruction word uses byte mode due to the 20-bit data length with bits A/L:B/W = 01.



4.4.3 Symbolic Mode

The Symbolic mode calculates the address of the operand by adding the signed index to the PC. The Symbolic mode has three addressing possibilities:

- Symbolic mode in lower 64-KB memory
- MSP430 instruction with Symbolic mode addressing memory above the lower 64-KB memory.
- MSP430X instruction with Symbolic mode

4.4.3.1 Symbolic Mode in Lower 64 KB

If the PC points to an address in the lower 64 KB of the memory range, the calculated memory address bits 19:16 are cleared after the addition of the PC and the signed 16-bit index. This means the calculated memory address is always located in the lower 64 KB and does not overflow or underflow out of the lower 64-KB memory space. The RAM and the peripheral registers can be accessed this way and existing MSP430 software is usable without modifications as shown in Figure 4-19.

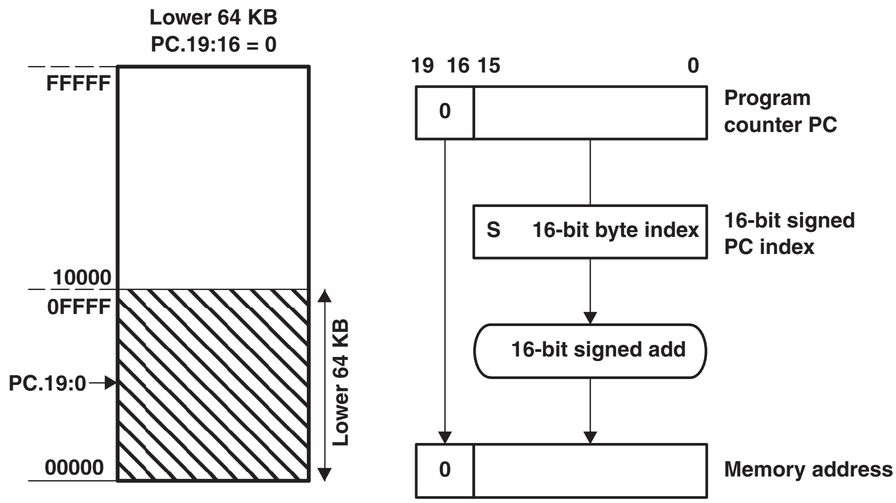
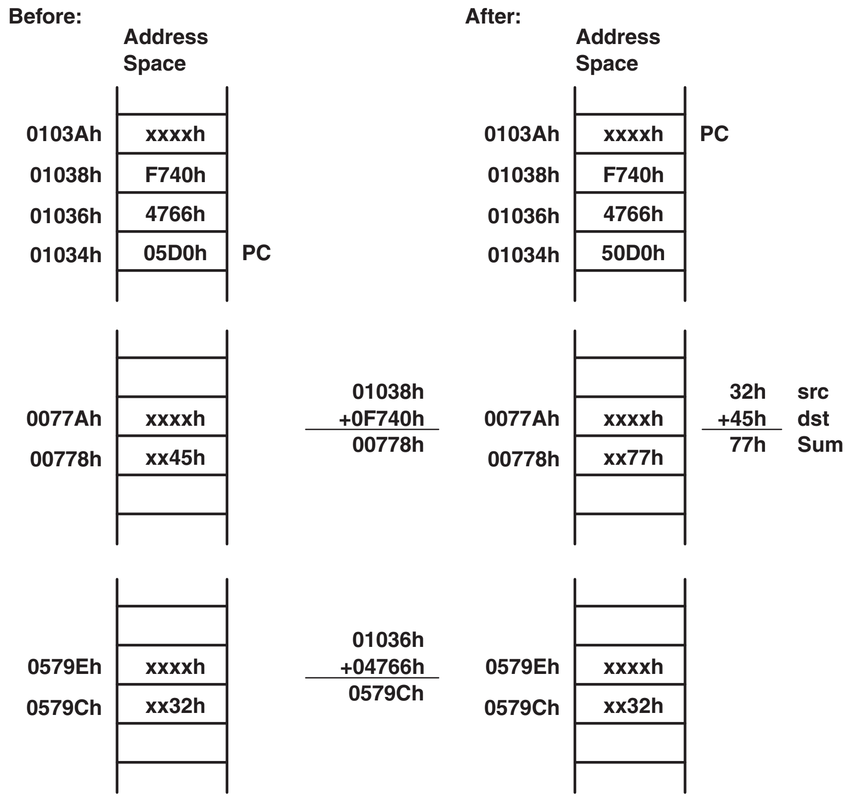


Figure 4-19. Symbolic Mode Running in Lower 64 KB

Operation:	The signed 16-bit index in the next word after the instruction is added temporarily to the PC. The resulting bits 19:16 are cleared giving a truncated 16-bit memory address, which points to an operand address in the range 00000h to 0FFFFh. The operand is the content of the addressed memory location.
Length:	Two or three words
Comment:	Valid for source and destination. The assembler calculates the PC index and inserts it.
Example:	`ADD.B EDE,TONI ;`
	This instruction adds the 8-bit data contained in source byte EDE and destination byte TONI and places the result into the destination byte TONI. Bytes EDE and TONI and the program are located in the lower 64 KB.
Source:	Byte EDE located at address 0579Ch, pointed to by PC + 4766h, where the PC index 4766h is the result of 0579Ch – 01036h = 04766h. Address 01036h is the location of the index for this example.
Destination:	Byte TONI located at address 00778h, pointed to by PC + F740h, is the truncated 16-bit result of 00778h – 1038h = FF740h. Address 01038h is the location of the index for this example.



4.4.3.2 MSP430 Instruction With Symbolic Mode in Upper Memory

If the PC points to an address above the lower 64-KB memory, the PC bits 19:16 are used for the address calculation of the operand. The operand may be located in memory in the range PC ± 32 KB, because the index, X, is a signed 16-bit value. In this case, the address of the operand can overflow or underflow into the lower 64-KB memory space as shown in Figure 4-20 and Figure 4-21.

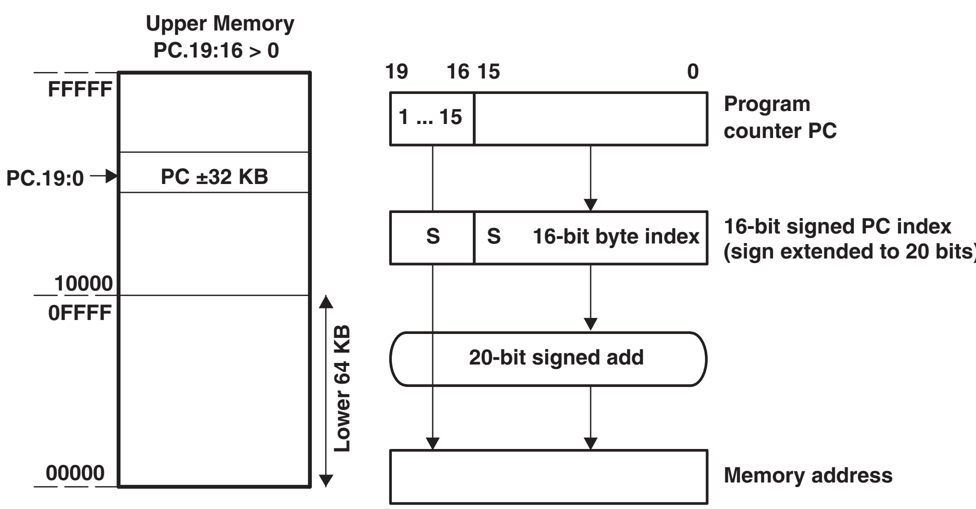


Figure 4-20. Symbolic Mode Running in Upper Memory

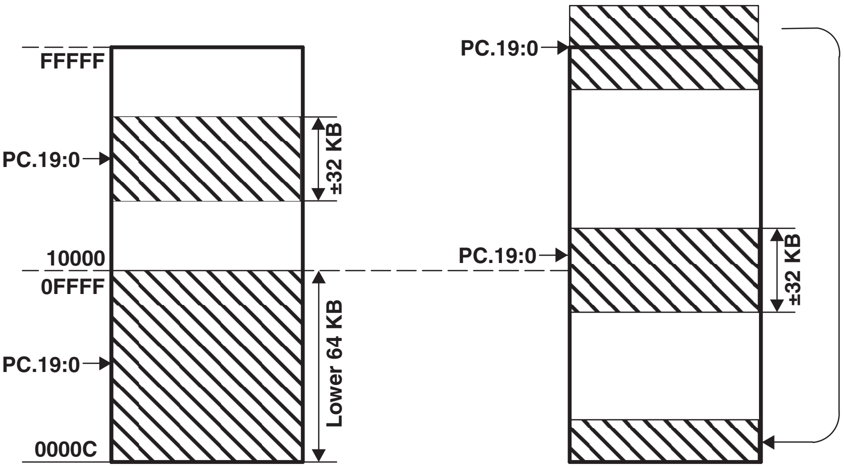
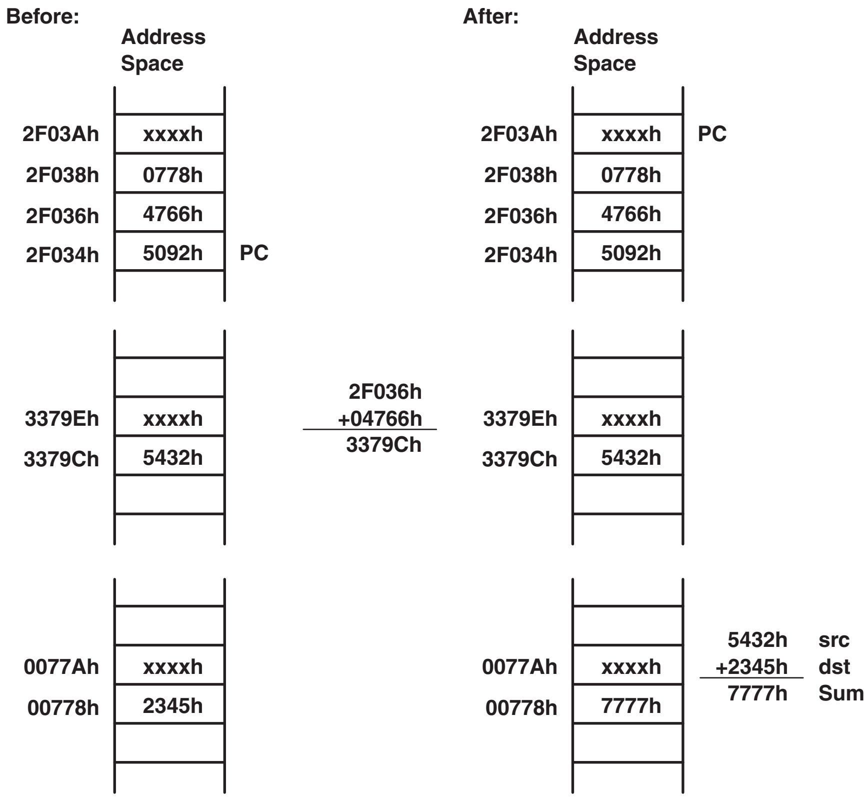


Figure 4-21. Overflow and Underflow for Symbolic Mode

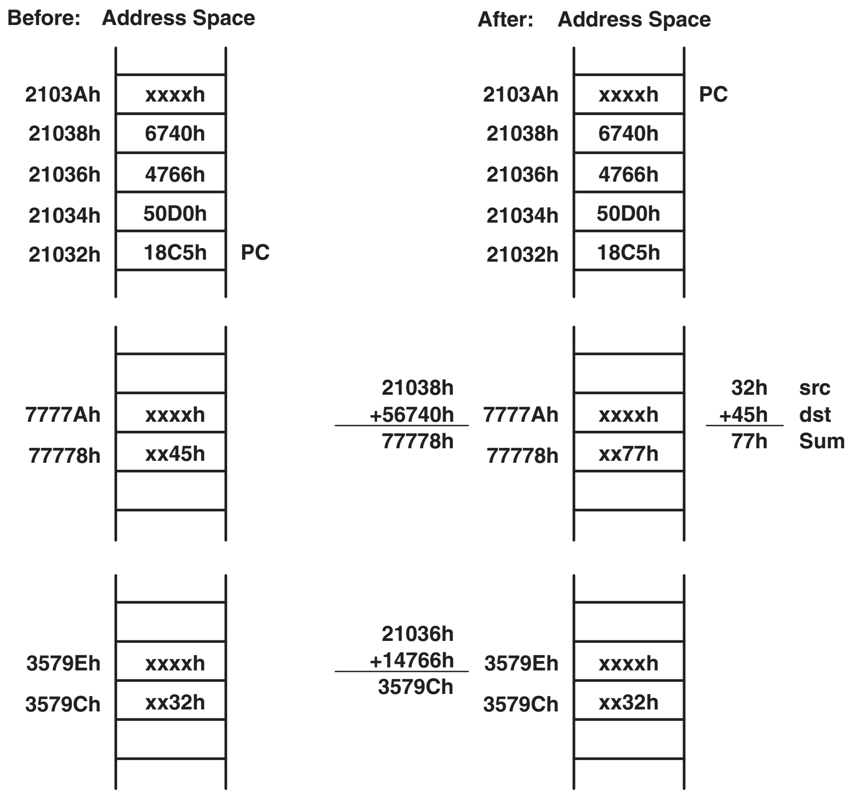
Length:	Two or three words
Operation:	The sign-extended 16-bit index in the next word after the instruction is added to the 20 bits of the PC. This delivers a 20-bit address, which points to an address in the range 0 to FFFFFh. The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the PC index and inserts it
Example:	`ADD.W EDE,&TONI ;`
	This instruction adds the 16-bit data contained in source word EDE and destination word TONI and places the 16-bit result into the destination word TONI. For this example, the instruction is located at address 2F034h.
Source:	Word EDE at address 3379Ch, pointed to by PC + 4766h, which is the 16-bit result of 3379Ch – 2F036h = 04766h. Address 2F036h is the location of the index for this example.
Destination:	Word TONI located at address 00778h pointed to by the absolute address 00778h



4.4.3.3 MSP430X Instruction With Symbolic Mode

When using an MSP430X instruction with Symbolic mode, the operand can be located anywhere in the range of PC + 19 bits.

Length:	Three or four words
Operation:	The operand address is the sum of the 20-bit PC and the 20-bit index. The 4 MSBs of the index are contained in the extension word; the 16 LSBs are contained in the word following the instruction.
Comment:	Valid for source and destination. The assembler calculates the register index and inserts it.
Example:	`ADDX.B EDE,TONI ;`
	This instruction adds the 8-bit data contained in source byte EDE and destination byte TONI and places the result into the destination byte TONI.
Source:	Byte EDE located at address 3579Ch, pointed to by PC + 14766h, is the 20-bit result of 3579Ch – 21036h = 14766h. Address 21036h is the address of the index in this example.
Destination:	Byte TONI located at address 77778h, pointed to by PC + 56740h, is the 20-bit result of 77778h – 21038h = 56740h. Address 21038h is the address of the index in this example.



4.4.4 Absolute Mode

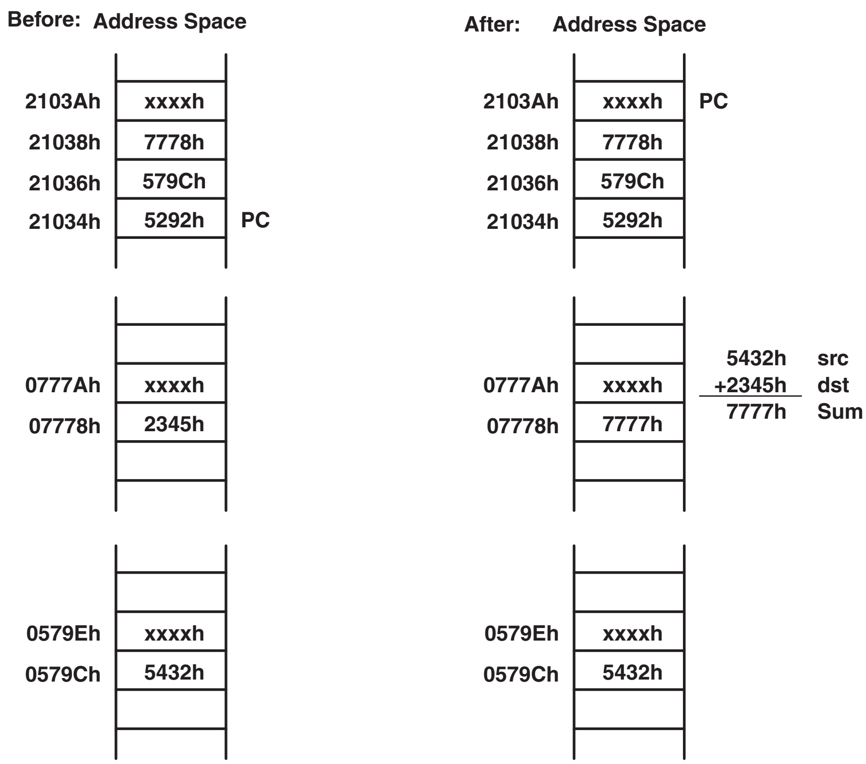
The Absolute mode uses the contents of the word following the instruction as the address of the operand. The Absolute mode has two addressing possibilities:

- Absolute mode in lower 64-KB memory
- MSP430X instruction with Absolute mode

4.4.4.1 Absolute Mode in Lower 64 KB

If an MSP430 instruction is used with Absolute addressing mode, the absolute address is a 16-bit value and, therefore, points to an address in the lower 64 KB of the memory range. The address is calculated as an index from 0 and is stored in the word following the instruction The RAM and the peripheral registers can be accessed this way and existing MSP430 software is usable without modifications.

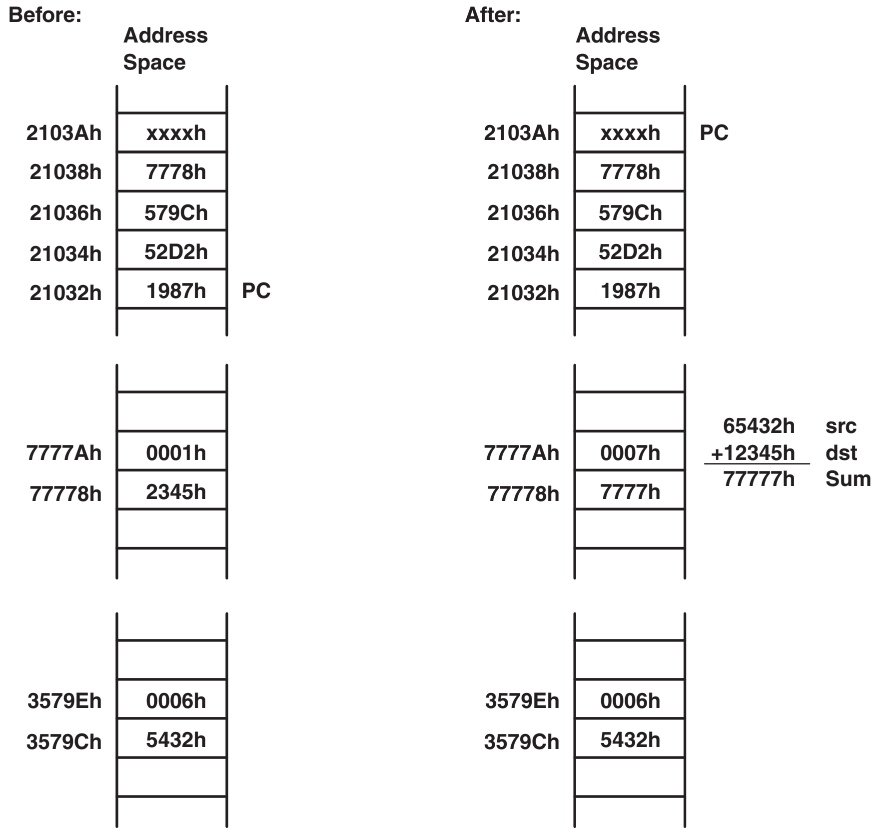
Length:	Two or three words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the index from 0 and inserts it.
Example:	`ADD.W &EDE,&TONI ;`
	This instruction adds the 16-bit data contained in the absolute source and destination addresses and places the result into the destination.
Source:	Word at address EDE
Destination:	Word at address TONI



4.4.4.2 MSP430X Instruction With Absolute Mode

If an MSP430X instruction is used with Absolute addressing mode, the absolute address is a 20-bit value and, therefore, points to any address in the memory range. The address value is calculated as an index from 0. The 4 MSBs of the index are contained in the extension word, and the 16 LSBs are contained in the word following the instruction.

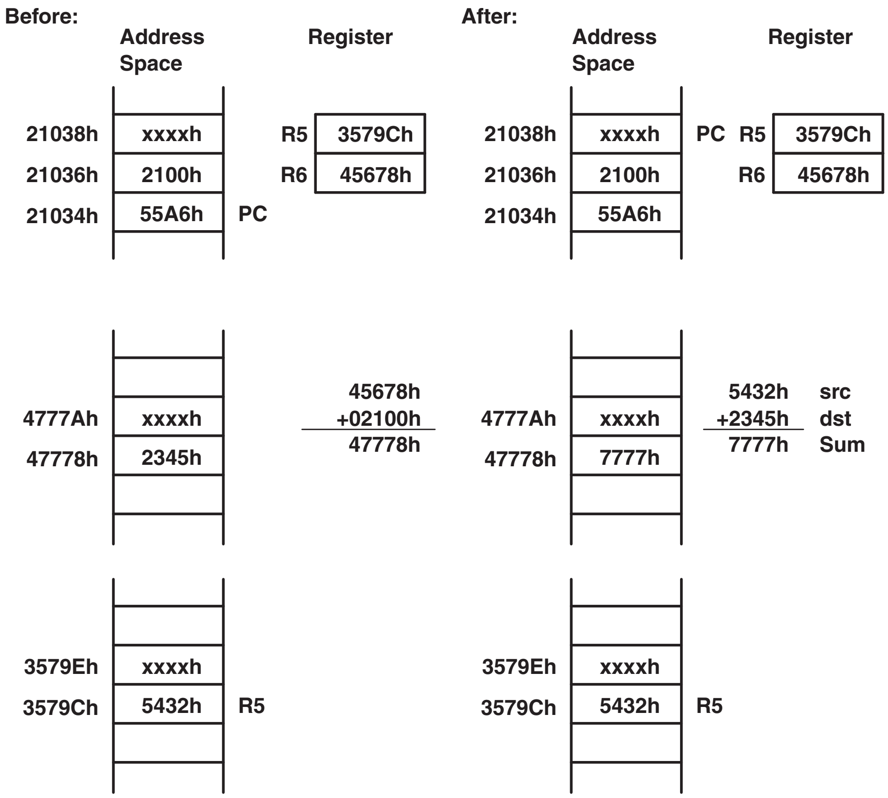
Length:	Three or four words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the index from 0 and inserts it.
Example:	`ADDX.A &EDE,&TONI ;`
	This instruction adds the 20-bit data contained in the absolute source and destination addresses and places the result into the destination.
Source:	Two words beginning with address EDE
Destination:	Two words beginning with address TONI



4.4.5 Indirect Register Mode

The Indirect Register mode uses the contents of the CPU register Rsrc as the source operand. The Indirect Register mode always uses a 20-bit address.

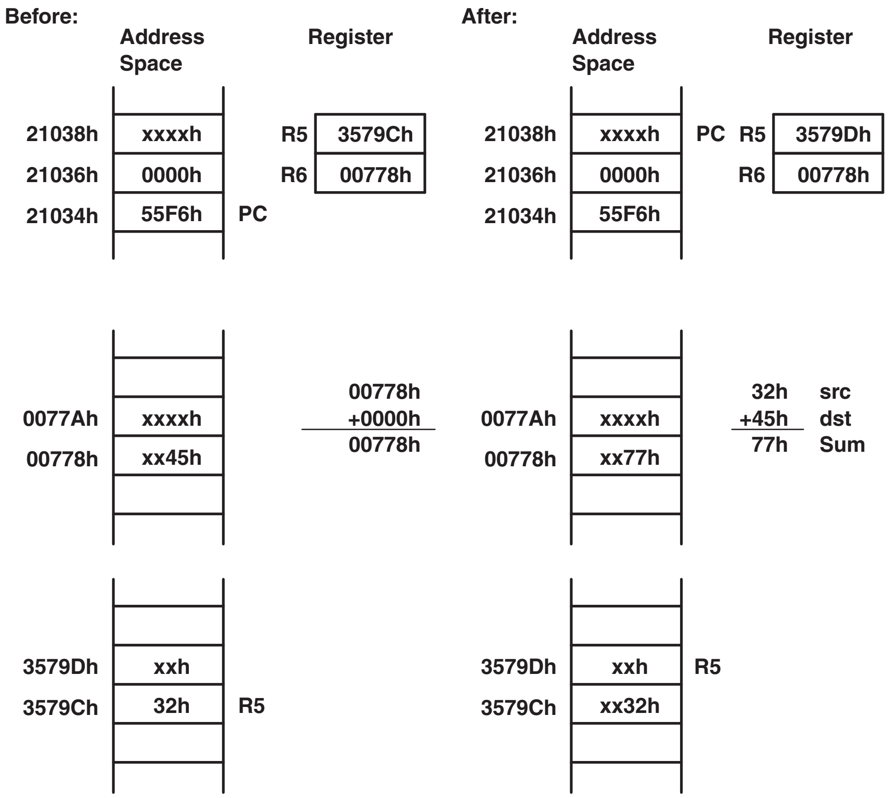
Length:	One, two, or three words
Operation:	The operand is the content the addressed memory location. The source register Rsrc is not modified.
Comment:	Valid only for the source operand. The substitute for the destination operand is 0(Rdst).
Example:	`ADDX.W @R5,2100h(R6)`
	This instruction adds the two 16-bit operands contained in the source and the destination addresses and places the result into the destination.
Source:	Word pointed to by R5. R5 contains address 3579Ch for this example.
Destination:	Word pointed to by R6 + 2100h, which results in address 45678h + 2100h = 7778h



4.4.6 Indirect Autoincrement Mode

The Indirect Autoincrement mode uses the contents of the CPU register Rsrc as the source operand. Rsrc is then automatically incremented by 1 for byte instructions, by 2 for word instructions, and by 4 for address-word instructions immediately after accessing the source operand. If the same register is used for source and destination, it contains the incremented address for the destination access. Indirect Autoincrement mode always uses 20-bit addresses.

Length:	One, two, or three words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid only for the source operand
Example:	`ADD.B @R5+,0(R6)`
	This instruction adds the 8-bit data contained in the source and the destination addresses and places the result into the destination.
Source:	Byte pointed to by R5. R5 contains address 3579Ch for this example.
Destination:	Byte pointed to by R6 + 0h, which results in address 0778h for this example



4.4.7 Immediate Mode

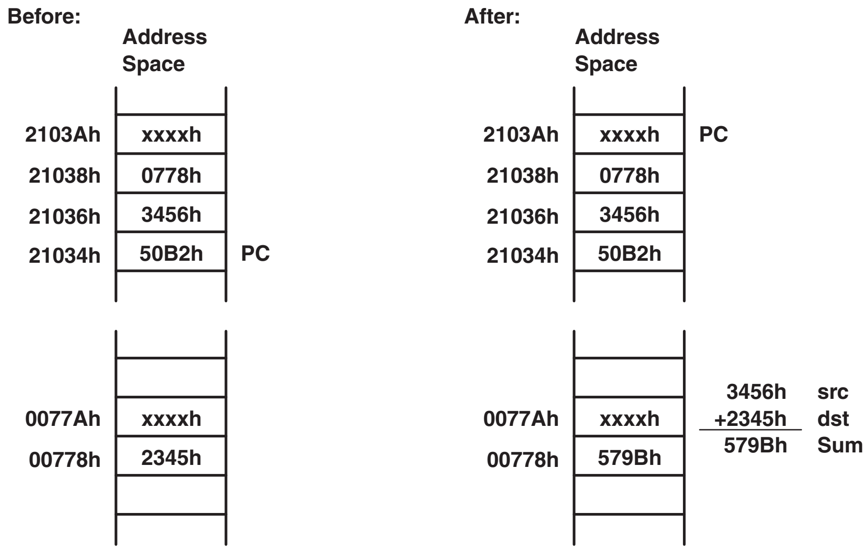
The Immediate mode allows accessing constants as operands by including the constant in the memory location following the instruction. The PC is used with the Indirect Autoincrement mode. The PC points to the immediate value contained in the next word. After the fetching of the immediate operand, the PC is incremented by 2 for byte, word, or address-word instructions. The Immediate mode has two addressing possibilities:

- 8-bit or 16-bit constants with MSP430 instructions
- 20-bit constants with MSP430X instruction

4.4.7.1 MSP430 Instructions With Immediate Mode

If an MSP430 instruction is used with Immediate addressing mode, the constant is an 8- or 16-bit value and is stored in the word following the instruction.

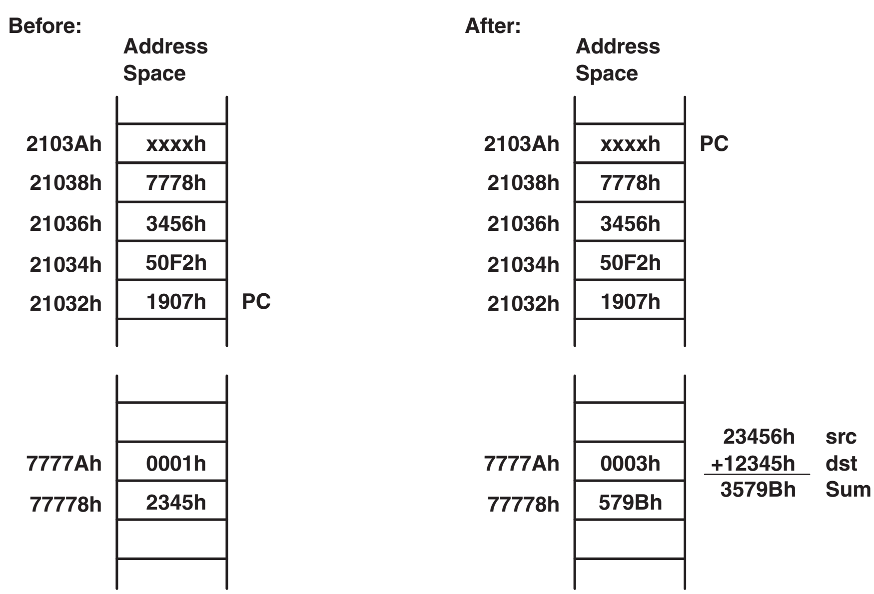
Length:	Two or three words. One word less if a constant of the constant generator can be used for the immediate operand.
Operation:	The 16-bit immediate source operand is used together with the 16-bit destination operand.
Comment:	Valid only for the source operand
Example:	`ADD #3456h,&TONI`
	This instruction adds the 16-bit immediate operand 3456h to the data in the destination address TONI.
Source:	16-bit immediate value 3456h
Destination:	Word at address TONI



4.4.7.2 MSP430X Instructions With Immediate Mode

If an MSP430X instruction is used with Immediate addressing mode, the constant is a 20-bit value. The 4 MSBs of the constant are stored in the extension word, and the 16 LSBs of the constant are stored in the word following the instruction.

Length:	Three or four words. One word less if a constant of the constant generator can be used for the immediate operand.
Operation:	The 20-bit immediate source operand is used together with the 20-bit destination operand.
Comment:	Valid only for the source operand
Example:	`ADDX.A #23456h,&TONI ;`
	This instruction adds the 20-bit immediate operand 23456h to the data in the destination address TONI.
Source:	20-bit immediate value 23456h
Destination:	Two words beginning with address TONI



4.5 MSP430 and MSP430X Instructions

MSP430 instructions are the 27 implemented instructions of the MSP430 CPU. These instructions are used throughout the 1-MB memory range unless their 16-bit capability is exceeded. The MSP430X instructions are used when the addressing of the operands, or the data length exceeds the 16-bit capability of the MSP430 instructions.

There are three possibilities when choosing between an MSP430 and MSP430X instruction:

- To use only the MSP430 instructions – The only exceptions are the CALLA and the RETA instruction. This can be done if a few, simple rules are met:
  - Placement of all constants, variables, arrays, tables, and data in the lower 64 KB. This allows the use of MSP430 instructions with 16-bit addressing for all data accesses. No pointers with 20-bit addresses are needed.
  - Placement of subroutine constants immediately after the subroutine code. This allows the use of the symbolic addressing mode with its 16-bit index to reach addresses within the range of PC + 32 KB.
- To use only MSP430X instructions – The disadvantages of this method are the reduced speed due to the additional CPU cycles and the increased program space due to the necessary extension word for any double operand instruction.
- Use the best fitting instruction where needed.

The following sections list and describe the MSP430 and MSP430X instructions.

4.5.1 MSP430 Instructions

The MSP430 instructions can be used, regardless if the program resides in the lower 64 KB or beyond it. The only exceptions are the instructions CALL and RET, which are limited to the lower 64-KB address range. CALLA and RETA instructions have been added to the MSP430X CPU to handle subroutines in the entire address range with no code size overhead.

4.5.1.1 MSP430 Double-Operand (Format I) Instructions

Figure 4-22 shows the format of the MSP430 double-operand instructions. Source and destination words are appended for the Indexed, Symbolic, Absolute, and Immediate modes. Table 4-4 lists the 12 MSP430 double-operand instructions.

15 – 12	11 – 8	7	6	5 – 4	0
Op-code	Rsrc	Ad	B/W	As	Rdst
Source or Destination 15:0					
Destination 15:0					

Figure 4-22. MSP430 Double-Operand Instruction Format

Table 4-4. MSP430 Double-Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits [(1)]			
			V	N	Z	C
MOV(.B)	src,dst	src → dst	–	–	–	–
ADD(.B)	src,dst	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	src + dst + C → dst	*	*	*	*
SUB(.B)	src,dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	dst + .not.src + C → dst	*	*	*	*
CMP(.B)	src,dst	dst - src	*	*	*	*
DADD(.B)	src,dst	src + dst + C → dst (decimally)	*	*	*	*
BIT(.B)	src,dst	src .and. dst	0	*	*	Z
BIC(.B)	src,dst	.not.src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	src .or. dst → dst	–	–	–	–
XOR(.B)	src,dst	src .xor. dst → dst	*	*	*	Z
AND(.B)	src,dst	src .and. dst → dst	0	*	*	Z

(1) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

4.5.1.2 MSP430 Single-Operand (Format II) Instructions

Figure 4-23 shows the format for MSP430 single-operand instructions, except RETI. The destination word is appended for the Indexed, Symbolic, Absolute, and Immediate modes. Table 4-5 lists the seven single-operand instructions.

15 … 7	6	5 4	… 0
Op-code	B/W	Ad	Rdst
Destination 15:0			

Figure 4-23. MSP430 Single-Operand Instructions

Table 4-5. MSP430 Single-Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits [(1)]			
			V	N	Z	C
RRC(.B)	dst	C → MSB →.......LSB → C	*	*	*	*
RRA(.B)	dst	MSB → MSB →....LSB → C	0	*	*	*
PUSH(.B)	src	SP - 2 → SP, src → SP	–	–	–	–
SWPB	dst	bit 15...bit 8 ↔ bit 7...bit 0	–	–	–	–
CALL	dst	Call subroutine in lower 64 KB	–	–	–	–
RETI		TOS → SR, SP + 2 → SP	*	*	*	*
		TOS → PC,SP + 2 → SP				
SXT	dst	Register mode: bit 7 → bit 8...bit 19 Other modes: bit 7 → bit 8...bit 15	0	*	*	Z

(1) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

4.5.1.3 Jump Instructions

Figure 4-24 shows the format for MSP430 and MSP430X jump instructions. The signed 10-bit word offset of the jump instruction is multiplied by two, sign-extended to a 20-bit address, and added to the 20-bit PC. This allows jumps in a range of –511 to +512 words relative to the PC in the full 20-bit address space. Jumps do not affect the status bits. Table 4-6 lists and describes the eight jump instructions.

15 – 13	12 – 10	9	8 – 0
Op-Code	Condition	S	10-Bit Signed PC Offset

Figure 4-24. Format of Conditional Jump Instructions

Table 4-6. Conditional Jump Instructions

Mnemonic	S-Reg, D-Reg	Operation
`JEQ/JZ`	Label	Jump to label if zero bit is set
`JNE/JNZ`	Label	Jump to label if zero bit is reset
`JC`	Label	Jump to label if carry bit is set
`JNC`	Label	Jump to label if carry bit is reset
`JN`	Label	Jump to label if negative bit is set
`JGE`	Label	Jump to label if (N .XOR. V) = 0
`JL`	Label	Jump to label if (N .XOR. V) = 1
`JMP`	Label	Jump to label unconditionally

4.5.1.4 Emulated Instructions

In addition to the MSP430 and MSP430X instructions, emulated instructions are instructions that make code easier to write and read, but do not have op-codes themselves. Instead, they are replaced automatically by the assembler with a core instruction. There is no code or performance penalty for using emulated instructions. The emulated instructions are listed in Table 4-7.

Table 4-7. Emulated Instructions

Instruction	Explanation	Emulation	Status Bits [(1)]			
			V	N	Z	C
`ADC(.B) dst`	Add Carry to dst	`ADDC(.B) #0,dst`	*	*	*	*
`BR dst`	Branch indirectly dst	`MOV dst,PC`	–	–	–	–
`CLR(.B) dst`	Clear dst	`MOV(.B) #0,dst`	–	–	–	–
`CLRC`	Clear Carry bit	`BIC #1,SR`	–	–	–	0
`CLRN`	Clear Negative bit	`BIC #4,SR`	–	0	–	–
`CLRZ`	Clear Zero bit	`BIC #2,SR`	–	–	0	–
`DADC(.B) dst`	Add Carry to dst decimally	`DADD(.B) #0,dst`	*	*	*	*
`DEC(.B) dst`	Decrement dst by 1	`SUB(.B) #1,dst`	*	*	*	*
`DECD(.B) dst`	Decrement dst by 2	`SUB(.B) #2,dst`	*	*	*	*
`DINT`	Disable interrupt	`BIC #8,SR`	–	–	–	–
`EINT`	Enable interrupt	`BIS #8,SR`	–	–	–	–
`INC(.B) dst`	Increment dst by 1	`ADD(.B) #1,dst`	*	*	*	*
`INCD(.B) dst`	Increment dst by 2	`ADD(.B) #2,dst`	*	*	*	*
`INV(.B) dst`	Invert dst	`XOR(.B) #–1,dst`	*	*	*	*

(1) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

Table 4-7. Emulated Instructions (continued)

Instruction	Explanation	Emulation	Status Bits [(1)] V	N	Z	C
`NOP`	No operation	`MOV R3,R3`	–	–	–	–
`POP dst`	Pop operand from stack	`MOV @SP+,dst`	–	–	–	–
`RET`	Return from subroutine	`MOV @SP+,PC`	–	–	–	–
`RLA(.B) dst`	Shift left dst arithmetically	`ADD(.B) dst,dst`	*	*	*	*
`RLC(.B) dst`	Shift left dst logically through Carry	`ADDC(.B) dst,dst`	*	*	*	*
`SBC(.B) dst`	Subtract Carry from dst	`SUBC(.B) #0,dst`	*	*	*	*
`SETC`	Set Carry bit	`BIS #1,SR`	–	–	–	1
`SETN`	Set Negative bit	`BIS #4,SR`	–	1	–	–
`SETZ`	Set Zero bit	`BIS #2,SR`	–	–	1	–
`TST(.B) dst`	Test dst (compare with 0)	`CMP(.B) #0,dst`	0	*	*	1

4.5.1.5 MSP430 Instruction Execution

The number of CPU clock cycles required for an instruction depends on the instruction format and the addressing modes used – not the instruction itself. The number of clock cycles refers to MCLK.

Instruction Cycles and Length for Interrupt, Reset, and Subroutines

Table 4-8 lists the length and the CPU cycles for reset, interrupts, and subroutines.

Table 4-8. Interrupt, Return, and Reset Cycles and Length

Action	Execution Time (MCLK Cycles)	Length of Instruction (Words)
Return from interrupt RETI	5	1
Return from subroutine RET	4	1
Interrupt request service (cycles needed before first instruction)	6	–
WDT reset	4	–
Reset ($\overline{RST}$/NMI)	4	–

Format II (Single-Operand) Instruction Cycles and Lengths

Table 4-9 lists the length and the CPU cycles for all addressing modes of the MSP430 single-operand instructions.

Table 4-9. MSP430 Format II Instruction Cycles and Length

Addressing Mode	No. of Cycles RRA, RRC SWPB, SXT	PUSH	CALL	Length of Instruction	Example
Rn	1	3	4	1	`SWPB R5`
@Rn	3	3	4	1	`RRC @R9`
@Rn+	3	3	4	1	`SWPB @R10+`
#N	N/A	3	4	2	`CALL #LABEL`
X(Rn)	4	4	5	2	`CALL 2(R7)`
EDE	4	4	5	2	`PUSH EDE`
&EDE	4	4	6	2	`SXT &EDE`

Jump Instructions Cycles and Lengths

All jump instructions require one code word and take two CPU cycles to execute, regardless of whether the jump is taken or not.

Format I (Double-Operand) Instruction Cycles and Lengths

Table 4-10 lists the length and CPU cycles for all addressing modes of the MSP430 Format I instructions.

Table 4-10. MSP430 Format I Instructions Cycles and Length

Addressing Mode		No. of Cycles	Length of Instruction	Example
Source	Destination			
Rn	Rm	1	1	`MOV R5,R8`
	PC	3	1	`BR R9`
	x(Rm)	4[1]	2	`ADD R5,4(R6)`
	EDE	4[1]	2	`XOR R8,EDE`
	&EDE	4[1]	2	`MOV R5,&EDE`
@Rn	Rm	2	1	`AND @R4,R5`
	PC	4	1	`BR @R8`
	x(Rm)	5[1]	2	`XOR @R5,8(R6)`
	EDE	5[1]	2	`MOV @R5,EDE`
	&EDE	5[1]	2	`XOR @R5,&EDE`
@Rn+	Rm	2	1	`ADD @R5+,R6`
	PC	4	1	`BR @R9+`
	x(Rm)	5[1]	2	`XOR @R5,8(R6)`
	EDE	5[1]	2	`MOV @R9+,EDE`
	&EDE	5[1]	2	`MOV @R9+,&EDE`
#N	Rm	2	2	`MOV #20,R9`
	PC	3	2	`BR #2AEh`
	x(Rm)	5[1]	3	`MOV #0300h,0(SP)`
	EDE	5[1]	3	`ADD #33,EDE`
	&EDE	5[1]	3	`ADD #33,&EDE`
x(Rn)	Rm	3	2	`MOV 2(R5),R7`
	PC	5	2	`BR 2(R6)`
	TONI	6[1]	3	`MOV 4(R7),TONI`
	x(Rm)	6[1]	3	`ADD 4(R4),6(R9)`
	&TONI	6[1]	3	`MOV 2(R4),&TONI`
EDE	Rm	3	2	`AND EDE,R6`
	PC	5	2	`BR EDE`
	TONI	6[1]	3	`CMP EDE,TONI`
	x(Rm)	6[1]	3	`MOV EDE,0(SP)`
	&TONI	6[1]	3	`MOV EDE,&TONI`
&EDE	Rm	3	2	`MOV &EDE,R8`
	PC	5	2	`BR &EDE`
	TONI	6[1]	3	`MOV &EDE,TONI`
	x(Rm)	6[1]	3	`MOV &EDE,0(SP)`
	&TONI	6[1]	3	`MOV &EDE,&TONI`

(1) MOV, BIT, and CMP instructions execute in one fewer cycle.

4.5.2 MSP430X Extended Instructions

The extended MSP430X instructions give the MSP430X CPU full access to its 20-bit address space. Most MSP430X instructions require an additional word of op-code called the extension word. Some extended instructions do not require an additional word and are noted in the instruction description. All addresses, indexes, and immediate numbers have 20-bit values when preceded by the extension word.

There are two types of extension words:

- Register/register mode for Format I instructions and register mode for Format II instructions
- Extension word for all other address mode combinations

4.5.2.1 Register Mode Extension Word

The register mode extension word is shown in Figure 4-25 and described in Table 4-11. An example is shown in Figure 4-27.

15–12	11	10–9	8	7	6	5	4	3–0
0001	1	00	ZC	#	A/L	0	0	(n-1)/Rn

Figure 4-25. Extension Word for Register Modes

Table 4-11. Description of the Extension Word Bits for Register Mode

Bit	Description
15:11	Extension word op-code. Op-codes 1800h to 1FFFh are extension words.
10:9	Reserved
ZC	Zero carry 0 The executed instruction uses the status of the carry bit C. 1 The executed instruction uses the carry bit as 0. The carry bit is defined by the result of the final operation after instruction execution.
#	Repetition 0 The number of instruction repetitions is set by extension word bits 3:0. 1 The number of 6instructions repetitions is defined by the value of the four LSBs of Rn. See description for bits 3:0.
A/L	Data length extension. Together with the B/W bits of the following MSP430 instruction, the AL bit defines the used data length of the instruction. **A/L B/W Comment** 0 0 Reserved 0 1 20-bit address word 1 0 16-bit word 1 1 8-bit byte
5:4	Reserved
3:0	Repetition count # = 0 These four bits set the repetition count n. These bits contain n – 1. # = 1 These four bits define the CPU register whose bits 3:0 set the number of repetitions. Rn.3:0 contain n – 1.

4.5.2.2 Non-Register Mode Extension Word

The extension word for non-register modes is shown in Figure 4-26 and described in Table 4-12. An example is shown in Figure 4-28.

15	14	13	12	11	10–7	6	5	4	3–0
0	0	0	1	1	Source bits 19:16	A/L	0	0	Destination bits 19:16

Figure 4-26. Extension Word for Non-Register Modes

Table 4-12. Description of Extension Word Bits for Non-Register Modes

Bit	Description
15:11	Extension word op-code. Op-codes 1800h to 1FFFh are extension words.
Source Bits 19:16	The four MSBs of the 20-bit source. Depending on the source addressing mode, these four MSBs may belong to an immediate operand, an index or to an absolute address.
A/L	Data length extension. Together with the B/W bits of the following MSP430 instruction, the AL bit defines the used data length of the instruction. **A/L B/W Comment** 0 0 Reserved 0 1 20-bit address word 1 0 16-bit word 1 1 8-bit byte
5:4	Reserved
Destination Bits 19:16	The four MSBs of the 20-bit destination. Depending on the destination addressing mode, these four MSBs may belong to an index or to an absolute address.

NOTE: B/W and A/L bit settings for SWPBX and SXTX

A/L	B/W	
0	0	SWPBX.A, SXTX.A
0	1	N/A
1	0	SWPB.W, SXTX.W
1	1	N/A

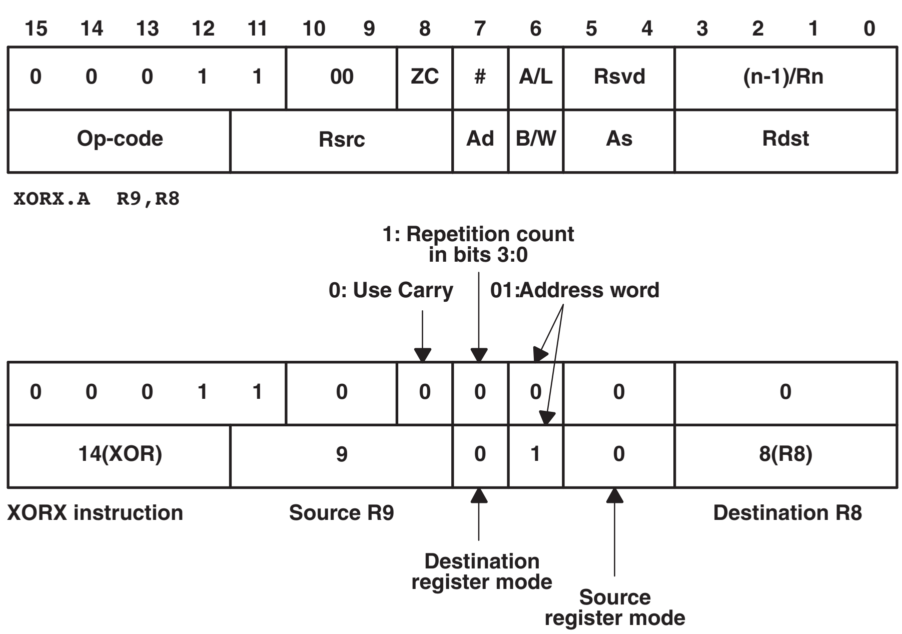


Figure 4-27. Example for Extended Register/Register Instruction

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	Source 19:16				A/L	Rsvd		Destination 19:16			
Op-code				Rsrc				Ad	B/W	As		Rdst			
Source 15:0															
Destination 15:0															

```
XORX.A #12345h, 45678h(R15)
```

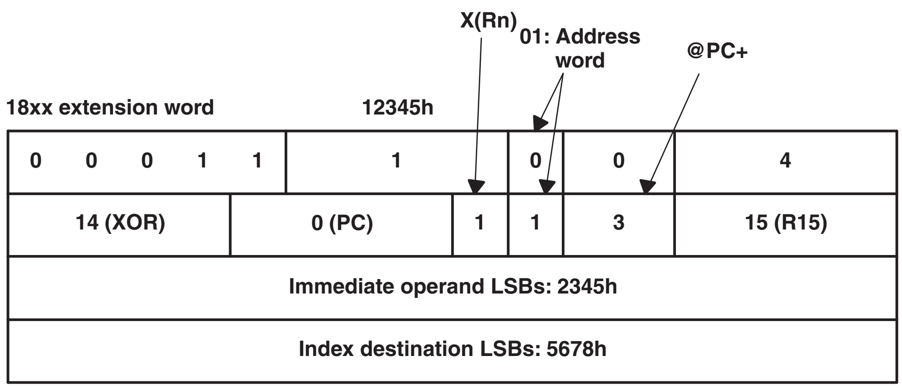


Figure 4-28. Example for Extended Immediate/Indexed Instruction

4.5.2.3 Extended Double-Operand (Format I) Instructions

All 12 double-operand instructions have extended versions as listed in Table 4-13.

Table 4-13. Extended Double-Operand Instructions

Mnemonic	Operands	Operation	Status Bits [(1)]			
			V	N	Z	C
`MOVX(.B,.A)`	src,dst	src → dst	–	–	–	–
`ADDX(.B,.A)`	src,dst	src + dst → dst	*	*	*	*
`ADDCX(.B,.A)`	src,dst	src + dst + C → dst	*	*	*	*
`SUBX(.B,.A)`	src,dst	dst + .not.src + 1 → dst	*	*	*	*
`SUBCX(.B,.A)`	src,dst	dst + .not.src + C → dst	*	*	*	*
`CMPX(.B,.A)`	src,dst	dst – src	*	*	*	*
`DADDX(.B,.A)`	src,dst	src + dst + C → dst (decimal)	*	*	*	*
`BITX(.B,.A)`	src,dst	src .and. dst	0	*	*	Z
`BICX(.B,.A)`	src,dst	.not.src .and. dst → dst	–	–	–	–
`BISX(.B,.A)`	src,dst	src .or. dst → dst	–	–	–	–
`XORX(.B,.A)`	src,dst	src .xor. dst → dst	*	*	*	Z
`ANDX(.B,.A)`	src,dst	src .and. dst → dst	0	*	*	Z

(1) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

The four possible addressing combinations for the extension word for Format I instructions are shown in Figure 4-29.

15	14	13	12	11	10	9	8	7	6	5	4	3			0
0	0	0	1	1	0	0	ZC	#	A/L	0	0	n-1/Rn			
Op-code				src				0	B/W	0	0	dst			

0	0	0	1	1	src.19:16				A/L	0	0	0	0	0	0
Op-code				src				Ad	B/W	As		dst			
src.15:0															

0	0	0	1	1	0	0	0	0	A/L	0	0	dst.19:16			
Op-code				src				Ad	B/W	As		dst			
dst.15:0															

0	0	0	1	1	src.19:16				A/L	0	0	dst.19:16			
Op-code				src				Ad	B/W	As		dst			
src.15:0															
dst.15:0															

Figure 4-29. Extended Format I Instruction Formats

If the 20-bit address of a source or destination operand is located in memory, not in a CPU register, then two words are used for this operand as shown in Figure 4-30.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Address+2	0 0												19:16			
Address	Operand LSBs 15:0															

Figure 4-30. 20-Bit Addresses in Memory

4.5.2.4 Extended Single-Operand (Format II) Instructions

Extended MSP430X Format II instructions are listed in Table 4-14.

Table 4-14. Extended Single-Operand Instructions

Mnemonic	Operands	Operation	n	Status Bits [(1)]			
				V	N	Z	C
CALLA	dst	Call indirect to subroutine (20-bit address)		–	–	–	–
POPM.A	#n,Rdst	Pop n 20-bit registers from stack	1 to 16	*	*	*	*
POPM.W	#n,Rdst	Pop n 16-bit registers from stack	1 to 16	*	*	*	*
PUSHM.A	#n,Rsrc	Push n 20-bit registers to stack	1 to 16	*	*	*	*
PUSHM.W	#n,Rsrc	Push n 16-bit registers to stack	1 to 16	*	*	*	*
PUSHX(.B,.A)	src	Push 8/16/20-bit source to stack		*	*	*	*
RRCM(.A)	#n,Rdst	Rotate right Rdst n bits through carry (16-/20-bit register)	1 to 4	*	*	*	*
RRUM(.A)	#n,Rdst	Rotate right Rdst n bits unsigned (16-/20-bit register)	1 to 4	0	*	*	Z
RRAM(.A)	#n,Rdst	Rotate right Rdst n bits arithmetically (16-/20-bit register)	1 to 4	*	*	*	*
RLAM(.A)	#n,Rdst	Rotate left Rdst n bits arithmetically (16-/20-bit register)	1 to 4	*	*	*	*
RRCX(.B,.A)	dst	Rotate right dst through carry (8-/16-/20-bit data)	1	*	*	*	Z
RRUX(.B,.A)	Rdst	Rotate right dst unsigned (8-/16-/20-bit)	1	0	*	*	Z
RRAX(.B,.A)	dst	Rotate right dst arithmetically	1				
SWPBX(.A)	dst	Exchange low byte with high byte	1				
SXTX(.A)	Rdst	Bit7 → bit8 ... bit19	1				
SXTX(.A)	dst	Bit7 → bit8 ... MSB	1				

(1) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

The three possible addressing mode combinations for Format II instructions are shown in Figure 4-31.

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th></th><th></th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>ZC</td><td>#</td><td>A/L</td><td>0</td><td>0</td><td colspan="4">n-1/Rn</td></tr>
<tr><td colspan="9">Op-code</td><td>B/W</td><td>0</td><td>0</td><td colspan="4">dst</td></tr>
</table>

<table>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>A/L</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="9">Op-code</td><td>B/W</td><td>1</td><td>x</td><td colspan="4">dst</td></tr>
</table>

<table>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>A/L</td><td>0</td><td>0</td><td colspan="4">dst.19:16</td></tr>
<tr><td colspan="9">Op-code</td><td>B/W</td><td>x</td><td>1</td><td colspan="4">dst</td></tr>
<tr><td colspan="16">dst.15:0</td></tr>
</table>

Figure 4-31. Extended Format II Instruction Format

Extended Format II Instruction Format Exceptions

Exceptions for the Format II instruction formats are shown in Figure 4-32 through Figure 4-35.

15 ... 8	7 ... 4	3 ... 0
Op-code	n-1	Rdst - n+1

Figure 4-32. PUSHM/POPM Instruction Format

15 ... 12	11 ... 10	9 ... 4	3 ... 0
C	n-1	Op-code	Rdst

Figure 4-33. RRCM, RRAM, RRUM, and RLAM Instruction Format

15 ... 12	11 ... 8	7 ... 4	3 ... 0
C	Rsrc	Op-code	0(PC)

C	#imm/abs19:16	Op-code	0(PC)
#imm15:0 / &abs15:0			

C	Rsrc	Op-code	0(PC)
index15:0			

Figure 4-34. BRA Instruction Format

15 ... 4	3 ... 0
Op-code	Rdst

Op-code	Rdst
index15:0	

Op-code	#imm/ix/abs19:16
#imm15:0 / index15:0 / &abs15:0	

Figure 4-35. CALLA Instruction Format

4.5.2.5 Extended Emulated Instructions

The extended instructions together with the constant generator form the extended emulated instructions. Table 4-15 lists the emulated instructions.

Table 4-15. Extended Emulated Instructions

Instruction	Explanation	Emulation
`ADCX(.B,.A) dst`	Add carry to dst	`ADDCX(.B,.A) #0,dst`
`BRA dst`	Branch indirect dst	`MOVA dst,PC`
`RETA`	Return from subroutine	`MOVA @SP+,PC`
`CLRA Rdst`	Clear Rdst	`MOV #0,Rdst`
`CLRX(.B,.A) dst`	Clear dst	`MOVX(.B,.A) #0,dst`
`DADCX(.B,.A) dst`	Add carry to dst decimally	`DADDX(.B,.A) #0,dst`
`DECX(.B,.A) dst`	Decrement dst by 1	`SUBX(.B,.A) #1,dst`
`DECDA Rdst`	Decrement Rdst by 2	`SUBA #2,Rdst`
`DECDX(.B,.A) dst`	Decrement dst by 2	`SUBX(.B,.A) #2,dst`
`INCX(.B,.A) dst`	Increment dst by 1	`ADDX(.B,.A) #1,dst`
`INCDA Rdst`	Increment Rdst by 2	`ADDA #2,Rdst`
`INCDX(.B,.A) dst`	Increment dst by 2	`ADDX(.B,.A) #2,dst`
`INVX(.B,.A) dst`	Invert dst	`XORX(.B,.A) #-1,dst`
`RLAX(.B,.A) dst`	Shift left dst arithmetically	`ADDX(.B,.A) dst,dst`
`RLCX(.B,.A) dst`	Shift left dst logically through carry	`ADDCX(.B,.A) dst,dst`
`SBCX(.B,.A) dst`	Subtract carry from dst	`SUBCX(.B,.A) #0,dst`
`TSTA Rdst`	Test Rdst (compare with 0)	`CMPA #0,Rdst`
`TSTX(.B,.A) dst`	Test dst (compare with 0)	`CMPX(.B,.A) #0,dst`
`POPX dst`	Pop to dst	`MOVX(.B, .A) @SP+,dst`

4.5.2.6 MSP430X Address Instructions

MSP430X address instructions are instructions that support 20-bit operands but have restricted addressing modes. The addressing modes are restricted to the Register mode and the Immediate mode, except for the MOVA instruction as listed in Table 4-16. Restricting the addressing modes removes the need for the additional extension-word op-code improving code density and execution time. Address instructions should be used any time an MSP430X instruction is needed with the corresponding restricted addressing mode.

Table 4-16. Address Instructions, Operate on 20-Bit Register Data

Mnemonic	Operands	Operation	Status Bits [(1)]			
			V	N	Z	C
ADDA	Rsrc,Rdst	Add source to destination register	*	*	*	*
	#imm20,Rdst					
MOVA	Rsrc,Rdst	Move source to destination	–	–	–	–
	#imm20,Rdst					
	z16(Rsrc),Rdst					
	EDE,Rdst					
	&abs20,Rdst					
	@Rsrc,Rdst					
	@Rsrc+,Rdst					
	Rsrc,z16(Rdst)					
	Rsrc,&abs20					
CMPA	Rsrc,Rdst	Compare source to destination register	*	*	*	*
	#imm20,Rdst					
SUBA	Rsrc,Rdst	Subtract source from destination register	*	*	*	*
	#imm20,Rdst					

(1) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

4.5.2.7 MSP430X Instruction Execution

The number of CPU clock cycles required for an MSP430X instruction depends on the instruction format and the addressing modes used, not the instruction itself. The number of clock cycles refers to MCLK.

MSP430X Format II (Single-Operand) Instruction Cycles and Lengths

Table 4-17 lists the length and the CPU cycles for all addressing modes of the MSP430X extended single-operand instructions.

Table 4-17. MSP430X Format II Instruction Cycles and Length

Instruction	Execution Cycles/Length of Instruction (Words)						
	Rn	@Rn	@Rn+	#N	X(Rn)	EDE	&EDE
RRAM	n/1	–	–	–	–	–	–
RRCM	n/1	–	–	–	–	–	–
RRUM	n/1	–	–	–	–	–	–
RLAM	n/1	–	–	–	–	–	–
PUSHM	2+n/1	–	–	–	–	–	–
PUSHM.A	2+2n/1	–	–	–	–	–	–
POPM	2+n/1	–	–	–	–	–	–
POPM.A	2+2n/1	–	–	–	–	–	–
CALLA	5/1	6/1	6/1	5/2	5[1]/2	7/2	7/2
RRAX(.B)	1+n/2	4/2	4/2	–	5/3	5/3	5/3
RRAX.A	1+n/2	6/2	6/2	–	7/3	7/3	7/3
RRCX(.B)	1+n/2	4/2	4/2	–	5/3	5/3	5/3
RRCX.A	1+n/2	6/2	6/2	–	7/3	7/3	7/3
PUSHX(.B)	4/2	4/2	4/2	4/3	5[1]/3	5/3	5/3
PUSHX.A	5/2	6/2	6/2	5/3	7[1]/3	7/3	7/3
POPX(.B)	3/2	–	–	–	5/3	5/3	5/3
POPX.A	4/2	–	–	–	7/3	7/3	7/3

(1) Add one cycle when Rn = SP

MSP430X Format I (Double-Operand) Instruction Cycles and Lengths

Table 4-18 lists the length and CPU cycles for all addressing modes of the MSP430X extended Format I instructions.

Table 4-18. MSP430X Format I Instruction Cycles and Length

Addressing Mode		No. of Cycles		Length of Instruction	Examples
Source	Destination	.B/.W	.A	.B/.W/.A	
Rn	Rm[1]	2	2	2	`BITX.B R5,R8`
	PC	4	4	2	`ADDX R9,PC`
	x(Rm)	5[2]	7[3]	3	`ANDX.A R5,4(R6)`
	EDE	5[2]	7[3]	3	`XORX R8,EDE`
	&EDE	5[2]	7[3]	3	`BITX.W R5,&EDE`
@Rn	Rm	3	4	2	`BITX @R5,R8`
	PC	5	6	2	`ADDX @R9,PC`
	x(Rm)	6[2]	9[3]	3	`ANDX.A @R5,4(R6)`
	EDE	6[2]	9[3]	3	`XORX @R8,EDE`
	&EDE	6[2]	9[3]	3	`BITX.B @R5,&EDE`
@Rn+	Rm	3	4	2	`BITX @R5+,R8`
	PC	5	6	2	`ADDX.A @R9+,PC`
	x(Rm)	6[2]	9[3]	3	`ANDX @R5+,4(R6)`
	EDE	6[2]	9[3]	3	`XORX.B @R8+,EDE`
	&EDE	6[2]	9[3]	3	`BITX @R5+,&EDE`
#N	Rm	3	3	3	`BITX #20,R8`
	PC[4]	4	4	3	`ADDX.A #FE000h,PC`
	x(Rm)	6[2]	8[3]	4	`ANDX #1234,4(R6)`
	EDE	6[2]	8[3]	4	`XORX #A5A5h,EDE`
	&EDE	6[2]	8[3]	4	`BITX.B #12,&EDE`
x(Rn)	Rm	4	5	3	`BITX 2(R5),R8`
	PC[4]	6	7	3	`SUBX.A 2(R6),PC`
	TONI	7[2]	10[3]	4	`ANDX 4(R7),4(R6)`
	x(Rm)	7[2]	10[3]	4	`XORX.B 2(R6),EDE`
	&TONI	7[2]	10[3]	4	`BITX 8(SP),&EDE`
EDE	Rm	4	5	3	`BITX.B EDE,R8`
	PC[4]	6	7	3	`ADDX.A EDE,PC`
	TONI	7[2]	10[3]	4	`ANDX EDE,4(R6)`
	x(Rm)	7[2]	10[3]	4	`ANDX EDE,TONI`
	&TONI	7[2]	10[3]	4	`BITX EDE,&TONI`
&EDE	Rm	4	5	3	`BITX &EDE,R8`
	PC[4]	6	7	3	`ADDX.A &EDE,PC`
	TONI	7[2]	10[3]	4	`ANDX.B &EDE,4(R6)`
	x(Rm)	7[2]	10[3]	4	`XORX &EDE,TONI`
	&TONI	7[2]	10[3]	4	`BITX &EDE,&TONI`

(1) Repeat instructions require n + 1 cycles, where n is the number of times the instruction is executed.
(2) Reduce the cycle count by one for MOV, BIT, and CMP instructions.
(3) Reduce the cycle count by two for MOV, BIT, and CMP instructions.
(4) Reduce the cycle count by one for MOV, ADD, and SUB instructions.

MSP430X Address Instruction Cycles and Lengths

Table 4-19 lists the length and the CPU cycles for all addressing modes of the MSP430X address instructions.

Table 4-19. Address Instruction Cycles and Length

Addressing Mode		Execution Time (MCLK Cycles)		Length of Instruction (Words)		Example
Source	Destination	MOVA BRA	CMPA ADDA SUBA	MOVA	CMPA ADDA SUBA	
Rn	Rn	1	1	1	1	`CMPA R5,R8`
	PC	3	3	1	1	`SUBA R9,PC`
	x(Rm)	4	–	2	–	`MOVA R5,4(R6)`
	EDE	4	–	2	–	`MOVA R8,EDE`
	&EDE	4	–	2	–	`MOVA R5,&EDE`
@Rn	Rm	3	–	1	–	`MOVA @R5,R8`
	PC	5	–	1	–	`MOVA @R9,PC`
@Rn+	Rm	3	–	1	–	`MOVA @R5+,R8`
	PC	5	–	1	–	`MOVA @R9+,PC`
#N	Rm	2	3	2	2	`CMPA #20,R8`
	PC	3	3	2	2	`SUBA #FE000h,PC`
x(Rn)	Rm	4	–	2	–	`MOVA 2(R5),R8`
	PC	6	–	2	–	`MOVA 2(R6),PC`
EDE	Rm	4	–	2	–	`MOVA EDE,R8`
	PC	6	–	2	–	`MOVA EDE,PC`
&EDE	Rm	4	–	2	–	`MOVA &EDE,R8`
	PC	6	–	2	–	`MOVA &EDE,PC`

4.6 Instruction Set Description

Table 4-20 shows all available instructions:

Table 4-20. Instruction Map of MSP430X

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx	MOVA, CMPA, ADDA, SUBA, RRCM, RRAM, RLAM, RRUM															
10xx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI	CALLA		
14xx	PUSHM.A, POPM.A, PUSHM.W, POPM.W															
18xx 1Cxx	Extension word for Format I and Format II instructions															
20xx	JNE/JNZ															
24xx	JEQ/JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															

4.6.1 Extended Instruction Binary Descriptions

Detailed MSP430X instruction binary descriptions are shown in the following tables.

Instruction	Instruction Group				src or data.19:16	Instruction Identifier				dst	
	15			12	11 8	7			4	3 0	
MOVA	0	0	0	0	src	0	0	0	0	dst	MOVA @Rsrc,Rdst
	0	0	0	0	src	0	0	0	1	dst	MOVA @Rsrc+,Rdst
	0	0	0	0	&abs.19:16	0	0	1	0	dst	MOVA &abs20,Rdst
	&abs.15:0										
	0	0	0	0	src	0	0	1	1	dst	MOVA x(Rsrc),Rdst
	x.15:0										±15-bit index x
	0	0	0	0	src	0	1	1	0	&abs.19:16	MOVA Rsrc,&abs20
	&abs.15:0										
	0	0	0	0	src	0	1	1	1	dst	MOVA Rsrc,X(Rdst)
	x.15:0										±15-bit index x
	0	0	0	0	imm.19:16	1	0	0	0	dst	MOVA #imm20,Rdst
	imm.15:0										
CMPA	0	0	0	0	imm.19:16	1	0	0	1	dst	CMPA #imm20,Rdst
	imm.15:0										
ADDA	0	0	0	0	imm.19:16	1	0	1	0	dst	ADDA #imm20,Rdst
	imm.15:0										
SUBA	0	0	0	0	imm.19:16	1	0	1	1	dst	SUBA #imm20,Rdst
	imm.15:0										
MOVA	0	0	0	0	src	1	1	0	0	dst	MOVA Rsrc,Rdst
CMPA	0	0	0	0	src	1	1	0	1	dst	CMPA Rsrc,Rdst
ADDA	0	0	0	0	src	1	1	1	0	dst	ADDA Rsrc,Rdst
SUBA	0	0	0	0	src	1	1	1	1	dst	SUBA Rsrc,Rdst

Instruction	Instruction Group				Bit Loc.	Inst. ID		Instruction Identifier				dst	
	15			12	11 10	9	8	7			4	3 0	
RRCM.A	0	0	0	0	n – 1	0	0	0	1	0	0	dst	RRCM.A #n,Rdst
RRAM.A	0	0	0	0	n – 1	0	1	0	1	0	0	dst	RRAM.A #n,Rdst
RLAM.A	0	0	0	0	n – 1	1	0	0	1	0	0	dst	RLAM.A #n,Rdst
RRUM.A	0	0	0	0	n – 1	1	1	0	1	0	0	dst	RRUM.A #n,Rdst
RRCM.W	0	0	0	0	n – 1	0	0	0	1	0	1	dst	RRCM.W #n,Rdst
RRAM.W	0	0	0	0	n – 1	0	1	0	1	0	1	dst	RRAM.W #n,Rdst
RLAM.W	0	0	0	0	n – 1	1	0	0	1	0	1	dst	RLAM.W #n,Rdst
RRUM.W	0	0	0	0	n – 1	1	1	0	1	0	1	dst	RRUM.W #n,Rdst

Instruction	Instruction Identifier												dst				
	15			12	11			8	7	6	5	4	3			0	
RETI	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	
CALLA	0	0	0	1	0	0	1	1	0	1	0	0	dst				CALLA Rdst
	0	0	0	1	0	0	1	1	0	1	0	1	dst				CALLA x(Rdst)
	x.15:0																
	0	0	0	1	0	0	1	1	0	1	1	0	dst				CALLA @Rdst
	0	0	0	1	0	0	1	1	0	1	1	1	dst				CALLA @Rdst+
	0	0	0	1	0	0	1	1	1	0	0	0	&abs.19:16				CALLA &abs20
	&abs.15:0																
	0	0	0	1	0	0	1	1	1	0	0	1	x.19:16				CALLA EDE
	x.15:0																CALLA x(PC)
	0	0	0	1	0	0	1	1	1	0	1	1	imm.19:16				CALLA #imm20
	imm.15:0																
Reserved	0	0	0	1	0	0	1	1	1	0	1	0	x	x	x	x	
Reserved	0	0	0	1	0	0	1	1	1	1	x	x	x	x	x	x	
PUSHM.A	0	0	0	1	0	1	0	0	n – 1				dst				PUSHM.A #n,Rdst
PUSHM.W	0	0	0	1	0	1	0	1	n – 1				dst				PUSHM.W #n,Rdst
POPM.A	0	0	0	1	0	1	1	0	n – 1				dst – n + 1				POPM.A #n,Rdst
POPM.W	0	0	0	1	0	1	1	1	n – 1				dst – n + 1				POPM.W #n,Rdst

4.6.2 MSP430 Instructions

The MSP430 instructions are listed and described on the following pages.

*** ADC[.W]** Add carry to destination
*** ADC.B** Add carry to destination

Syntax `ADC dst` or `ADC.W dst`
`ADC.B dst`

Operation dst + C → dst

Emulation `ADDC #0,dst`
`ADDC.B #0,dst`

Description The carry bit (C) is added to the destination operand. The previous contents of the destination are lost.

Status Bits
- N: Set if result is negative, reset if positive
- Z: Set if result is zero, reset otherwise
- C: Set if dst was incremented from 0FFFFh to 0000, reset otherwise
  Set if dst was incremented from 0FFh to 00, reset otherwise
- V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 16-bit counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
ADD     @R13,0(R12)      ; Add LSDs
ADC     2(R12)           ; Add carry to MSD
```

Example The 8-bit counter pointed to by R13 is added to a 16-bit counter pointed to by R12.

```
ADD.B   @R13,0(R12)     ; Add LSDs
ADC.B   1(R12)          ; Add carry to MSD
```

ADD[.W] Add source word to destination word

ADD.B Add source byte to destination byte

Syntax `ADD src,dst` or `ADD.W src,dst`

`ADD.B src,dst`

Operation src + dst → dst

Description The source operand is added to the destination operand. The previous content of the destination is lost.

Status Bits
- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if there is a carry from the MSB of the result, reset otherwise
- V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Ten is added to the 16-bit counter CNTR located in lower 64 K.

```
ADD.W   #10,&CNTR      ; Add 10 to 16-bit counter
```

Example A table word pointed to by R5 (20-bit address in R5) is added to R6. The jump to label TONI is performed on a carry.

```
ADD.W    @R5,R6        ; Add table word to R6. R6.19:16 = 0
JC       TONI          ; Jump if carry
...                    ; No carry
```

Example A table byte pointed to by R5 (20-bit address) is added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1. R6.19:8 = 0

```
ADD.B    @R5+,R6       ; Add byte to R6. R5 + 1. R6: 000xxh
JNC      TONI          ; Jump if no carry
...                    ; Carry occurred
```

ADDC[.W] Add source word and carry to destination word

ADDC.B Add source byte and carry to destination byte

Syntax `ADDC src,dst` or `ADDC.W src,dst`

`ADDC.B src,dst`

Operation src + dst + C → dst

Description The source operand and the carry bit C are added to the destination operand. The previous content of the destination is lost.

Status Bits

- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if there is a carry from the MSB of the result, reset otherwise
- V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Constant value 15 and the carry of the previous instruction are added to the 16-bit counter CNTR located in lower 64 K.

```
ADDC.W   #15,&CNTR      ; Add 15 + C to 16-bit CNTR
```

Example A table word pointed to by R5 (20-bit address) and the carry C are added to R6. The jump to label TONI is performed on a carry. R6.19:16 = 0

```
ADDC.W   @R5,R6         ; Add table word + C to R6
JC       TONI           ; Jump if carry
...                     ; No carry
```

Example A table byte pointed to by R5 (20-bit address) and the carry bit C are added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1. R6.19:8 = 0

```
ADDC.B   @R5+,R6        ; Add table byte + C to R6. R5 + 1
JNC      TONI           ; Jump if no carry
...                     ; Carry occurred
```

AND[.W] Logical AND of source word with destination word

AND.B Logical AND of source byte with destination byte

Syntax `AND src,dst` or `AND.W src,dst`

`AND.B src,dst`

Operation src .and. dst → dst

Description The source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected.

Status Bits
- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if the result is not zero, reset otherwise. C = (.not. Z)
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The bits set in R5 (16-bit data) are used as a mask (AA55h) for the word TOM located in the lower 64 K. If the result is zero, a branch is taken to label TONI. R5.19:16 = 0

```
MOV     #AA55h,R5      ; Load 16-bit mask to R5
AND     R5,&TOM        ; TOM .and. R5 -> TOM
JZ      TONI           ; Jump if result 0
...                    ; Result > 0
```

or shorter:

```
AND     #AA55h,&TOM    ; TOM .and. AA55h -> TOM
JZ      TONI           ; Jump if result 0
```

Example A table byte pointed to by R5 (20-bit address) is logically ANDed with R6. R5 is incremented by 1 after the fetching of the byte. R6.19:8 = 0

```
AND.B   @R5+,R6        ; AND table byte with R6. R5 + 1
```

BIC[.W] Clear bits set in source word in destination word

BIC.B Clear bits set in source byte in destination byte

Syntax `BIC src,dst` or `BIC.W src,dst`

`BIC.B src,dst`

Operation (.not. src) .and. dst → dst

Description The inverted source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected.

Status Bits
- N: Not affected
- Z: Not affected
- C: Not affected
- V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The bits 15:14 of R5 (16-bit data) are cleared. R5.19:16 = 0

```
BIC     #0C000h,R5     ; Clear R5.19:14 bits
```

Example A table word pointed to by R5 (20-bit address) is used to clear bits in R7. R7.19:16 = 0

```
BIC.W   @R5,R7         ; Clear bits in R7 set in @R5
```

Example A table byte pointed to by R5 (20-bit address) is used to clear bits in Port1.

```
BIC.B   @R5,&P1OUT     ; Clear I/O port P1 bits set in @R5
```

BIS[.W] Set bits set in source word in destination word

BIS.B Set bits set in source byte in destination byte

Syntax `BIS src,dst` or `BIS.W src,dst`

`BIS.B src,dst`

Operation src .or. dst → dst

Description The source operand and the destination operand are logically ORed. The result is placed into the destination. The source operand is not affected.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Bits 15 and 13 of R5 (16-bit data) are set to one. R5.19:16 = 0

```
BIS      #A000h,R5       ; Set R5 bits
```

Example A table word pointed to by R5 (20-bit address) is used to set bits in R7. R7.19:16 = 0

```
BIS.W   @R5,R7           ; Set bits in R7
```

Example A table byte pointed to by R5 (20-bit address) is used to set bits in Port1. R5 is incremented by 1 afterwards.

```
BIS.B   @R5+,&P1OUT      ; Set I/O port P1 bits. R5 + 1
```

BIT[.W] Test bits set in source word in destination word

BIT.B Test bits set in source byte in destination byte

Syntax `BIT src,dst` or `BIT.W src,dst`

`BIT.B src,dst`

Operation src .and. dst

Description The source operand and the destination operand are logically ANDed. The result affects only the status bits in SR.

Register mode: the register bits Rdst.19:16 (.W) resp. Rdst. 19:8 (.B) are not cleared!

Status Bits
- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if the result is not zero, reset otherwise. C = (.not. Z)
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Test if one (or both) of bits 15 and 14 of R5 (16-bit data) is set. Jump to label TONI if this is the case. R5.19:16 are not affected.

```
BIT     #C000h,R5        ; Test R5.15:14 bits
JNZ     TONI             ; At least one bit is set in R5
...                      ; Both bits are reset
```

Example A table word pointed to by R5 (20-bit address) is used to test bits in R7. Jump to label TONI if at least one bit is set. R7.19:16 are not affected.

```
BIT.W   @R5,R7           ; Test bits in R7
JC      TONI             ; At least one bit is set
...                      ; Both are reset
```

Example A table byte pointed to by R5 (20-bit address) is used to test bits in output Port1. Jump to label TONI if no bit is set. The next table byte is addressed.

```
BIT.B   @R5+,&P1OUT      ; Test I/O port P1 bits. R5 + 1
JNC     TONI             ; No corresponding bit is set
...                      ; At least one bit is set
```

*** BR, BRANCH** Branch to destination in lower 64K address space

Syntax `BR dst`

Operation dst → PC

Emulation `MOV dst,PC`

Description An unconditional branch is taken to an address anywhere in the lower 64K address space. All source addressing modes can be used. The branch instruction is a word instruction.

Status Bits Status bits are not affected.

Example Examples for all addressing modes are given.

```
BR     #EXEC     ; Branch to label EXEC or direct branch (e.g. #0A4h)
                 ; Core instruction MOV @PC+,PC

BR     EXEC      ; Branch to the address contained in EXEC
                 ; Core instruction MOV X(PC),PC
                 ; Indirect address

BR     &EXEC     ; Branch to the address contained in absolute
                 ; address EXEC
                 ; Core instruction MOV X(0),PC
                 ; Indirect address

BR     R5        ; Branch to the address contained in R5
                 ; Core instruction MOV R5,PC
                 ; Indirect R5

BR     @R5       ; Branch to the address contained in the word
                 ; pointed to by R5.
                 ; Core instruction MOV @R5,PC
                 ; Indirect, indirect R5

BR     @R5+      ; Branch to the address contained in the word pointed
                 ; to by R5 and increment pointer in R5 afterwards.
                 ; The next time-S/W flow uses R5 pointer-it can
                 ; alter program execution due to access to
                 ; next address in a table pointed to by R5
                 ; Core instruction MOV @R5,PC
                 ; Indirect, indirect R5 with autoincrement

BR     X(R5)     ; Branch to the address contained in the address
                 ; pointed to by R5 + X (e.g. table with address
                 ; starting at X). X can be an address or a label
                 ; Core instruction MOV X(R5),PC
                 ; Indirect, indirect R5 + X
```

CALL Call a subroutine in lower 64 K

Syntax `CALL dst`

Operation

dst → PC 16-bit dst is evaluated and stored

SP – 2 → SP

PC → @SP updated PC with return address to TOS

tmp → PC saved 16-bit dst to PC

Description A subroutine call is made from an address in the lower 64 K to a subroutine address in the lower 64 K. All seven source addressing modes can be used. The call instruction is a word instruction. The return is made with the RET instruction.

Status Bits Status bits are not affected.
PC.19:16 cleared (address in lower 64 K)

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Examples Examples for all addressing modes are given.

Immediate Mode: Call a subroutine at label EXEC (lower 64 K) or call directly to address.

```
CALL    #EXEC               ; Start address EXEC
CALL    #0AA04h             ; Start address 0AA04h
```

Symbolic Mode: Call a subroutine at the 16-bit address contained in address EXEC. EXEC is located at the address (PC + X) where X is within PC + 32 K.

```
CALL    EXEC                ; Start address at @EXEC. z16(PC)
```

Absolute Mode: Call a subroutine at the 16-bit address contained in absolute address EXEC in the lower 64 K.

```
CALL    &EXEC               ; Start address at @EXEC
```

Register mode: Call a subroutine at the 16-bit address contained in register R5.15:0.

```
CALL    R5                  ; Start address at R5
```

Indirect Mode: Call a subroutine at the 16-bit address contained in the word pointed to by register R5 (20-bit address).

```
CALL    @R5                 ; Start address at @R5
```

*** CLR[.W]** Clear destination
*** CLR.B** Clear destination

Syntax `CLR dst` or `CLR.W dst`
`CLR.B dst`

Operation 0 → dst

Emulation `MOV #0,dst`
`MOV.B #0,dst`

Description The destination operand is cleared.

Status Bits Status bits are not affected.

Example RAM word TONI is cleared.

```
CLR     TONI      ; 0 -> TONI
```

Example Register R5 is cleared.

```
CLR     R5
```

Example RAM byte TONI is cleared.

```
CLR.B   TONI      ; 0 -> TONI
```

*** CLRC** Clear carry bit

Syntax `CLRC`

Operation $0 \rightarrow C$

Emulation `BIC #1,SR`

Description The carry bit (C) is cleared. The clear carry instruction is a word instruction.

Status Bits
N: Not affected
Z: Not affected
C: Cleared
V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 16-bit decimal counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
CLRC                    ; C=0: defines start
DADD    @R13,0(R12)     ; add 16-bit counter to low word of 32-bit counter
DADC    2(R12)          ; add carry to high word of 32-bit counter
```

*** CLRN** Clear negative bit

Syntax `CLRN`

Operation $0 \rightarrow N$

or

(.NOT.src .AND. dst → dst)

Emulation `BIC #4,SR`

Description The constant 04h is inverted (0FFFBh) and is logically ANDed with the destination operand. The result is placed into the destination. The clear negative bit instruction is a word instruction.

Status Bits
N: Reset to 0
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The negative bit in the SR is cleared. This avoids special treatment with negative numbers of the subroutine called.

```
            CLRN
            CALL   SUBR
            ......
            ......
SUBR        JN     SUBRET     ; If input is negative: do nothing and return
            ......
            ......
            ......
SUBRET      RET
```

*** CLRZ** Clear zero bit

Syntax `CLRZ`

Operation $0 \rightarrow Z$

or

(.NOT.src .AND. dst → dst)

Emulation `BIC #2,SR`

Description The constant 02h is inverted (0FFFDh) and logically ANDed with the destination operand. The result is placed into the destination. The clear zero bit instruction is a word instruction.

Status Bits
- N: Not affected
- Z: Reset to 0
- C: Not affected
- V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The zero bit in the SR is cleared.

```
CLRZ
```

Indirect, Auto-Increment mode: Call a subroutine at the 16-bit address contained in the word pointed to by register R5 (20-bit address) and increment the 16-bit address in R5 afterwards by 2. The next time the software uses R5 as a pointer, it can alter the program execution due to access to the next word address in the table pointed to by R5.

```
CALL   @R5+          ; Start address at @R5. R5 + 2
```

Indexed mode: Call a subroutine at the 16-bit address contained in the 20-bit address pointed to by register (R5 + X), e.g., a table with addresses starting at X. The address is within the lower 64 KB. X is within +32 KB.

```
CALL   X(R5)         ; Start address at @(R5+X). z16(R5)
```

CMP[.W] Compare source word and destination word

CMP.B Compare source byte and destination byte

Syntax `CMP src,dst` or `CMP.W src,dst`

`CMP.B src,dst`

Operation (.not.src) + 1 + dst

or

dst – src

Emulation `BIC #2,SR`

Description The source operand is subtracted from the destination operand. This is made by adding the 1s complement of the source + 1 to the destination. The result affects only the status bits in SR.

Register mode: the register bits Rdst.19:16 (.W) resp. Rdst. 19:8 (.B) are not cleared.

Status Bits

N: Set if result is negative (src > dst), reset if positive (src = dst)

Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)

C: Set if there is a carry from the MSB, reset otherwise

V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow).

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Compare word EDE with a 16-bit constant 1800h. Jump to label TONI if EDE equals the constant. The address of EDE is within PC + 32 K.

```
CMP     #01800h,EDE      ; Compare word EDE with 1800h
JEQ     TONI             ; EDE contains 1800h
...                      ; Not equal
```

Example A table word pointed to by (R5 + 10) is compared with R7. Jump to label TONI if R7 contains a lower, signed 16-bit number. R7.19:16 is not cleared. The address of the source operand is a 20-bit address in full memory range.

```
CMP.W   10(R5),R7        ; Compare two signed numbers
JL      TONI             ; R7 < 10(R5)
...                      ; R7 >= 10(R5)
```

Example A table byte pointed to by R5 (20-bit address) is compared to the value in output Port1. Jump to label TONI if values are equal. The next table byte is addressed.

```
CMP.B   @R5+,&P1OUT      ; Compare P1 bits with table. R5 + 1
JEQ     TONI             ; Equal contents
...                      ; Not equal
```

*** DADC[.W]** Add carry decimally to destination

*** DADC.B** Add carry decimally to destination

Syntax `DADC dst` or `DADC.W dst`

`DADC.B dst`

Operation dst + C → dst (decimally)

Emulation `DADD #0,dst` `DADD.B #0,dst`

Description The carry bit (C) is added decimally to the destination.

Status Bits

N: Set if MSB is 1

Z: Set if dst is 0, reset otherwise

C: Set if destination increments from 9999 to 0000, reset otherwise

Set if destination increments from 99 to 00, reset otherwise

V: Undefined

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The four-digit decimal number contained in R5 is added to an eight-digit decimal number pointed to by R8.

```
CLRC                ; Reset carry
                    ; next instruction's start condition is defined
DADD   R5,0(R8)     ; Add LSDs + C
DADC   2(R8)        ; Add carry to MSD
```

Example The two-digit decimal number contained in R5 is added to a four-digit decimal number pointed to by R8.

```
CLRC                ; Reset carry
                    ; next instruction's start condition is defined
DADD.B   R5,0(R8)   ; Add LSDs + C
DADC     1(R8)      ; Add carry to MSDs
```

*** DADD[.W]** Add source word and carry decimally to destination word

*** DADD.B** Add source byte and carry decimally to destination byte

Syntax `DADD src,dst` or `DADD.W src,dst`

`DADD.B src,dst`

Operation src + dst + C → dst (decimally)

Description The source operand and the destination operand are treated as two (.B) or four (.W) binary coded decimals (BCD) with positive signs. The source operand and the carry bit C are added decimally to the destination operand. The source operand is not affected. The previous content of the destination is lost. The result is not defined for non-BCD numbers.

Status Bits
- N: Set if MSB of result is 1 (word > 7999h, byte > 79h), reset if MSB is 0
- Z: Set if result is zero, reset otherwise
- C: Set if the BCD result is too large (word > 9999h, byte > 99h), reset otherwise
- V: Undefined

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Decimal 10 is added to the 16-bit BCD counter DECCNTR.

```
DADD   #10h,&DECCNTR   ; Add 10 to 4-digit BCD counter
```

Example The eight-digit BCD number contained in 16-bit RAM addresses BCD and BCD+2 is added decimally to an eight-digit BCD number contained in R4 and R5 (BCD+2 and R5 contain the MSDs). The carry C is added, and cleared.

```
CLRC                   ; Clear carry
DADD.W   &BCD,R4       ; Add LSDs. R4.19:16 = 0
DADD.W   &BCD+2,R5     ; Add MSDs with carry. R5.19:16 = 0
JC       OVERFLOW      ; Result >9999,9999: go to error routine
...                    ; Result ok
```

Example The two-digit BCD number contained in word BCD (16-bit address) is added decimally to a two-digit BCD number contained in R4. The carry C is added, also. R4.19:8 = 0CLRC ; Clear carryDADD.B &BCD,R4 ; Add BCD to R4 decimally. R4: 0,00ddh

```
CLRC                   ; Clear carry
DADD.B   &BCD,R4       ; Add BCD to R4 decimally.
                         R4: 0,00ddh
```

*** DEC[.W]** Decrement destination

*** DEC.B** Decrement destination

Syntax `DEC dst` or `DEC.W dst`

`DEC.B dst`

Operation dst – 1 → dst

Emulation `SUB #1,dst`

`SUB.B #1,dst`

Description The destination operand is decremented by one. The original contents are lost.

Status Bits
- N: Set if result is negative, reset if positive
- Z: Set if dst contained 1, reset otherwise
- C: Reset if dst contained 0, set otherwise
- V: Set if an arithmetic overflow occurs, otherwise reset.
  Set if initial value of destination was 08000h, otherwise reset.
  Set if initial value of destination was 080h, otherwise reset.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example R10 is decremented by 1.

```
        DEC     R10                 ; Decrement R10

; Move a block of 255 bytes from memory location starting with EDE to
; memory location starting with TONI. Tables should not overlap: start of
; destination address TONI must not be within the range EDE to EDE+0FEh

        MOV     #EDE,R6
        MOV     #510,R10
L$1     MOV     @R6+,TONI-EDE-1(R6)
        DEC     R10
        JNZ     L$1
```

Do not transfer tables using the routine above with the overlap shown in Figure 4-36.

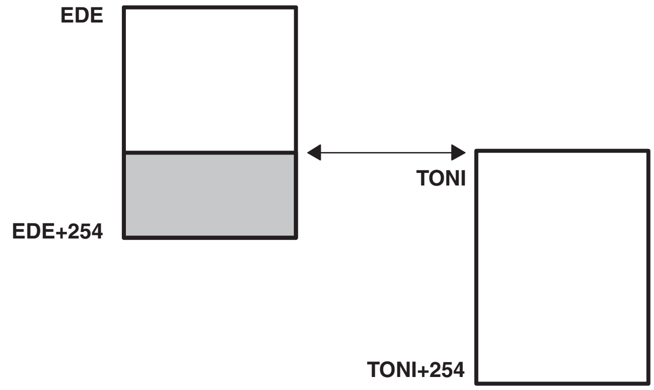


Figure 4-36. Decrement Overlap

*** DECD[.W]** Double-decrement destination

*** DECD.B** Double-decrement destination

Syntax `DECD dst` or `DECD.W dst`

`DECD.B dst`

Operation dst – 2 → dst

Emulation `SUB #2,dst`

`SUB.B #2,dst`

Description The destination operand is decremented by two. The original contents are lost.

Status Bits
- N: Set if result is negative, reset if positive
- Z: Set if dst contained 2, reset otherwise
- C: Reset if dst contained 0 or 1, set otherwise
- V: Set if an arithmetic overflow occurs, otherwise reset
  Set if initial value of destination was 08001 or 08000h, otherwise reset
  Set if initial value of destination was 081 or 080h, otherwise reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example R10 is decremented by 2.

```
        DECD    R10             ; Decrement R10 by two

; Move a block of 255 bytes from memory location starting with EDE to
; memory location starting with TONI.
; Tables should not overlap: start of destination address TONI must not
; be within the range EDE to EDE+0FEh

        MOV     #EDE,R6
        MOV     #255,R10
L$1     MOV.B   @R6+,TONI-EDE-2(R6)
        DECD    R10
        JNZ     L$1
```

Example Memory at location LEO is decremented by two.

```
        DECD.B  LEO             ; Decrement MEM(LEO)
```

Decrement status byte STATUS by two

```
        DECD.B  STATUS
```

*** DINT** Disable (general) interrupts

Syntax `DINT`

Operation 0 → GIE
or
(0FFF7h .AND. SR → SR / .NOT.src .AND. dst → dst)

Emulation `BIC #8,SR`

Description All interrupts are disabled.
The constant 08h is inverted and logically ANDed with the SR. The result is placed into the SR.

Status Bits Status bits are not affected.

Mode Bits GIE is reset. OSCOFF and CPUOFF are not affected.

Example The general interrupt enable (GIE) bit in the SR is cleared to allow a nondisrupted move of a 32-bit counter. This ensures that the counter is not modified during the move by any interrupt.

```
DINT                 ; All interrupt events using the GIE bit are disabled
NOP
MOV    COUNTHI,R5    ; Copy counter
MOV    COUNTLO,R6
EINT                 ; All interrupt events using the GIE bit are enabled
```

NOTE: Disable interrupt

If any code sequence needs to be protected from interruption, DINT should be executed at least one instruction before the beginning of the uninterruptible sequence, or it should be followed by a NOP instruction.

*** EINT** Enable (general) interrupts

Syntax `EINT`

Operation 1 → GIE
or
(0008h .OR. SR → SR / .src .OR. dst → dst)

Emulation `BIS #8,SR`

Description All interrupts are enabled.
The constant #08h and the SR are logically ORed. The result is placed into the SR.

Status Bits Status bits are not affected.

Mode Bits GIE is set. OSCOFF and CPUOFF are not affected.

Example The general interrupt enable (GIE) bit in the SR is set.

```
         PUSH.B    &P1IN
         BIC.B     @SP,&P1IFG  ; Reset only accepted flags
         EINT                  ; Preset port 1 interrupt flags stored on stack
                               ; other interrupts are allowed
         BIT    #Mask,@SP
         JEQ    MaskOK         ; Flags are present identically to mask: jump
         ......
MaskOK   BIC    #Mask,@SP
         ......
         INCD    SP            ; Housekeeping: inverse to PUSH instruction
                               ; at the start of interrupt subroutine. Corrects
                               ; the stack pointer.
         RETI
```

NOTE: Enable interrupt

The instruction following the enable interrupt instruction (EINT) is always executed, even if an interrupt service request is pending when the interrupts are enabled.

*** INC[.W]** Increment destination

*** INC.B** Increment destination

Syntax `INC dst` or `INC.W dst`

`INC.B dst`

Operation dst + 1 → dst

Emulation `ADD #1,dst`

Description The destination operand is incremented by one. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive

Z: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise

C: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise

V: Set if dst contained 07FFFh, reset otherwise
Set if dst contained 07Fh, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The status byte, STATUS, of a process is incremented. When it is equal to 11, a branch to OVFL is taken.

```
INC.B   STATUS
CMP.B   #11,STATUS
JEQ     OVFL
```

*** INCD[.W]** Double-increment destination
*** INCD.B** Double-increment destination

Syntax `INCD dst` or `INCD.W dst`
`INCD.B dst`

Operation dst + 2 → dst

Emulation `ADD #2,dst`

Description The destination operand is incremented by two. The original contents are lost.

Status Bits
- N: Set if result is negative, reset if positive
- Z: Set if dst contained 0FFFEh, reset otherwise
  Set if dst contained 0FEh, reset otherwise
- C: Set if dst contained 0FFFEh or 0FFFFh, reset otherwise
  Set if dst contained 0FEh or 0FFh, reset otherwise
- V: Set if dst contained 07FFEh or 07FFFh, reset otherwise
  Set if dst contained 07Eh or 07Fh, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The item on the top of the stack (TOS) is removed without using a register.

```
.......
PUSH    R5       ; R5 is the result of a calculation, which is stored
                 ; in the system stack
INCD    SP       ; Remove TOS by double-increment from stack
                 ; Do not use INCD.B, SP is a word-aligned register
RET
```

Example The byte on the top of the stack is incremented by two.

```
INCD.B   0(SP)   ; Byte on TOS is increment by two
```

*** INV[.W]** Invert destination

*** INV.B** Invert destination

Syntax `INV dst` or `INV.W dst`

`INV.B dst`

Operation .not.dst → dst

Emulation `XOR #0FFFFh,dst`

`XOR.B #0FFh,dst`

Description The destination operand is inverted. The original contents are lost.

Status Bits

- N: Set if result is negative, reset if positive
- Z: Set if dst contained 0FFFFh, reset otherwise
  Set if dst contained 0FFh, reset otherwise
- C: Set if result is not zero, reset otherwise (= .NOT. Zero)
- V: Set if initial destination operand was negative, otherwise reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Content of R5 is negated (2s complement).

```
MOV     #00AEh,R5     ;                      R5 = 000AEh
INV     R5            ; Invert R5,           R5 = 0FF51h
INC     R5            ; R5 is now negated,   R5 = 0FF52h
```

Example Content of memory byte LEO is negated.

```
MOV.B   #0AEh,LEO     ;                      MEM(LEO) = 0AEh
INV.B   LEO           ; Invert LEO,          MEM(LEO) = 051h
INC.B   LEO           ; MEM(LEO) is negated, MEM(LEO) = 052h
```

JC Jump if carry

JHS Jump if higher or same (unsigned)

Syntax `JC label`

`JHS label`

Operation If C = 1: PC + (2 × Offset) → PC
If C = 0: execute the following instruction

Description The carry bit C in the SR is tested. If it is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If C is reset, the instruction after the jump is executed.

JC is used for the test of the carry bit C.

JHS is used for the comparison of unsigned numbers.

Status Bits Status bits are not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The state of the port 1 pin P1IN.1 bit defines the program flow.

```
BIT.B   #2,&P1IN       ; Port 1, bit 1 set? Bit -> C
JC      Label1         ; Yes, proceed at Label1
...                    ; No, continue
```

Example If R5 ≥ R6 (unsigned), the program continues at Label2.

```
CMP     R6,R 5         ; Is R5 >= R6? Info to C
JHS     Label2         ; Yes, C = 1
...                    ; No, R5 < R6. Continue
```

Example If R5 ≥ 12345h (unsigned operands), the program continues at Label2.

```
CMPA    #12345h,R5     ; Is R5 >= 12345h? Info to C
JHS     Label2         ; Yes, 12344h < R5 <= F,FFFFh. C = 1
...                    ; No, R5 < 12345h. Continue
```

JEQ Jump if equal

JZ Jump if zero

Syntax `JEQ label`

`JZ label`

Operation If Z = 1: PC + (2 × Offset) → PC
If Z = 0: execute following instruction

Description The zero bit Z in the SR is tested. If it is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If Z is reset, the instruction after the jump is executed.

JZ is used for the test of the zero bit Z.

JEQ is used for the comparison of operands.

Status Bits Status bits are not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The state of the P2IN.0 bit defines the program flow.

```
BIT.B   #1,&P2IN      ; Port 2, bit 0 reset?
JZ      Label1        ; Yes, proceed at Label1
...                   ; No, set, continue
```

Example If R5 = 15000h (20-bit data), the program continues at Label2.

```
CMPA    #15000h,R5    ; Is R5 = 15000h? Info to SR
JEQ     Label2        ; Yes, R5 = 15000h. Z = 1
...                   ; No, R5 not equal 15000h. Continue
```

Example R7 (20-bit counter) is incremented. If its content is zero, the program continues at Label4.

```
ADDA    #1,R7         ; Increment R7
JZ      Label4        ; Zero reached: Go to Label4
...                   ; R7 not equal 0. Continue here.
```

JGE Jump if greater or equal (signed)

Syntax `JGE label`

Operation If (N .xor. V) = 0: PC + (2 × Offset) → PC
If (N .xor. V) = 1: execute following instruction

Description The negative bit N and the overflow bit V in the SR are tested. If both bits are set or both are reset, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range -511 to +512 words relative to the PC in full Memory range. If only one bit is set, the instruction after the jump is executed.

JGE is used for the comparison of signed operands: also for incorrect results due to overflow, the decision made by the JGE instruction is correct.

Note that JGE emulates the nonimplemented JP (jump if positive) instruction if used after the instructions AND, BIT, RRA, SXTX, and TST. These instructions clear the V bit.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example If byte EDE (lower 64 K) contains positive data, go to Label1. Software can run in the full memory range.

```
TST.B   &EDE           ; Is EDE positive? V <- 0
JGE     Label1         ; Yes, JGE emulates JP
...                    ; No, 80h <= EDE <= FFh
```

Example If the content of R6 is greater than or equal to the memory pointed to by R7, the program continues a Label5. Signed data. Data and program in full memory range.

```
CMP     @R7,R6         ; Is R6 >= @R7?
JGE     Label5         ; Yes, go to Label5
...                    ; No, continue here
```

Example If R5 ≥ 12345h (signed operands), the program continues at Label2. Program in full memory range.

```
CMPA    #12345h,R5     ; Is R5 >= 12345h?
JGE     Label2         ; Yes, 12344h < R5 <= 7FFFFh
...                    ; No, 80000h <= R5 < 12345h
```

JL Jump if less (signed)

Syntax `JL label`

Operation If (N .xor. V) = 1: PC + (2 × Offset) → PC
If (N .xor. V) = 0: execute following instruction

Description The negative bit N and the overflow bit V in the SR are tested. If only one is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range –511 to +512 words relative to the PC in full memory range. If both bits N and V are set or both are reset, the instruction after the jump is executed.

JL is used for the comparison of signed operands: also for incorrect results due to overflow, the decision made by the JL instruction is correct.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example If byte EDE contains a smaller, signed operand than byte TONI, continue at Label1. The address EDE is within PC ± 32 K.

```
CMP.B   &TONI,EDE     ; Is EDE < TONI
JL      Label1        ; Yes
...                   ; No, TONI <= EDE
```

Example If the signed content of R6 is less than the memory pointed to by R7 (20-bit address), the program continues at Label5. Data and program in full memory range.

```
CMP     @R7,R6        ; Is R6 < @R7?
JL      Label5        ; Yes, go to Label5
...                   ; No, continue here
```

Example If R5 < 12345h (signed operands), the program continues at Label2. Data and program in full memory range.

```
CMPA    #12345h,R5    ; Is R5 < 12345h?
JL      Label2        ; Yes, 80000h =< R5 < 12345h
...                   ; No, 12344h < R5 <= 7FFFFh
```

JMP Jump unconditionally

Syntax `JMP label`

Operation PC + (2 × Offset) → PC

Description The signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means an unconditional jump in the range –511 to +512 words relative to the PC in the full memory. The JMP instruction may be used as a BR or BRA instruction within its limited range relative to the PC.

Status Bits Status bits are not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The byte STATUS is set to 10. Then a jump to label MAINLOOP is made. Data in lower 64 K, program in full memory range.

```
MOV.B   #10,&STATUS    ; Set STATUS to 10
JMP     MAINLOOP       ; Go to main loop
```

Example The interrupt vector TAIV of Timer_A3 is read and used for the program flow. Program in full memory range, but interrupt handlers always starts in lower 64 K.

```
ADD     &TAIV,PC       ; Add Timer_A interrupt vector to PC
RETI                   ; No Timer_A interrupt pending
JMP     IHCCR1         ; Timer block 1 caused interrupt
JMP     IHCCR2         ; Timer block 2 caused interrupt
RETI                   ; No legal interrupt, return
```

JN Jump if negative

Syntax `JN label`

Operation If N = 1: PC + (2 × Offset) → PC
If N = 0: execute following instruction

Description The negative bit N in the SR is tested. If it is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit program PC. This means a jump in the range -511 to +512 words relative to the PC in the full memory range. If N is reset, the instruction after the jump is executed.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The byte COUNT is tested. If it is negative, program execution continues at Label0. Data in lower 64 K, program in full memory range.

```
TST.B   &COUNT      ; Is byte COUNT negative?
JN      Label0      ; Yes, proceed at Label0
...                 ; COUNT >= 0
```

Example R6 is subtracted from R5. If the result is negative, program continues at Label2. Program in full memory range.

```
SUB     R6,R5       ; R5 - R6 -> R5
JN      Label2      ; R5 is negative: R6 > R5 (N = 1)
...                 ; R5 >= 0. Continue here.
```

Example R7 (20-bit counter) is decremented. If its content is below zero, the program continues at Label4. Program in full memory range.

```
SUBA    #1,R7       ; Decrement R7
JN      Label4      ; R7 < 0: Go to Label4
...                 ; R7 >= 0. Continue here.
```

JNC Jump if no carry

JLO Jump if lower (unsigned)

Syntax `JNC label`

`JLO label`

Operation If C = 0: PC + (2 × Offset) → PC
If C = 1: execute following instruction

Description The carry bit C in the SR is tested. If it is reset, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If C is set, the instruction after the jump is executed.

JNC is used for the test of the carry bit C.

JLO is used for the comparison of unsigned numbers.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example If byte EDE < 15, the program continues at Label2. Unsigned data. Data in lower 64 K, program in full memory range.

```
CMP.B   #15,&EDE      ; Is EDE < 15? Info to C
JLO     Label2        ; Yes, EDE < 15. C = 0
...                   ; No, EDE >= 15. Continue
```

Example The word TONI is added to R5. If no carry occurs, continue at Label0. The address of TONI is within PC ± 32 K.

```
ADD     TONI,R5       ; TONI + R5 -> R5. Carry -> C
JNC     Label0        ; No carry
...                   ; Carry = 1: continue here
```

JNZ Jump if not zero

JNE Jump if not equal

Syntax `JNZ label`

`JNE label`

Operation If Z = 0: PC + (2 × Offset) → PC
If Z = 1: execute following instruction

Description The zero bit Z in the SR is tested. If it is reset, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If Z is set, the instruction after the jump is executed.

JNZ is used for the test of the zero bit Z.

JNE is used for the comparison of operands.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The byte STATUS is tested. If it is not zero, the program continues at Label3. The address of STATUS is within PC ± 32 K.

```
TST.B   STATUS          ; Is STATUS = 0?
JNZ     Label3          ; No, proceed at Label3
...                     ; Yes, continue here
```

Example If word EDE ≠ 1500, the program continues at Label2. Data in lower 64 K, program in full memory range.

```
CMP     #1500,&EDE      ; Is EDE = 1500? Info to SR
JNE     Label2          ; No, EDE not equal 1500.
...                     ; Yes, R5 = 1500. Continue
```

Example R7 (20-bit counter) is decremented. If its content is not zero, the program continues at Label4. Program in full memory range.

```
SUBA     #1,R7          ; Decrement R7
JNZ      Label4         ; Zero not reached: Go to Label4
...                     ; Yes, R7 = 0. Continue here.
```

MOV[.W] Move source word to destination word

MOV.B Move source byte to destination byte

Syntax `MOV src,dst` or `MOV.W src,dst`

`MOV.B src,dst`

Operation src → dst

Description The source operand is copied to the destination. The source operand is not affected.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Move a 16-bit constant 1800h to absolute address-word EDE (lower 64 K)

```
MOV    #01800h,&EDE          ; Move 1800h to EDE
```

Example The contents of table EDE (word data, 16-bit addresses) are copied to table TOM. The length of the tables is 030h words. Both tables reside in the lower 64 K.

```
        MOV     #EDE,R10            ; Prepare pointer (16-bit address)
Loop    MOV     @R10+,TOM-EDE-2(R10) ; R10 points to both tables.
                                    ; R10+2
        CMP     #EDE+60h,R10        ; End of table reached?
        JLO     Loop                ; Not yet
        ...                         ; Copy completed
```

Example The contents of table EDE (byte data, 16-bit addresses) are copied to table TOM. The length of the tables is 020h bytes. Both tables may reside in full memory range, but must be within R10 ± 32 K.

```
        MOVA    #EDE,R10            ; Prepare pointer (20-bit)
        MOV     #20h,R9             ; Prepare counter
Loop    MOV.B   @R10+,TOM-EDE-1(R10) ; R10 points to both tables.
                                    ; R10+1
        DEC     R9                  ; Decrement counter
        JNZ     Loop                ; Not yet done
        ...                         ; Copy completed
```

*** NOP** No operation

Syntax `NOP`

Operation None

Emulation `MOV #0, R3`

Description No operation is performed. The instruction may be used for the elimination of instructions during the software check or for defined waiting times.

Status Bits Status bits are not affected.

*** POP[.W]** Pop word from stack to destination

*** POP.B** Pop byte from stack to destination

Syntax
`POP dst`
`POP.B dst`

Operation
@SP → temp
SP + 2 → SP
temp → dst

Emulation
`MOV @SP+,dst` or `MOV.W @SP+,dst`
`MOV.B @SP+,dst`

Description The stack location pointed to by the SP (TOS) is moved to the destination. The SP is incremented by two afterwards.

Status Bits Status bits are not affected.

Example The contents of R7 and the SR are restored from the stack.

```
POP     R7      ; Restore R7
POP     SR      ; Restore status register
```

Example The contents of RAM byte LEO is restored from the stack.

```
POP.B   LEO     ; The low byte of the stack is moved to LEO.
```

Example The contents of R7 is restored from the stack.

```
POP.B   R7      ; The low byte of the stack is moved to R7,
                ; the high byte of R7 is 00h
```

Example The contents of the memory pointed to by R7 and the SR are restored from the stack.

```
POP.B   0(R7)   ; The low byte of the stack is moved to the
                ; the byte which is pointed to by R7
                : Example:   R7 = 203h
                ;            Mem(R7) = low byte of system stack
                : Example:   R7 = 20Ah
                ;            Mem(R7) = low byte of system stack
POP     SR      ; Last word on stack moved to the SR
```

NOTE: System stack pointer

The system SP is always incremented by two, independent of the byte suffix.

PUSH[.W] Save a word on the stack

PUSH.B Save a byte on the stack

Syntax `PUSH dst` or `PUSH.W dst`

`PUSH.B dst`

Operation SP – 2 → SP
dst → @SP

Description The 20-bit SP SP is decremented by two. The operand is then copied to the RAM word addressed by the SP. A pushed byte is stored in the low byte; the high byte is not affected.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Save the two 16-bit registers R9 and R10 on the stack

```
PUSH     R9       ; Save R9 and R10 XXXXh
PUSH     R10      ; YYYYh
```

Example Save the two bytes EDE and TONI on the stack. The addresses EDE and TONI are within PC ± 32 K.

```
PUSH.B   EDE      ; Save EDE   xxXXh
PUSH.B   TONI     ; Save TONI   xxYYh
```

RET Return from subroutine

Syntax `RET`

Operation @SP →PC.15:0 Saved PC to PC.15:0. PC.19:16 ← 0
SP + 2 → SP

Description The 16-bit return address (lower 64 K), pushed onto the stack by a CALL instruction is restored to the PC. The program continues at the address following the subroutine call. The four MSBs of the PC.19:16 are cleared.

Status Bits Status bits are not affected.
PC.19:16: Cleared

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Call a subroutine SUBR in the lower 64 K and return to the address in the lower 64 K after the CALL.

```
        CALL    #SUBR       ; Call subroutine starting at SUBR
        ...                 ; Return by RET to here
SUBR    PUSH    R14         ; Save R14 (16 bit data)
        ...                 ; Subroutine code
        POP     R14         ; Restore R14
        RET                 ; Return to lower 64 K
```

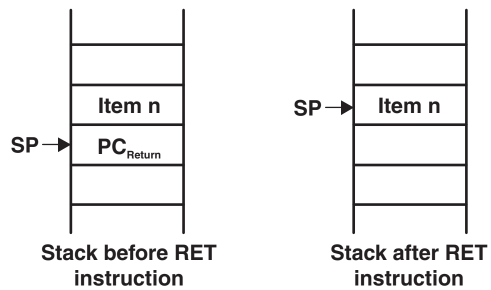


Figure 4-37. Stack After a RET Instruction

RETI Return from interrupt

Syntax `RETI`

Operation

@SP → SR.15:0 SP + 2 → SP	Restore saved SR with PC.19:16
@SP → PC.15:0 SP + 2 → SP	Restore saved PC.15:0 Housekeeping

Description The SR is restored to the value at the beginning of the interrupt service routine. This includes the four MSBs of the PC.19:16. The SP is incremented by two afterward.

The 20-bit PC is restored from PC.19:16 (from same stack location as the status bits) and PC.15:0. The 20-bit PC is restored to the value at the beginning of the interrupt service routine. The program continues at the address following the last executed instruction when the interrupt was granted. The SP is incremented by two afterward.

Status Bits

N: Restored from stack

C: Restored from stack

Z: Restored from stack

V: Restored from stack

Mode Bits OSCOFF, CPUOFF, and GIE are restored from stack.

Example Interrupt handler in the lower 64 K. A 20-bit return address is stored on the stack.

```
INTRPT  PUSHM.A   #2,R14    ; Save R14 and R13 (20-bit data)
        ...                 ; Interrupt handler code
        POPM.A    #2,R14    ; Restore R13 and R14 (20-bit data)
        RETI                ; Return to 20-bit address in full memory range
```

*** RLA[.W]** Rotate left arithmetically

*** RLA.B** Rotate left arithmetically

Syntax `RLA dst` or `RLA.W dst`

`RLA.B dst`

Operation C ← MSB ← MSB-1 .... LSB+1 ← LSB ← 0

Emulation `ADD dst,dst` `ADD.B dst,dst`

Description The destination operand is shifted left one position as shown in Figure 4-38. The MSB is shifted into the carry bit (C) and the LSB is filled with 0. The RLA instruction acts as a signed multiplication by 2.

An overflow occurs if dst ≥ 04000h and dst < 0C000h before operation is performed; the result has changed sign.

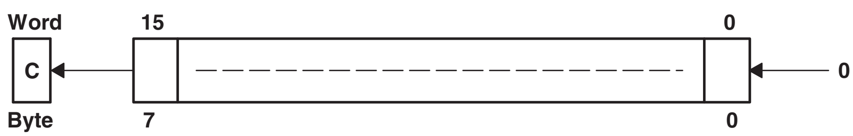


Figure 4-38. Destination Operand—Arithmetic Shift Left

An overflow occurs if dst ≥ 040h and dst < 0C0h before the operation is performed; the result has changed sign.

Status Bits

- N: Set if result is negative, reset if positive
- Z: Set if result is zero, reset otherwise
- C: Loaded from the MSB
- V: Set if an arithmetic overflow occurs; the initial value is 04000h ≤ dst < 0C000h, reset otherwise

  Set if an arithmetic overflow occurs; the initial value is 040h ≤ dst < 0C0h, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example R7 is multiplied by 2.

```
RLA     R7     ; Shift left R7  (x 2)
```

Example The low byte of R7 is multiplied by 4.

```
RLA.B   R7     ; Shift left low byte of R7  (x 2)
RLA.B   R7     ; Shift left low byte of R7  (x 4)
```

NOTE: RLA substitution

The assembler does not recognize the instructions:

```
RLA   @R5+           RLA.B   @R5+           RLA(.B) @R5
```

They must be substituted by:

```
ADD   @R5+,-2(R5)    ADD.B   @R5+,-1(R5)    ADD(.B) @R5
```

*** RLC[.W]** Rotate left through carry

*** RLC.B** Rotate left through carry

Syntax `RLC dst` or `RLC.W dst`

`RLC.B dst`

Operation C ← MSB ← MSB-1 .... LSB+1 ← LSB ← C

Emulation `ADDC dst,dst`

Description The destination operand is shifted left one position as shown in Figure 4-39. The carry bit (C) is shifted into the LSB, and the MSB is shifted into the carry bit (C).

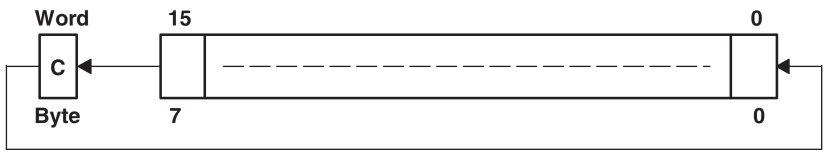


Figure 4-39. Destination Operand—Carry Left Shift

Status Bits

N: Set if result is negative, reset if positive

Z: Set if result is zero, reset otherwise

C: Loaded from the MSB

V: Set if an arithmetic overflow occurs; the initial value is 04000h ≤ dst < 0C000h, reset otherwise

Set if an arithmetic overflow occurs; the initial value is 040h ≤ dst < 0C0h, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example R5 is shifted left one position.

```
RLC     R5            ; (R5 x 2) + C -> R5
```

Example The input P1IN.1 information is shifted into the LSB of R5.

```
BIT.B   #2,&P1IN      ; Information -> Carry
RLC     R5            ; Carry=P0in.1 -> LSB of R5
```

Example The MEM(LEO) content is shifted left one position.

```
RLC.B   LEO           ; Mem(LEO) x 2 + C -> Mem(LEO)
```

NOTE: RLA substitution

The assembler does not recognize the instructions:

```
RLC  @R5+          RLC.B  @R5+          RLC(.B) @R5
```

They must be substituted by:

```
ADDC  @R5+,-2(R5)  ADDC.B  @R5+,-1(R5)  ADDC(.B) @R5
```

RRA[.W] Rotate right arithmetically destination word

RRA.B Rotate right arithmetically destination byte

Syntax `RRA.B dst` or `RRA.W dst`

Operation MSB → MSB → MSB–1 → ... LSB+1 → LSB → C

Description The destination operand is shifted right arithmetically by one bit position as shown in Figure 4-40. The MSB retains its value (sign). RRA operates equal to a signed division by 2. The MSB is retained and shifted into the MSB–1. The LSB+1 is shifted into the LSB. The previous LSB is shifted into the carry bit C.

Status Bits
- N: Set if result is negative (MSB = 1), reset otherwise (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Loaded from the LSB
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The signed 16-bit number in R5 is shifted arithmetically right one position.

```
RRA     R5              ; R5/2 -> R5
```

Example The signed RAM byte EDE is shifted arithmetically right one position.

```
RRA.B   EDE             ; EDE/2 -> EDE
```

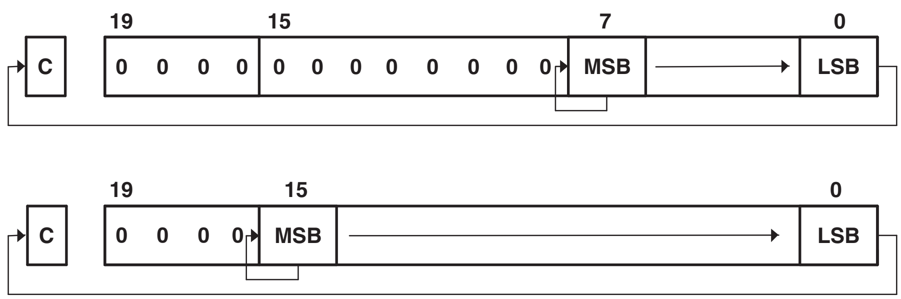


Figure 4-40. Rotate Right Arithmetically RRA.B and RRA.W

RRC[.W] Rotate right through carry destination word

RRC.B Rotate right through carry destination byte

Syntax `RRC dst` or `RRC.W dst`

`RRC.B dst`

Operation C → MSB → MSB–1 → ... LSB+1 → LSB → C

Description The destination operand is shifted right by one bit position as shown in Figure 4-41. The carry bit C is shifted into the MSB and the LSB is shifted into the carry bit C.

Status Bits
- N: Set if result is negative (MSB = 1), reset otherwise (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Loaded from the LSB
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example RAM word EDE is shifted right one bit position. The MSB is loaded with 1.

```
SETC            ; Prepare carry for MSB
RRC     EDE     ; EDE = EDE >> 1 + 8000h
```

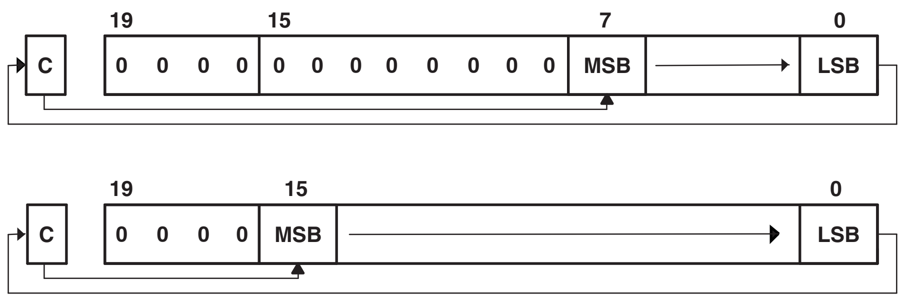


Figure 4-41. Rotate Right Through Carry RRC.B and RRC.W

*** SBC[.W]** Subtract borrow (.NOT. carry) from destination

*** SBC.B** Subtract borrow (.NOT. carry) from destination

Syntax `SBC dst` or `SBC.W dst`

`SBC.B dst`

Operation dst + 0FFFFh + C → dst

dst + 0FFh + C → dst

Emulation `SUBC #0,dst`

`SUBC.B #0,dst`

Description The carry bit (C) is added to the destination operand minus one. The previous contents of the destination are lost.

Status Bits

- N: Set if result is negative, reset if positive
- Z: Set if result is zero, reset otherwise
- C: Set if there is a carry from the MSB of the result, reset otherwise
  Set to 1 if no borrow, reset if borrow
- V: Set if an arithmetic overflow occurs, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 16-bit counter pointed to by R13 is subtracted from a 32-bit counter pointed to by R12.

```
SUB     @R13,0(R12)    ; Subtract LSDs
SBC     2(R12)         ; Subtract carry from MSD
```

Example The 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.

```
SUB.B   @R13,0(R12)    ; Subtract LSDs
SBC.B   1(R12)         ; Subtract carry from MSD
```

NOTE: Borrow implementation

The borrow is treated as a .NOT. carry:

Borrow	Carry Bit
Yes	0
No	1

*** SETC** Set carry bit

Syntax `SETC`

Operation 1 → C

Emulation `BIS #1,SR`

Description The carry bit (C) is set.

Status Bits
N: Not affected
Z: Not affected
C: Set
V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Emulation of the decimal subtraction:
Subtract R5 from R6 decimally.
Assume that R5 = 03987h and R6 = 04137h.

```
DSUB   ADD    #06666h,R5     ; Move content R5 from 0-9 to 6-0Fh
                             ; R5 = 03987h + 06666h = 09FEDh
       INV    R5             ; Invert this (result back to 0-9)
                             ; R5 = .NOT. R5 = 06012h
       SETC                  ; Prepare carry = 1
       DADD   R5,R6          ; Emulate subtraction by addition of:
                             ; (010000h - R5 - 1)
                             ; R6 = R6 + R5 + 1
                             ; R6 = 0150h
```

*** SETN**	Set negative bit
Syntax	`SETN`
Operation	1 → N
Emulation	`BIS #4,SR`
Description	The negative bit (N) is set.
Status Bits	N: Set
	Z: Not affected
	C: Not affected
	V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.

*** SETZ** Set zero bit

Syntax `SETZ`

Operation 1 → N

Emulation `BIS #2,SR`

Description The zero bit (Z) is set.

Status Bits
- N: Not affected
- Z: Set
- C: Not affected
- V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

SUB[.W] Subtract source word from destination word

SUB.B Subtract source byte from destination byte

Syntax `SUB src,dst` or `SUB.W src,dst`

`SUB.B src,dst`

Operation (.not.src) + 1 + dst → dst or dst – src → dst

Description The source operand is subtracted from the destination operand. This is made by adding the 1s complement of the source + 1 to the destination. The source operand is not affected, the result is written to the destination operand.

Status Bits

- N: Set if result is negative (src > dst), reset if positive (src ≤ dst)
- Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)
- C: Set if there is a carry from the MSB, reset otherwise
- V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example A 16-bit constant 7654h is subtracted from RAM word EDE.

```
SUB     #7654h,&EDE     ; Subtract 7654h from EDE
```

Example A table word pointed to by R5 (20-bit address) is subtracted from R7. Afterwards, if R7 contains zero, jump to label TONI. R5 is then auto-incremented by 2. R7.19:16 = 0.

```
SUB     @R5+,R7        ; Subtract table number from R7. R5 + 2
JZ      TONI           ; R7 = @R5 (before subtraction)
...                    ; R7 <> @R5 (before subtraction)
```

Example Byte CNT is subtracted from byte R12 points to. The address of CNT is within PC ± 32K. The address R12 points to is in full memory range.

```
SUB.B   CNT,0(R12)     ; Subtract CNT from @R12
```

SUBC[.W] Subtract source word with carry from destination word

SUBC.B Subtract source byte with carry from destination byte

Syntax `SUBC src,dst` or `SUBC.W src,dst`

`SUBC.B src,dst`

Operation (.not.src) + C + dst → dst or dst – (src – 1) + C → dst

Description The source operand is subtracted from the destination operand. This is done by adding the 1s complement of the source + carry to the destination. The source operand is not affected, the result is written to the destination operand. Used for 32, 48, and 64-bit operands.

Status Bits
- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if there is a carry from the MSB, reset otherwise
- V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example A 16-bit constant 7654h is subtracted from R5 with the carry from the previous instruction. R5.19:16 = 0

```
SUBC.W   #7654h,R5       ; Subtract 7654h + C from R5
```

Example A 48-bit number (3 words) pointed to by R5 (20-bit address) is subtracted from a 48-bit counter in RAM, pointed to by R7. R5 points to the next 48-bit number afterwards. The address R7 points to is in full memory range.

```
SUB      @R5+,0(R7)      ; Subtract LSBs. R5 + 2
SUBC     @R5+,2(R7)      ; Subtract MIDs with C. R5 + 2
SUBC     @R5+,4(R7)      ; Subtract MSBs with C. R5 + 2
```

Example Byte CNT is subtracted from the byte, R12 points to. The carry of the previous instruction is used. The address of CNT is in lower 64 K.

```
SUBC.B   &CNT,0(R12)     ; Subtract byte CNT from @R12
```

SWPB Swap bytes

Syntax `SWPB dst`

Operation dst.15:8 ↔ dst.7:0

Description The high and the low byte of the operand are exchanged. PC.19:16 bits are cleared in register mode.

Status Bits Status bits are not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Exchange the bytes of RAM word EDE (lower 64 K)

```
MOV     #1234h,&EDE       ; 1234h -> EDE
SWPB    &EDE              ; 3412h -> EDE
```

Before SWPB

15 8	7 0
High Byte	Low Byte

After SWPB

15 8	7 0
Low Byte	High Byte

Figure 4-42. Swap Bytes in Memory

Before SWPB

19 16	15 8	7 0
x	High Byte	Low Byte

After SWPB

19 16	15 8	7 0
0 ... 0	Low Byte	High Byte

Figure 4-43. Swap Bytes in a Register

SXT Extend sign

Syntax `SXT dst`

Operation dst.7 → dst.15:8, dst.7 → dst.19:8 (register mode)

Description Register mode: the sign of the low byte of the operand is extended into the bits Rdst.19:8.

Rdst.7 = 0: Rdst.19:8 = 000h afterwards

Rdst.7 = 1: Rdst.19:8 = FFFh afterwards

Other modes: the sign of the low byte of the operand is extended into the high byte.

dst.7 = 0: high byte = 00h afterwards

dst.7 = 1: high byte = FFh afterwards

Status Bits
- N: Set if result is negative, reset otherwise
- Z: Set if result is zero, reset otherwise
- C: Set if result is not zero, reset otherwise (C = .not.Z)
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The signed 8-bit data in EDE (lower 64 K) is sign extended and added to the 16-bit signed data in R7.

```
MOV.B   &EDE,R5      ; EDE -> R5. 00XXh
SXT     R5           ; Sign extend low byte to R5.19:8
ADD     R5,R7        ; Add signed 16-bit values
```

Example The signed 8-bit data in EDE (PC +32 K) is sign extended and added to the 20-bit data in R7.

```
MOV.B   EDE,R5       ; EDE -> R5. 00XXh
SXT     R5           ; Sign extend low byte to R5.19:8
ADDA    R5,R7        ; Add signed 20-bit values
```

*** TST[.W]** Test destination

*** TST.B** Test destination

Syntax `TST dst` or `TST.W dst`

`TST.B dst`

Operation dst + 0FFFFh + 1

dst + 0FFh + 1

Emulation CMP #0,dst

CMP.B #0,dst

Description The destination operand is compared with zero. The status bits are set according to the result. The destination is not affected.

Status Bits
- N: Set if destination is negative, reset if positive
- Z: Set if destination contains zero, reset otherwise
- C: Set
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
         TST     R7        ; Test R7
         JN      R7NEG     ; R7 is negative
         JZ      R7ZERO    ; R7 is zero
R7POS    ......            ; R7 is positive but not zero
R7NEG    ......            ; R7 is negative
R7ZERO   ......            ; R7 is zero
```

Example The low byte of R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
         TST.B   R7        ; Test low byte of R7
         JN      R7NEG     ; Low byte of R7 is negative
         JZ      R7ZERO    ; Low byte of R7 is zero
R7POS    ......            ; Low byte of R7 is positive but not zero
R7NEG    .....             ; Low byte of R7 is negative
R7ZERO   ......            ; Low byte of R7 is zero
```

XOR[.W] Exclusive OR source word with destination word

XOR.B Exclusive OR source byte with destination byte

Syntax `XOR src,dst` or `XOR.W src,dst`

`XOR.B src,dst`

Operation src .xor. dst → dst

Description The source and destination operands are exclusively ORed. The result is placed into the destination. The source operand is not affected. The previous content of the destination is lost.

Status Bits
- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if result is not zero, reset otherwise (C = .not. Z)
- V: Set if both operands are negative before execution, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Toggle bits in word CNTR (16-bit data) with information (bit = 1) in address-word TONI. Both operands are located in lower 64 K.

```
XOR     &TONI,&CNTR     ; Toggle bits in CNTR
```

Example A table word pointed to by R5 (20-bit address) is used to toggle bits in R6. R6.19:16 = 0.

```
XOR     @R5,R6          ; Toggle bits in R6
```

Example Reset to zero those bits in the low byte of R7 that are different from the bits in byte EDE. R7.19:8 = 0. The address of EDE is within PC ± 32 K.

```
XOR.B   EDE,R7          ; Set different bits to 1 in R7.
INV.B   R7              ; Invert low byte of R7, high byte is 0h
```

4.6.3 Extended Instructions

The extended MSP430X instructions give the MSP430X CPU full access to its 20-bit address space. MSP430X instructions require an additional word of op-code called the extension word. All addresses, indexes, and immediate numbers have 20-bit values when preceded by the extension word. The MSP430X extended instructions are listed and described in the following pages.

*** ADCX.A** Add carry to destination address-word

*** ADCX.[W]** Add carry to destination word

*** ADCX.B** Add carry to destination byte

Syntax `ADCX.A dst`

`ADCX dst` or `ADCX.W dst`

`ADCX.B dst`

Operation dst + C → dst

Emulation `ADDCX.A #0,dst`

`ADDCX #0,dst`

`ADDCX.B #0,dst`

Description The carry bit (C) is added to the destination operand. The previous contents of the destination are lost.

Status Bits

- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if there is a carry from the MSB of the result, reset otherwise
- V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 40-bit counter, pointed to by R12 and R13, is incremented.

```
INCX.A   @R12     ; Increment lower 20 bits
ADCX.A   @R13     ; Add carry to upper 20 bits
```

ADDX.A Add source address-word to destination address-word

ADDX.[W] Add source word to destination word

ADDX.B Add source byte to destination byte

Syntax `ADDX.A src,dst`

`ADDX src,dst` or `ADDX.W src,dst`

`ADDX.B src,dst`

Operation src + dst → dst

Description The source operand is added to the destination operand. The previous contents of the destination are lost. Both operands can be located in the full address space.

Status Bits
- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if there is a carry from the MSB of the result, reset otherwise
- V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Ten is added to the 20-bit pointer CNTR located in two words CNTR (LSBs) and CNTR+2 (MSBs).

```
ADDX.A   #10,CNTR     ; Add 10 to 20-bit pointer
```

Example A table word (16-bit) pointed to by R5 (20-bit address) is added to R6. The jump to label TONI is performed on a carry.

```
ADDX.W   @R5,R6       ; Add table word to R6
JC       TONI         ; Jump if carry
...                   ; No carry
```

Example A table byte pointed to by R5 (20-bit address) is added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1.

```
ADDX.B   @R5+,R6      ; Add table byte to R6. R5 + 1. R6: 000xxh
JNC      TONI         ; Jump if no carry
...                   ; Carry occurred
```

Note: Use ADDA for the following two cases for better code density and execution.

```
ADDX.A   Rsrc,Rdst
ADDX.A   #imm20,Rdst
```

ADDCX.A Add source address-word and carry to destination address-word

ADDCX.[W] Add source word and carry to destination word

ADDCX.B Add source byte and carry to destination byte

Syntax

`ADDCX.A src,dst`

`ADDCX src,dst` or `ADDCX.W src,dst`

`ADDCX.B src,dst`

Operation src + dst + C → dst

Description The source operand and the carry bit C are added to the destination operand. The previous contents of the destination are lost. Both operands may be located in the full address space.

Status Bits

- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if there is a carry from the MSB of the result, reset otherwise
- V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Constant 15 and the carry of the previous instruction are added to the 20-bit counter CNTR located in two words.

```
ADDCX.A   #15,&CNTR   ; Add 15 + C to 20-bit CNTR
```

Example A table word pointed to by R5 (20-bit address) and the carry C are added to R6. The jump to label TONI is performed on a carry.

```
ADDCX.W   @R5,R6      ; Add table word + C to R6
JC        TONI        ; Jump if carry
...                   ; No carry
```

Example A table byte pointed to by R5 (20-bit address) and the carry bit C are added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1.

```
ADDCX.B   @R5+,R6     ; Add table byte + C to R6. R5 + 1
JNC       TONI        ; Jump if no carry
...                   ; Carry occurred
```

ANDX.A Logical AND of source address-word with destination address-word

ANDX.[W] Logical AND of source word with destination word

ANDX.B Logical AND of source byte with destination byte

Syntax `ANDX.A src,dst`

`ANDX src,dst` or `ANDX.W src,dst`

`ANDX.B src,dst`

Operation src .and. dst → dst

Description The source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected. Both operands may be located in the full address space.

Status Bits
- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if the result is not zero, reset otherwise. C = (.not. Z)
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The bits set in R5 (20-bit data) are used as a mask (AAA55h) for the address-word TOM located in two words. If the result is zero, a branch is taken to label TONI.

```
MOVA     #AAA55h,R5     ; Load 20-bit mask to R5
ANDX.A   R5,TOM         ; TOM .and. R5 -> TOM
JZ       TONI           ; Jump if result 0
...                     ; Result > 0
```

or shorter:

```
ANDX.A   #AAA55h,TOM    ; TOM .and. AAA55h -> TOM
JZ       TONI           ; Jump if result 0
```

Example A table byte pointed to by R5 (20-bit address) is logically ANDed with R6. R6.19:8 = 0. The table pointer is auto-incremented by 1.

```
ANDX.B   @R5+,R6        ; AND table byte with R6. R5 + 1
```

BICX.A Clear bits set in source address-word in destination address-word

BICX.[W] Clear bits set in source word in destination word

BICX.B Clear bits set in source byte in destination byte

Syntax `BICX.A src,dst`

`BICX src,dst` or `BICX.W src,dst`

`BICX.B src,dst`

Operation (.not. src) .and. dst → dst

Description The inverted source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected. Both operands may be located in the full address space.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The bits 19:15 of R5 (20-bit data) are cleared.

```
BICX.A   #0F8000h,R5     ; Clear R5.19:15 bits
```

Example A table word pointed to by R5 (20-bit address) is used to clear bits in R7. R7.19:16 = 0.

```
BICX.W   @R5,R7          ; Clear bits in R7
```

Example A table byte pointed to by R5 (20-bit address) is used to clear bits in output Port1.

```
BICX.B   @R5,&P1OUT      ; Clear I/O port P1 bits
```

BISX.A Set bits set in source address-word in destination address-word

BISX.[W] Set bits set in source word in destination word

BISX.B Set bits set in source byte in destination byte

Syntax `BISX.A src,dst`

`BISX src,dst` or `BISX.W src,dst`

`BISX.B src,dst`

Operation src .or. dst → dst

Description The source operand and the destination operand are logically ORed. The result is placed into the destination. The source operand is not affected. Both operands may be located in the full address space.

Status Bits
- N: Not affected
- Z: Not affected
- C: Not affected
- V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Bits 16 and 15 of R5 (20-bit data) are set to one.

```
BISX.A    #018000h,R5     ; Set R5.16:15 bits
```

Example A table word pointed to by R5 (20-bit address) is used to set bits in R7.

```
BISX.W   @R5,R7           ; Set bits in R7
```

Example A table byte pointed to by R5 (20-bit address) is used to set bits in output Port1.

```
BISX.B   @R5,&P1OUT       ; Set I/O port P1 bits
```

BITX.A Test bits set in source address-word in destination address-word

BITX.[W] Test bits set in source word in destination word

BITX.B Test bits set in source byte in destination byte

Syntax

`BITX.A src,dst`

`BITX src,dst` or `BITX.W src,dst`

`BITX.B src,dst`

Operation src .and. dst → dst

Description The source operand and the destination operand are logically ANDed. The result affects only the status bits. Both operands may be located in the full address space.

Status Bits

- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if the result is not zero, reset otherwise. C = (.not. Z)
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Test if bit 16 or 15 of R5 (20-bit data) is set. Jump to label TONI if so.

```
BITX.A   #018000h,R5     ; Test R5.16:15 bits
JNZ      TONI            ; At least one bit is set
...                      ; Both are reset
```

Example A table word pointed to by R5 (20-bit address) is used to test bits in R7. Jump to label TONI if at least one bit is set.

```
BITX.W   @R5,R7           ; Test bits in R7: C = .not.Z
JC       TONI             ; At least one is set
...                       ; Both are reset
```

Example A table byte pointed to by R5 (20-bit address) is used to test bits in input Port1. Jump to label TONI if no bit is set. The next table byte is addressed.

```
BITX.B   @R5+,&P1IN       ; Test input P1 bits. R5 + 1
JNC      TONI             ; No corresponding input bit is set
...                       ; At least one bit is set
```

*** CLRX.A** Clear destination address-word
*** CLRX.[W]** Clear destination word
*** CLRX.B** Clear destination byte

Syntax
`CLRX.A dst`
`CLRX dst` or `CLRX.W dst`
`CLRX.B dst`

Operation 0 → dst

Emulation
`MOVX.A #0,dst`
`MOVX #0,dst`
`MOVX.B #0,dst`

Description The destination operand is cleared.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example RAM address-word TONI is cleared.

```
CLRX.A    TONI    ; 0 -> TONI
```

CMPX.A Compare source address-word and destination address-word

CMPX.[W] Compare source word and destination word

CMPX.B Compare source byte and destination byte

Syntax `CMPX.A src,dst`

`CMPX src,dst` or `CMPX.W src,dst`

`CMPX.B src,dst`

Operation (.not. src) + 1 + dst or dst – src

Description The source operand is subtracted from the destination operand by adding the 1s complement of the source + 1 to the destination. The result affects only the status bits. Both operands may be located in the full address space.

Status Bits

- N: Set if result is negative (src > dst), reset if positive (src ≤ dst)
- Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)
- C: Set if there is a carry from the MSB, reset otherwise
- V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Compare EDE with a 20-bit constant 18000h. Jump to label TONI if EDE equals the constant.

```
CMPX.A   #018000h,EDE      ; Compare EDE with 18000h
JEQ      TONI              ; EDE contains 18000h
...                        ; Not equal
```

Example A table word pointed to by R5 (20-bit address) is compared with R7. Jump to label TONI if R7 contains a lower, signed, 16-bit number.

```
CMPX.W   @R5,R7            ; Compare two signed numbers
JL       TONI              ; R7 < @R5
...                        ; R7 >= @R5
```

Example A table byte pointed to by R5 (20-bit address) is compared to the input in I/O Port1. Jump to label TONI if the values are equal. The next table byte is addressed.

```
CMPX.B   @R5+,&P1IN        ; Compare P1 bits with table. R5 + 1
JEQ      TONI              ; Equal contents
...                        ; Not equal
```

Note: Use CMPA for the following two cases for better density and execution.

```
CMPA     Rsrc,Rdst
CMPA     #imm20,Rdst
```

*** DADCX.A** Add carry decimally to destination address-word
*** DADCX.[W]** Add carry decimally to destination word
*** DADCX.B** Add carry decimally to destination byte

Syntax `DADCX.A dst`
`DADCX dst` or `DADCX.W dst`
`DADCX.B dst`

Operation dst + C → dst (decimally)

Emulation `DADDX.A #0,dst`
`DADDX #0,dst`
`DADDX.B #0,dst`

Description The carry bit (C) is added decimally to the destination.

Status Bits
N: Set if MSB of result is 1 (address-word > 79999h, word > 7999h, byte > 79h), reset if MSB is 0
Z: Set if result is zero, reset otherwise
C: Set if the BCD result is too large (address-word > 99999h, word > 9999h, byte > 99h), reset otherwise
V: Undefined

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 40-bit counter, pointed to by R12 and R13, is incremented decimally.

```
DADDX.A   #1,0(R12)     ; Increment lower 20 bits
DADCX.A   0(R13)        ; Add carry to upper 20 bits
```

DADDX.A Add source address-word and carry decimally to destination address-word
DADDX.[W] Add source word and carry decimally to destination word
DADDX.B Add source byte and carry decimally to destination byte

Syntax `DADDX.A src,dst`

`DADDX src,dst` or `DADDX.W src,dst`

`DADDX.B src,dst`

Operation src + dst + C → dst (decimally)

Description The source operand and the destination operand are treated as two (.B), four (.W), or five (.A) binary coded decimals (BCD) with positive signs. The source operand and the carry bit C are added decimally to the destination operand. The source operand is not affected. The previous contents of the destination are lost. The result is not defined for non-BCD numbers. Both operands may be located in the full address space.

Status Bits
- N: Set if MSB of result is 1 (address-word > 79999h, word > 7999h, byte > 79h), reset if MSB is 0.
- Z: Set if result is zero, reset otherwise
- C: Set if the BCD result is too large (address-word > 99999h, word > 9999h, byte > 99h), reset otherwise
- V: Undefined

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Decimal 10 is added to the 20-bit BCD counter DECCNTR located in two words.

```
DADDX.A   #10h,&DECCNTR     ; Add 10 to 20-bit BCD counter
```

Example The eight-digit BCD number contained in 20-bit addresses BCD and BCD+2 is added decimally to an eight-digit BCD number contained in R4 and R5 (BCD+2 and R5 contain the MSDs).

```
CLRC                        ; Clear carry
DADDX.W   BCD,R4            ; Add LSDs
DADDX.W   BCD+2,R5          ; Add MSDs with carry
JC        OVERFLOW          ; Result >99999999: go to error routine
...                         ;    Result ok
```

Example The two-digit BCD number contained in 20-bit address BCD is added decimally to a two-digit BCD number contained in R4.

```
CLRC                        ; Clear carry
DADDX.B   BCD,R4            ; Add BCD to R4 decimally.
                            ; R4: 000ddh
```

*** DECX.A** Decrement destination address-word
*** DECX.[W]** Decrement destination word
*** DECX.B** Decrement destination byte

Syntax
`DECX.A dst`
`DECX dst` or `DECX.W dst`
`DECX.B dst`

Operation dst – 1 → dst

Emulation
`SUBX.A #1,dst`
`SUBX #1,dst`
`SUBX.B #1,dst`

Description The destination operand is decremented by one. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 1, reset otherwise
C: Reset if dst contained 0, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example RAM address-word TONI is decremented by one.

```
DECX.A   TONI     ; Decrement TONI
```

*** DECDX.A** Double-decrement destination address-word
*** DECDX.[W]** Double-decrement destination word
*** DECDX.B** Double-decrement destination byte

Syntax
`DECDX.A dst`
`DECDX dst` or `DECDX.W dst`
`DECDX.B dst`

Operation dst – 2 → dst

Emulation
`SUBX.A #2,dst`
`SUBX #2,dst`
`SUBX.B #2,dst`

Description The destination operand is decremented by two. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 2, reset otherwise
C: Reset if dst contained 0 or 1, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example RAM address-word TONI is decremented by two.

```
DECDX.A   TONI     ; Decrement TONI
```

*** INCX.A** Increment destination address-word

*** INCX.[W]** Increment destination word

*** INCX.B** Increment destination byte

Syntax `INCX.A dst`

`INCX dst` or `INCX.W dst`

`INCX.B dst`

Operation dst + 1 → dst

Emulation `ADDX.A #1,dst`

`ADDX #1,dst`

`ADDX.B #1,dst`

Description The destination operand is incremented by one. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFFh, reset otherwise
Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
C: Set if dst contained 0FFFFFh, reset otherwise
Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
V: Set if dst contained 07FFFh, reset otherwise
Set if dst contained 07FFFh, reset otherwise
Set if dst contained 07Fh, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example RAM address-wordTONI is incremented by one.

```
INCX.A   TONI     ; Increment TONI (20-bits)
```

*** INCDX.A** Double-increment destination address-word
*** INCDX.[W]** Double-increment destination word
*** INCDX.B** Double-increment destination byte

Syntax
`INCDX.A dst`
`INCDX dst` or `INCDX.W dst`
`INCDX.B dst`

Operation dst + 2 → dst

Emulation
`ADDX.A #2,dst`
`ADDX #2,dst`
`ADDX.B #2,dst`

Description The destination operand is incremented by two. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFEh, reset otherwise
Set if dst contained 0FFFEh, reset otherwise
Set if dst contained 0FEh, reset otherwise
C: Set if dst contained 0FFFFEh or 0FFFFFh, reset otherwise
Set if dst contained 0FFFEh or 0FFFFh, reset otherwise
Set if dst contained 0FEh or 0FFh, reset otherwise
V: Set if dst contained 07FFFEh or 07FFFFh, reset otherwise
Set if dst contained 07FFEh or 07FFFh, reset otherwise
Set if dst contained 07Eh or 07Fh, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example RAM byte LEO is incremented by two; PC points to upper memory.

```
INCDX.B   LEO     ; Increment LEO by two
```

*** INVX.A** Invert destination
*** INVX.[W]** Invert destination
*** INVX.B** Invert destination

Syntax
`INVX.A dst`
`INVX dst` or `INVX.W dst`
`INVX.B dst`

Operation .NOT.dst → dst

Emulation
`XORX.A #0FFFFFh,dst`
`XORX #0FFFFh,dst`
`XORX.B #0FFh,dst`

Description The destination operand is inverted. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFFh, reset otherwise
Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Set if initial destination operand was negative, otherwise reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example 20-bit content of R5 is negated (2s complement).

```
INVX.A   R5       ; Invert R5
INCX.A   R5       ; R5 is now negated
```

Example Content of memory byte LEO is negated. PC is pointing to upper memory.

```
INVX.B   LEO      ; Invert LEO
INCX.B   LEO      ; MEM(LEO) is negated
```

MOVX.A Move source address-word to destination address-word

MOVX.[W] Move source word to destination word

MOVX.B Move source byte to destination byte

Syntax `MOVX.A src,dst`

`MOVX src,dst` or `MOVX.W src,dst`

`MOVX.B src,dst`

Operation src → dst

Description The source operand is copied to the destination. The source operand is not affected. Both operands may be located in the full address space.

Status Bits
- N: Not affected
- Z: Not affected
- C: Not affected
- V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Move a 20-bit constant 18000h to absolute address-word EDE

```
MOVX.A   #018000h,&EDE         ; Move 18000h to EDE
```

Example The contents of table EDE (word data, 20-bit addresses) are copied to table TOM. The length of the table is 030h words.

```
      MOVA     #EDE,R10               ; Prepare pointer (20-bit address)
Loop  MOVX.W   @R10+,TOM-EDE-2(R10)   ; R10 points to both tables.
                                      ; R10+2
      CMPA     #EDE+60h,R10           ; End of table reached?
      JLO      Loop                   ; Not yet
      ...                             ; Copy completed
```

Example The contents of table EDE (byte data, 20-bit addresses) are copied to table TOM. The length of the table is 020h bytes.

```
      MOVA     #EDE,R10               ; Prepare pointer (20-bit)
      MOV      #20h,R9                ; Prepare counter
Loop  MOVX.W   @R10+,TOM-EDE-2(R10)   ; R10 points to both tables.
                                      ; R10+1
      DEC      R9                     ; Decrement counter
      JNZ      Loop                   ; Not yet done
      ...                             ; Copy completed
```

Ten of the 28 possible addressing combinations of the MOVX.A instruction can use the MOVA instruction. This saves two bytes and code cycles. Examples for the addressing combinations are:

```
MOVX.A   Rsrc,Rdst          MOVA   Rsrc,Rdst         ; Reg/Reg
MOVX.A   #imm20,Rdst        MOVA   #imm20,Rdst       ; Immediate/Reg
MOVX.A   &abs20,Rdst        MOVA   &abs20,Rdst       ; Absolute/Reg
MOVX.A   @Rsrc,Rdst         MOVA   @Rsrc,Rdst        ; Indirect/Reg
MOVX.A   @Rsrc+,Rdst        MOVA   @Rsrc+,Rdst       ; Indirect,Auto/Reg
MOVX.A   Rsrc,&abs20        MOVA   Rsrc,&abs20       ; Reg/Absolute
```

The next four replacements are possible only if 16-bit indexes are sufficient for the addressing:

```
MOVX.A   z20(Rsrc),Rdst      MOVA   z16(Rsrc),Rdst    ; Indexed/Reg
MOVX.A   Rsrc,z20(Rdst)      MOVA   Rsrc,z16(Rdst)    ; Reg/Indexed
MOVX.A   symb20,Rdst         MOVA   symb16,Rdst       ; Symbolic/Reg
MOVX.A   Rsrc,symb20         MOVA   Rsrc,symb16       ; Reg/Symbolic
```

POPM.A Restore n CPU registers (20-bit data) from the stack

POPM.[W] Restore n CPU registers (16-bit data) from the stack

Syntax
`POPM.A #n,Rdst` $1 \le n \le 16$

`POPM.W #n,Rdst` or `POPM #n,Rdst` $1 \le n \le 16$

Operation
POPM.A: Restore the register values from stack to the specified CPU registers. The SP is incremented by four for each register restored from stack. The 20-bit values from stack (two words per register) are restored to the registers.

POPM.W: Restore the 16-bit register values from stack to the specified CPU registers. The SP is incremented by two for each register restored from stack. The 16-bit values from stack (one word per register) are restored to the CPU registers.

Note : This instruction does not use the extension word.

Description
POPM.A: The CPU registers pushed on the stack are moved to the extended CPU registers, starting with the CPU register (Rdst – n + 1). The SP is incremented by (n × 4) after the operation.

POPM.W: The 16-bit registers pushed on the stack are moved back to the CPU registers, starting with CPU register (Rdst – n + 1). The SP is incremented by (n × 2) after the instruction. The MSBs (Rdst.19:16) of the restored CPU registers are cleared.

Status Bits Status bits are not affected, except SR is included in the operation.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Restore the 20-bit registers R9, R10, R11, R12, R13 from the stack

```
POPM.A   #5,R13     ; Restore R9, R10, R11, R12, R13
```

Example Restore the 16-bit registers R9, R10, R11, R12, R13 from the stack.

```
POPM.W   #5,R13     ; Restore R9, R10, R11, R12, R13
```

PUSHM.A Save n CPU registers (20-bit data) on the stack

PUSHM.[W] Save n CPU registers (16-bit words) on the stack

Syntax `PUSHM.A #n,Rdst` $1 \leq n \leq 16$

`PUSHM.W #n,Rdst` or `PUSHM #n,Rdst` $1 \leq n \leq 16$

Operation PUSHM.A: Save the 20-bit CPU register values on the stack. The SP is decremented by four for each register stored on the stack. The MSBs are stored first (higher address).

PUSHM.W: Save the 16-bit CPU register values on the stack. The SP is decremented by two for each register stored on the stack.

Description PUSHM.A: The n CPU registers, starting with Rdst backwards, are stored on the stack. The SP is decremented by (n × 4) after the operation. The data (Rn.19:0) of the pushed CPU registers is not affected.

PUSHM.W: The n registers, starting with Rdst backwards, are stored on the stack. The SP is decremented by (n × 2) after the operation. The data (Rn.19:0) of the pushed CPU registers is not affected.

Note : This instruction does not use the extension word.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Save the five 20-bit registers R9, R10, R11, R12, R13 on the stack

```
PUSHM.A   #5,R13     ; Save R13, R12, R11, R10, R9
```

Example Save the five 16-bit registers R9, R10, R11, R12, R13 on the stack

```
PUSHM.W   #5,R13     ; Save R13, R12, R11, R10, R9
```

*** POPX.A** Restore single address-word from the stack
*** POPX.[W]** Restore single word from the stack
*** POPX.B** Restore single byte from the stack

Syntax `POPX.A dst`

`POPX dst` or `POPX.W dst`

`POPX.B dst`

Operation Restore the 8-/16-/20-bit value from the stack to the destination. 20-bit addresses are possible. The SP is incremented by two (byte and word operands) and by four (address-word operand).

Emulation `MOVX(.B,.A) @SP+,dst`

Description The item on TOS is written to the destination operand. Register mode, Indexed mode, Symbolic mode, and Absolute mode are possible. The SP is incremented by two or four.

Note: the SP is incremented by two also for byte operations.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Write the 16-bit value on TOS to the 20-bit address &EDE

```
POPX.W   &EDE     ; Write word to address EDE
```

Example Write the 20-bit value on TOS to R9

```
POPX.A   R9     ; Write address-word to R9
```

PUSHX.A Save single address-word to the stack
PUSHX.[W] Save single word to the stack
PUSHX.B Save single byte to the stack

Syntax `PUSHX.A src`

`PUSHX src` or `PUSHX.W src`

`PUSHX.B src`

Operation Save the 8-/16-/20-bit value of the source operand on the TOS. 20-bit addresses are possible. The SP is decremented by two (byte and word operands) or by four (address-word operand) before the write operation.

Description The SP is decremented by two (byte and word operands) or by four (address-word operand). Then the source operand is written to the TOS. All seven addressing modes are possible for the source operand.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Save the byte at the 20-bit address &EDE on the stack

```
PUSHX.B   &EDE     ; Save byte at address EDE
```

Example Save the 20-bit value in R9 on the stack.

```
PUSHX.A   R9       ; Save address-word in R9
```

RLAM.A Rotate left arithmetically the 20-bit CPU register content

RLAM.[W] Rotate left arithmetically the 16-bit CPU register content

Syntax `RLAM.A #n,Rdst` 1 ≤ n ≤ 4

`RLAM.W #n,Rdst` or `RLAM #n,Rdst` 1 ≤ n ≤ 4

Operation C ← MSB ← MSB-1 .... LSB+1 ← LSB ← 0

Description The destination operand is shifted arithmetically left one, two, three, or four positions as shown in Figure 4-44. RLAM works as a multiplication (signed and unsigned) with 2, 4, 8, or 16. The word instruction RLAM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits

N: Set if result is negative

.A: Rdst.19 = 1, reset if Rdst.19 = 0

.W: Rdst.15 = 1, reset if Rdst.15 = 0

Z: Set if result is zero, reset otherwise

C: Loaded from the MSB (n = 1), MSB-1 (n = 2), MSB-2 (n = 3), MSB-3 (n = 4)

V: Undefined

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit operand in R5 is shifted left by three positions. It operates equal to an arithmetic multiplication by 8.

```
RLAM.A   #3,R5     ; R5 = R5 x 8
```

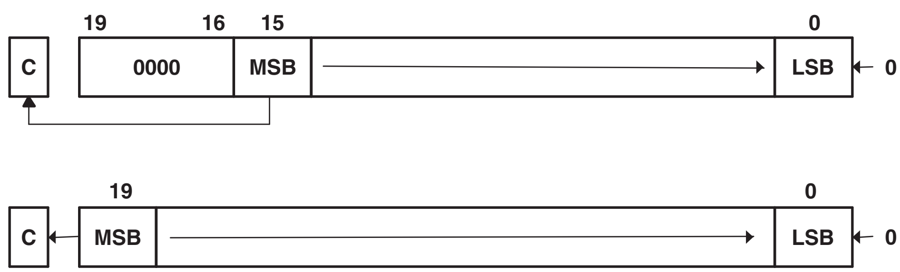


Figure 4-44. Rotate Left Arithmetically—RLAM[.W] and RLAM.A

*** RLAX.A** Rotate left arithmetically address-word

*** RLAX.[W]** Rotate left arithmetically word

*** RLAX.B** Rotate left arithmetically byte

Syntax
`RLAX.A dst`
`RLAX dst` or `RLAX.W dst`
`RLAX.B dst`

Operation C ← MSB ← MSB-1 .... LSB+1 ← LSB ← 0

Emulation
`ADDX.A dst,dst`
`ADDX dst,dst`
`ADDX.B dst,dst`

Description The destination operand is shifted left one position as shown in Figure 4-45. The MSB is shifted into the carry bit (C) and the LSB is filled with 0. The RLAX instruction acts as a signed multiplication by 2.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the MSB
V: Set if an arithmetic overflow occurs: the initial value is 040000h ≤ dst < 0C0000h; reset otherwise
Set if an arithmetic overflow occurs: the initial value is 04000h ≤ dst < 0C000h; reset otherwise
Set if an arithmetic overflow occurs: the initial value is 040h ≤ dst < 0C0h; reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit value in R7 is multiplied by 2

```
RLAX.A   R7     ; Shift left R7 (20-bit)
```

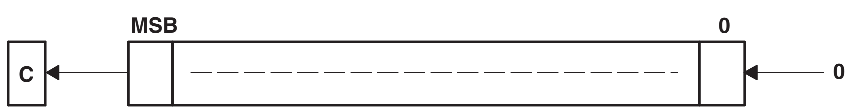


Figure 4-45. Destination Operand-Arithmetic Shift Left

*** RLCX.A** Rotate left through carry address-word
*** RLCX.[W]** Rotate left through carry word
*** RLCX.B** Rotate left through carry byte

Syntax
`RLCX.A dst`
`RLCX dst` or `RLCX.W dst`
`RLCX.B dst`

Operation C ← MSB ← MSB-1 .... LSB+1 ← LSB ← C

Emulation
`ADDCX.A dst,dst`
`ADDCX dst,dst`
`ADDCX.B dst,dst`

Description The destination operand is shifted left one position as shown in Figure 4-46. The carry bit (C) is shifted into the LSB and the MSB is shifted into the carry bit (C).

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the MSB
V: Set if an arithmetic overflow occurs: the initial value is 040000h ≤ dst < 0C0000h; reset otherwise
Set if an arithmetic overflow occurs: the initial value is 04000h ≤ dst < 0C000h; reset otherwise
Set if an arithmetic overflow occurs: the initial value is 040h ≤ dst < 0C0h; reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit value in R5 is shifted left one position.

```
RLCX.A   R5     ; (R5 x 2) + C -> R5
```

Example The RAM byte LEO is shifted left one position. PC is pointing to upper memory.

```
RLCX.B   LEO    ; RAM(LEO) x 2 + C -> RAM(LEO)
```

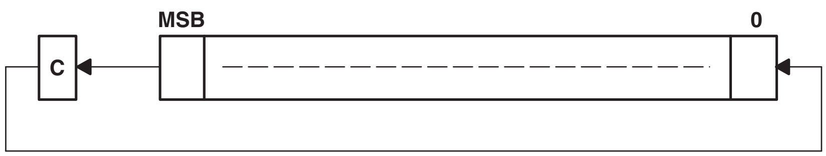


Figure 4-46. Destination Operand-Carry Left Shift

RRAM.A Rotate right arithmetically the 20-bit CPU register content
RRAM.[W] Rotate right arithmetically the 16-bit CPU register content

Syntax `RRAM.A #n,Rdst` $1 \le n \le 4$

`RRAM.W #n,Rdst` or `RRAM #n,Rdst` $1 \le n \le 4$

Operation MSB → MSB → MSB–1 ... LSB+1 → LSB → C

Description The destination operand is shifted right arithmetically by one, two, three, or four bit positions as shown in Figure 4-47. The MSB retains its value (sign). RRAM operates equal to a signed division by 2/4/8/16. The MSB is retained and shifted into MSB-1. The LSB+1 is shifted into the LSB, and the LSB is shifted into the carry bit C. The word instruction RRAM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits
- N: Set if result is negative
  - .A: Rdst.19 = 1, reset if Rdst.19 = 0
  - .W: Rdst.15 = 1, reset if Rdst.15 = 0
- Z: Set if result is zero, reset otherwise
- C: Loaded from the LSB (n = 1), LSB+1 (n = 2), LSB+2 (n = 3), or LSB+3 (n = 4)
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The signed 20-bit number in R5 is shifted arithmetically right two positions.

```
RRAM.A   #2,R5          ; R5/4 -> R5
```

Example The signed 20-bit value in R15 is multiplied by 0.75. (0.5 + 0.25) × R15.

```
PUSHM.A   #1,R15        ; Save extended R15 on stack
RRAM.A    #1,R15        ; R15 y 0.5 -> R15
ADDX.A    @SP+,R15      ; R15 y 0.5 + R15 = 1.5 y R15 -> R15
RRAM.A    #1,R15        ; (1.5 y R15) y 0.5 = 0.75 y R15 -> R15
```

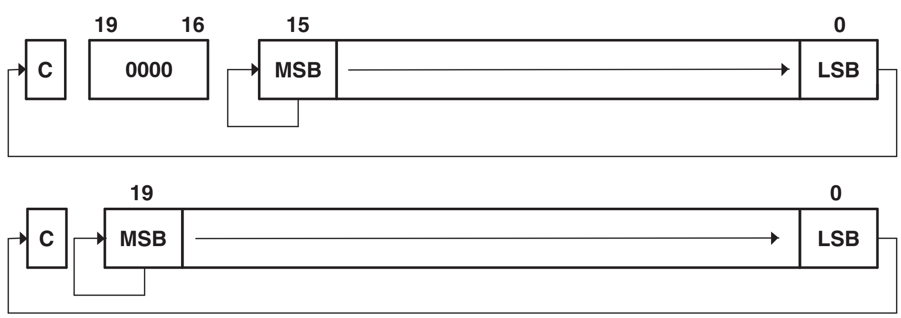


Figure 4-47. Rotate Right Arithmetically RRAM[.W] and RRAM.A

RRAX.A Rotate right arithmetically the 20-bit operand

RRAX.[W] Rotate right arithmetically the 16-bit operand

RRAX.B Rotate right arithmetically the 8-bit operand

Syntax

`RRAX.A Rdst`

`RRAX.W Rdst`

`RRAX Rdst`

`RRAX.B Rdst`

`RRAX.A dst`

`RRAX dst` or `RRAX.W dst`

`RRAX.B dst`

Operation MSB → MSB → MSB–1 ... LSB+1 → LSB → C

Description Register mode for the destination: the destination operand is shifted right by one bit position as shown in Figure 4-48. The MSB retains its value (sign). The word instruction RRAX.W clears the bits Rdst.19:16, the byte instruction RRAX.B clears the bits Rdst.19:8. The MSB retains its value (sign), the LSB is shifted into the carry bit. RRAX here operates equal to a signed division by 2.

All other modes for the destination: the destination operand is shifted right arithmetically by one bit position as shown in Figure 4-49. The MSB retains its value (sign), the LSB is shifted into the carry bit. RRAX here operates equal to a signed division by 2. All addressing modes, with the exception of the Immediate mode, are possible in the full memory.

Status Bits

N: Set if result is negative, reset if positive
 .A: dst.19 = 1, reset if dst.19 = 0
 .W: dst.15 = 1, reset if dst.15 = 0
 .B: dst.7 = 1, reset if dst.7 = 0

Z: Set if result is zero, reset otherwise

C: Loaded from the LSB

V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The signed 20-bit number in R5 is shifted arithmetically right four positions.

```
RPT     #4
RRAX.A  R5          ; R5/16 -> R5
```

Example The signed 8-bit value in EDE is multiplied by 0.5.

```
RRAX.B    &EDE       ; EDE/2 -> EDE
```

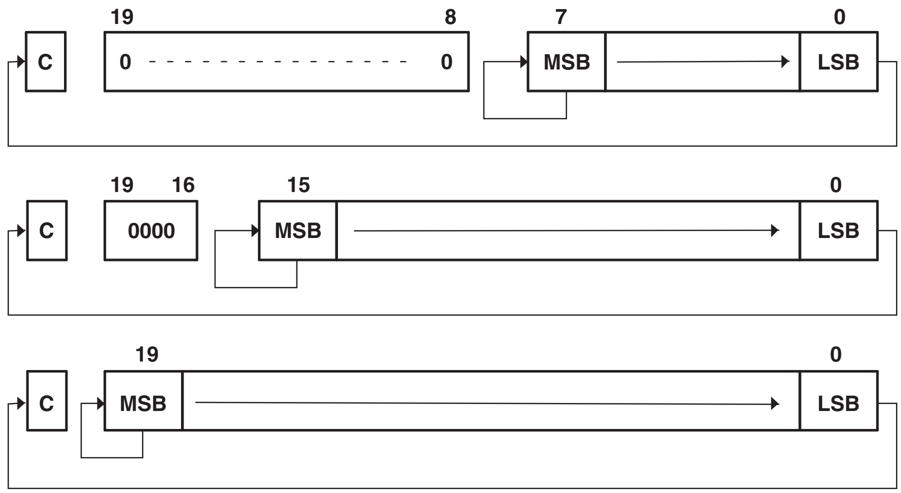


Figure 4-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode

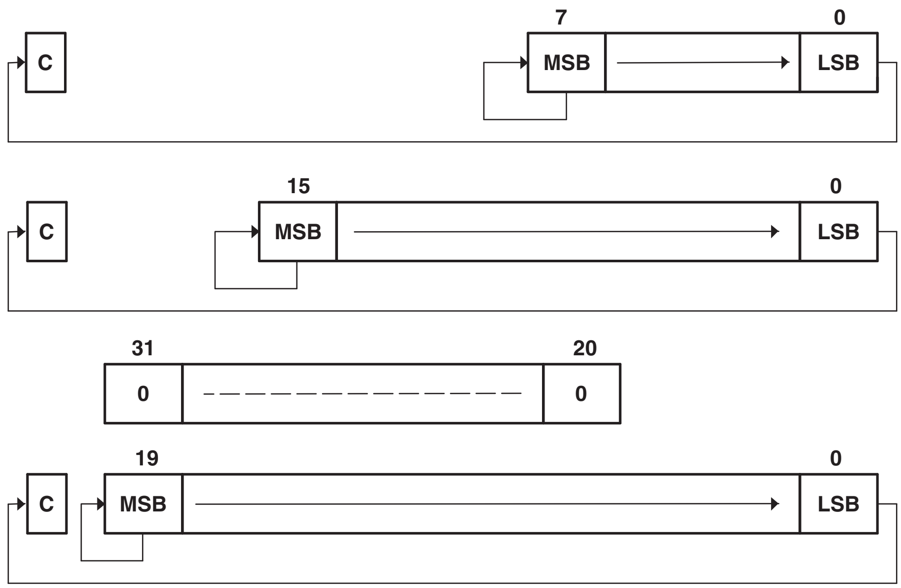


Figure 4-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode

RRCM.A Rotate right through carry the 20-bit CPU register content

RRCM.[W] Rotate right through carry the 16-bit CPU register content

Syntax `RRCM.A #n,Rdst` $1 \le n \le 4$

`RRCM.W #n,Rdst` or `RRCM #n,Rdst` $1 \le n \le 4$

Operation C → MSB → MSB–1 ... LSB+1 → LSB → C

Description The destination operand is shifted right by one, two, three, or four bit positions as shown in Figure 4-50. The carry bit C is shifted into the MSB, the LSB is shifted into the carry bit. The word instruction RRCM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits
N: Set if result is negative
.A: Rdst.19 = 1, reset if Rdst.19 = 0
.W: Rdst.15 = 1, reset if Rdst.15 = 0
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB (n = 1), LSB+1 (n = 2), LSB+2 (n = 3), or LSB+3 (n = 4)
V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The address-word in R5 is shifted right by three positions. The MSB–2 is loaded with 1.

```
SETC                ; Prepare carry for MSB-2
RRCM.A   #3,R5      ; R5 = R5 » 3 + 20000h
```

Example The word in R6 is shifted right by two positions. The MSB is loaded with the LSB. The MSB–1 is loaded with the contents of the carry flag.

```
RRCM.W   #2,R6      ; R6 = R6 » 2. R6.19:16 = 0
```

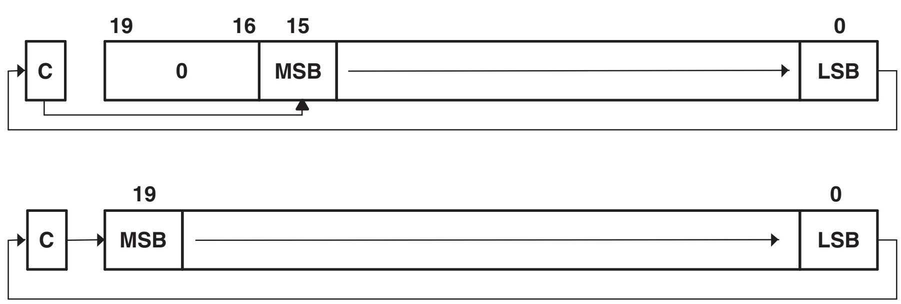


Figure 4-50. Rotate Right Through Carry RRCM[.W] and RRCM.A

RRCX.A Rotate right through carry the 20-bit operand
RRCX.[W] Rotate right through carry the 16-bit operand
RRCX.B Rotate right through carry the 8-bit operand

Syntax
`RRCX.A Rdst`
`RRCX.W Rdst`
`RRCX Rdst`
`RRCX.B Rdst`

`RRCX.A dst`
`RRCX dst` or `RRCX.W dst`
`RRCX.B dst`

Operation C → MSB → MSB–1 ... LSB+1 → LSB → C

Description Register mode for the destination: the destination operand is shifted right by one bit position as shown in Figure 4-51. The word instruction RRCX.W clears the bits Rdst.19:16, the byte instruction RRCX.B clears the bits Rdst.19:8. The carry bit C is shifted into the MSB, the LSB is shifted into the carry bit.

All other modes for the destination: the destination operand is shifted right by one bit position as shown in Figure 4-52. The carry bit C is shifted into the MSB, the LSB is shifted into the carry bit. All addressing modes, with the exception of the Immediate mode, are possible in the full memory.

Status Bits
N: Set if result is negative
.A: dst.19 = 1, reset if dst.19 = 0
.W: dst.15 = 1, reset if dst.15 = 0
.B: dst.7 = 1, reset if dst.7 = 0
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB
V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit operand at address EDE is shifted right by one position. The MSB is loaded with 1.

```
SETC                ; Prepare carry for MSB
RRCX.A    EDE       ; EDE = EDE » 1 + 80000h
```

Example The word in R6 is shifted right by 12 positions.

```
RPT      #12
RRCX.W   R6        ; R6 = R6 » 12. R6.19:16 = 0
```

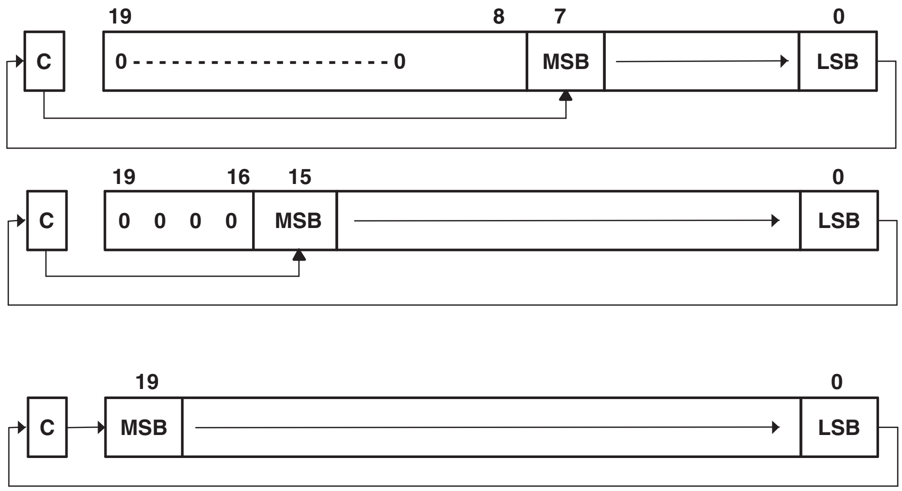


Figure 4-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode

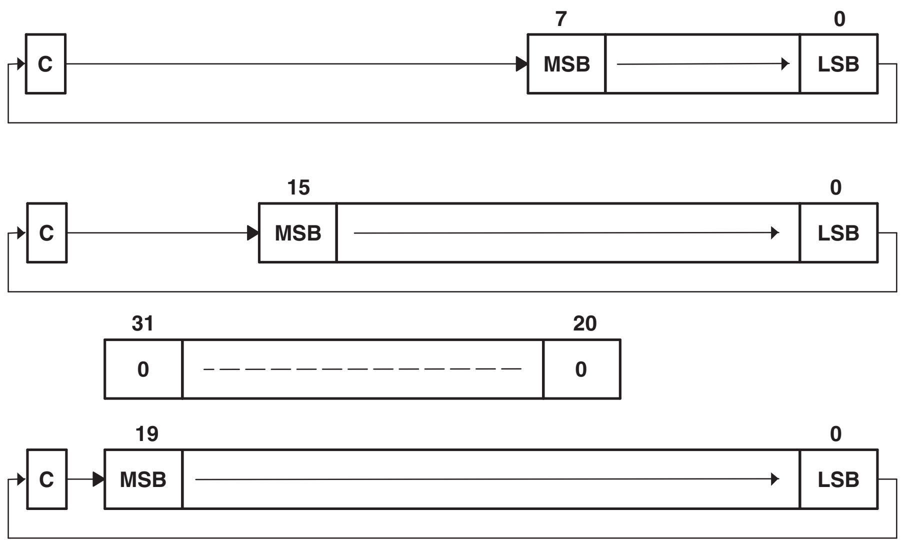


Figure 4-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode

RRUM.A Rotate right through carry the 20-bit CPU register content

RRUM.[W] Rotate right through carry the 16-bit CPU register content

Syntax `RRUM.A #n,Rdst` $1 \le n \le 4$

`RRUM.W #n,Rdst` or `RRUM #n,Rdst` $1 \le n \le 4$

Operation 0 → MSB → MSB–1 ... LSB+1 → LSB → C

Description The destination operand is shifted right by one, two, three, or four bit positions as shown in Figure 4-53. Zero is shifted into the MSB, the LSB is shifted into the carry bit. RRUM works like an unsigned division by 2, 4, 8, or 16. The word instruction RRUM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits
- N: Set if result is negative
  - .A: Rdst.19 = 1, reset if Rdst.19 = 0
  - .W: Rdst.15 = 1, reset if Rdst.15 = 0
- Z: Set if result is zero, reset otherwise
- C: Loaded from the LSB (n = 1), LSB+1 (n = 2), LSB+2 (n = 3), or LSB+3 (n = 4)
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The unsigned address-word in R5 is divided by 16.

```
RRUM.A   #4,R5      ; R5 = R5 » 4. R5/16
```

Example The word in R6 is shifted right by one bit. The MSB R6.15 is loaded with 0.

```
RRUM.W   #1,R6      ; R6 = R6/2. R6.19:15 = 0
```

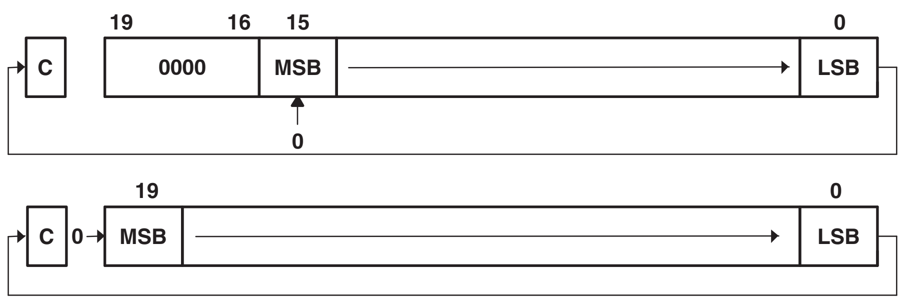


Figure 4-53. Rotate Right Unsigned RRUM[.W] and RRUM.A

RRUX.A Shift right unsigned the 20-bit CPU register content
RRUX.[W] Shift right unsigned the 16-bit CPU register content
RRUX.B Shift right unsigned the 8-bit CPU register content

Syntax
`RRUX.A Rdst`
`RRUX.W Rdst`
`RRUX Rdst`
`RRUX.B Rdst`

Operation C=0 → MSB → MSB–1 ... LSB+1 → LSB → C

Description RRUX is valid for register mode only: the destination operand is shifted right by one bit position as shown in Figure 4-54. The word instruction RRUX.W clears the bits Rdst.19:16. The byte instruction RRUX.B clears the bits Rdst.19:8. Zero is shifted into the MSB, the LSB is shifted into the carry bit.

Status Bits
N: Set if result is negative
.A: dst.19 = 1, reset if dst.19 = 0
.W: dst.15 = 1, reset if dst.15 = 0
.B: dst.7 = 1, reset if dst.7 = 0
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB
V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The word in R6 is shifted right by 12 positions.

```
RPT     #12
RRUX.W  R6       ; R6 = R6 » 12. R6.19:16 = 0
```

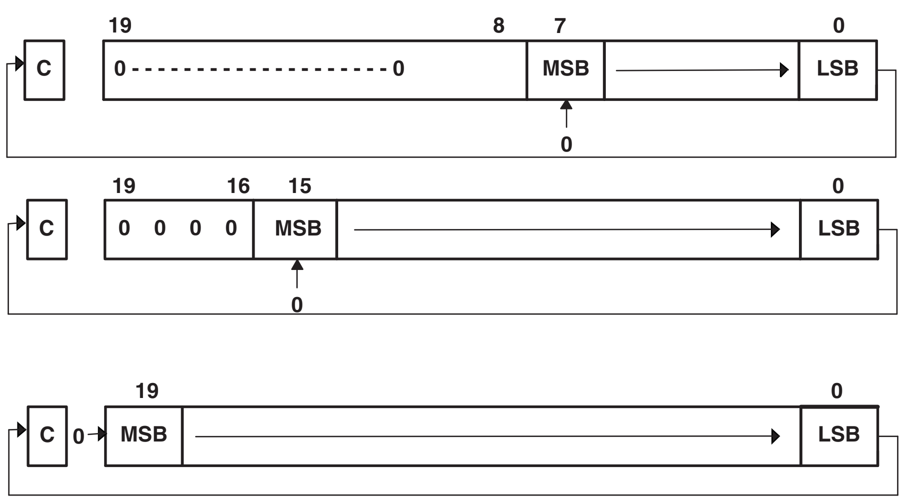


Figure 4-54. Rotate Right Unsigned RRUX(.B,.A) – Register Mode

*** SBCX.A** Subtract borrow (.NOT. carry) from destination address-word

*** SBCX.[W]** Subtract borrow (.NOT. carry) from destination word

*** SBCX.B** Subtract borrow (.NOT. carry) from destination byte

Syntax `SBCX.A dst`

`SBCX dst` or `SBCX.W dst`

`SBCX.B dst`

Operation dst + 0FFFFFh + C → dst

dst + 0FFFFh + C → dst

dst + 0FFh + C → dst

Emulation `SBCX.A #0,dst`

`SBCX #0,dst`

`SBCX.B #0,dst`

Description The carry bit (C) is added to the destination operand minus one. The previous contents of the destination are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset otherwise
Set to 1 if no borrow, reset if borrow
V: Set if an arithmetic overflow occurs, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.

```
SUBX.B   @R13,0(R12)      ; Subtract LSDs
SBCX.B   1(R12)           ; Subtract carry from MSD
```

NOTE: Borrow implementation

The borrow is treated as a .NOT. carry:

Borrow	Carry Bit
Yes	0
No	1

SUBX.A Subtract source address-word from destination address-word

SUBX.[W] Subtract source word from destination word

SUBX.B Subtract source byte from destination byte

Syntax
`SUBX.A src,dst`
`SUBX src,dst` or `SUBX.W src,dst`
`SUBX.B src,dst`

Operation (.not. src) + 1 + dst → dst   or   dst – src → dst

Description The source operand is subtracted from the destination operand. This is done by adding the 1s complement of the source + 1 to the destination. The source operand is not affected. The result is written to the destination operand. Both operands may be located in the full address space.

Status Bits
- N: Set if result is negative (src > dst), reset if positive (src ≤ dst)
- Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)
- C: Set if there is a carry from the MSB, reset otherwise
- V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example A 20-bit constant 87654h is subtracted from EDE (LSBs) and EDE+2 (MSBs).

```
SUBX.A   #87654h,EDE     ; Subtract 87654h from EDE+2|EDE
```

Example A table word pointed to by R5 (20-bit address) is subtracted from R7. Jump to label TONI if R7 contains zero after the instruction. R5 is auto-incremented by two. R7.19:16 = 0.

```
SUBX.W   @R5+,R7         ; Subtract table number from R7. R5 + 2
JZ       TONI            ; R7 = @R5 (before subtraction)
...                      ; R7 <> @R5 (before subtraction)
```

Example Byte CNT is subtracted from the byte R12 points to in the full address space. Address of CNT is within PC ± 512 K.

```
SUBX.B   CNT,0(R12)      ; Subtract CNT from @R12
```

Note: Use SUBA for the following two cases for better density and execution.

```
SUBX.A   Rsrc,Rdst
SUBX.A   #imm20,Rdst
```

SUBCX.A Subtract source address-word with carry from destination address-word
SUBCX.[W] Subtract source word with carry from destination word
SUBCX.B Subtract source byte with carry from destination byte

Syntax
`SUBCX.A src,dst`
`SUBCX src,dst` or `SUBCX.W src,dst`
`SUBCX.B src,dst`

Operation (.not. src) + C + dst → dst or dst – (src – 1) + C → dst

Description The source operand is subtracted from the destination operand. This is made by adding the 1s complement of the source + carry to the destination. The source operand is not affected, the result is written to the destination operand. Both operands may be located in the full address space.

Status Bits
- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if there is a carry from the MSB, reset otherwise
- V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow).

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example A 20-bit constant 87654h is subtracted from R5 with the carry from the previous instruction.

```
SUBCX.A   #87654h,R5      ; Subtract 87654h + C from R5
```

Example A 48-bit number (3 words) pointed to by R5 (20-bit address) is subtracted from a 48-bit counter in RAM, pointed to by R7. R5 auto-increments to point to the next 48-bit number.

```
SUBX.W    @R5+,0(R7)      ; Subtract LSBs. R5 + 2
SUBCX.W   @R5+,2(R7)      ; Subtract MIDs with C. R5 + 2
SUBCX.W   @R5+,4(R7)      ; Subtract MSBs with C. R5 + 2
```

Example Byte CNT is subtracted from the byte R12 points to. The carry of the previous instruction is used. 20-bit addresses.

```
SUBCX.B   &CNT,0(R12)     ; Subtract byte CNT from @R12
```

SWPBX.A Swap bytes of lower word

SWPBX.[W] Swap bytes of word

Syntax `SWPBX.A dst`

`SWPBX dst` or `SWPBX.W dst`

Operation dst.15:8 ↔ dst.7:0

Description Register mode: Rn.15:8 are swapped with Rn.7:0. When the .A extension is used, Rn.19:16 are unchanged. When the .W extension is used, Rn.19:16 are cleared.

Other modes: When the .A extension is used, bits 31:20 of the destination address are cleared, bits 19:16 are left unchanged, and bits 15:8 are swapped with bits 7:0. When the .W extension is used, bits 15:8 are swapped with bits 7:0 of the addressed word.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Exchange the bytes of RAM address-word EDE

```
MOVX.A     #23456h,&EDE       ; 23456h -> EDE
SWPBX.A    EDE                ; 25634h -> EDE
```

Example Exchange the bytes of R5

```
MOVA       #23456h,R5         ; 23456h -> R5
SWPBX.W    R5                 ; 05634h -> R5
```

Before SWPBX.A

19 16	15 8	7 0
X	High Byte	Low Byte

After SWPBX.A

19 16	15 8	7 0
X	Low Byte	High Byte

Figure 4-55. Swap Bytes SWPBX.A Register Mode

Before SWPBX.A

31 20	19 16	15 8	7 0
X	X	High Byte	Low Byte

After SWPBX.A

31 20	19 16	15 8	7 0
0	X	Low Byte	High Byte

Figure 4-56. Swap Bytes SWPBX.A In Memory

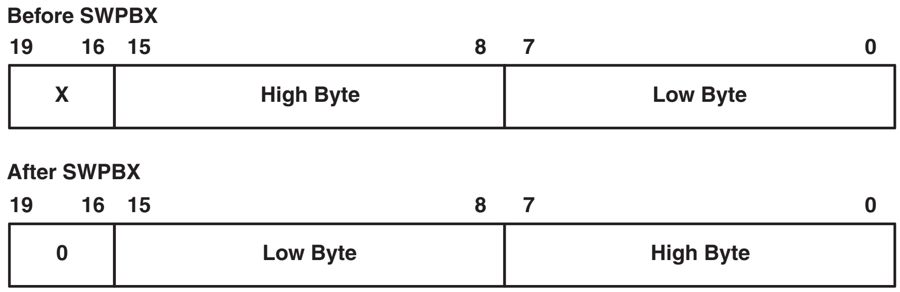


Figure 4-57. Swap Bytes SWPBX[.W] Register Mode

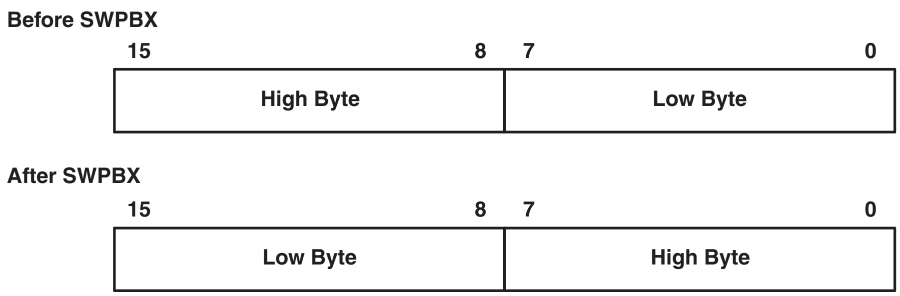


Figure 4-58. Swap Bytes SWPBX[.W] In Memory

SXTX.A Extend sign of lower byte to address-word

SXTX.[W] Extend sign of lower byte to word

Syntax `SXTX.A dst`

`SXTX dst` or `SXTX.W dst`

Operation dst.7 → dst.15:8, Rdst.7 → Rdst.19:8 (Register mode)

Description Register mode: The sign of the low byte of the operand (Rdst.7) is extended into the bits Rdst.19:8.

Other modes: SXTX.A: the sign of the low byte of the operand (dst.7) is extended into dst.19:8. The bits dst.31:20 are cleared.

SXTX[.W]: the sign of the low byte of the operand (dst.7) is extended into dst.15:8.

Status Bits
- N: Set if result is negative, reset otherwise
- Z: Set if result is zero, reset otherwise
- C: Set if result is not zero, reset otherwise (C = .not.Z)
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The signed 8-bit data in EDE.7:0 is sign extended to 20 bits: EDE.19:8. Bits 31:20 located in EDE+2 are cleared.

```
SXTX.A      &EDE            ; Sign extended EDE -> EDE+2/EDE
```

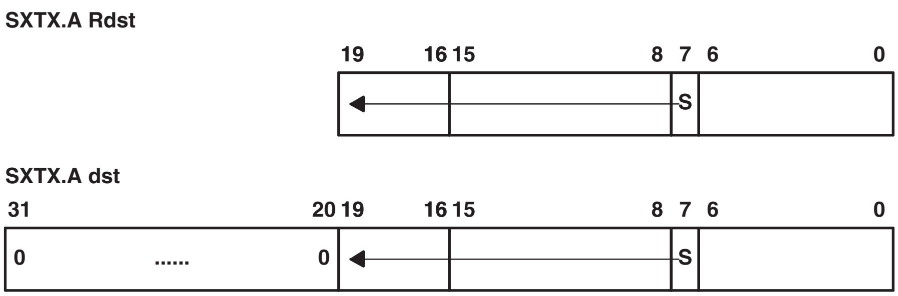


Figure 4-59. Sign Extend SXTX.A

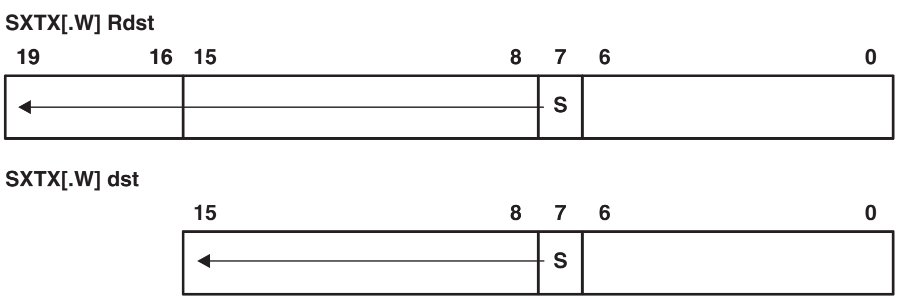


Figure 4-60. Sign Extend SXTX[.W]

*** TSTX.A** Test destination address-word
*** TSTX.[W]** Test destination word
*** TSTX.B** Test destination byte

Syntax
`TSTX.A dst`
`TSTX dst` or `TSTX.W dst`
`TSTX.B dst`

Operation
dst + 0FFFFFh + 1
dst + 0FFFFh + 1
dst + 0FFh + 1

Emulation
`CMPX.A #0,dst`
`CMPX #0,dst`
`CMPX.B #0,dst`

Description The destination operand is compared with zero. The status bits are set according to the result. The destination is not affected.

Status Bits
N: Set if destination is negative, reset if positive
Z: Set if destination contains zero, reset otherwise
C: Set
V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example RAM byte LEO is tested; PC is pointing to upper memory. If it is negative, continue at LEONEG; if it is positive but not zero, continue at LEOPOS.

```
          TSTX.B   LEO          ; Test LEO
          JN       LEONEG       ; LEO is negative
          JZ       LEOZERO      ; LEO is zero
LEOPOS    ......                ; LEO is positive but not zero
LEONEG    ......                ; LEO is negative
LEOZERO   ......                ; LEO is zero
```

XORX.A Exclusive OR source address-word with destination address-word

XORX.[W] Exclusive OR source word with destination word

XORX.B Exclusive OR source byte with destination byte

Syntax
`XORX.A src,dst`
`XORX src,dst` or `XORX.W src,dst`
`XORX.B src,dst`

Operation src .xor. dst → dst

Description The source and destination operands are exclusively ORed. The result is placed into the destination. The source operand is not affected. The previous contents of the destination are lost. Both operands may be located in the full address space.

Status Bits
- N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if result is not zero, reset otherwise (carry = .not. Zero)
- V: Set if both operands are negative (before execution), reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Toggle bits in address-word CNTR (20-bit data) with information in address-word TONI (20-bit address)

```
XORX.A   TONI,&CNTR       ; Toggle bits in CNTR
```

Example A table word pointed to by R5 (20-bit address) is used to toggle bits in R6.

```
XORX.W   @R5,R6           ; Toggle bits in R6. R6.19:16 = 0
```

Example Reset to zero those bits in the low byte of R7 that are different from the bits in byte EDE (20-bit address)

```
XORX.B   EDE,R7           ; Set different bits to 1 in R7
INV.B    R7               ; Invert low byte of R7. R7.19:8 = 0.
```

4.6.4 Address Instructions

MSP430X address instructions are instructions that support 20-bit operands but have restricted addressing modes. The addressing modes are restricted to the Register mode and the Immediate mode, except for the MOVA instruction. Restricting the addressing modes removes the need for the additional extension-word op-code improving code density and execution time. The MSP430X address instructions are listed and described in the following pages.

ADDA Add 20-bit source to a 20-bit destination register

Syntax `ADDA Rsrc,Rdst`

`ADDA #imm20,Rdst`

Operation src + Rdst → Rdst

Description The 20-bit source operand is added to the 20-bit destination CPU register. The previous contents of the destination are lost. The source operand is not affected.

Status Bits
- N: Set if result is negative (Rdst.19 = 1), reset if positive (Rdst.19 = 0)
- Z: Set if result is zero, reset otherwise
- C: Set if there is a carry from the 20-bit result, reset otherwise
- V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example R5 is increased by 0A4320h. The jump to TONI is performed if a carry occurs.

```
ADDA    #0A4320h,R5     ; Add A4320h to 20-bit R5
JC      TONI            ; Jump on carry
...                     ; No carry occurred
```

*** BRA** Branch to destination

Syntax `BRA dst`

Operation dst → PC

Emulation `MOVA dst,PC`

Description An unconditional branch is taken to a 20-bit address anywhere in the full address space. All seven source addressing modes can be used. The branch instruction is an address-word instruction. If the destination address is contained in a memory location X, it is contained in two ascending words: X (LSBs) and (X + 2) (MSBs).

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Examples Examples for all addressing modes are given.

Immediate mode: Branch to label EDE located anywhere in the 20-bit address space or branch directly to address.

```
BRA     #EDE        ; MOVA   #imm20,PC
BRA     #01AA04h
```

Symbolic mode: Branch to the 20-bit address contained in addresses EXEC (LSBs) and EXEC+2 (MSBs). EXEC is located at the address (PC + X) where X is within +32 K. Indirect addressing.

```
BRA     EXEC        ; MOVA   z16(PC),PC
```

Note: If the 16-bit index is not sufficient, a 20-bit index may be used with the following instruction.

```
MOVX.A  EXEC,PC     ; 1M byte range with 20-bit index
```

Absolute mode: Branch to the 20-bit address contained in absolute addresses EXEC (LSBs) and EXEC+2 (MSBs). Indirect addressing.

```
BRA     &EXEC       ; MOVA   &abs20,PC
```

Register mode: Branch to the 20-bit address contained in register R5. Indirect R5.

```
BRA     R5          ; MOVA   R5,PC
```

Indirect mode: Branch to the 20-bit address contained in the word pointed to by register R5 (LSBs). The MSBs have the address (R5 + 2). Indirect, indirect R5.

```
BRA     @R5         ; MOVA   @R5,PC
```

Indirect, Auto-Increment mode: Branch to the 20-bit address contained in the words pointed to by register R5 and increment the address in R5 afterwards by 4. The next time the S/W flow uses R5 as a pointer, it can alter the program execution due to access to the next address in the table pointed to by R5. Indirect, indirect R5.

```
BRA     @R5+        ; MOVA   @R5+,PC. R5 + 4
```

Indexed mode: Branch to the 20-bit address contained in the address pointed to by register (R5 + X) (e.g., a table with addresses starting at X). (R5 + X) points to the LSBs, (R5 + X + 2) points to the MSBs of the address. X is within R5 + 32 K. Indirect, indirect (R5 + X).

```
BRA      X(R5)        ; MOVA   z16(R5),PC
```

Note: If the 16-bit index is not sufficient, a 20-bit index X may be used with the following instruction:

```
MOVX.A   X(R5),PC     ; 1M byte range with 20-bit index
```

CALLA Call a subroutine

Syntax `CALLA dst`

Operation

dst → tmp 20-bit dst is evaluated and stored

SP – 2 → SP

PC.19:16 → @SP updated PC with return address to TOS (MSBs)

SP – 2 → SP

PC.15:0 → @SP updated PC to TOS (LSBs)

tmp → PC saved 20-bit dst to PC

Description A subroutine call is made to a 20-bit address anywhere in the full address space. All seven source addressing modes can be used. The call instruction is an address-word instruction. If the destination address is contained in a memory location X, it is contained in two ascending words, X (LSBs) and (X + 2) (MSBs). Two words on the stack are needed for the return address. The return is made with the instruction RETA.

Status Bits

N: Not affected

Z: Not affected

C: Not affected

V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Examples Examples for all addressing modes are given.

Immediate mode: Call a subroutine at label EXEC or call directly an address.

```
CALLA   #EXEC        ; Start address EXEC
CALLA   #01AA04h     ; Start address 01AA04h
```

Symbolic mode: Call a subroutine at the 20-bit address contained in addresses EXEC (LSBs) and EXEC+2 (MSBs). EXEC is located at the address (PC + X) where X is within +32 K. Indirect addressing.

```
CALLA   EXEC         ; Start address at @EXEC. z16(PC)
```

Absolute mode: Call a subroutine at the 20-bit address contained in absolute addresses EXEC (LSBs) and EXEC+2 (MSBs). Indirect addressing.

```
CALLA   &EXEC        ; Start address at @EXEC
```

Register mode: Call a subroutine at the 20-bit address contained in register R5. Indirect R5.

```
CALLA   R5           ; Start address at @R5
```

Indirect mode: Call a subroutine at the 20-bit address contained in the word pointed to by register R5 (LSBs). The MSBs have the address (R5 + 2). Indirect, indirect R5.

```
CALLA   @R5          ; Start address at @R5
```

Indirect, Auto-Increment mode: Call a subroutine at the 20-bit address contained in the words pointed to by register R5 and increment the 20-bit address in R5 afterwards by 4. The next time the S/W flow uses R5 as a pointer, it can alter the program execution due to access to the next word address in the table pointed to by R5. Indirect, indirect R5.

```
CALLA   @R5+         ; Start address at @R5. R5 + 4
```

Indexed mode: Call a subroutine at the 20-bit address contained in the address pointed to by register (R5 + X); e.g., a table with addresses starting at X. (R5 + X) points to the LSBs, (R5 + X + 2) points to the MSBs of the word address. X is within R5 + 32 K. Indirect, indirect (R5 + X).

```
CALLA   X(R5)          ; Start address at @(R5+X). z16(R5)
```

*** CLRA** Clear 20-bit destination register

Syntax `CLRA Rdst`

Operation 0 → Rdst

Emulation `MOVA #0,Rdst`

Description The destination register is cleared.

Status Bits Status bits are not affected.

Example The 20-bit value in R10 is cleared.

```
CLRA    R10       ; 0 -> R10
```

CMPA Compare the 20-bit source with a 20-bit destination register

Syntax `CMPA Rsrc,Rdst`

`CMPA #imm20,Rdst`

Operation (.not. src) + 1 + Rdst or Rdst – src

Description The 20-bit source operand is subtracted from the 20-bit destination CPU register. This is made by adding the 1s complement of the source + 1 to the destination register. The result affects only the status bits.

Status Bits

N: Set if result is negative (src > dst), reset if positive (src ≤ dst)

Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)

C: Set if there is a carry from the MSB, reset otherwise

V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example A 20-bit immediate operand and R6 are compared. If they are equal, the program continues at label EQUAL.

```
CMPA   #12345h,R6       ; Compare R6 with 12345h
JEQ    EQUAL            ; R5 = 12345h
...                     ; Not equal
```

Example The 20-bit values in R5 and R6 are compared. If R5 is greater than (signed) or equal to R6, the program continues at label GRE.

```
CMPA   R6,R5            ; Compare R6 with R5 (R5 - R6)
JGE    GRE              ; R5 >= R6
...                     ; R5 < R6
```

*** DECDA** Double-decrement 20-bit destination register

Syntax `DECDA Rdst`

Operation Rdst – 2 → Rdst

Emulation `SUBA #2,Rdst`

Description The destination register is decremented by two. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if Rdst contained 2, reset otherwise
C: Reset if Rdst contained 0 or 1, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit value in R5 is decremented by 2.

```
DECDA   R5      ; Decrement R5 by two
```

*** INCDA** Double-increment 20-bit destination register

Syntax `INCDA Rdst`

Operation Rdst + 2 → Rdst

Emulation `ADDA #2,Rdst`

Description The destination register is incremented by two. The original contents are lost.

Status Bits
- N: Set if result is negative, reset if positive
- Z: Set if Rdst contained 0FFFFEh, reset otherwise
  Set if Rdst contained 0FFFEh, reset otherwise
  Set if Rdst contained 0FEh, reset otherwise
- C: Set if Rdst contained 0FFFFEh or 0FFFFFh, reset otherwise
  Set if Rdst contained 0FFFEh or 0FFFFh, reset otherwise
  Set if Rdst contained 0FEh or 0FFh, reset otherwise
- V: Set if Rdst contained 07FFFEh or 07FFFFh, reset otherwise
  Set if Rdst contained 07FFEh or 07FFFh, reset otherwise
  Set if Rdst contained 07Eh or 07Fh, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit value in R5 is incremented by two.

```
INCDA   R5      ; Increment R5 by two
```

MOVA Move the 20-bit source to the 20-bit destination

Syntax

```
MOVA Rsrc,Rdst
MOVA #imm20,Rdst
MOVA z16(Rsrc),Rdst
MOVA EDE,Rdst
MOVA &abs20,Rdst
MOVA @Rsrc,Rdst
MOVA @Rsrc+,Rdst
MOVA Rsrc,z16(Rdst)
MOVA Rsrc,&abs20
```

Operation

src → Rdst

Rsrc → dst

Description The 20-bit source operand is moved to the 20-bit destination. The source operand is not affected. The previous content of the destination is lost.

Status Bits

N: Not affected

Z: Not affected

C: Not affected

V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Examples Copy 20-bit value in R9 to R8

```
MOVA    R9,R8          ; R9 -> R8
```

Write 20-bit immediate value 12345h to R12

```
MOVA    #12345h,R12    ; 12345h -> R12
```

Copy 20-bit value addressed by (R9 + 100h) to R8. Source operand in addresses (R9 + 100h) LSBs and (R9 + 102h) MSBs.

```
MOVA    100h(R9),R8      ; Index: + 32 K. 2 words transferred
```

Move 20-bit value in 20-bit absolute addresses EDE (LSBs) and EDE+2 (MSBs) to R12

```
MOVA    &EDE,R12         ; &EDE -> R12. 2 words transferred
```

Move 20-bit value in 20-bit addresses EDE (LSBs) and EDE+2 (MSBs) to R12. PC index ± 32 K.

```
MOVA    EDE,R12          ; EDE -> R12. 2 words transferred
```

Copy 20-bit value R9 points to (20 bit address) to R8. Source operand in addresses @R9 LSBs and @(R9 + 2) MSBs.

```
MOVA    @R9,R8           ; @R9 -> R8. 2 words transferred
```

Copy 20-bit value R9 points to (20 bit address) to R8. R9 is incremented by four afterwards. Source operand in addresses @R9 LSBs and @(R9 + 2) MSBs.

```
MOVA    @R9+,R8          ; @R9 -> R8. R9 + 4. 2 words transferred.
```

Copy 20-bit value in R8 to destination addressed by (R9 + 100h). Destination operand in addresses @(R9 + 100h) LSBs and @(R9 + 102h) MSBs.

```
MOVA   R8,100h(R9)      ; Index: +- 32 K. 2 words transferred
```

Move 20-bit value in R13 to 20-bit absolute addresses EDE (LSBs) and EDE+2 (MSBs)

```
MOVA   R13,&EDE         ; R13 -> EDE. 2 words transferred
```

Move 20-bit value in R13 to 20-bit addresses EDE (LSBs) and EDE+2 (MSBs). PC index ± 32 K.

```
MOVA   R13,EDE          ; R13 -> EDE. 2 words transferred
```

*** RETA** Return from subroutine

Syntax `RETA`

Operation
@SP → PC.15:0 LSBs (15:0) of saved PC to PC.15:0
SP + 2 → SP
@SP → PC.19:16 MSBs (19:16) of saved PC to PC.19:16
SP + 2 → SP

Emulation `MOVA @SP+,PC`

Description The 20-bit return address information, pushed onto the stack by a CALLA instruction, is restored to the PC. The program continues at the address following the subroutine call. The SR bits SR.11:0 are not affected. This allows the transfer of information with these bits.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Call a subroutine SUBR from anywhere in the 20-bit address space and return to the address after the CALLA

```
       CALLA    #SUBR     ; Call subroutine starting at SUBR
       ...                ; Return by RETA to here
SUBR   PUSHM.A  #2,R14    ; Save R14 and R13 (20 bit data)
       ...                ; Subroutine code
       POPM.A   #2,R14    ; Restore R13 and R14 (20 bit data)
       RETA               ; Return (to full address space)
```

*** TSTA** Test 20-bit destination register

Syntax `TSTA Rdst`

Operation dst + 0FFFFFh + 1

dst + 0FFFFh + 1

dst + 0FFh + 1

Emulation `CMPA #0,Rdst`

Description The destination register is compared with zero. The status bits are set according to the result. The destination register is not affected.

Status Bits N: Set if destination register is negative, reset if positive

Z: Set if destination register contains zero, reset otherwise

C: Set

V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit value in R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
         TSTA   R7            ; Test R7
         JN     R7NEG         ; R7 is negative
         JZ     R7ZERO        ; R7 is zero
R7POS    ......               ; R7 is positive but not zero
R7NEG    ......               ; R7 is negative
R7ZERO   ......               ; R7 is zero
```

SUBA Subtract 20-bit source from 20-bit destination register

Syntax `SUBA Rsrc,Rdst`

`SUBA #imm20,Rdst`

Operation (.not.src) + 1 + Rdst → Rdst or Rdst – src → Rdst

Description The 20-bit source operand is subtracted from the 20-bit destination register. This is made by adding the 1s complement of the source + 1 to the destination. The result is written to the destination register, the source is not affected.

Status Bits
- N: Set if result is negative (src > dst), reset if positive (src ≤ dst)
- Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)
- C: Set if there is a carry from the MSB (Rdst.19), reset otherwise
- V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit value in R5 is subtracted from R6. If a carry occurs, the program continues at label TONI.

```
SUBA   R5,R6      ; R6 - R5 -> R6
JC     TONI       ; Carry occurred
...               ; No carry
```

Chapter 5

SLAU208F–June 2008–Revised March 2010

Flash Memory Controller

This chapter describes the operation of the flash memory controller.

Topic **Page**

5.1 Flash Memory Introduction 256
5.2 Flash Memory Segmentation 257
5.3 Flash Memory Operation 259
5.4 Flash Memory Registers 273

5.1 Flash Memory Introduction

The flash memory is byte, word, and long-word addressable and programmable. The flash memory module has an integrated controller that controls programming and erase operations. The module contains three registers, a timing generator, and a voltage generator to supply program and erase voltages. The cumulative high-voltage time must not be exceeded, and each 32-bit word can be written not more than four times (in byte, word, or long word write modes) before another erase cycle (see device-specific data sheet for details).

The flash memory features include:

- Internal programming voltage generation
- Byte, word (2 bytes), and long (4 bytes) programmable
- Ultralow-power operation
- Segment erase, bank erase (device specific), and mass erase
- Marginal 0 and marginal 1 read modes
- Each bank (device specific) can be erased individually while program execution can proceed in a different flash bank.

NOTE: Bank operations are not supported on all devices. See the device-specific data sheet for banks supported and their respective sizes.

The block diagram of the flash memory and controller is shown in Figure 5-1.

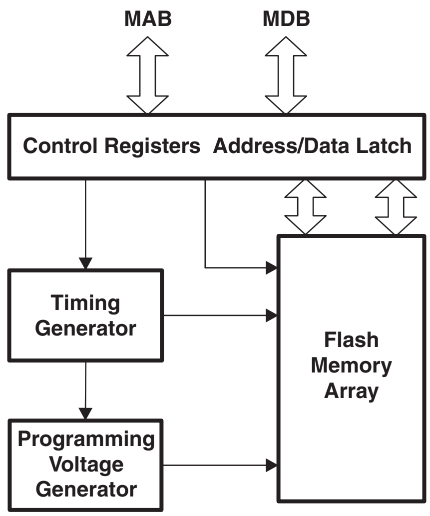


Figure 5-1. Flash Memory Module Block Diagram

5.2 Flash Memory Segmentation

The flash main memory is partitioned into 512-byte segments. Single bits, bytes, or words can be written to flash memory, but a segment is the smallest size of the flash memory that can be erased.

The flash memory is partitioned into main and information memory sections. There is no difference in the operation of the main and information memory sections. Code and data can be located in either section. The difference between the sections is the segment size.

There are four information memory segments, A through D. Each information memory segment contains 128 bytes and can be erased individually.

The bootstrap loader (BSL) memory consists of four segments, A through D. Each BSL memory segment contains 512 bytes and can be erased individually.

The main memory segment size is 512 byte. See the device-specific data sheet for the start and end addresses of each bank, when available, and for the complete memory map of a device.

Figure 5-2 shows the flash segmentation using an example of 256-KB flash that has four banks of 64 KB (segments A through D) and information memory.

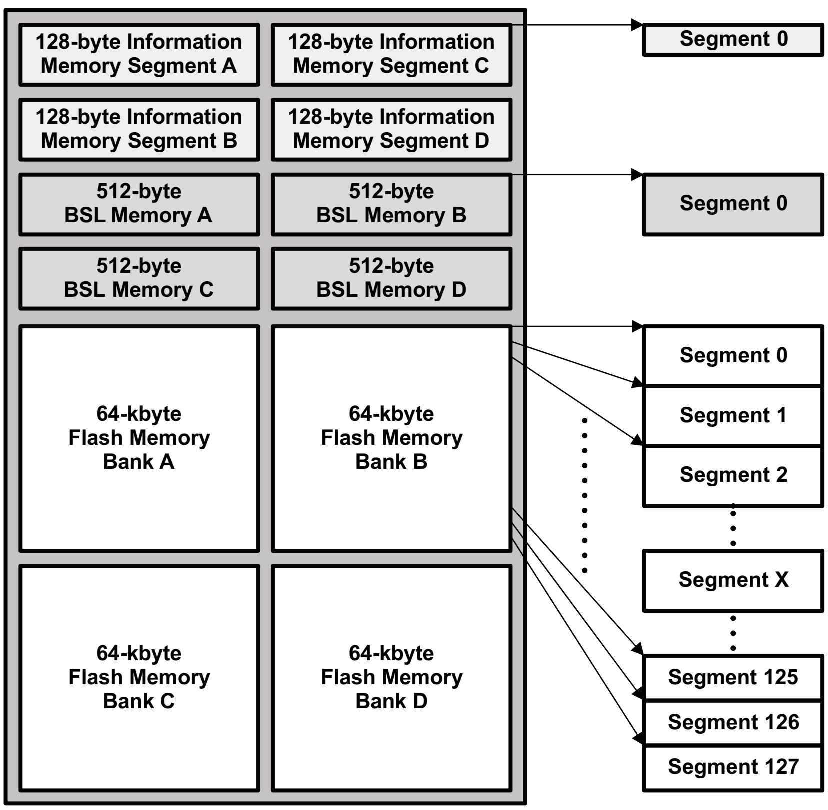


Figure 5-2. 256-KB Flash Memory Segments Example

5.2.1 Segment A

Segment A of the information memory is locked separately from all other segments with the LOCKA bit. If LOCKA = 1, segment A cannot be written or erased, and all information memory is protected from being segment erased. If LOCKA = 0, segment A can be erased and written like any other flash memory segment.

The state of the LOCKA bit is toggled when a 1 is written to it. Writing a 0 to LOCKA has no effect. This allows existing flash programming routines to be used unchanged.

```
; Unlock Info Memory
   BIC      #FWPW+LOCKINFO, &FCTL4   ; Clear LOCKINFO
; Unlock SegmentA
   BIT      #LOCKA,&FCTL3            ; Test LOCKA
   JZ       SEGA_UNLOCKED            ; Already unlocked?
   MOV      #FWPW+LOCKA,&FCTL3       ; No, unlock SegmentA
SEGA_UNLOCKED                        ; Yes, continue
; SegmentA is unlocked


; Lock SegmentA
   BIT      #LOCKA,&FCTL3            ; Test LOCKA
   JNZ      SEGA_LOCKED              ; Already locked?
   MOV      #FWPW+LOCKA,&FCTL3       ; No, lock SegmentA
SEGA_LOCKED                          ; Yes, continue
; SegmentA is locked
; Lock Info Memory
   BIS      #FWPW+LOCKINFO,&FCTL4    ; Set LOCKINFO
```

5.3 Flash Memory Operation

The default mode of the flash memory is read mode. In read mode, the flash memory is not being erased or written, the flash timing generator and voltage generator are off, and the memory operates identically to ROM.

Read and fetch while erase – The flash memory allows execution of a program from flash while a different flash bank is erased. Data reads are also possible from any flash bank not being erased.

NOTE: Read and fetch while erase

The read and fetch while erase feature is available in flash memory configurations where more than one flash bank is available. If there is one flash bank available, holding the complete flash program memory, the read from the program memory and information memory and BSL memory during the erase is not provided.

Flash memory is in-system programmable (ISP) without the need for additional external voltage. The CPU can program the flash memory. The flash memory write/erase modes are selected by the BLKWRT, WRT, MERAS, and ERASE bits and are:

- Byte/word/long-word (32-bit) write
- Block write
- Segment erase
- Bank erase (only main memory)
- Mass erase (all main memory banks)
- Read during bank erase (except for the one currently read from)

Reading or writing to flash memory while it is busy programming or erasing (page, mass, or bank) from the same bank is prohibited. Any flash erase or programming can be initiated from within flash memory or RAM.

5.3.1 Erasing Flash Memory

The logical value of an erased flash memory bit is 1. Each bit can be programmed from 1 to 0 individually, but to reprogram from 0 to 1 requires an erase cycle. The smallest amount of flash that can be erased is one segment. There are three erase modes selected by the ERASE and MERAS bits listed in Table 5-1.

Table 5-1. Erase Modes

MERAS	ERASE	Erase Mode
0	1	Segment erase
1	0	Bank erase (of one bank) selected by the dummy write address[1]
1	1	Mass erase (all memory banks, information memory A to D and BSL segments A to D are not erased)

(1) Bank operations are not supported on all devices. See the device-specific data sheet for support of bank operations.

5.3.1.1 Erase Cycle

An erase cycle is initiated by a dummy write to the address range of the segment to be erased. The dummy write starts the erase operation. Figure 5-3 shows the erase cycle timing. The BUSY bit is set immediately after the dummy write and remains set throughout the erase cycle. BUSY, MERAS, and ERASE are automatically cleared when the cycle completes. The mass erase cycle timing is not dependent on the amount of flash memory present on a device. Erase cycle times are equivalent for all devices.

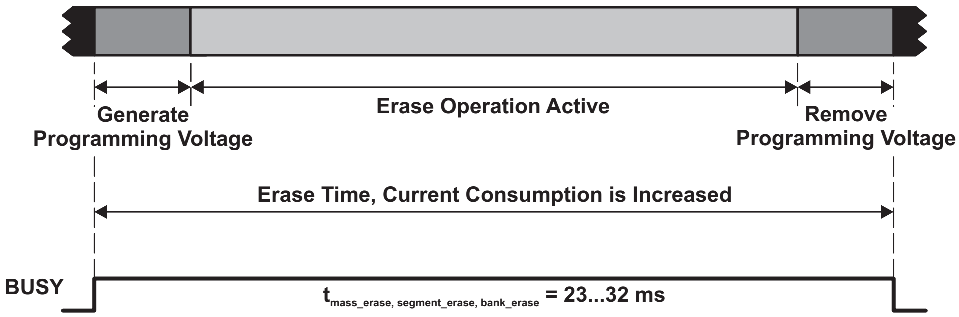


Figure 5-3. Erase Cycle Timing

5.3.1.2 Erasing Main Memory

The main memory consists of one or more banks. Each bank can be erased individually (bank erase). All main memory banks can be erased in the mass erase mode.

5.3.1.3 Erasing Information Memory or Flash Segments

The information memory A to D and the BSL segments A to D can be erased in segment erase mode. They are not erased during a bank erase or a mass erase.

5.3.1.4 Initiating Erase From Flash

An erase cycle can be initiated from within flash memory. Code can be executed from flash or RAM during a bank erase. The executed code cannot be located in a bank to be erased.

During a segment erase, the CPU is held until the erase cycle completes. After the erase cycle ends, the CPU resumes code execution with the instruction following the dummy write.

When initiating an erase cycle from within flash memory, it is possible to erase the code needed for execution after the erase operation. If this occurs, CPU execution is unpredictable after the erase cycle.

The flow to initiate an erase from flash is shown in Figure 5-4.

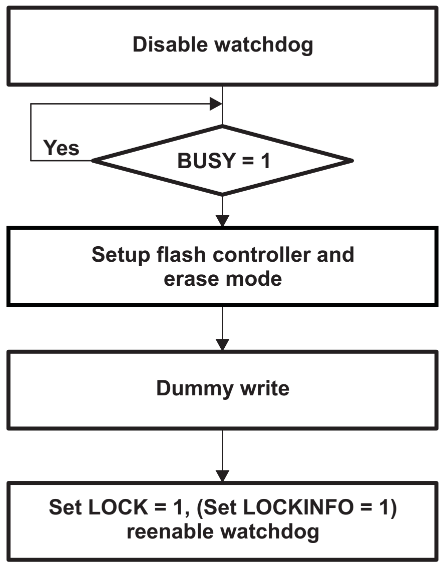


Figure 5-4. Erase Cycle From Flash

```
; Segment Erase from flash.
; Assumes Program Memory. Information memory or BSL
; requires LOCKINFO to be cleared as well.
; Assumes ACCVIE = NMIIE = OFIE = 0.
    MOV   #WDTPW+WDTHOLD,&WDTCTL     ; Disable WDT
L1  BIT   #BUSY,&FCTL3               ; Test BUSY
    JNZ   L1                         ; Loop while busy
    MOV   #FWPW,&FCTL3               ; Clear LOCK
    MOV   #FWPW+ERASE,&FCTL1         ; Enable segment erase
    CLR   &0FC10h                    ; Dummy write
L2  BIT   #BUSY,&FCTL3               ; Test BUSY
    JNZ   L2                         ; Loop while busy
    MOV   #FWPW+LOCK,&FCTL3          ; Done, set LOCK
    ...                              ; Re-enable WDT?
```

5.3.1.5 Initiating Erase From RAM

An erase cycle can be initiated from RAM. In this case, the CPU is not held and continues to execute code from RAM. The mass erase (all main memory banks) operation is initiated while executing from RAM. The BUSY bit is used to determine the end of the erase cycle. If the flash is busy completing a bank erase, flash addresses of a different bank can be used to read data or to fetch instructions. While the flash is BUSY, starting an erase cycle or a programming cycle causes an access violation, ACCIFG is set to 1, and the result of the erase operation is unpredictable.

The flow to initiate an erase from flash from RAM is shown in Figure 5-5.

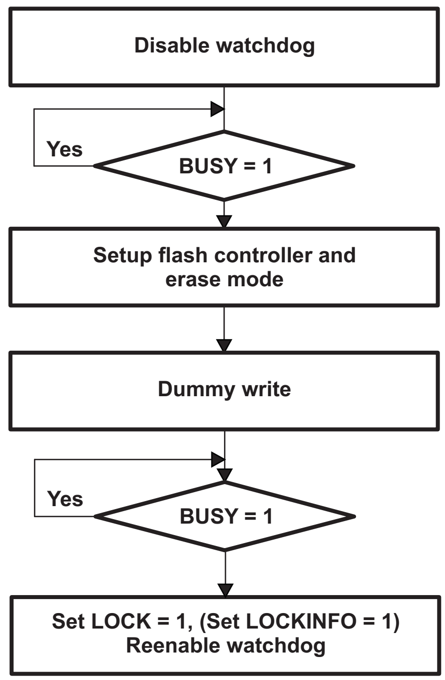


Figure 5-5. Erase Cycle From RAM

```
; segment Erase from RAM.
; Assumes Program Memory. Information memory or BSL
; requires LOCKINFO to be cleared as well.
; Assumes ACCVIE = NMIIE = OFIE = 0.
    MOV    #WDTPW+WDTHOLD,&WDTCTL      ; Disable WDT
L1  BIT    #BUSY,&FCTL3                ; Test BUSY
    JNZ    L1                          ; Loop while busy
    MOV    #FWPW,&FCTL3                ; Clear LOCK
    MOV    #FWPW+ERASE,&FCTL1          ; Enable page erase
    CLR    &0FC10h                     ; Dummy write
L2  BIT    #BUSY,&FCTL3                ; Test BUSY
    JNZ    L2                          ; Loop while busy
    MOV    #FWPW+LOCK,&FCTL3           ; Done, set LOCK
    ...                                ; Re-enable WDT?
```

5.3.2 Writing Flash Memory

The write modes, selected by the WRT and BLKWRT bits, are listed in Table 5-2.

Table 5-2. Write Modes

BLKWRT	WRT	Write Mode
0	1	Byte/word write
1	0	Long-word write
1	1	Long-word block write

The write modes use a sequence of individual write instructions. Using the long-word write mode is approximately twice as fast as the byte/word mode. Using the long-word block write mode is approximately four times faster than byte/word mode, because the voltage generator remains on for the complete block write, and long-words are written in parallel. Any instruction that modifies a destination can be used to modify a flash location in either byte/word write mode, long-word write mode, or block long-word write mode.

The BUSY bit is set while the write operation is active and cleared when the operation completes. If the write operation is initiated from RAM, the CPU must not access flash while BUSY is set to 1. Otherwise, an access violation occurs, ACCVIFG is set, and the flash write is unpredictable.

5.3.2.1 Byte/Word Write

A byte/word write operation can be initiated from within flash memory or from RAM. When initiating from within flash memory, the CPU is held while the write completes. After the write completes, the CPU resumes code execution with the instruction following the write access. The byte/word write timing is shown in Figure 5-6.

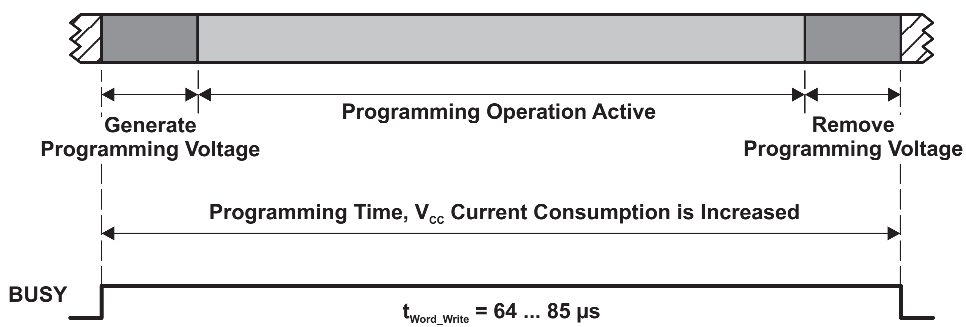


Figure 5-6. Byte/Word/Long-Word Write Timing

When a byte/word write is executed from RAM, the CPU continues to execute code from RAM. The BUSY bit must be zero before the CPU accesses flash again, otherwise an access violation occurs, ACCVIFG is set, and the write result is unpredictable.

In byte/word write mode, the internally-generated programming voltage is applied to the complete 128-byte block. The cumulative programming time, t_{CPT}, must not be exceeded for any block. Each byte or word write adds to the cumulative program time of a segment. If the maximum cumulative program time is reached or exceeded, the segment must be erased. Further programming or using the data returns unpredictable results (see the device-specific data sheet for specifications).

5.3.2.2 Initiating Byte/Word Write From Flash

The flow to initiate a byte/word write from flash is shown in Figure 5-7.

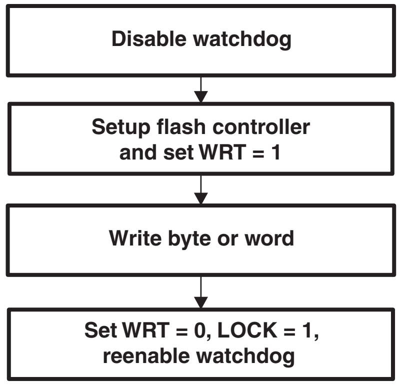


Figure 5-7. Initiating a Byte/Word Write From Flash

```
; Byte/word write from flash.
; Assumes 0x0FF1E is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL     ; Disable WDT
   MOV   #FWPW,&FCTL3               ; Clear LOCK
   MOV   #FWPW+WRT,&FCTL1           ; Enable write
   MOV   #0123h,&0FF1Eh             ; 0123h -> 0x0FF1E
   MOV   #FWPW,&FCTL1               ; Done. Clear WRT
   MOV   #FWPW+LOCK,&FCTL3          ; Set LOCK
   ...                              ; Re-enable WDT?
```

5.3.2.3 Initiating Byte/Word Write From RAM

The flow to initiate a byte/word write from RAM is shown in Figure 5-8.

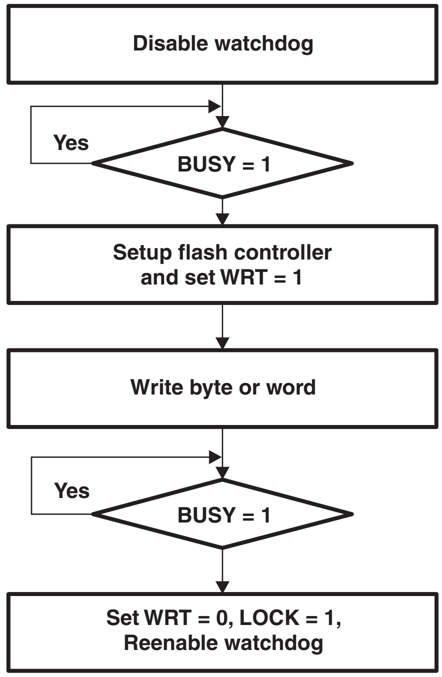


Figure 5-8. Initiating a Byte/Word Write From RAM

```
; Byte/word write from RAM.
; Assumes 0x0FF1E is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
    MOV   #WDTPW+WDTHOLD,&WDTCTL     ; Disable WDT
L1  BIT   #BUSY,&FCTL3               ; Test BUSY
    JNZ   L1                         ; Loop while busy
    MOV   #FWPW,&FCTL3               ; Clear LOCK
    MOV   #FWPW+WRT,&FCTL1           ; Enable write
    MOV   #0123h,&0FF1Eh             ; 0123h -> 0x0FF1E
L2  BIT   #BUSY,&FCTL3               ; Test BUSY
    JNZ   L2                         ; Loop while busy
    MOV   #FWPW,&FCTL1               ; Clear WRT
    MOV   #FWPW+LOCK,&FCTL3          ; Set LOCK
    ...                              ; Re-enable WDT?
```

5.3.2.4 Long-Word Write

A long-word write operation can be initiated from within flash memory or from RAM. The BUSY bit is set to 1 after 32 bits are written to the flash controller and the programming cycle starts. When initiating from within flash memory, the CPU is held while the write completes. After the write completes, the CPU resumes code execution with the instruction following the write access. The long-word write timing is shown in Figure 5-6.

A long-word consists of four consecutive bytes aligned to at 32-bit address (only the lower two address bits are different). The bytes can be written in any order or any combination of bytes and words. If a byte or word is written more than once, the last data written to the four bytes are stored into the flash memory.

If a write to a flash address outside of the 32-bit address happens before all four bytes are available, the data written so far is discarded, and the latest byte/word written defines the new 32-bit aligned address.

When 32 bits are available, the write cycle is executed. When executing from RAM, the CPU continues to execute code. The BUSY bit must be zero before the CPU accesses flash again, otherwise an access violation occurs, ACCVIFG is set, and the write result is unpredictable.

In long-word write mode, the internally-generated programming voltage is applied to a complete 128-byte block. The cumulative programming time, t_{CPT}, must not be exceeded for any block. Each byte or word write adds to the cumulative program time of a segment. If the maximum cumulative program time is reached or exceeded, the segment must be erased. Further programming or using the data returns unpredictable results.

With each byte or word write, the amount of time the block is subjected to the programming voltage accumulates. If the cumulative programming time is reached or exceeded, the block must be erased before further programming or use (see the device-specific data sheet for specifications).

5.3.2.5 Initiating Long-Word Write From Flash

The flow to initiate a long-word write from flash is shown in Figure 5-9.

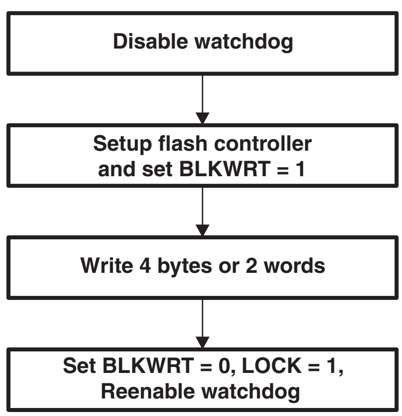


Figure 5-9. Initiating Long-Word Write From Flash

```
; Long-word write from flash.
; Assumes 0x0FF1C and 0x0FF1E is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL     ; Disable WDT
   MOV   #FWPW,&FCTL3               ; Clear LOCK
   MOV   #FWPW+BLKWRT,&FCTL1        ; Enable 2-word write
   MOV   #0123h,&0FF1Ch             ; 0123h -> 0x0FF1C
   MOV   #45676h,&0FF1Eh            ; 04567h -> 0x0FF1E
   MOV   #FWPW,&FCTL1               ; Done. Clear BLKWRT
   MOV   #FWPW+LOCK,&FCTL3          ; Set LOCK
   ...                              ; Re-enable WDT?
```

5.3.2.6 Initiating Long-Word Write From RAM

The flow to initiate a long-word write from RAM is shown in Figure 5-10.

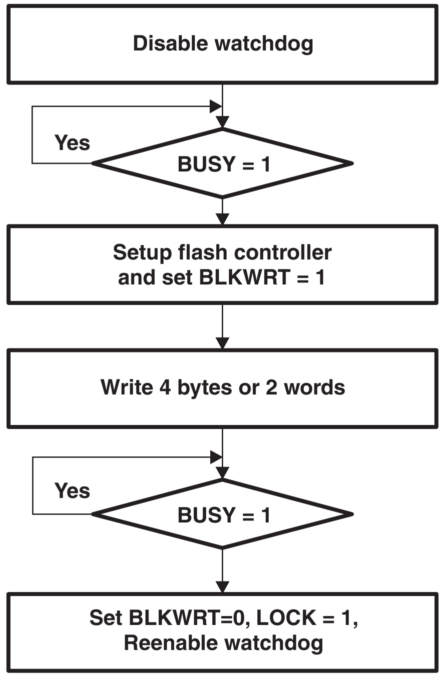


Figure 5-10. Initiating Long-Word Write from RAM

```
; Two 16-bit word writes from RAM.
; Assumes 0x0FF1C and 0x0FF1E is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
    MOV   #WDTPW+WDTHOLD,&WDTCTL     ; Disable WDT
L1  BIT   #BUSY,&FCTL3               ; Test BUSY
    JNZ   L1                         ; Loop while busy
    MOV   #FWPW,&FCTL3               ; Clear LOCK
    MOV   #FWPW+BLKWRT,&FCTL1        ; Enable write
    MOV   #0123h,&0FF1Ch             ; 0123h -> 0x0FF1C
    MOV   #4567h,&0FF1Eh             ; 4567h -> 0x0FF1E
L2  BIT   #BUSY,&FCTL3               ; Test BUSY
    JNZ   L2                         ; Loop while busy
    MOV   #FWPW,&FCTL1               ; Clear WRT
    MOV   #FWPW+LOCK,&FCTL3          ; Set LOCK
    ...                              ; Re-enable WDT?
```

5.3.2.7 Block Write

The block write can be used to accelerate the flash write process when many sequential bytes or words need to be programmed. The flash programming voltage remains on for the duration of writing the 128-byte row. The cumulative programming time, t_{CPT}, must not be exceeded for any row during a block write.

A block write cannot be initiated from within flash memory. The block write must be initiated from RAM. The BUSY bit remains set throughout the duration of the block write. The WAIT bit must be checked between writing four bytes, or two words, to the block. When WAIT is set, then four bytes, or two 16-bit words, of the block can be written. When writing successive blocks, the BLKWRT bit must be cleared after the current block is completed. BLKWRT can be set initiating the next block write after the required flash recovery time given by t_{END}. BUSY is cleared following each block write completion, indicating the next block can be written. Figure 5-11 shows the block write timing.

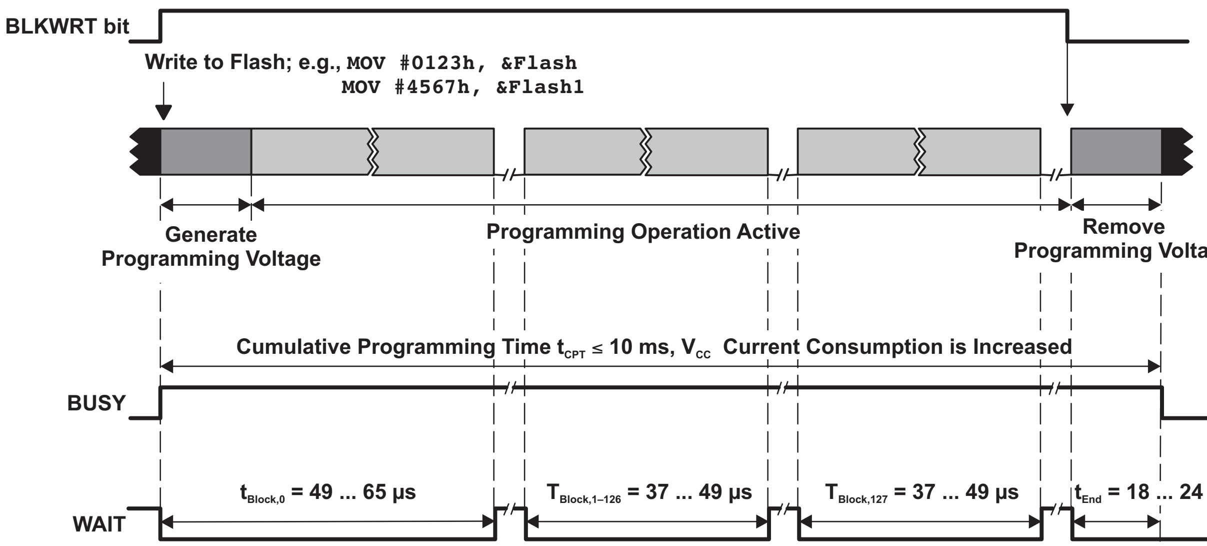


Figure 5-11. Block-Write Cycle Timing

5.3.2.8 Block Write Flow and Example

A block write flow is shown in Figure 5-12 and the following code example.

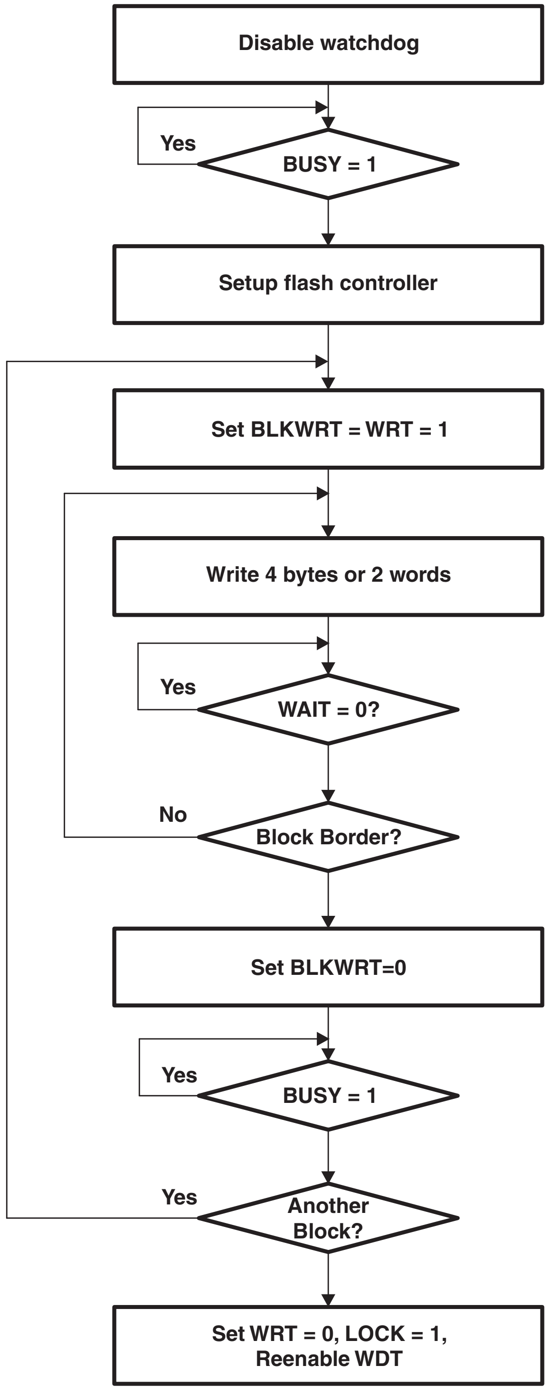


Figure 5-12. Block Write Flow

```
; Write one block starting at 0F000h.
; Must be executed from RAM, Assumes Flash is already erased.
; Assumes ACCVIE = NMIIE = OFIE = 0.
    MOV    #32,R5                    ; Use as write counter
    MOV    #0F000h,R6                ; Write pointer
    MOV    #WDTPW+WDTHOLD,&WDTCTL    ; Disable WDT
L1  BIT    #BUSY,&FCTL3              ; Test BUSY
    JNZ    L1                        ; Loop while busy
    MOV    #FWPW,&FCTL3              ; Clear LOCK
    MOV    #FWPW+BLKWRT+WRT,&FCTL1   ; Enable block write
L2  MOV    Write_Value1,0(R6)        ; Write 1st location
    MOV    Write_Value2,2(R6)        ; Write 2nd word
L3  BIT    #WAIT,&FCTL3              ; Test WAIT
    JZ     L3                        ; Loop while WAIT=0
    INCD   R6                        ; Point to next words
    INCD   R6                        ; Point to next words
    DEC    R5                        ; Decrement write counter
    JNZ    L2                        ; End of block?
    MOV    #FWPW,&FCTL1              ; Clear WRT, BLKWRT
L4  BIT    #BUSY,&FCTL3              ; Test BUSY
    JNZ    L4                        ; Loop while busy
    MOV    #FWPW+LOCK,&FCTL3         ; Set LOCK
    ...                              ; Re-enable WDT if needed
```

5.3.3 Flash Memory Access During Write or Erase

When a write or an erase operation is initiated from RAM while BUSY = 1, the CPU may not write to any flash location. Otherwise, an access violation occurs, ACCVIFG is set, and the result is unpredictable.

When a write operation is initiated from within flash memory, the CPU continues code execution with the next instruction fetch after the write cycle completed (BUSY = 0).

The op-code 3FFFh is the JMP PC instruction. This causes the CPU to loop until the flash operation is finished. When the operation is finished and BUSY = 0, the flash controller allows the CPU to fetch the op-code and program execution resumes.

The flash access conditions while BUSY = 1 are listed in Table 5-3.

Table 5-3. Flash Access While Flash is Busy (BUSY = 1)

Flash Operation	Flash Access	WAIT	Result
Bank erase	Read	0	From the erased bank: ACCVIFG = 0. 03FFFh is the value read. From any other flash location: ACCVIFG = 0. Valid read.
	Write	0	ACCVIFG = 1. Write is ignored.
	Instruction fetch	0	From the erased bank: ACCVIFG = 0. CPU fetches 03FFFh. This is the JMP PC instruction. From any other flash location: ACCVIFG = 0. Valid instruction fetch.
Segment erase	Read	0	ACCVIFG = 0: 03FFFh is the value read.
	Write	0	ACCVIFG = 1: Write is ignored.
	Instruction fetch	0	ACCVIFG = 0: CPU fetches 03FFFh. This is the JMP PC instruction.
Word/byte write or long-word write	Read	0	ACCVIFG = 0: 03FFFh is the value read.
	Write	0	ACCVIFG = 1: Write is ignored.
	Instruction fetch	0	ACCVIFG = 0: CPU fetches 03FFFh. This is the JMP PC instruction.
Block write	Any	0	ACCVIFG = 1: LOCK = 1, block write is exited.
	Read	1	ACCVIFG = 0: 03FFFh is the value read.
	Write	1	ACCVIFG = 0: Valid write
	Instruction fetch	1	ACCVIFG = 1: LOCK = 1, block write is exited

Interrupts are automatically disabled during any flash operation.

The watchdog timer (in watchdog mode) should be disabled before a flash erase cycle. A reset aborts the erase and the result is unpredictable. After the erase cycle has completed, the watchdog may be reenabled.

5.3.4 Checking Flash memory

The result of a programming cycle of the flash memory can be checked by calculating and storing a checksum (CRC) of parts and/or the complete flash memory content. The CRC module can be used for this purpose (see the device-specific data sheet). During the runtime of the system, the known checksums can be recalculated and compared with the expected values stored in the flash memory. The program checking the flash memory content is executed in RAM.

To get an early indication of weak memory cells, reading the flash can be done in combination with the device-specific marginal read modes. The marginal read modes are controlled by the FCTL4.MRG0 and FCTL4.MRG1 register bits if available (device specific). During marginal read mode, marginally programmed flash memory bit locations can be detected. One method for identifying such memory locations would be to periodically perform a checksum calculation over a section of flash memory (for example, a flash segment) and repeating this procedure with the marginal read mode enabled. If they do not match, it could indicate an insufficiently programmed flash memory location. It is possible to refresh the affected Flash memory segment by disabling marginal read mode, copying to RAM, erasing the flash segment, and writing back to it from RAM.

The program checking the flash memory contents must be executed from RAM. Executing code from flash will automatically disable the marginal read mode. The marginal read modes are controlled by the MRG0 and MRG1 register bits. Setting MRG1 is used to detect insufficiently programmed flash cells containing a “1” (erased bits). Setting MRG0 is used to detect insufficiently programmed flash cells containing a “0” (programmed bits). Only one of these bits should be set at a time. Therefore, a full marginal read check will require two passes of checking the flash memory content’s integrity. During marginal read mode, the flash access speed (MCLK) must be limited to 1 MHz (see the device-specific data sheet).

5.3.5 Configuring and Accessing the Flash Memory Controller

The FCTLx registers are 16-bit password-protected read/write registers. Any read or write access must use word instructions, and write accesses must include the write password 0A5h in the upper byte. Any write to any FCTLx register with a value other than 0A5h in the upper byte is a password violation, sets the KEYV flag, and triggers a PUC system reset. Any read of any FCTLx registers reads 096h in the upper byte.

Any write to FCTL1 during an erase or byte/word/double-word write operation is an access violation and sets ACCVIFG. Writing to FCTL1 is allowed in block write mode when WAIT = 1, but writing to FCTL1 in block write mode when WAIT = 0 is an access violation and sets ACCVIFG.

Any write to FCTL2 (this register is currently not implemented) when BUSY = 1 is an access violation.

Any FCTLx register may be read when BUSY = 1. A read does not cause an access violation.

5.3.6 Flash Memory Controller Interrupts

The flash controller has two interrupt sources, KEYV and ACCVIFG. ACCVIFG is set when an access violation occurs. When the ACCVIE bit is reenabled after a flash write or erase, a set ACCVIFG flag generates an interrupt request. The ACCVIE bit resides in the the Special Function Register, SFRIE1 (see the *System Resets, Interrupts, and Operating Modes, System Control Module (SYS)* chapter for details). ACCVIFG sources the NMI interrupt vector, so it is not necessary for GIE to be set for ACCVIFG to request an interrupt. ACCVIFG may also be checked by software to determine if an access violation occurred. ACCVIFG must be reset by software.

The password violation flag, KEYV, is set when any of the flash control registers are written with an incorrect password. When this occurs, a PUC is generated immediately, resetting the device.

5.3.7 Programming Flash Memory Devices

There are three options for programming a flash device. All options support in-system programming.

- Program via JTAG
- Program via the BSL
- Program via a custom solution

5.3.7.1 Programming Flash Memory Via JTAG

Devices can be programmed via the JTAG port. The JTAG interface requires four signals (five signals on 20- and 28-pin devices), ground, and optionally VCC and $\overline{RST}$/NMI.

The JTAG port is protected with a fuse. Blowing the fuse completely disables the JTAG port and is not reversible. Further access to the device via JTAG is not possible For more details see the application report *Programming a Flash-Based MSP430 Using the JTAG Interface* at www.ti.com/msp430.

5.3.7.2 Programming Flash Memory Via Bootstrap Loader (BSL)

Every flash device contains a BSL. The BSL enables users to read or program the flash memory or RAM using a UART serial interface. Access to the flash memory via the BSL is protected by a 256-bit user-defined password. For more details, see the application report *Features of the MSP430 Bootstrap Loader* at www.ti.com/msp430.

5.3.7.3 Programming Flash Memory Via Custom Solution

The ability of the MSP430 CPU to write to its own flash memory allows for in-system and external custom programming solutions as shown in Figure 5-13. The user can choose to provide data through any means available (UART, SPI, etc.). User-developed software can receive the data and program the flash memory. Since this type of solution is developed by the user, it can be completely customized to fit the application needs for programming, erasing, or updating the flash memory.

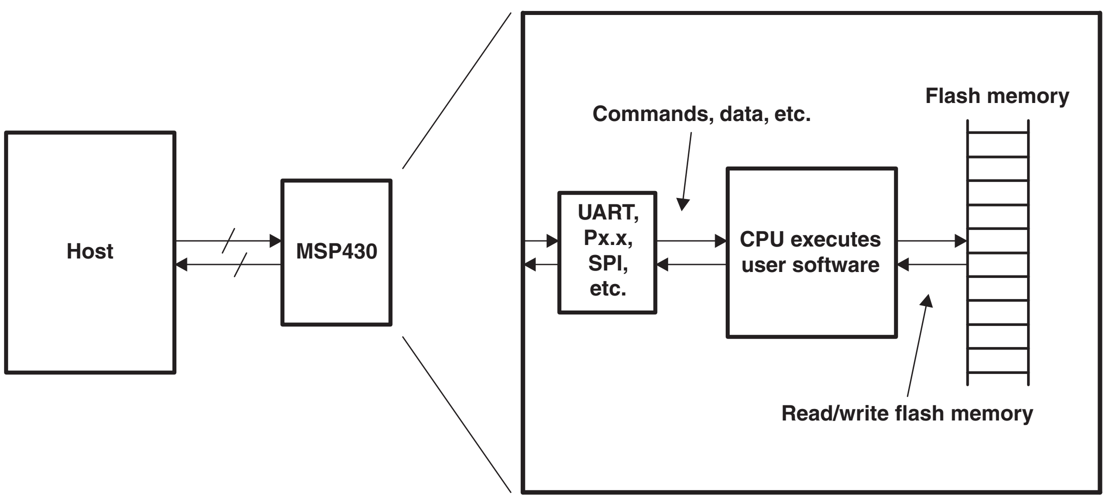


Figure 5-13. User-Developed Programming Solution

5.4 Flash Memory Registers

The flash memory registers are listed in Table 5-4. The base address can be found in the device-specific data sheet. The address offset is given in Table 5-4.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 5-4. Flash Controller Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Flash Memory Control 1	FCTL1	Read/write	Word	00h	9600h
	FCTL1_L	Read/Write	Byte	00h	00h
	FCTL1_H	Read/Write	Byte	01h	96h
Flash Memory Control 3	FCTL3	Read/write	Word	04h	9658h
	FCTL3_L	Read/Write	Byte	04h	58h
	FCTL3_H	Read/Write	Byte	05h	96h
Flash Memory Control 4	FCTL4	Read/write	Word	06h	9600h
	FCTL4_L	Read/Write	Byte	06h	00h
	FCTL4_H	Read/Write	Byte	07h	96h

Flash Memory Control 1 Register (FCTL1)

15	14	13	12	11	10	9	8
FRPW, Read as 096h FWPW, Must be written as 0A5h							

7	6	5	4	3	2	1	0
BLKWRT	**WRT**	**SWRT**	**Reserved**	**Reserved**	**MERAS**	**ERASE**	**Reserved**
rw-0	rw-0	rw-0	r-0	r-0	rw-0	rw-0	r-0

FRPW/FWPW Bits 15–8 FCTL password. Always read as 096h. Must be written as 0A5h or a PUC is generated.

BLKWRT Bit 7 See following table

WRT Bit 6 See following table

BLKWRT	WRT	Write Mode
0	1	Byte/word write
1	0	Long-word write
1	1	Long-word block write

SWRT Bit 5 Smart write. If this bit is set, the program time is shortened. The programming quality has to be checked by marginal read modes.

Reserved Bits 4-3 Reserved. Must be written to 0. Always read 0.

MERAS Bit 2
ERASE Bit 1
Mass erase and erase. These bits are used together to select the erase mode. MERAS and ERASE are automatically reset when a flash erase operation has completed.

MERAS	ERASE	Erase Cycle
0	0	No erase
0	1	Segment erase
1	0	Bank erase (of one bank)
1	1	Mass erase (Erase all flash memory banks)

Reserved Bit 0 Reserved. Always read 0.

Flash Memory Control 3 Register (FCTL3)

15	14	13	12	11	10	9	8
FRPW, Read as 096h FWPW, Must be written as 0A5h							

7	6	5	4	3	2	1	0
Reserved	**LOCKA**	**Reserved**	**LOCK**	**WAIT**	**ACCVIFG**	**KEYV**	**BUSY**
r-0	rw-1	rw-0	rw-1	r-1	rw-0	rw-(0)	rw-0

FRPW/FWPW	Bits 15–8	FCTLx password. Always read as 096h. Must be written as 0A5h or a PUC is generated.
Reserved	Bit 7	Reserved. Always read 0.
LOCKA	Bit 6	Segment A lock. Write a 1 to this bit to change its state. Writing 0 has no effect. 0 Segment A, B, C, D are unlocked. and are erased during a mass erase. 1 Segment A of the information memory is write protected. Segment B, C, and D are protected from all erase.
Reserved	Bit 5	Reserved. Must be written with 0.
LOCK	Bit 4	Lock. This bit unlocks the flash memory for writing or erasing. The LOCK bit can be set any time during a byte/word write or erase operation, and the operation completes normally. In the block write mode, if the LOCK bit is set while BLKWRT = WAIT = 1, BLKWRT and WAIT are reset and the mode ends normally. 0 Unlocked 1 Locked
WAIT	Bit 3	Wait. Indicates the flash memory is being written to. 0 Flash memory is not ready for the next byte/word write. 1 Flash memory is ready for the next byte/word write.
ACCVIFG	Bit 2	Access violation interrupt flag 0 No interrupt pending 1 Interrupt pending
KEYV	Bit 1	Flash password violation. This bit indicates an incorrect FCTLx password was written to any flash control register and generates a PUC when set. KEYV must be reset with software. 0 FCTLx password was written correctly. 1 FCTLx password was written incorrectly.
BUSY	Bit 0	Busy. This bit indicates if the flash is currently busy erasing or programming. 0 Not busy 1 Busy

Flash Memory Control 4 Register (FCTL4)

15	14	13	12	11	10	9	8
FRPW, Read as 096h FWPW, Must be written as 0A5h							

7	6	5	4	3	2	1	0
LOCKINFO	**Reserved**	**MRG1**	**MRG0**	**Reserved**			**VPE**
rw-0	r-0	rw-0	rw-0	r-0	r-0	r-0	rw-0

FRPW/FWPW	Bits 15–8	FCTLx password. Always read as 096h. Must be written as 0A5h or a PUC is generated.
LOCKINFO	Bit 7	Lock information memory. If set, the information memory cannot be erased in segment erase mode and cannot be written to.
Reserved	Bit 6	Reserved. Always read as 0.
MRG1	Bit 5	Marginal read 1 mode. This bit enables the marginal 1 read mode. The marginal read 1 bit is valid for reads from the flash memory only. During a fetch cycle, the marginal mode is turned off automatically. If both MRG1 and MRG0 are set, MRG1 is active and MRG0 is ignored. 0 Marginal 1 read mode is disabled. 1 Marginal 1 read mode is enabled.
MRG0	Bit 4	Marginal read 0 mode. This bit enables the marginal 0 read mode. The marginal read 1 bit is valid for reads from the flash memory only. During a fetch cycle, the marginal mode is turned off automatically. If both MRG1 and MRG0 are set, MRG1 is active and MRG0 is ignored. 0 Marginal 0 read mode is disabled. 1 Marginal 0 read mode is enabled.
Reserved	Bits 3–1	Reserved. Always read as 0.
VPE	Bit 0	Voltage changed during program error. This bit is set by hardware and can only be cleared by software. If DVCC changed significantly during programming, this bit is set to indicate an invalid result. The ACCVIFG bit is set if VPE is set.

Interrupt Enable 1 Register (SFRIE1, SFRIE1_L, SFRIE1_H)

15	14	13	12	11	10	9	8

7	6	5	4	3	2	1	0
		ACCVIE					
		rw-0					

	Bits 15–6, 4–0	These bits may be used by other modules (see the device-specific data sheet and SYS chapter for details).
ACCVIE	Bit 5	Flash memory access violation interrupt enable. This bit enables the ACCVIFG interrupt. Because other bits in SFRIE1 may be used for other modules, it is recommended to set or clear this bit using `BIS` or `BIC` instructions, rather than `MOV` or `CLR` instructions. See the *System Resets, Interrupts, and Operating Modes, System Control Module (SYS)* chapter for more details. 0 Interrupt not enabled 1 Interrupt enabled

RAM Controller

The RAM controller (RAMCTL) allows control of the operation of the RAM.

Topic **Page**

6.1 Ram Controller (RAMCTL) Introduction 278
6.2 RAMCTL Operation 278
6.3 RAMCTL Module Registers 279

6.1 Ram Controller (RAMCTL) Introduction

The RAMCTL provides access to the different power modes of the RAM. The RAMCTL allows the ability to reduce the leakage current while the CPU is off. The RAM can also be switched off. In retention mode, the RAM content is saved while the RAM content is lost in off mode. The RAM is partitioned in sectors, typically of 4KB (sector) size. See the device-specific data sheet for actual block allocation and size. Each sector is controlled by the RAM controller RAM Sector Off control bit (RCRSyOFF) of the RAMCTL Control 0 register (RCCTL0). The RCCTL0 register is protected with a key. Only if the correct key is written during a word write, the RCCTL0 register content can be modified. Byte write accesses or write accesses with a wrong key are ignored.

6.2 RAMCTL Operation

Active mode

In active mode, the RAM can be read and written at any time. If a RAM address of a sector must hold data, the whole sector cannot be switched off.

Low-power modes

In all low-power modes, the CPU is switched off. As soon as the CPU is switched off, the RAM enters retention mode to reduce the leakage current.

RAM off mode

Each sector can be turned off independently of each other by setting the respective RCRSyOFF bit to 1. Reading from a switched off RAM sector returns 0 as data. All data previously stored into a switched off RAM sector is lost and cannot be read, even if the sector is turned on again.

Stack pointer

The program stack is located in RAM. Sectors holding the stack must not be turned off if an interrupt has to be executed, or a low-power mode is entered.

USB buffer memory

On devices with USB, the USB buffer memory is located in RAM. Sector 7 is used for this purpose. RCRS7OFF can be set to switch off this memory if it is not required for USB operation or is not being utilized in normal operation.

6.3 RAMCTL Module Registers

The RAMCTL module register is listed in Table 6-1. The base address can be found in the device-specific data sheet. The address offset is given in Table 6-1.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 6-1. RAMCTL Module Register

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
RAM Controller Control 0	RCCTL0	Read/write	Word	00h	6900h
	RCCTL0_L	Read/write	Byte	00h	00h
	RCCTL0_H	Read/write	Byte	01h	69h

RAM Controller Control 0 Register (RCCTL0)

15	14	13	12	11	10	9	8
RCKEY Always reads as 69h Must be written as 5Ah							
rw-0	rw-1	rw-1	rw-0	rw-1	rw-0	rw-0	rw-1

7	6	5	4	3	2	1	0
RCRS7OFF	**Reserved**			**RCRS3OFF**	**RCRS2OFF**	**RCRS1OFF**	**RCRS0OFF**
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

RCKEY	Bits 15-8	RAM controller key. Always read as 69h. Must be written as 5Ah, otherwise the RAMCTL write is ignored.
RCRS7OFF	Bit 7	RAM controller RAM sector 7 off. Setting the bit to 1 turns off the RAM sector 7. All data of the RAM sector 7 is lost. On devices with USB, this sector is also used as USB buffer memory. See the device-specific data sheet to find the address range and size of each RAM sector.
Reserved	Bits 6-4	Reserved. Always read as 0.
RCRSyOFF	Bits 3-0	RAM controller RAM sector y off. Setting the bit to 1 turns off the RAM sector *y*. All data of the RAM sector *y* is lost. See the device-specific data sheet to find the address range and size of each RAM sector.

DMA Controller

The direct memory access (DMA) controller module transfers data from one address to another, without CPU intervention. This chapter describes the operation of the DMA controller.

Topic **Page**

7.1 Direct Memory Access (DMA) Introduction **282**
7.2 DMA Operation **284**
7.3 DMA Registers **295**

7.1 Direct Memory Access (DMA) Introduction

The DMA controller transfers data from one address to another, without CPU intervention, across the entire address range. For example, the DMA controller can move data from the ADC conversion memory to RAM.

Devices that contain a DMA controller may have up to eight DMA channels available. Therefore, depending on the number of DMA channels available, some features described in this chapter are not applicable to all devices. See the device-specific data sheet for number of channels supported.

Using the DMA controller can increase the throughput of peripheral modules. It can also reduce system power consumption by allowing the CPU to remain in a low-power mode, without having to awaken to move data to or from a peripheral.

DMA controller features include:

- Up to eight independent transfer channels
- Configurable DMA channel priorities
- Requires only two MCLK clock cycles per transfer
- Byte or word and mixed byte/word transfer capability
- Block sizes up to 65535 bytes or words
- Configurable transfer trigger selections
- Selectable-edge or level-triggered transfer
- Four addressing modes
- Single, block, or burst-block transfer modes

The DMA controller block diagram is shown in Figure 7-1.

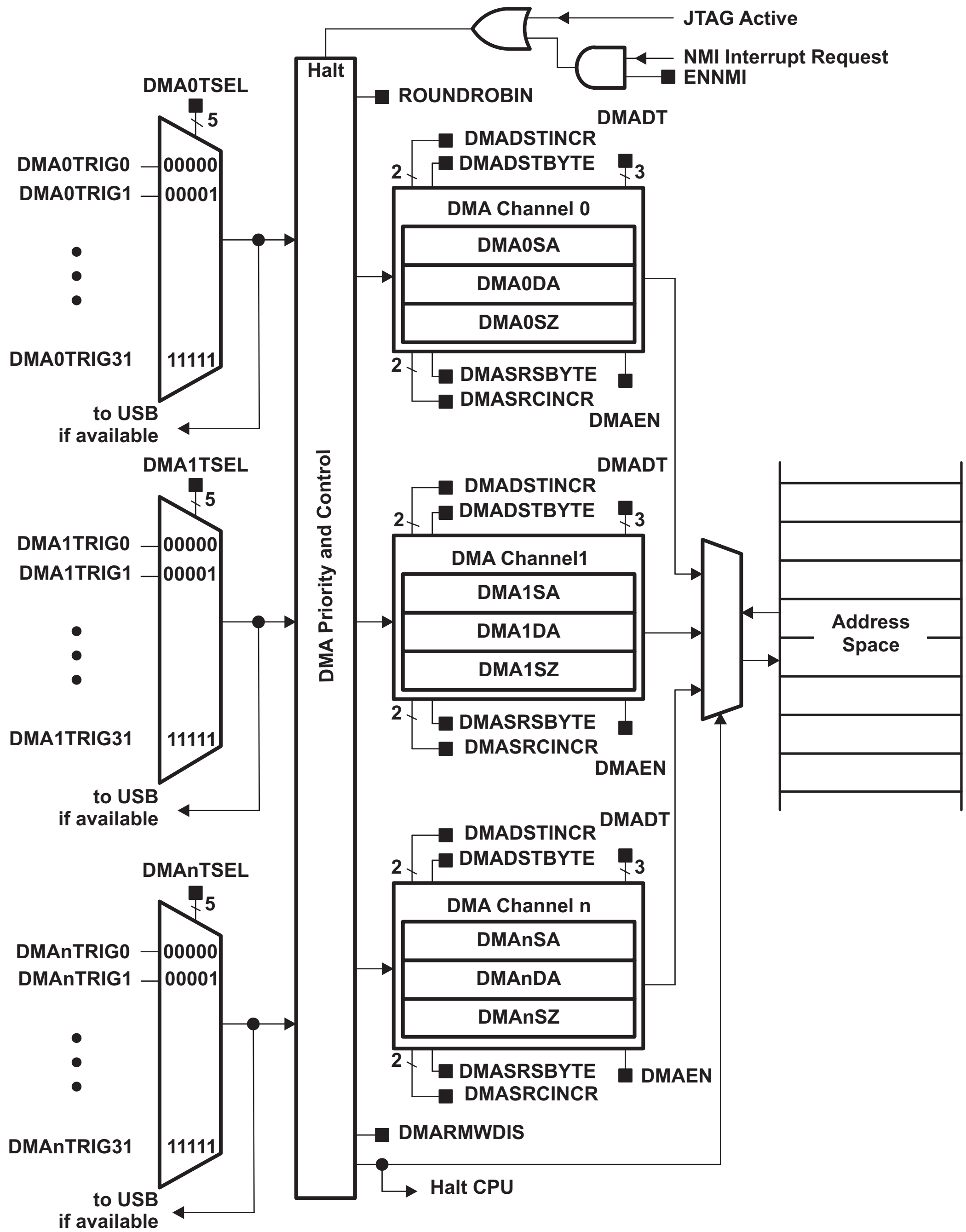


Figure 7-1. DMA Controller Block Diagram

7.2 DMA Operation

The DMA controller is configured with user software. The setup and operation of the DMA is discussed in the following sections.

7.2.1 DMA Addressing Modes

The DMA controller has four addressing modes. The addressing mode for each DMA channel is independently configurable. For example, channel 0 may transfer between two fixed addresses, while channel 1 transfers between two blocks of addresses. The addressing modes are shown in Figure 7-2. The addressing modes are:

- Fixed address to fixed address
- Fixed address to block of addresses
- Block of addresses to fixed address
- Block of addresses to block of addresses

The addressing modes are configured with the DMASRCINCR and DMADSTINCR control bits. The DMASRCINCR bits select if the source address is incremented, decremented, or unchanged after each transfer. The DMADSTINCR bits select if the destination address is incremented, decremented, or unchanged after each transfer.

Transfers may be byte to byte, word to word, byte to word, or word to byte. When transferring word to byte, only the lower byte of the source-word transfers. When transferring byte to word, the upper byte of the destination-word is cleared when the transfer occurs.

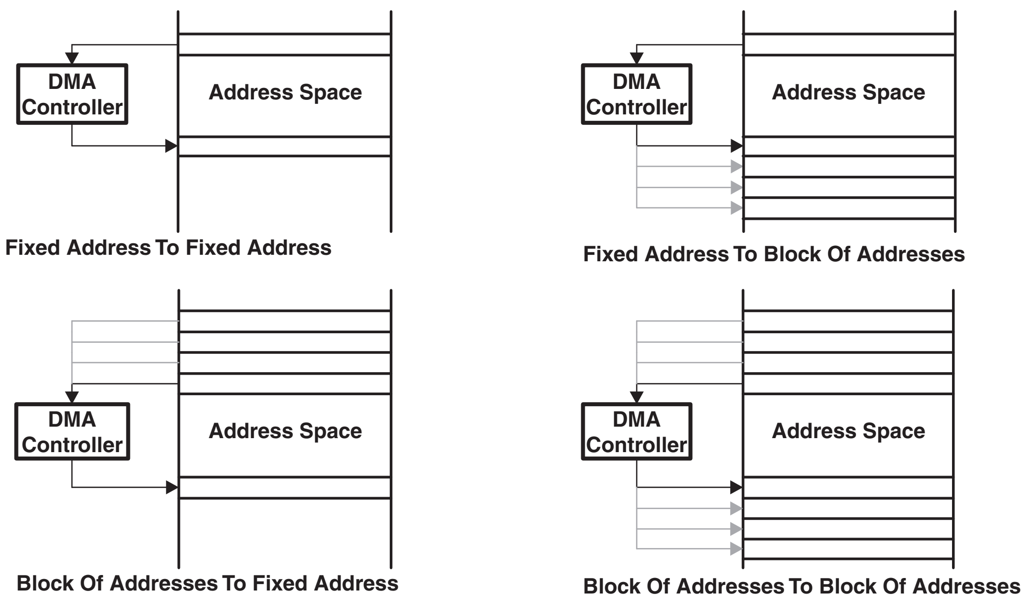


Figure 7-2. DMA Addressing Modes

7.2.2 DMA Transfer Modes

The DMA controller has six transfer modes selected by the DMADT bits as listed in Table 7-1. Each channel is individually configurable for its transfer mode. For example, channel 0 may be configured in single transfer mode, while channel 1 is configured for burst-block transfer mode, and channel 2 operates in repeated block mode. The transfer mode is configured independently from the addressing mode. Any addressing mode can be used with any transfer mode.

Two types of data can be transferred selectable by the DMAxCTL DSTBYTE and SRCBYTE fields. The source and/or destination location can be either byte or word data. It is also possible to transfer byte to byte, word to word, or any combination.

Table 7-1. DMA Transfer Modes

DMADT	Transfer Mode	Description
000	Single transfer	Each transfer requires a trigger. DMAEN is automatically cleared when DMAxSZ transfers have been made.
001	Block transfer	A complete block is transferred with one trigger. DMAEN is automatically cleared at the end of the block transfer.
010, 011	Burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.
100	Repeated single transfer	Each transfer requires a trigger. DMAEN remains enabled.
101	Repeated block transfer	A complete block is transferred with one trigger. DMAEN remains enabled.
110, 111	Repeated burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN remains enabled.

7.2.2.1 Single Transfer

In single transfer mode, each byte/word transfer requires a separate trigger. The single transfer state diagram is shown in Figure 7-3.

The DMAxSZ register is used to define the number of transfers to be made. The DMADSTINCR and DMASRCINCR bits select if the destination address and the source address are incremented or decremented after each transfer. If DMAxSZ = 0, no transfers occur.

The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer. The DMAxSZ register is decremented after each transfer. When the DMAxSZ register decrements to zero, it is reloaded from its temporary register and the corresponding DMAIFG flag is set. When DMADT = {0}, the DMAEN bit is cleared automatically when DMAxSZ decrements to zero and must be set again for another transfer to occur.

In repeated single transfer mode, the DMA controller remains enabled with DMAEN = 1, and a transfer occurs every time a trigger occurs.

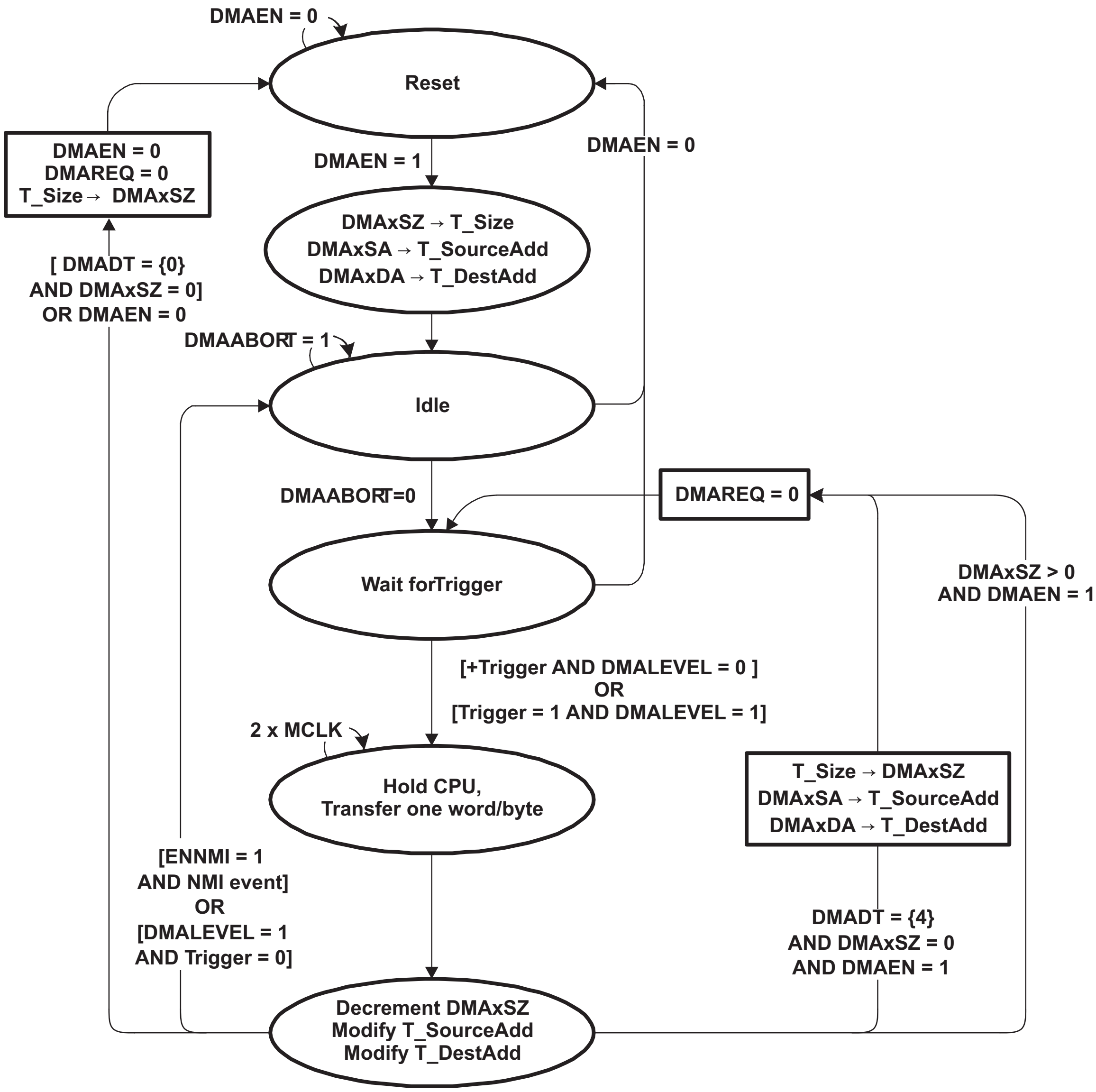


Figure 7-3. DMA Single Transfer State Diagram

7.2.2.2 Block Transfer

In block transfer mode, a transfer of a complete block of data occurs after one trigger. When DMADT = {1} ,the DMAEN bit is cleared after the completion of the block transfer and must be set again before another block transfer can be triggered. After a block transfer has been triggered, further trigger signals occurring during the block transfer are ignored. The block transfer state diagram is shown in Figure 7-4.

The DMAxSZ register is used to define the size of the block, and the DMADSTINCR and DMASRCINCR bits select if the destination address and the source address are incremented or decremented after each transfer of the block. If DMAxSZ = 0, no transfers occur.

The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer in the block. The DMAxSZ register is decremented after each transfer of the block and shows the number of transfers remaining in the block. When the DMAxSZ register decrements to zero, it is reloaded from its temporary register and the corresponding DMAIFG flag is set.

During a block transfer, the CPU is halted until the complete block has been transferred. The block transfer takes 2 × MCLK × DMAxSZ clock cycles to complete. CPU execution resumes with its previous state after the block transfer is complete.

In repeated block transfer mode, the DMAEN bit remains set after completion of the block transfer. The next trigger after the completion of a repeated block transfer triggers another block transfer.

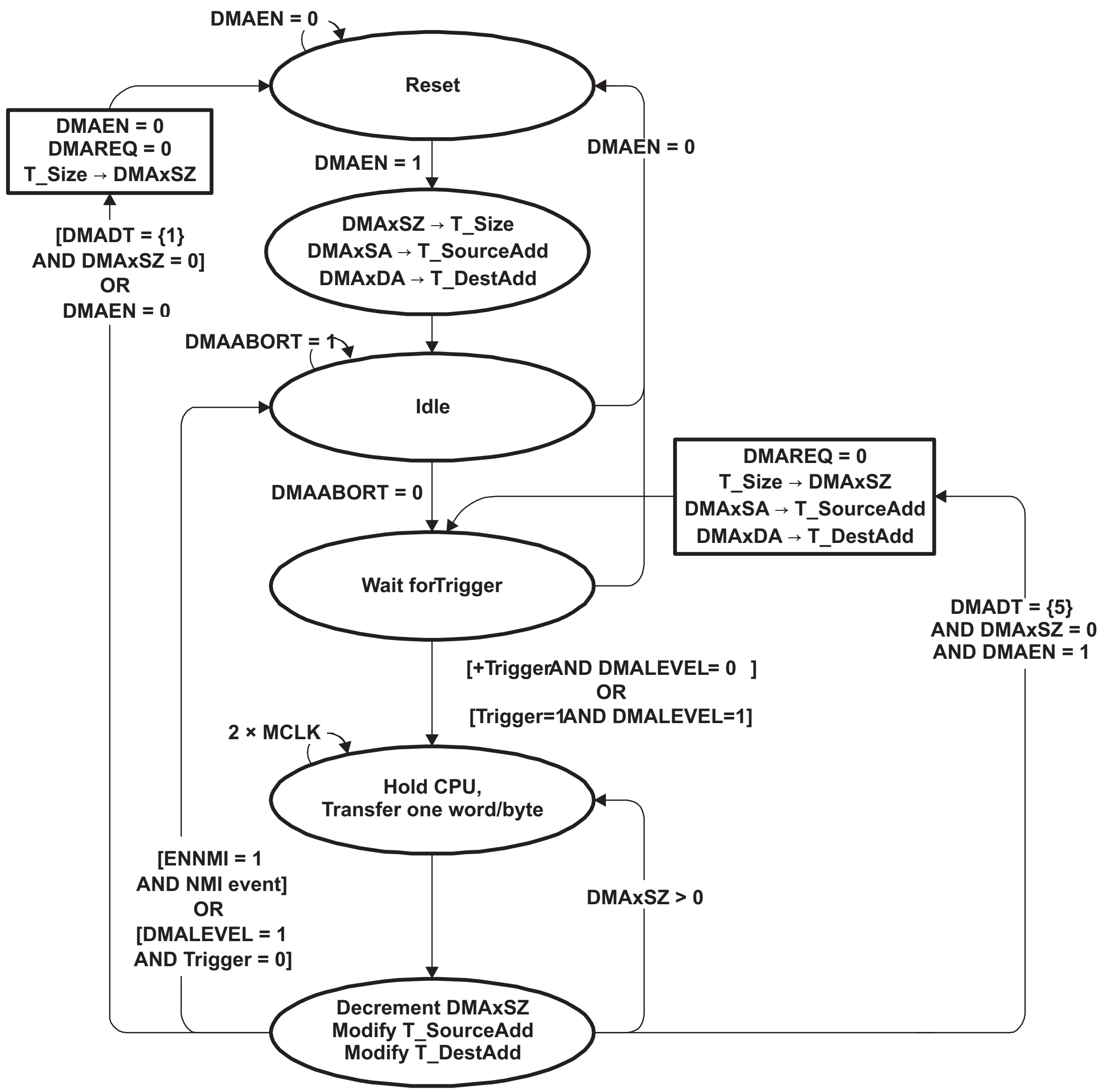


Figure 7-4. DMA Block Transfer State Diagram

7.2.2.3 Burst-Block Transfer

In burst-block mode, transfers are block transfers with CPU activity interleaved. The CPU executes two MCLK cycles after every four byte/word transfers of the block, resulting in 20% CPU execution capacity. After the burst-block, CPU execution resumes at 100% capacity and the DMAEN bit is cleared. DMAEN must be set again before another burst-block transfer can be triggered. After a burst-block transfer has been triggered, further trigger signals occurring during the burst-block transfer are ignored. The burst-block transfer state diagram is shown in Figure 7-5.

The DMAxSZ register is used to define the size of the block, and the DMADSTINCR and DMASRCINCR bits select if the destination address and the source address are incremented or decremented after each transfer of the block. If DMAxSZ = 0, no transfers occur.

The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer in the block. The DMAxSZ register is decremented after each transfer of the block and shows the number of transfers remaining in the block. When the DMAxSZ register decrements to zero, it is reloaded from its temporary register and the corresponding DMAIFG flag is set.

In repeated burst-block mode, the DMAEN bit remains set after completion of the burst-block transfer and no further trigger signals are required to initiate another burst-block transfer. Another burst-block transfer begins immediately after completion of a burst-block transfer. In this case, the transfers must be stopped by clearing the DMAEN bit, or by an (non)maskable interrupt (NMI) when ENNMI is set. In repeated burst-block mode the CPU executes at 20% capacity continuously until the repeated burst-block transfer is stopped.

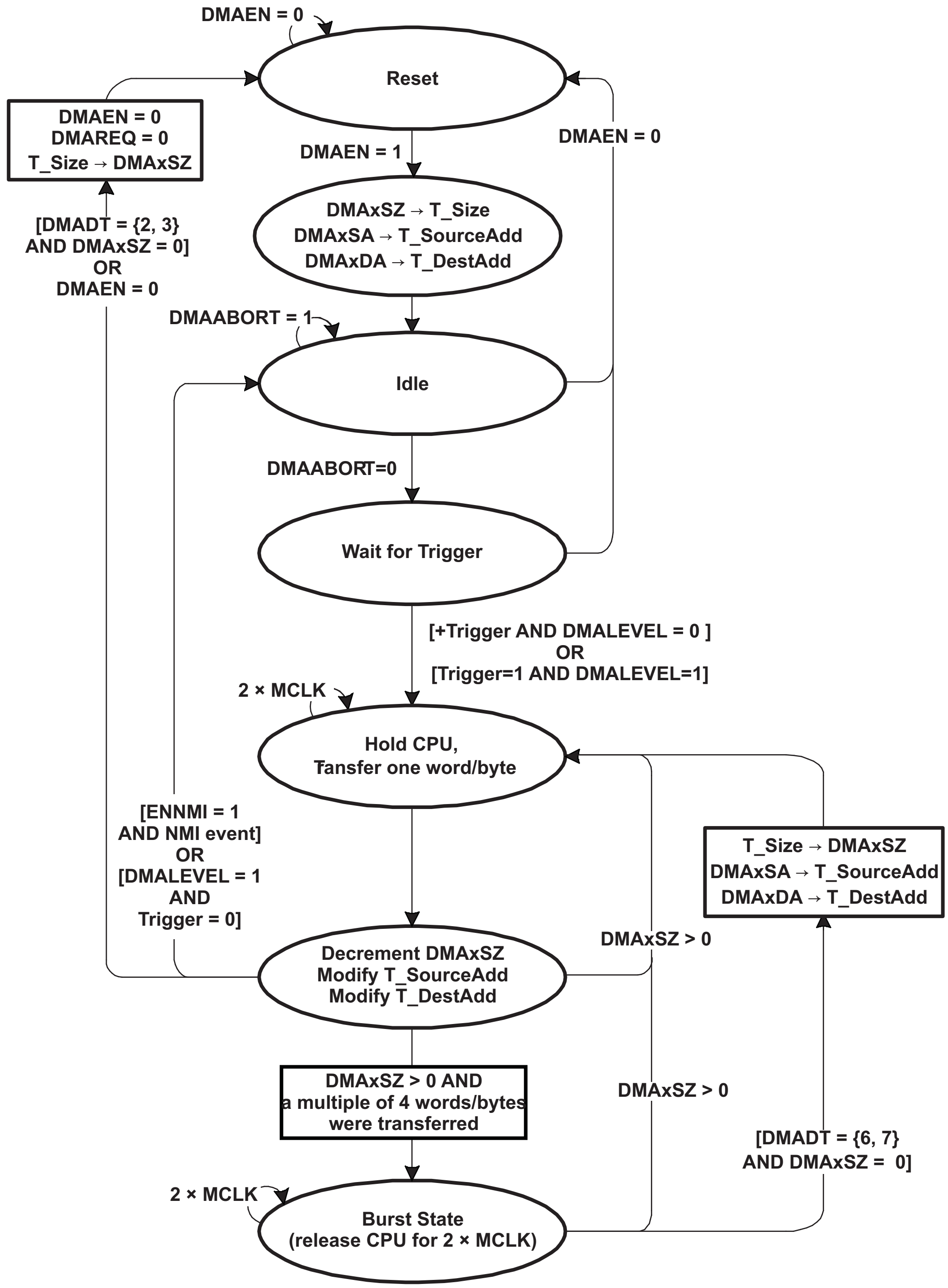


Figure 7-5. DMA Burst-Block Transfer State Diagram

7.2.3 Initiating DMA Transfers

Each DMA channel is independently configured for its trigger source with the DMAxTSEL. The DMAxTSEL bits should be modified only when the DMACTLx DMAEN bit is 0. Otherwise, unpredictable DMA triggers may occur.Table 7-2 describes the trigger operation for each type of module. See the device-specific data sheet for the list of triggers available, along with their respective DMAxTSEL values.

When selecting the trigger, the trigger must not have already occurred, or the transfer does not take place.

NOTE: DMA trigger selection and USB

On devices that contain a USB module, the triggers selection from DMA channels 0, 1, or 2 can be used for the USB time stamp event selection (see the USB module description for further details).

7.2.3.1 Edge-Sensitive Triggers

When DMALEVEL = 0, edge-sensitive triggers are used, and the rising edge of the trigger signal initiates the transfer. In single-transfer mode, each transfer requires its own trigger. When using block or burst-block modes, only one trigger is required to initiate the block or burst-block transfer.

7.2.3.2 Level-Sensitive Triggers

When DMALEVEL = 1, level-sensitive triggers are used. For proper operation, level-sensitive triggers can only be used when external trigger DMAE0 is selected as the trigger. DMA transfers are triggered as long as the trigger signal is high and the DMAEN bit remains set.

The trigger signal must remain high for a block or burst-block transfer to complete. If the trigger signal goes low during a block or burst-block transfer, the DMA controller is held in its current state until the trigger goes back high or until the DMA registers are modified by software. If the DMA registers are not modified by software, when the trigger signal goes high again, the transfer resumes from where it was when the trigger signal went low.

When DMALEVEL = 1, transfer modes selected when DMADT = {0, 1, 2, 3} are recommended because the DMAEN bit is automatically reset after the configured transfer.

7.2.4 Halting Executing Instructions for DMA Transfers

The DMARMWDIS bit controls when the CPU is halted for DMA transfers. When DMARMWDIS = 0, the CPU is halted immediately and the transfer begins when a trigger is received. In this case, it is possible that CPU read-modify-write operations can be interrupted by a DMA transfer. When DMARMWDIS = 1, the CPU finishes the currently executing read-modify-write operation before the DMA controller halts the CPU and the transfer begins (see Table 7-2).

Table 7-2. DMA Trigger Operation

Module	Operation
DMA	A transfer is triggered when the DMAREQ bit is set. The DMAREQ bit is automatically reset when the transfer starts. A transfer is triggered when the DMAxIFG flag is set. DMA0IFG triggers channel 1, DMA1IFG triggers channel 2, and DMA2IFG triggers channel 0. None of the DMAxIFG flags are automatically reset when the transfer starts. A transfer is triggered by the external trigger DMAE0.
Timer_A	A transfer is triggered when the TAxCCR0 CCIFG flag is set. The TAxCCR0 CCIFG flag is automatically reset when the transfer starts. If the TAxCCR0 CCIE bit is set, the TAxCCR0 CCIFG flag dies not trigger a transfer. A transfer is triggered when the TAxCCR2 CCIFG flag is set. The TAxCCR2 CCIFG flag is automatically reset when the transfer starts. If the TAxCCR2 CCIE bit is set, the TAxCCR2 CCIFG flag does not trigger a transfer.
Timer_B	A transfer is triggered when the TBxCCR0 CCIFG flag is set. The TBxCCR0 CCIFG flag is automatically reset when the transfer starts. If the TBxCCR0 CCIE bit is set, the TBxCCR0 CCIFG flag does not trigger a transfer. A transfer is triggered when the TBxCCR2 CCIFG flag is set. The TBxCCR2 CCIFG flag is automatically reset when the transfer starts. If the TBxCCR2 CCIE bit is set, the TBxCCR2 CCIFG flag does not trigger a transfer.
USCI_Ax	A transfer is triggered when USCI_Ax receives new data. UCAxRXIFG is automatically reset when the transfer starts. If UCAxRXIE is set, the UCAxRXIFG does not trigger a transfer. A transfer is triggered when USCI_Ax is ready to transmit new data. UCAxTXIFG is automatically reset when the transfer starts. If UCAxTXIE is set, the UCAxTXIFG does not trigger a transfer.
USCI_Bx	A transfer is triggered when USCI_Bx receives new data. UCBxRXIFG is automatically reset when the transfer starts. If UCBxRXIE is set, the UCBxRXIFG does not trigger a transfer. A transfer is triggered when USCI_Bx is ready to transmit new data. UCBxTXIFG is automatically reset when the transfer starts. If UCBxTXIE is set, the UCBxTXIFG does not trigger a transfer.
DAC12_A	A transfer is triggered when the DAC12_xCTL0 DAC12IFG flag is set. The DAC12_xCTL0 DAC12IFG flag is automatically cleared when the transfer starts. If the DAC12_xCTL0 DAC12IE bit is set, the DAC12_xCTL0 DAC12IFG flag does not trigger a transfer.
ADC12_A	A transfer is triggered by an ADC12IFG flag. When single-channel conversions are performed, the corresponding ADC12IFG is the trigger. When sequences are used, the ADC12IFG for the last conversion in the sequence is the trigger. A transfer is triggered when the conversion is completed and the ADC12IFG is set. Setting the ADC12IFG with software does not trigger a transfer. All ADC12IFG flags are automatically reset when the associated ADC12MEMx register is accessed by the DMA controller.
MPY	A transfer is triggered when the hardware multiplier is ready for a new operand.
Reserved	No transfer is triggered.

7.2.5 Stopping DMA Transfers

There are two ways to stop DMA transfers in progress:

- A single, block, or burst-block transfer may be stopped with an NMI, if the ENNMI bit is set in register DMACTL1.
- A burst-block transfer may be stopped by clearing the DMAEN bit.

7.2.6 DMA Channel Priorities

The default DMA channel priorities are DMA0 through DMA7. If two or three triggers happen simultaneously or are pending, the channel with the highest priority completes its transfer (single, block, or burst-block transfer) first, then the second priority channel, then the third priority channel. Transfers in progress are not halted if a higher-priority channel is triggered. The higher-priority channel waits until the transfer in progress completes before starting.

The DMA channel priorities are configurable with the ROUNDROBIN bit. When the ROUNDROBIN bit is set, the channel that completes a transfer becomes the lowest priority. The *order* of the priority of the channels always stays the same, DMA0-DMA1-DMA2, for example, for three channels. When the ROUNDROBIN bit is cleared, the channel priority returns to the default priority.

DMA Priority	Transfer Occurs	New DMA Priority
DMA0-DMA1-DMA2	DMA1	DMA2-DMA0-DMA1
DMA2-DMA0-DMA1	DMA2	DMA0-DMA1-DMA2
DMA0-DMA1-DMA2	DMA0	DMA1-DMA2-DMA0

7.2.7 DMA Transfer Cycle Time

The DMA controller requires one or two MCLK clock cycles to synchronize before each single transfer or complete block or burst-block transfer. Each byte/word transfer requires two MCLK cycles after synchronization, and one cycle of wait time after the transfer. Because the DMA controller uses MCLK, the DMA cycle time is dependent on the MSP430 operating mode and clock system setup.

If the MCLK source is active but the CPU is off, the DMA controller uses the MCLK source for each transfer, without reenabling the CPU. If the MCLK source is off, the DMA controller temporarily restarts MCLK, sourced with DCOCLK, for the single transfer or complete block or burst-block transfer. The CPU remains off and after the transfer completes, MCLK is turned off. The maximum DMA cycle time for all operating modes is shown in Table 7-3.

Table 7-3. Maximum Single-Transfer DMA Cycle Time

CPU Operating Mode Clock Source	Maximum DMA Cycle Time
Active mode MCLK = DCOCLK	4 MCLK cycles
Active mode MCLK = LFXT1CLK	4 MCLK cycles
Low-power mode LPM0/1 MCLK = DCOCLK	5 MCLK cycles
Low-power mode LPM3/4 MCLK = DCOCLK	5 MCLK cycles + 5 μs(1)
Low-power mode LPM0/1 MCLK = LFXT1CLK	5 MCLK cycles
Low-power mode LPM3 MCLK = LFXT1CLK	5 MCLK cycles
Low-power mode LPM4 MCLK = LFXT1CLK	5 MCLK cycles + 5 μs(1)

(1) The additional 5 μs are needed to start the DCOCLK. It is the $t_{(LPMx)}$ parameter in the data sheet.

7.2.8 Using DMA With System Interrupts

DMA transfers are not interruptible by system interrupts. System interrupts remain pending until the completion of the transfer. NMIs can interrupt the DMA controller if the ENNMI bit is set.

System interrupt service routines are interrupted by DMA transfers. If an interrupt service routine or other routine must execute with no interruptions, the DMA controller should be disabled prior to executing the routine.

7.2.9 DMA Controller Interrupts

Each DMA channel has its own DMAIFG flag. Each DMAIFG flag is set in any mode when the corresponding DMAxSZ register counts to zero. If the corresponding DMAIE and GIE bits are set, an interrupt request is generated.

All DMAIFG flags are prioritized, with DMA0IFG being the highest, and combined to source a single interrupt vector. The highest-priority enabled interrupt generates a number in the DMAIV register. This number can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled DMA interrupts do not affect the DMAIV value.

Any access, read or write, of the DMAIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, assume that DMA0 has the highest priority. If the DMA0IFG and DMA2IFG flags are set when the interrupt service routine accesses the DMAIV register, DMA0IFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the DMA2IFG generates another interrupt.

7.2.9.1 DMAIV Software Example

The following software example shows the recommended use of DMAIV and the handling overhead for an eight channel DMA controller. The DMAIV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself.

```
;Interrupt handler for DMAxIFG                    Cycles
```

```
DMA_HND      ...                ; Interrupt latency          6
        ADD      &DMAIV,PC      ; Add offset to Jump table   3
        RETI                    ; Vector  0: No interrupt    5
        JMP      DMA0_HND       ; Vector  2: DMA channel 0   2
        JMP      DMA1_HND       ; Vector  4: DMA channel 1   2
        JMP      DMA2_HND       ; Vector  6: DMA channel 2   2
        JMP      DMA3_HND       ; Vector  8: DMA channel 3   2
        JMP      DMA4_HND       ; Vector 10: DMA channel 4   2
        JMP      DMA5_HND       ; Vector 12: DMA channel 5   2
        JMP      DMA6_HND       ; Vector 14: DMA channel 6   2
        JMP      DMA7_HND       ; Vector 16: DMA channel 7   2

DMA7_HND                        ; Vector 16: DMA channel 7
        ...                     ; Task starts here
        RETI                    ; Back to main program       5

DMA6_HND                        ; Vector 14: DMA channel 6
        ...                     ; Task starts here
        RETI                    ; Back to main program       5

DMA5_HND                        ; Vector 12: DMA channel 5
        ...                     ; Task starts here
        RETI                    ; Back to main program       5

DMA4_HND                        ; Vector 10: DMA channel 4
        ...                     ; Task starts here
        RETI                    ; Back to main program       5

DMA3_HND                        ; Vector 8: DMA channel 3
        ...                     ; Task starts here
        RETI                    ; Back to main program       5

DMA2_HND                        ; Vector 6: DMA channel 2
        ...                     ; Task starts here
        RETI                    ; Back to main program       5

DMA1_HND                        ; Vector 4: DMA channel 1
        ...                     ; Task starts here
        RETI                    ; Back to main program       5

DMA0_HND                        ; Vector 2: DMA channel 0
        ...                     ; Task starts here
        RETI                    ; Back to main program       5
```

7.2.10 Using the USCI_B I2C Module With the DMA Controller

The USCI_B I2C module provides two trigger sources for the DMA controller. The USCI_B I2C module can trigger a transfer when new I2C data is received and the when the transmit data is needed.

7.2.11 Using ADC12 With the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data from any ADC12MEMx register to another location. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput of the ADC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

DMA transfers can be triggered from any ADC12IFG flag. When CONSEQx = {0,2}, the ADC12IFG flag for the ADC12MEMx used for the conversion can trigger a DMA transfer. When CONSEQx = {1,3}, the ADC12IFG flag for the last ADC12MEMx in the sequence can trigger a DMA transfer. Any ADC12IFG flag is automatically cleared when the DMA controller accesses the corresponding ADC12MEMx.

7.2.12 Using DAC12 With the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data to the DAC12_xDAT register. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput to the DAC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

Applications requiring periodic waveform generation can benefit from using the DMA controller with the DAC12. For example, an application that produces a sinusoidal waveform may store the sinusoid values in a table. The DMA controller can continuously and automatically transfer the values to the DAC12 at specific intervals creating the sinusoid with zero CPU execution. The DAC12_xCTL DAC12IFG flag is automatically cleared when the DMA controller accesses the DAC12_xDAT register.

7.3 DMA Registers

The DMA module registers are listed in Table 7-4. The base addresses can be found in the device-specific data sheet. Each channel starts at its respective base address. The address offsets are listed in Table 7-4.

Table 7-4. DMA Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
DMA Control 0	DMACTL0	Read/write	Word	00h	0000h
DMA Control 1	DMACTL1	Read/write	Word	02h	0000h
DMA Control 2	DMACTL2	Read/write	Word	04h	0000h
DMA Control 3	DMACTL3	Read/write	Word	06h	0000h
DMA Control 4	DMACTL4	Read/write	Word	08h	0000h
DMA Interrupt Vector	DMAIV	Read only	Word	0Eh	0000h
DMA Channel 0 Control	DMA0CTL	Read/write	Word	00h	0000h
DMA Channel 0 Source Address	DMA0SA	Read/write	Word, double word	02h	undefined
DMA Channel 0 Destination Address	DMA0DA	Read/write	Word, double word	06h	undefined
DMA Channel 0 Transfer Size	DMA0SZ	Read/write	Word	0Ah	undefined
DMA Channel 1 Control	DMA1CTL	Read/write	Word	00h	0000h
DMA Channel 1 Source Address	DMA1SA	Read/write	Word, double word	02h	undefined
DMA Channel 1 Destination Address	DMA1DA	Read/write	Word, double word	06h	undefined
DMA Channel 1 Transfer Size	DMA1SZ	Read/write	Word	0Ah	undefined
DMA Channel 2 Control	DMA2CTL	Read/write	Word	00h	0000h
DMA Channel 2 Source Address	DMA2SA	Read/write	Word, double word	02h	undefined
DMA Channel 2 Destination Address	DMA2DA	Read/write	Word, double word	06h	undefined
DMA Channel 2 Transfer Size	DMA2SZ	Read/write	Word	0Ah	undefined
DMA Channel 3 Control	DMA3CTL	Read/write	Word	00h	0000h
DMA Channel 3 Source Address	DMA3SA	Read/write	Word, double word	02h	undefined
DMA Channel 3 Destination Address	DMA3DA	Read/write	Word, double word	06h	undefined
DMA Channel 3 Transfer Size	DMA3SZ	Read/write	Word	0Ah	undefined
DMA Channel 4 Control	DMA4CTL	Read/write	Word	00h	0000h
DMA Channel 4 Source Address	DMA4SA	Read/write	Word, double word	02h	undefined
DMA Channel 4 Destination Address	DMA4DA	Read/write	Word, double word	06h	undefined
DMA Channel 4 Transfer Size	DMA4SZ	Read/write	Word	0Ah	undefined
DMA Channel 5 Control	DMA5CTL	Read/write	Word	00h	0000h
DMA Channel 5 Source Address	DMA5SA	Read/write	Word, double word	02h	undefined
DMA Channel 5 Destination Address	DMA5DA	Read/write	Word, double word	06h	undefined
DMA Channel 5 Transfer Size	DMA5SZ	Read/write	Word	0Ah	undefined
DMA Channel 6 Control	DMA6CTL	Read/write	Word	00h	0000h
DMA Channel 6 Source Address	DMA6SA	Read/write	Word, double word	02h	undefined
DMA Channel 6 Destination Address	DMA6DA	Read/write	Word, double word	06h	undefined
DMA Channel 6 Transfer Size	DMA6SZ	Read/write	Word	0Ah	undefined

Table 7-4. DMA Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
DMA Channel 7 Control	DMA7CTL	Read/write	Word	00h	0000h
DMA Channel 7 Source Address	DMA7SA	Read/write	Word, double word	02h	undefined
DMA Channel 7 Destination Address	DMA7DA	Read/write	Word, double word	06h	undefined
DMA Channel 7 Transfer Size	DMA7SZ	Read/write	Word	0Ah	undefined

7.3.1 DMA Control 0 Register (DMACTL0)

15	14	13	12	11	10	9	8
Reserved			DMA1TSEL				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
Reserved			DMA0TSEL				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved	Bits 15-13	Reserved. Read only. Always read as 0.
DMA1TSEL	Bits 12-8	DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000 DMA1TRIG0 00001 DMA1TRIG1 00010 DMA1TRIG2 ⋮ 11110 DMA1TRIG30 11111 DMA1TRIG31
Reserved	Bits 7-5	Reserved. Read only. Always read as 0.
DMA0TSEL	Bits 4-0	Same as DMA1TSEL

7.3.2 DMA Control 1 Register (DMACTL1)

15	14	13	12	11	10	9	8
Reserved			DMA3TSEL				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
Reserved			DMA2TSEL				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved	Bits 15-13	Reserved. Read only. Always read as 0.
DMA3TSEL	Bits 12-8	DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000 DMA3TRIG0 00001 DMA3TRIG1 00010 DMA3TRIG2 ⋮ 11110 DMA3TRIG30 11111 DMA3TRIG31
Reserved	Bits 7-5	Reserved. Read only. Always read as 0.
DMA2TSEL	Bits 4-0	Same as DMA3TSEL

7.3.3 DMA Control 2 Register (DMACTL2)

15	14	13	12	11	10	9	8
Reserved			**DMA5TSEL**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved			**DMA4TSEL**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-13 Reserved. Read only. Always read as 0.

DMA5TSEL Bits 12-8 DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment.

00000 DMA5TRIG0
00001 DMA5TRIG1
00010 DMA5TRIG2
⋮
11110 DMA5TRIG30
11111 DMA5TRIG31

Reserved Bits 7-5 Reserved. Read only. Always read as 0.

DMA4TSEL Bits 4-0 Same as DMA5TSEL

7.3.4 DMA Control 3 Register (DMACTL3)

15	14	13	12	11	10	9	8
Reserved			**DMA7TSEL**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved			**DMA6TSEL**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-13 Reserved. Read only. Always read as 0.

DMA7TSEL Bits 12-8 DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment.

00000 DMA7TRIG0
00001 DMA7TRIG1
00010 DMA7TRIG2
⋮
11110 DMA7TRIG30
11111 DMA7TRIG31

Reserved Bits 7-5 Reserved. Read only. Always read as 0.

DMA6TSEL Bits 4-0 Same as DMA7TSEL

7.3.5 DMA Control 4 Register (DMACTL4)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**DMARMWDIS**	**ROUND ROBIN**	**ENNMI**
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

Reserved	Bits 15-3	Reserved. Read only. Always read as 0.
DMARMWDIS	Bit 2	Read-modify-write disable. When set, this bit inhibits any DMA transfers from occurring during CPU read-modify-write operations. 0 DMA transfers can occur during read-modify-write CPU operations. 1 DMA transfers inhibited during read-modify-write CPU operations
ROUNDROBIN	Bit 1	Round robin. This bit enables the round-robin DMA channel priorities. 0 DMA channel priority is DMA0-DMA1-DMA2 - ...... -DMA7. 1 DMA channel priority changes with each transfer.
ENNMI	Bit 0	Enable NMI. This bit enables the interruption of a DMA transfer by an NMI. When an NMI interrupts a DMA transfer, the current transfer is completed normally, further transfers are stopped and DMAABORT is set. 0 NMI does not interrupt DMA transfer. 1 NMI interrupts a DMA transfer.

7.3.6 DMA Channel x Control Register (DMAxCTL)

15	14	13	12	11	10	9	8
Reserved	**DMADT**			**DMADSTINCR**		**DMASRCINCR**	
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
DMA DSTBYTE	**DMA SRCBYTE**	**DMALEVEL**	**DMAEN**	**DMAIFG**	**DMAIE**	**DMAABORT**	**DMAREQ**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bit 15 Reserved. Read only. Always read as 0.

DMADT Bits 14-12 DMA transfer mode

- 000 Single transfer
- 001 Block transfer
- 010 Burst-block transfer
- 011 Burst-block transfer
- 100 Repeated single transfer
- 101 Repeated block transfer
- 110 Repeated burst-block transfer
- 111 Repeated burst-block transfer

DMADSTINCR Bits 11-10 DMA destination increment. This bit selects automatic incrementing or decrementing of the destination address after each byte or word transfer. When DMADSTBYTE = 1, the destination address increments/decrements by one. When DMADSTBYTE = 0, the destination address increments/decrements by two. The DMAxDA is copied into a temporary register and the temporary register is incremented or decremented. DMAxDA is not incremented or decremented.

- 00 Destination address is unchanged.
- 01 Destination address is unchanged.
- 10 Destination address is decremented.
- 11 Destination address is incremented.

DMASRCINCR Bits 9-8 DMA source increment. This bit selects automatic incrementing or decrementing of the source address for each byte or word transfer. When DMASRCBYTE = 1, the source address increments/decrements by one. When DMASRCBYTE = 0, the source address increments/decrements by two. The DMAxSA is copied into a temporary register and the temporary register is incremented or decremented. DMAxSA is not incremented or decremented.

- 00 Source address is unchanged.
- 01 Source address is unchanged.
- 10 Source address is decremented.
- 11 Source address is incremented.

DMADSTBYTE Bit 7 DMA destination byte. This bit selects the destination as a byte or word.

- 0 Word
- 1 Byte

DMASRCBYTE Bit 6 DMA source byte. This bit selects the source as a byte or word.

- 0 Word
- 1 Byte

DMALEVEL Bit 5 DMA level. This bit selects between edge-sensitive and level-sensitive triggers.

- 0 Edge sensitive (rising edge)
- 1 Level sensitive (high level)

DMAEN Bit 4 DMA enable

- 0 Disabled
- 1 Enabled

DMAIFG	Bit 3	DMA interrupt flag	
		0	No interrupt pending
		1	Interrupt pending
DMAIE	Bit 2	DMA interrupt enable	
		0	Disabled
		1	Enabled
DMAABORT	Bit 1	DMA abort. This bit indicates if a DMA transfer was interrupt by an NMI.	
		0	DMA transfer not interrupted
		1	DMA transfer interrupted by NMI
DMAREQ	Bit 0	DMA request. Software-controlled DMA start. DMAREQ is reset automatically.	
		0	No DMA start
		1	Start DMA

7.3.7 DMA Source Address Register (DMAxSA)

31	30	29	28	27	26	25	24
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
23	22	21	20	19	18	17	16
Reserved				**DMAxSA**			
r0	r0	r0	r0	rw	rw	rw	rw
15	14	13	12	11	10	9	8
DMAxSA							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
DMAxSA							
rw	rw	rw	rw	rw	rw	rw	rw

Reserved	Bits 31-20	Reserved. Read only. Always read as 0.
DMAxSA	Bits 15-0	DMA source address. The source address register points to the DMA source address for single transfers or the first source address for block transfers. The source address register remains unchanged during block and burst-block transfers. There are two words for the DMAxSA register. Bits 31–20 are reserved and always read as zero. Reading or writing bits 19-16 requires the use of extended instructions. When writing to DMAxSA with word instructions, bits 19–16 are cleared.

7.3.8 DMA Destination Address Register (DMAxDA)

31	30	29	28	27	26	25	24
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
23	22	21	20	19	18	17	16
Reserved				**DMAxDA**			
r0	r0	r0	r0	rw	rw	rw	rw
15	14	13	12	11	10	9	8
DMAxDA							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
DMAxDA							
rw	rw	rw	rw	rw	rw	rw	rw

Reserved	Bits 31-20	Reserved. Read only. Always read as 0.
DMAxDA	Bits 15-0	DMA destination address. The destination address register points to the DMA destination address for single transfers or the first destination address for block transfers. The destination address register remains unchanged during block and burst-block transfers. There are two words for the DMAxDA register. Bits 31–20 are reserved and always read as zero. Reading or writing bits 19–16 requires the use of extended instructions. When writing to DMAxDA with word instructions, bits 19–16 are cleared.

7.3.9 DMA Size Address Register (DMAxSZ)

15	14	13	12	11	10	9	8
DMAxSZ							
rw	rw	rw	rw	rw	rw	rw	rw

7	6	5	4	3	2	1	0
DMAxSZ							
rw	rw	rw	rw	rw	rw	rw	rw

DMAxSZ Bits 15-0 DMA size. The DMA size register defines the number of byte/word data per block transfer. DMAxSZ register decrements with each word or byte transfer. When DMAxSZ decrements to 0, it is immediately and automatically reloaded with its previously initialized value.

00000h	Transfer is disabled.
00001h	One byte or word is transferred.
00002h	Two bytes or words are transferred.
⋮	
0FFFFh	65535 bytes or words are transferred.

7.3.10 DMA Interrupt Vector Register (DMAIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**DMAIV**					**0**
r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)	r0

DMAIV Bits 15-0 DMA interrupt vector value

DMAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	DMA channel 0	DMA0IFG	Highest
04h	DMA channel 1	DMA1IFG	
06h	DMA channel 2	DMA2IFG	
08h	DMA channel 3	DMA3IFG	
0Ah	DMA channel 4	DMA4IFG	
0Ch	DMA channel 5	DMA5IFG	
0Eh	DMA channel 6	DMA6IFG	
10h	DMA channel 7	DMA7IFG	Lowest

Digital I/O

This chapter describes the operation of the digital I/O ports in all devices.

Topic **Page**

8.1	**Digital I/O Introduction**	**304**
8.2	**Digital I/O Operation**	**305**
8.3	**I/O Configuration and LPMx.5 Low-Power Modes**	**308**
8.4	**Digital I/O Registers**	**310**

8.1 Digital I/O Introduction

The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts. Some devices may include additional port interrupts.
- Independent input and output data registers
- Individually configurable pullup or pulldown resistors

Devices within the family may have up to twelve digital I/O ports implemented (P1 to P11 and PJ). Most ports contain eight I/O lines; however, some ports may contain less (see the device-specific data sheet for ports available). Each I/O line is individually configurable for input or output direction, and each can be individually read or written. Each I/O line is individually configurable for pullup or pulldown resistors, as well as, configurable drive strength, full or reduced. PJ contains only four I/O lines.

Ports P1 and P2 always have interrupt capability. Each interrupt for the P1 and P2 I/O lines can be individually enabled and configured to provide an interrupt on a rising or falling edge of an input signal. All P1 I/O lines source a single interrupt vector P1IV, and all P2 I/O lines source a different, single interrupt vector P2IV. On some devices, additional ports with interrupt capability may be available (see the device-specific data sheet for details) and contain their own respective interrupt vectors.

Individual ports can be accessed as byte-wide ports or can be combined into word-wide ports and accessed via word formats. Port pairs P1/P2, P3/P4, P5/P6, P7/P8, etc., are associated with the names PA, PB, PC, PD, etc., respectively. All port registers are handled in this manner with this naming convention except for the interrupt vector registers, P1IV and P2IV; i.e. PAIV does not exist.

When writing to port PA with word operations, all 16 bits are written to the port. When writing to the lower byte of the PA port using byte operations, the upper byte remains unchanged. Similarly, writing to the upper byte of the PA port using byte instructions leaves the lower byte unchanged. When writing to a port that contains less than the maximum number of bits possible, the unused bits are a "don't care". Ports PB, PC, PD, PE, and PF behave similarly.

Reading of the PA port using word operations causes all 16 bits to be transferred to the destination. Reading the lower or upper byte of the PA port (P1 or P2) and storing to memory using byte operations causes only the lower or upper byte to be transferred to the destination, respectively. Reading of the PA port and storing to a general-purpose register using byte operations causes the byte transferred to be written to the least significant byte of the register. The upper significant byte of the destination register is cleared automatically. Ports PB, PC, PD, PE, and PF behave similarly. When reading from ports that contain less than the maximum bits possible, unused bits are read as zeros (similarly for port PJ).

8.2 Digital I/O Operation

The digital I/O are configured with user software. The setup and operation of the digital I/O are discussed in the following sections.

8.2.1 Input Registers PxIN

Each bit in each PxIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function. These registers are read only.

- Bit = 0: Input is low
- Bit = 1: Input is high

NOTE: Writing to read-only registers PxIN

Writing to these read-only registers results in increased current consumption while the write attempt is active.

8.2.2 Output Registers PxOUT

Each bit in each PxOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function, output direction.

- Bit = 0: Output is low
- Bit = 1: Output is high

If the pin is configured as I/O function, input direction and the pullup/pulldown resistor are enabled; the corresponding bit in the PxOUT register selects pullup or pulldown.

- Bit = 0: Pin is pulled down
- Bit = 1: Pin is pulled up

8.2.3 Direction Registers PxDIR

Each bit in each PxDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin. PxDIR bits for I/O pins that are selected for other functions must be set as required by the other function.

- Bit = 0: Port pin is switched to input direction
- Bit = 1: Port pin is switched to output direction

8.2.4 Pullup/Pulldown Resistor Enable Registers PxREN

Each bit in each PxREN register enables or disables the pullup/pulldown resistor of the corresponding I/O pin. The corresponding bit in the PxOUT register selects if the pin contains a pullup or pulldown.

- Bit = 0: Pullup/pulldown resistor disabled
- Bit = 1: Pullup/pulldown resistor enabled

Table 8-1 summarizes the usage of PxDIR, PxREN, and PxOUT for proper I/O configuration.

Table 8-1. I/O Configuration

PxDIR	PxREN	PxOUT	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

8.2.5 Output Drive Strength Registers PxDS

Each bit in each PxDS register selects either full drive or reduced drive strength. Default is reduced drive strength.

- Bit = 0: Reduced drive strength
- Bit = 1: Full drive strength

NOTE: Drive strength and EMI

All outputs default to reduced drive strength to reduce EMI. Using full drive strength can result in increased EMI.

8.2.6 Function Select Registers PxSEL

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each PxSEL bit is used to select the pin function – I/O port or peripheral module function.

- Bit = 0: I/O Function is selected for the pin
- Bit = 1: Peripheral module function is selected for the pin

Setting PxSEL = 1 does not automatically set the pin direction. Other peripheral module functions may require the PxDIR bits to be configured according to the direction needed for the module function. See the pin schematics in the device-specific data sheet.

NOTE: P1 and P2 interrupts are disabled when PxSEL = 1

When any PxSEL bit is set, the corresponding pin’s interrupt function is disabled. Therefore, signals on these pins does not generate P1 or P2 interrupts, regardless of the state of the corresponding P1IE or P2IE bit.

When a port pin is selected as an input to a peripheral, the input signal to the peripheral is a latched representation of the signal at the device pin. While its corresponding PxSEL = 1, the internal input signal follows the signal at the pin. However, if its PxSEL = 0, the input to the peripheral maintains the value of the input signal at the device pin before its corresponding PxSEL bit was reset.

8.2.7 P1 and P2 Interrupts, Port Interrupts

Each pin in ports P1 and P2 have interrupt capability, configured with the PxIFG, PxIE, and PxIES registers. All P1 interrupt flags are prioritized, with P1IFG.0 being the highest, and combined to source a single interrupt vector. The highest priority enabled interrupt generates a number in the P1IV register. This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled P1 interrupts do not affect the P1IV value. The same functionality exists for P2. The PxIV registers are word access only. Some devices may contain additional port interrupts besides P1 and P2. Please see the device specific data sheet to determine which port interrupts are available.

Each PxIFG bit is the interrupt flag for its corresponding I/O pin and is set when the selected input signal edge occurs at the pin. All PxIFG interrupt flags request an interrupt when their corresponding PxIE bit and the GIE bit are set. Software can also set each PxIFG flag, providing a way to generate a software-initiated interrupt.

- Bit = 0: No interrupt is pending
- Bit = 1: An interrupt is pending

Only transitions, not static levels, cause interrupts. If any PxIFG flag becomes set during a Px interrupt service routine, or is set after the RETI instruction of a Px interrupt service routine is executed, the set PxIFG flag generates another interrupt. This ensures that each transition is acknowledged.

NOTE: PxIFG flags when changing PxOUT, PxDIR, or PxREN

Writing to P1OUT, P1DIR, P1REN, P2OUT, P2DIR, or P2REN can result in setting the corresponding P1IFG or P2IFG flags.

Any access (read or write) of the P1IV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, assume that P1IFG.0 has the highest priority. If the P1IFG.0 and P1IFG.2 flags are set when the interrupt service routine accesses the P1IV register, P1IFG.0 is reset automatically. After the RETI instruction of the interrupt service routine is executed, the P1IFG.2 generates another interrupt.

Port P2 interrupts behave similarly, and source a separate single interrupt vector and utilize the P2IV register.

8.2.7.1 P1IV, P2IV Software Example

The following software example shows the recommended use of P1IV and the handling overhead. The P1IV value is added to the PC to automatically jump to the appropriate routine. The P2IV is similar.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself.

```
;Interrupt handler for P1                                      Cycles
P1_HND   ...                ; Interrupt latency                  6
         ADD      &P1IV,PC  ; Add offset to Jump table           3
         RETI               ; Vector  0: No interrupt            5
         JMP      P1_0_HND  ; Vector  2: Port 1 bit 0            2
         JMP      P1_1_HND  ; Vector  4: Port 1 bit 1            2
         JMP      P1_2_HND  ; Vector  6: Port 1 bit 2            2
         JMP      P1_3_HND  ; Vector  8: Port 1 bit 3            2
         JMP      P1_4_HND  ; Vector 10: Port 1 bit 4            2
         JMP      P1_5_HND  ; Vector 12: Port 1 bit 5            2
         JMP      P1_6_HND  ; Vector 14: Port 1 bit 6            2
         JMP      P1_7_HND  ; Vector 16: Port 1 bit 7            2

P1_7_HND                    ; Vector 16: Port 1 bit 7
         ...                ; Task starts here
         RETI               ; Back to main program               5

P1_6_HND                    ; Vector 14: Port 1 bit 6
         ...                ; Task starts here
         RETI               ; Back to main program               5

P1_5_HND                    ; Vector 12: Port 1 bit 5
         ...                ; Task starts here
         RETI               ; Back to main program               5

P1_4_HND                    ; Vector 10: Port 1 bit 4
         ...                ; Task starts here
         RETI               ; Back to main program               5

P1_3_HND                    ; Vector 8: Port 1 bit 3
         ...                ; Task starts here
         RETI               ; Back to main program               5

P1_2_HND                    ; Vector 6: Port 1 bit 2
         ...                ; Task starts here
         RETI               ; Back to main program               5

P1_1_HND                    ; Vector 4: Port 1 bit 1
         ...                ; Task starts here
         RETI               ; Back to main program               5
P1_0_HND                    ; Vector 2: Port 1 bit 0
         ...                ; Task starts here
         RETI               ; Back to main program               5
```

8.2.7.2 Interrupt Edge Select Registers P1IES, P2IES

Each PxIES bit selects the interrupt edge for the corresponding I/O pin.

- Bit = 0: Respective PxIFG flag is set with a low-to-high transition
- Bit = 1: Respective PxIFG flag is set with a high-to-low transition

NOTE: Writing to PxIES

Writing to P1IES or P2IES for each corresponding I/O can result in setting the corresponding interrupt flags.

PxIES	PxIN	PxIFG
$0 \rightarrow 1$	0	May be set
$0 \rightarrow 1$	1	Unchanged
$1 \rightarrow 0$	0	Unchanged
$1 \rightarrow 0$	1	May be set

8.2.7.3 Interrupt Enable P1IE, P2IE

Each PxIE bit enables the associated PxIFG interrupt flag.

- Bit = 0: The interrupt is disabled
- Bit = 1: The interrupt is enabled

8.2.8 Configuring Unused Port Pins

Unused I/O pins should be configured as I/O function, output direction, and left unconnected on the PC board, to prevent a floating input and reduce power consumption. The value of the PxOUT bit is don't care, because the pin is unconnected. Alternatively, the integrated pullup/pulldown resistor can be enabled by setting the PxREN bit of the unused pin to prevent the floating input. See the *System Resets, Interrupts, and Operating Modes, System Control Module (SYS))* chapter for termination of unused pins.

NOTE: Configuring port J and shared JTAG pins:

Application should ensure that port PJ is configured properly to prevent a floating input. Because port PJ is shared with the JTAG function, floating inputs may not be noticed when in an emulation environment. Port J is initialized to high-impedance inputs by default.

8.3 I/O Configuration and LPMx.5 Low-Power Modes

NOTE: The LPMx.5 low power modes may not be available on all devices. The LPM4.5 power mode allows for lowest power consumption and no clocks are available. The LPM3.5 power mode allows for RTC mode operation at the lowest power consumption available. Please refer to the *SYS* chapter for details, as well as, the device specific datasheet for LPMx.5 low power modes that are available. With respect to the digital I/O, this section is applicable for both LPM3.5 and LPM4.5.

The regulator of the Power Management Module (PMM) is disabled upon entering LPMx.5 (LPM3.5 or LPM4.5), which causes all I/O register configurations to be lost. Because the I/O register configurations are lost, the configuration of I/O pins must be handled differently to ensure that all pins in the application behave in a controlled manner upon entering and exiting LPMx.5. Properly setting the I/O pins is critical to achieving the lowest possible power consumption in LPMx.5, as well as preventing any possible uncontrolled input or output I/O state in the application. The application has complete control of the I/O pin conditions preventing the possibility of unwanted spurious activity upon entry and exit from LPMx.5. The detailed flow for entering and exiting LPMx.5 with respect to the I/O operation is as follows:

1. Set all I/Os to general purpose I/Os and configure as needed. Each I/O can be set to input high impedance, input with pulldown, input with pullup, output high (low or high drive strength), or output low (low or high drive strength). It is critical that no inputs are left floating in the application, otherwise excess current may be drawn in LPMx.5. Configuring the I/O in this manner ensures that each pin is in a safe condition prior to entering LPMx.5.Optionally, configure input interrupt pins for wake-up from LPMx.5.To wake the device from LPMx.5, a general-purpose I/O port must contain an input port with interrupt capability. Not all devices include wakeup from LPMx.5 via I/O, and not all inputs with interrupt capability offer wakeup from LPMx5. See the device-specific data sheet for availability. To configure a port to wake up the device, it should be configured properly prior to entering LPMx.5. Each port should be configured as general-purpose input. Pulldowns or pullups can be applied if required. Setting the PxIES bit of the corresponding register determines the edge transition that wakes the device. Lastly, the PxIE for the port must be enabled, as well as the general interrupt enable.
2. Enter LPMx.5 with LPMx.5 entry sequence, enable general interrupts for wake-up:

```
MOV.B #PMMPW_H, &PMMCTL0_H                  ; Open PMM registers for write
BIS.B #PMMREGOFF, &PMMCTL0_L                ;
BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR       ; Enter LPMx.5 when PMMREGOFF is set
```

3. Upon entry into LPMx.5, LOCKLPM5 residing in PM5CTL0 of the PMM module, is set automatically. The I/O pin states are held and locked based on the settings prior to LPMx.5 entry. Please note that only the pin conditions are retained. All other port configuration register settings such as PxDIR, PxREN, PxOUT, PxDS, PxIES, and PxIE contents are lost.
4. A LPMx.5 wakeup event e.g. an edge on a configured wakeup input pin, will start the BOR entry seqence together with the regulator. All peripheral registers are set to their default conditions. Upon exit from LPMx.5, the I/O pins remain locked while LOCKLPM5 remains set. Keeping the I/O pins locked ensures that all pin conditions remain stable upon entering the active mode regardless of the default I/O register settings.
5. Once in active mode, the I/O configuration and I/O interrupt configuration that was not retained during LPMx.5 should be restored to the values prior to entering LPMx.5. It is recommended to reconfigure the PxIES and PxIE to their previous settings to prevent a false port interrupt from occurring. The LOCKLPM5 bit can then be cleared, which releases the I/O pin conditions and I/O interrupt configuration. Any changes to the port configuration registers while LOCKLPM5 is set, have no effect on the I/O pins.
6. After enabling the I/O interrupts, the I/O nterrupt that caused the wakeup can be serviced indicated by the PxIFG flags. These flags can be used directly, or the corresponding PxIV register may be used. Please note that the PxIFG flag cannot be cleared until the LOCKLPM5 bit has been cleared.

NOTE: It is possible that multiple events occurred on various ports. In these cases, multiple PxIFG flags will be set and it cannot be determined which port has caused the I/O wakeup.

8.4 Digital I/O Registers

The digital I/O registers are listed in Table 8-2. The base addresses can be found in the device-specific data sheet. Each port grouping begins at its base address. The address offsets are given in Table 8-2.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 8-2. Digital I/O Registers

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port 1	Interrupt Vector	P1IV	Read only	Word	0Eh	0000h
		P1IV_L	Read only	Byte	0Eh	00h
		P1IV_H	Read only	Byte	0Fh	00h
Port 2	Interrupt Vector	P2IV	Read only	Word	1Eh	0000h
		P2IV_L	Read only	Byte	1Eh	00h
		P2IV_H	Read only	Byte	1Fh	00h
Port 1	Input	P1IN or PAIN_L	Read only	Byte	00h	
	Output	P1OUT or PAOUT_L	Read/write	Byte	02h	undefined
	Direction	P1DIR or PADIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P1REN or PAREN_L	Read/write	Byte	06h	00h
	Drive Strength	P1DS or PADS_L	Read/write	Byte	08h	00h
	Port Select	P1SEL or PASEL_L	Read/write	Byte	0Ah	00h
	Interrupt Edge Select	P1IES or PAIES_L	Read/write	Byte	18h	undefined
	Interrupt Enable	P1IE or PAIE_L	Read/write	Byte	1Ah	00h
	Interrupt Flag	P1IFG or PAIFG_L	Read/write	Byte	1Ch	00h
Port 2	Input	P2IN or PAIN_H	Read only	Byte	01h	
	Output	P2OUT or PAOUT_H	Read/write	Byte	03h	undefined
	Direction	P2DIR or PADIR_H	Read/write	Byte	05h	00h
	Resistor Enable	P2REN or PAREN_H	Read/write	Byte	07h	00h
	Drive Strength	P2DS or PADS_H	Read/write	Byte	09h	00h
	Port Select	P2SEL or PASEL_H	Read/write	Byte	0Bh	00h
	Interrupt Edge Select	P2IES or PAIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	P2IE or PAIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	P2IFG or PAIFG_H	Read/write	Byte	1Dh	00h

Table 8-2. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port 3	Input	P3IN or PBIN_L	Read only	Byte	00h	
	Output	P3OUT or PBOUT_L	Read/write	Byte	02h	undefined
	Direction	P3DIR or PBDIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P3REN or PBREN_L	Read/write	Byte	06h	00h
	Drive Strength	P3DS or PBDS_L	Read/write	Byte	08h	00h
	Port Select	P3SEL or PBSEL_L	Read/write	Byte	0Ah	00h
Port 4	Input	P4IN or PBIN_H	Read only	Byte	01h	
	Output	P4OUT or PBOUT_H	Read/write	Byte	03h	undefined
	Direction	P4DIR or PBDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	P4REN or PBREN_H	Read/write	Byte	07h	00h
	Drive Strength	P4DS or PBDS_H	Read/write	Byte	09h	00h
	Port Select	P4SEL or PBSEL_H	Read/write	Byte	0Bh	00h
Port 5	Input	P5IN or PCIN_L	Read only	Byte	00h	
	Output	P5OUT or PCOUT_L	Read/write	Byte	02h	undefined
	Direction	P5DIR or PCDIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P5REN or PCREN_L	Read/write	Byte	06h	00h
	Drive Strength	P5DS or PCDS_L	Read/write	Byte	08h	00h
	Port Select	P5SEL or PCSEL_L	Read/write	Byte	0Ah	00h
Port 6	Input	P6IN or PCIN_H	Read only	Byte	01h	
	Output	P6OUT or PCOUT_H	Read/write	Byte	03h	undefined
	Direction	P6DIR or PCDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	P6REN or PCREN_H	Read/write	Byte	07h	00h
	Drive Strength	P6DS or PCDS_H	Read/write	Byte	09h	00h
	Port Select	P6SEL or PCSEL_H	Read/write	Byte	0Bh	00h

Table 8-2. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port 7	Input	P7IN or PDIN_L	Read only	Byte	00h	
	Output	P7OUT or PDOUT_L	Read/write	Byte	02h	undefined
	Direction	P7DIR or PDDIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P7REN or PDREN_L	Read/write	Byte	06h	00h
	Drive Strength	P7DS or PDDS_L	Read/write	Byte	08h	00h
	Port Select	P7SEL or PDSEL_L	Read/write	Byte	0Ah	00h
Port 8	Input	P8IN or PDIN_H	Read only	Byte	01h	
	Output	P8OUT or PDOUT_H	Read/write	Byte	03h	undefined
	Direction	P8DIR or PDDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	P8REN or PDREN_H	Read/write	Byte	07h	00h
	Drive Strength	P8DS or PDDS_H	Read/write	Byte	09h	00h
	Port Select	P8SEL or PDSEL_H	Read/write	Byte	0Bh	00h
Port 9	Input	P9IN or PEIN_L	Read only	Byte	00h	
	Output	P9OUT or PEOUT_L	Read/write	Byte	02h	undefined
	Direction	P9DIR or PEDIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P9REN or PEREN_L	Read/write	Byte	06h	00h
	Drive Strength	P9DS or PEDS_L	Read/write	Byte	08h	00h
	Port Select	P9SEL or PESEL_L	Read/write	Byte	0Ah	00h
Port 10	Input	P10IN or PEIN_H	Read only	Byte	01h	
	Output	P10OUT or PEOUT_H	Read/write	Byte	03h	undefined
	Direction	P10DIR or PEDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	P10REN or PEREN_H	Read/write	Byte	07h	00h
	Drive Strength	P10DS or PEDS_H	Read/write	Byte	09h	00h
	Port Select	P10SEL or PESEL_H	Read/write	Byte	0Bh	00h

Table 8-2. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port 11	Input	P11IN or PFIN_L	Read only	Byte	00h	
	Output	P11OUT or PFOUT_L	Read/write	Byte	02h	undefined
	Direction	P11DIR or PFDIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P11REN or PFREN_L	Read/write	Byte	06h	00h
	Drive Strength	P11DS or PFDS_L	Read/write	Byte	08h	00h
	Port Select	P11SEL or PFSEL_L	Read/write	Byte	0Ah	00h
Port A	Input	PAIN	Read only	Word	00h	
		PAIN_L	Read only	Byte	00h	
		PAIN_H	Read only	Byte	01h	
	Output	PAOUT	Read/write	Word	02h	undefined
		PAOUT_L	Read/write	Byte	02h	undefined
		PAOUT_H	Read/write	Byte	03h	undefined
	Direction	PADIR	Read/write	Word	04h	0000h
		PADIR_L	Read/write	Byte	04h	00h
		PADIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PAREN	Read/write	Word	06h	0000h
		PAREN_L	Read/write	Byte	06h	00h
		PAREN_H	Read/write	Byte	07h	00h
	Drive Strength	PADS	Read/write	Word	08h	0000h
		PADS_L	Read/write	Byte	08h	00h
		PADS_H	Read/write	Byte	09h	00h
	Port Select	PASEL	Read/write	Word	0Ah	0000h
		PASEL_L	Read/write	Byte	0Ah	00h
		PASEL_H	Read/write	Byte	0Bh	00h
	Interrupt Edge Select	PAIES	Read/write	Word	18h	undefined
		PAIES_L	Read/write	Byte	18h	undefined
		PAIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	PAIE	Read/write	Word	1Ah	0000h
		PAIE_L	Read/write	Byte	1Ah	00h
		PAIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	PAIFG	Read/write	Word	1Ch	0000h
		PAIFG_L	Read/write	Byte	1Ch	00h
		PAIFG_H	Read/write	Byte	1Dh	00h

Table 8-2. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port B	Input	PBIN	Read only	Word	00h	
		PBIN_L	Read only	Byte	00h	
		PBIN_H	Read only	Byte	01h	
	Output	PBOUT	Read/write	Word	02h	undefined
		PBOUT_L	Read/write	Byte	02h	undefined
		PBOUT_H	Read/write	Byte	03h	undefined
	Direction	PBDIR	Read/write	Word	04h	0000h
		PBDIR_L	Read/write	Byte	04h	00h
		PBDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PBREN	Read/write	Word	06h	0000h
		PBREN_L	Read/write	Byte	06h	00h
		PBREN_H	Read/write	Byte	07h	00h
	Drive Strength	PBDS	Read/write	Word	08h	0000h
		PBDS_L	Read/write	Byte	08h	00h
		PBDS_H	Read/write	Byte	09h	00h
	Port Select	PBSEL	Read/write	Word	0Ah	0000h
		PBSEL_L	Read/write	Byte	0Ah	00h
		PBSEL_H	Read/write	Byte	0Bh	00h
Port C	Input	PCIN	Read only	Word	00h	
		PCIN_L	Read only	Byte	00h	
		PCIN_H	Read only	Byte	01h	
	Output	PCOUT	Read/write	Word	02h	undefined
		PCOUT_L	Read/write	Byte	02h	undefined
		PCOUT_H	Read/write	Byte	03h	undefined
	Direction	PCDIR	Read/write	Word	04h	0000h
		PCDIR_L	Read/write	Byte	04h	00h
		PCDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PCREN	Read/write	Word	06h	0000h
		PCREN_L	Read/write	Byte	06h	00h
		PCREN_H	Read/write	Byte	07h	00h
	Drive Strength	PCDS	Read/write	Word	08h	0000h
		PCDS_L	Read/write	Byte	08h	00h
		PCDS_H	Read/write	Byte	09h	00h
	Port Select	PCSEL	Read/write	Word	0Ah	0000h
		PCSEL_L	Read/write	Byte	0Ah	00h
		PCSEL_H	Read/write	Byte	0Bh	00h

Table 8-2. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port D	Input	PDIN	Read only	Word	00h	
		PDIN_L	Read only	Byte	00h	
		PDIN_H	Read only	Byte	01h	
	Output	PDOUT	Read/write	Word	02h	undefined
		PDOUT_L	Read/write	Byte	02h	undefined
		PDOUT_H	Read/write	Byte	03h	undefined
	Direction	PDDIR	Read/write	Word	04h	0000h
		PDDIR_L	Read/write	Byte	04h	00h
		PDDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PDREN	Read/write	Word	06h	0000h
		PDREN_L	Read/write	Byte	06h	00h
		PDREN_H	Read/write	Byte	07h	00h
	Drive Strength	PDDS	Read/write	Word	08h	0000h
		PDDS_L	Read/write	Byte	08h	00h
		PDDS_H	Read/write	Byte	09h	00h
	Port Select	PDSEL	Read/write	Word	0Ah	0000h
		PDSEL_L	Read/write	Byte	0Ah	00h
		PDSEL_H	Read/write	Byte	0Bh	00h
Port E	Input	PEIN	Read only	Word	00h	
		PEIN_L	Read only	Byte	00h	
		PEIN_H	Read only	Byte	01h	
	Output	PEOUT	Read/write	Word	02h	undefined
		PEOUT_L	Read/write	Byte	02h	undefined
		PEOUT_H	Read/write	Byte	03h	undefined
	Direction	PEDIR	Read/write	Word	04h	0000h
		PEDIR_L	Read/write	Byte	04h	00h
		PEDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PEREN	Read/write	Word	06h	0000h
		PEREN_L	Read/write	Byte	06h	00h
		PEREN_H	Read/write	Byte	07h	00h
	Drive Strength	PEDS	Read/write	Word	08h	0000h
		PEDS_L	Read/write	Byte	08h	00h
		PEDS_H	Read/write	Byte	09h	00h
	Port Select	PESEL	Read/write	Word	0Ah	0000h
		PESEL_L	Read/write	Byte	0Ah	00h
		PESEL_H	Read/write	Byte	0Bh	00h

Table 8-2. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port F	Input	PFIN	Read only	Word	00h	
		PFIN_L	Read only	Byte	00h	
		PFIN_H	Read only	Byte	01h	
	Output	PFOUT	Read/write	Word	02h	undefined
		PFOUT_L	Read/write	Byte	02h	undefined
		PFOUT_H	Read/write	Byte	03h	undefined
	Direction	PFDIR	Read/write	Word	04h	0000h
		PFDIR_L	Read/write	Byte	04h	00h
		PFDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PFREN	Read/write	Word	06h	0000h
		PFREN_L	Read/write	Byte	06h	00h
		PFREN_H	Read/write	Byte	07h	00h
	Drive Strength	PFDS	Read/write	Word	08h	0000h
		PFDS_L	Read/write	Byte	08h	00h
		PFDS_H	Read/write	Byte	09h	00h
	Port Select	PFSEL	Read/write	Word	0Ah	0000h
		PFSEL_L	Read/write	Byte	0Ah	00h
		PFSEL_H	Read/write	Byte	0Bh	00h
Port J	Input	PJIN	Read only	Word	00h	
		PJIN_L	Read only	Byte	00h	
		PJIN_H	Read only	Byte	01h	
	Output	PJOUT	Read/write	Word	02h	undefined
		PJOUT_L	Read/write	Byte	02h	undefined
		PJOUT_H	Read/write	Byte	03h	undefined
	Direction	PJDIR	Read/write	Word	04h	0000h
		PJDIR_L	Read/write	Byte	04h	00h
		PJDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PJREN	Read/write	Word	06h	0000h
		PJREN_L	Read/write	Byte	06h	00h
		PJREN_H	Read/write	Byte	07h	00h
	Drive Strength	PJDS	Read/write	Word	08h	0000h
		PJDS_L	Read/write	Byte	08h	00h
		PJDS_H	Read/write	Byte	09h	00h

Port 1 Interrupt Vector Register (P1IV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	0
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	0	0	**P1IV**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

P1IV Bits 15-0 Port 1 interrupt vector value

P1IV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Port 1.0 interrupt	P1IFG.0	Highest
04h	Port 1.1 interrupt	P1IFG.1	
06h	Port 1.2 interrupt	P1IFG.2	
08h	Port 1.3 interrupt	P1IFG.3	
0Ah	Port 1.4 interrupt	P1IFG.4	
0Ch	Port 1.5 interrupt	P1IFG.5	
0Eh	Port 1.6 interrupt	P1IFG.6	
10h	Port 1.7 interrupt	P1IFG.7	Lowest

Port 2 Interrupt Vector Register (P2IV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	0	0	**P2IV**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

P2IV Bits 15-0 Port 2 interrupt vector value

P2IV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Port 2.0 interrupt	P2IFG.0	Highest
04h	Port 2.1 interrupt	P2IFG.1	
06h	Port 2.2 interrupt	P2IFG.2	
08h	Port 2.3 interrupt	P2IFG.3	
0Ah	Port 2.4 interrupt	P2IFG.4	
0Ch	Port 2.5 interrupt	P2IFG.5	
0Eh	Port 2.6 interrupt	P2IFG.6	
10h	Port 2.7 interrupt	P2IFG.7	Lowest

Port 1 Interrupt Edge Select Register (P1IES)

7	6	5	4	3	2	1	0
P1IES							
rw	rw	rw	rw	rw	rw	rw	rw

P1IES Bits 7-0 Port 1 interrupt edge select

0 P1IFG flag is set with a low-to-high transition.

1 P1IFG flag is set with a high-to-low transition.

Port 1 Interrupt Enable Register (P1IE)

7	6	5	4	3	2	1	0
P1IE							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

P1IE	Bits 7-0	Port 1 interrupt enable 0 Corresponding port interrupt disabled 1 Corresponding port interrupt enabled

Port 1 Interrupt Flag Register (P1IFG)

7	6	5	4	3	2	1	0
P1IFG							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

P1IFG	Bits 7-0	Port 1 interrupt flag 0 No interrupt is pending. 1 Interrupt is pending.

Port 2 Interrupt Edge Select Register (P2IES)

7	6	5	4	3	2	1	0
P2IES							
rw	rw	rw	rw	rw	rw	rw	rw

P2IES	Bits 7-0	Port 2 interrupt edge select 0 P2IFG flag is set with a low-to-high transition. 1 P2IFG flag is set with a high-to-low transition.

Port 2 Interrupt Enable Register (P2IE)

7	6	5	4	3	2	1	0
P2IE							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

P2IE	Bits 7-0	Port 2 interrupt enable 0 Corresponding port interrupt disabled 1 Corresponding port interrupt enabled

Port 2 Interrupt Flag Register (P2IFG)

7	6	5	4	3	2	1	0
P2IFG							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

P2IFG	Bits 7-0	Port 2 interrupt flag 0 No interrupt is pending. 1 Interrupt is pending.

Port x Input Register (PxIN)

7	6	5	4	3	2	1	0
PxIN							
r	r	r	r	r	r	r	r

PxIN	Bits 7-0	Port x input. Read only.

Port x Output Register (PxOUT)

7	6	5	4	3	2	1	0
PxOUT							
rw	rw	rw	rw	rw	rw	rw	rw

PxOUT Bits 7-0 Port x output
When I/O configured to output mode:
- 0 Output is low.
- 1 Output is high.

When I/O configured to input mode and pullups/pulldowns enabled:
- 0 pulldown selected
- 1 pullup selected

Port x Direction Register (PxDIR)

7	6	5	4	3	2	1	0
PxDIR							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PxDIR Bits 7-0 Port x direction
- 0 Port configured as input
- 1 Port configured as output

Port x Drive Strength Register (PxDS)

7	6	5	4	3	2	1	0
PxDS							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PxDS Bits 7-0 Port x drive strength
- 0 Reduced output drive strength
- 1 Full output drive strength

Port Mapping Controller

The port mapping controller allows a flexible mapping of digital functions to port pins. This chapter describes the port mapping controller.

Topic **Page**

9.1 Port Mapping Controller Introduction 322
9.2 Port Mapping Controller Operation 322
9.3 Port Mapping Controller Registers 323

9.1 Port Mapping Controller Introduction

The port mapping controller allows the flexible and reconfigurable mapping of digital functions to port pins.

The port mapping controller features are:

- Configuration protected by write access key.
- Default mapping provided for each port pin (device-dependent, the device pinout in the device-specific data sheet).
- Mapping can be reconfigured during runtime.
- Each output signal can be mapped to several output pins.

9.2 Port Mapping Controller Operation

The port mapping is configured with user software. The setup is discussed in the following sections.

9.2.1 Access

To enable write access to any of the port mapping controller registers, the correct key must be written into the PMAPKEYID register. The PMAPKEYID register always reads 096A5h. Writing the key 02D52h grants write access to all port mapping controller registers. Read access is always possible.

If an invalid key is written while write access is granted, any further write accesses are prevented. It is recommended that the application completes mapping configuration by writing an invalid key.

There is a timeout counter implemented that is incremented with each (assembler) instruction, and when it counts to 32, the write access is locked again. Any access to the port mapping controller registers resets the counter. Interrupts should be disabled during the configuration process or the application should take precautions that the execution of an interrupt service routine does not accidentally cause a permanent lock of the port mapping registers; e.g., by using the reconfiguration capability (see Section 9.2.2).

The access status is reflected in the PMAPLOCK bit.

By default, the port mapping controller allows only one configuration after PUC. A second attempt to enable write access by writing the correct key is ignored, and the registers remain locked. A PUC is required to disable the permanent lock again. If it is necessary to reconfigure the mapping during runtime, the PMAPRECFG bit must be set during the first write access timeslot. If PMAPRECFG is cleared during later configuration sessions, no more configuration sessions are possible.

9.2.2 Mapping

For each port pin, Px.y, on ports providing the mapping functionality, a mapping register, PxMAPy, is available. Setting this register to a certain value maps a module's input and output signals to the respective port pin Px.y. The port pin itself is switched from a general purpose I/O to the selected peripheral/secondary function by setting the corresponding PxSEL.y bit to 1. If the input or the output function of the module is used, it is typically defined by the setting the PxDIR.y bit. If PxDIR.y = 0, the pin is an input, if PxDIR.y = 1, the pin is an output. There are also peripherals (e.g., the USCI module) that control the direction or even other functions of the pin (e.g., open drain), and these options are documented in the mapping table.

With the port mapping functionality the output of a module can be mapped to multiple pins. Also the input of a module can receive inputs from multiple pins. When mapping multiple inputs onto one function care needs to be taken because the input signals are logically ORed together without appling any priority - a logic one on any of the inputs will result in a logic one at the module. If the PxSEL.y bit is 0 the corresponding input signal is a logic zero.

The mapping is device-dependent; see the device-specific data sheet for available functions and specific values. The use of mapping-mnemonics to abstract the underlying PxMAPy values is recommended to allow simple portability between different devices. Table 9-1 shows some examples for mapping mnemonics of some common peripherals.

All mappable port pins provide the function PM_ANALOG (0FFh). Setting the port mapping register PxMAPy to PM_ANALOG together with PxSEL.y = 1 disables the output driver and the input Schmitt-trigger, to prevent parasitic cross currents when applying analog signals.

Table 9-1. Examples for Port Mapping Mnemonics and Functions

PxMAPy Mnemonic	Input Pin Function With PxSEL.y = 1 and PxDIR.y = 0	Output Pin Function With PxSEL.y = 1 and PxDIR.y = 1
PM_NONE	None	DVSS
PM_ACLK	None	ACLK
PM_MCLK	None	MCLK
PM_SMCLK	None	SMCLK
PM_TA0CLK	Timer_A0 clock input	DVSS
PM_TA0CCR0A	Timer_A0 CCR0 capture input CCI0A	TA0 CCR0 compare output Out0
PM_TA0CCR1A	Timer_A0 CCR1 capture input CCI1A	TA0 CCR1 compare output Out1
PM_TA0CCR2A	Timer_A0 CCR2 capture input CCI2A	TA0 CCR2 compare output Out2
PM_TA0CCR3A	Timer_A0 CCR3 capture input CCI3A	TA0 CCR3 compare output Out3
PM_TA0CCR4A	Timer_A0 CCR4 capture input CCI4A	TA0 CCR4 compare output Out4
PM_TA1CLK	Timer_A1 clock input	DVSS
PM_TA1CCR0A	Timer_A1 CCR0 capture input CCI0A	TA1 CCR0 compare output Out0
PM_TA1CCR1A	Timer_A1 CCR1 capture input CCI1A	TA1 CCR1 compare output Out1
PM_TA1CCR2A	Timer_A1 CCR2 capture input CCI2A	TA1 CCR2 compare output Out2
PM_TBCLK	Timer_B clock input	DVSS
PM_TBOUTH	Timer_B outputs high impedance	DVSS
PM_TBCCR0A	Timer_B CCR0 capture input CCI0A	TB CCR0 compare output Out0 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR1A	Timer_B CCR1 capture input CCI1A	TB CCR1 compare output Out1 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR2A	Timer_B CCR2 capture input CCI2A	TB CCR2 compare output Out2 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR3A	Timer_B CCR3 capture input CCI3A	TB CCR3 compare output Out3 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR4A	Timer_B CCR4 capture input CCI4A	TB CCR4 compare output Out4 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR5A	Timer_B CCR5 capture input CCI3A	TB CCR5 compare output Out5 [direction controlled by Timer_B (TBOUTH)]
PM_TBCCR6A	Timer_B CCR6 capture input CCI4A	TB CCR6 compare output Out6 [direction controlled by Timer_B (TBOUTH)]
PM_UCA0RXD	USCI_A0 UART RXD (direction controlled by USCI - input)	
PM_UCA0SOMI	USCI_A0 SPI slave out master in (direction controlled by USCI)	
PM_UCA0TXD	USCI_A0 UART TXD (direction controlled by USCI - output)	
PM_UCA0SIMO	USCI_A0 SPI slave in master out (direction controlled by USCI)	
PM_UCA0CLK	USCI_A0 clock input/output (direction controlled by USCI)	
PM_UCA0STE	USCI_A0 SPI slave transmit enable (direction controlled by USCI)	
PM_UCB0SOMI	USCI_B0 SPI slave out master in (direction controlled by USCI)	
PM_UCB0SCL	USCI_B0 I2C clock (open drain and direction controlled by USCI	
PM_UCB0SIMO	USCI_B0 SPI slave in master out (direction controlled by USCI)	
PM_UCB0SDA	USCI_B0 I2C data (open drain and direction controlled by USCI)	
PM_UCB0CLK	USCI_B0 clock input/output (direction controlled by USCI)	
PM_UCB0STE	USCI_B0 SPI slave transmit enable (direction controlled by USCI)	
PM_ANALOG	Disables the output driver and the input Schmitt-trigger to prevent parasitic cross currents when applying analog signals	

9.3 Port Mapping Controller Registers

The control register for the port mapping controller are listed in Table 9-2. The mapping registers are listed in Table 9-3. The mapping registers can also be accessed as words, as shown in Table 9-4.

Table 9-2. Port Mapping Control Registers

Register	Short Form	Register Type	Address Offset	Initial State
Port mapping key register	PMAPKEYID	Read/write	000h	Reset with PUC
Port mapping control register	PMAPCTL	Read/write	002h	Reset with PUC

Table 9-3. Port Mapping Registers for Port Px – Byte Access

Register	Short Form	Register Type	Address Offset	Initial State
Port Px.0 mapping register	PxMAP0	Read/write	000h	Device dependent
Port Px.1 mapping register	PxMAP1	Read/write	001h	Device dependent
Port Px.2 mapping register	PxMAP2	Read/write	002h	Device dependent
Port Px.3 mapping register	PxMAP3	Read/write	003h	Device dependent
Port Px.4 mapping register	PxMAP4	Read/write	004h	Device dependent
Port Px.5 mapping register	PxMAP5	Read/write	005h	Device dependent
Port Px.6 mapping register	PxMAP6	Read/write	006h	Device dependent
Port Px.7 mapping register	PxMAP7	Read/write	007h	Device dependent

Table 9-4. Port Mapping Registers for Port Px – Word Access

Register	Short Form	Register Type	Address Offset	Initial State
Port Px.0/Port Px.1 mapping register	PxMAP01	Read/write	000h	Device dependent
Port Px.2/Port Px.3 mapping register	PxMAP23	Read/write	002h	Device dependent
Port Px.4/Port Px.5 mapping register	PxMAP45	Read/write	004h	Device dependent
Port Px.6/Port Px.7 mapping register	PxMAP67	Read/write	006h	Device dependent

PMAPKEYID, Port Mapping Key Register

15	14	13	12	11	10	9	8
PMAPKEYx, read as 096A5h, must be written as 02D52h							

7	6	5	4	3	2	1	0
PMAPKEYx, read as 096A5h, must be written as 02D52h							

PMAPKEYx	Bits 15-0	Port write access key Always reads 096A5h. Must be written 02D52h for write access to the port mapping registers.

PMAPCTL, Port Mapping Control Register

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved						**PMAPRECFG**	**PMAPLOCKED**
r0	r0	r0	r0	r0	r0	rw-0	r-1

Reserved	Bits 15-2	Reserved
PMAPRECFG	Bit 1	Port mapping reconfiguration control bit 0 Configuration allowed only once 1 Allow reconfiguration of port mapping
PMAPLOCKED	Bit 0	Port mapping lock bit. Read only 0 Access to mapping registers is granted 1 Access to mapping registers is locked

PxMAPy, Port Px.y Mapping Register

7	6	5	4	3	2	1	0
PMAPx							
rw-0 (1)	rw-0 (1)	rw-0 (1)	rw-0 (1)	rw-0 (1)	rw-0 (1)	rw-0 (1)	rw-0 (1)

PMAPx	Bits 7-0	Selects secondary port function. Settings are device-dependent; see the device-specific data sheet.

(1) If not all bits are required to decode all provided functions, the unused bits are r0.

CRC Module

The cyclic redundancy check (CRC) module provides a signature for a given data sequence. This chapter describes the operation and use of the CRC module.

NOTE: The CRC module on the MSP430F543x and MSP430F541x non-A versions does not support the bit-wise reverse feature described in this module description. Registers CRCDIRB and CRCRESR, along with their respective functionality, are not available.

Topic **Page**

10.1 Cyclic Redundancy Check (CRC) Module Introduction **328**
10.2 CRC Checksum Generation **329**
10.3 CRC Module Registers **332**

10.1 Cyclic Redundancy Check (CRC) Module Introduction

The CRC module produces a signature for a given sequence of data values. The signature is generated through a feedback path from data bits 0, 4, 11, and 15 (see Figure 10-1). The CRC signature is based on the polynomial given in the CRC-CCITT-BR polynomial (see Equation 10) .

$$f(x) = x^{16} + x^{12} + x^{5} +1 \tag{10}$$

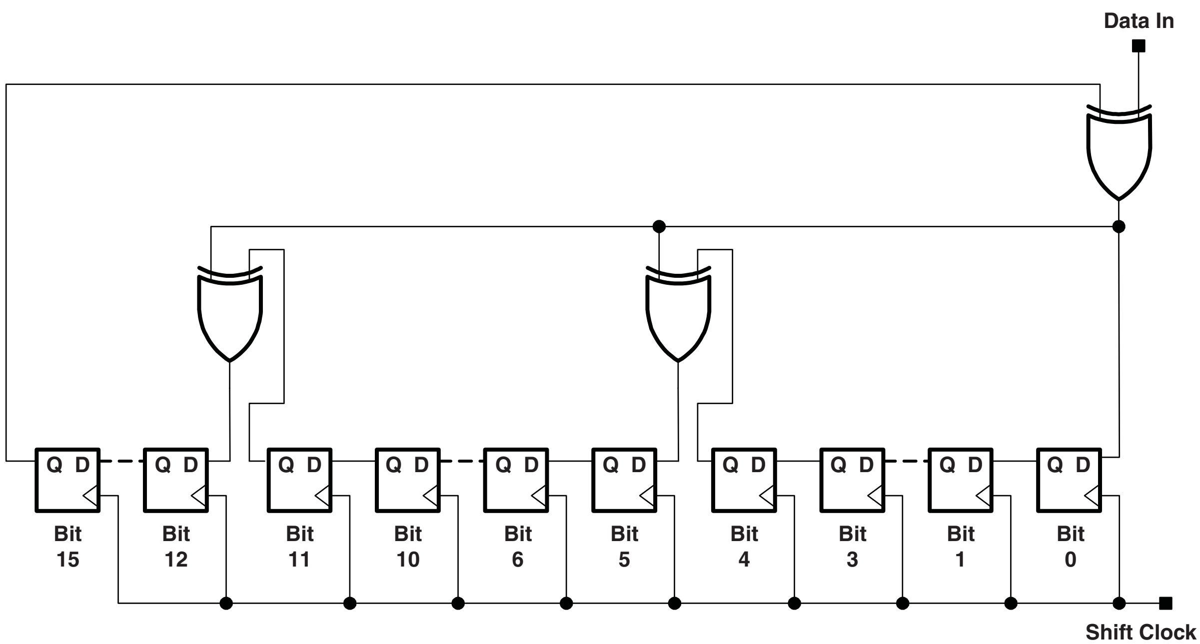


Figure 10-1. LFSR Implementation of CRC-CCITT Standard, Bit 0 is the MSB of the Result

Identical input data sequences result in identical signatures when the CRC is initialized with a fixed seed value, whereas different sequences of input data, in general, result in different signatures.

10.2 CRC Checksum Generation

The CRC generator is first initialized by writing a 16-bit word (seed) to the CRC Initialization and Result (CRCINIRES) register. Any data that should be included into the CRC calculation must be written to the CRC Data Input (CRCDI or CRCDIRB) register in the same order that the original CRC signature was calculated. The actual signature can be read from the CRCINIRES register to compare the computed checksum with the expected checksum.

Signature generation describes a method on how the result of a signature operation can be calculated. The calculated signature, which is computed by an external tool, is called checksum in the following text. The checksum is stored in the product's memory and is used to check the correctness of the CRC operation result.

10.2.1 CRC Implementation

To allow parallel processing of the CRC, the linear feedback shift register (LFSR) functionality is implemented with an XOR tree. This implementation shows the identical behavior as the LFSR approach after 8 bits of data are shifted in when the LSB is 'shifted' in first. The generation of a signature calculation has to be started by writing a seed to the CRCINIRES register to initialize the register. Software or hardware (e.g., DMA) can transfer data to the CRCDI or CRCDIRB register (e.g., from memory). The value in CRCDI or CRCDIRB is then included into the signature, and the result is available in the signature result registers at the next read access (CRCINIRES and CRCRESR). The signature can be generated using word or byte data.

If a word data is processed, the lower byte at the even address is used at the first clock (MCLK) cycle. During the second clock cycle, the higher byte is processed. Thus, it takes two clock cycles to process word data, while it takes only one clock (MCLK) cycle to process byte data.

Data bytes written to CRCDIRB in word mode or the data byte in byte mode are bit-wise reversed before the CRC engine adds them to the signature. The bits among each byte are reversed. Data bytes written to CRCDI in word mode or the data byte in byte mode are not bit reversed before use by the CRC engine.

If the Check Sum itself (with reversed bit order) is included into the CRC operation (as data written to CRCDI or CRCDIRB), the result in the CRCINIRES and CRCRESR registers must be zero.

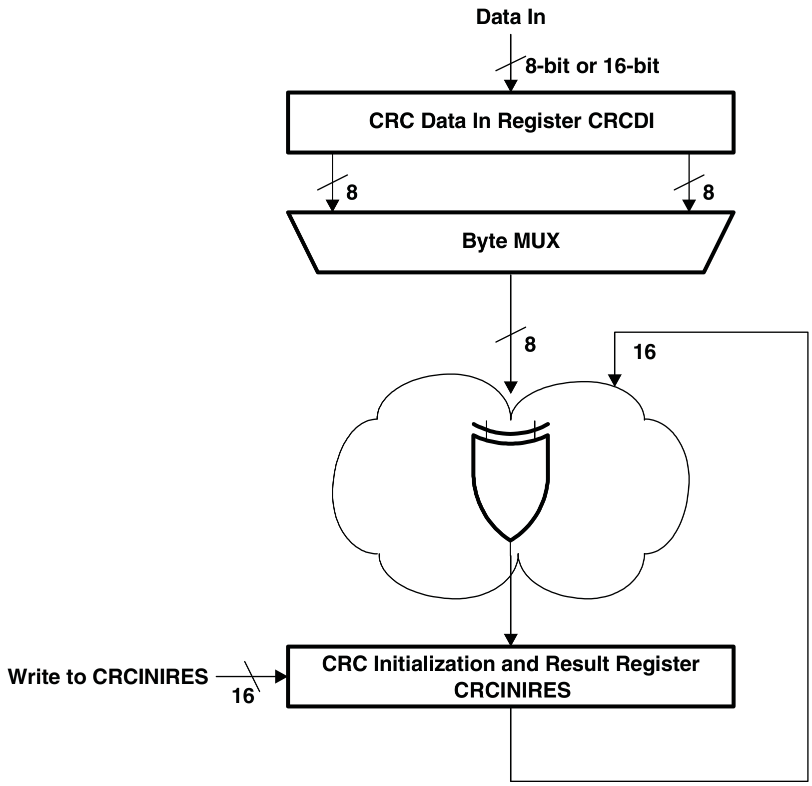


Figure 10-2. Implementation of CRC-CCITT using the CRCDI and CRCINIRES registers

10.2.2 Assembler Examples

10.2.2.1 General Assembler Example

This example demonstrates the operation of the on-chip CRC:

```
   ...
   PUSH   R4                   ; Save registers
   PUSH   R5
   MOV    #StartAddress,R4     ; StartAddress < EndAddress
   MOV    #EndAddress,R5
   MOV    &INIT, &CRCINIRES    ; INIT to CRCINIRES
L1 MOV    @R4+,&CRCDI          ; Item to Data In register
   CMP    R5,R4                ; End address reached?
   JLO    L1                   ; No
   MOV    &Check_Sum,&CRCDI    ; Yes, Include checksum
   TST    &CRCINIRES           ; Result = 0?
   JNZ    CRC_ERROR            ; No, CRCRES <> 0: error
   ...                         ; Yes, CRCRES=0:
                               ; information ok.
   POP    R5                   ; Restore registers
   POP    R4
```

10.2.2.2 Reference Data Sequence

The details of the implemented CRC algorithm is shown by the following data sequences using word or byte accesses and the CRC data-in as well as the CRC data-in reverse byte registers:

```
   ...
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.b   #00031h,&CRCDI_L    ; "1"
mov.b   #00032h,&CRCDI_L    ; "2"
mov.b   #00033h,&CRCDI_L    ; "3"
mov.b   #00034h,&CRCDI_L    ; "4"
mov.b   #00035h,&CRCDI_L    ; "5"
mov.b   #00036h,&CRCDI_L    ; "6"
mov.b   #00037h,&CRCDI_L    ; "7"
mov.b   #00038h,&CRCDI_L    ; "8"
mov.b   #00039h,&CRCDI_L    ; "9"

cmp     #089F6h,&CRCINIRES  ; compare result
                            ; CRCRESR contains 06F91h
jeq     &Success            ; no error
br      &Error              ; to error handler

mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.w   #03231h,&CRCDI      ; "1" & "2"
mov.w   #03433h,&CRCDI      ; "3" & "4"
mov.w   #03635h,&CRCDI      ; "5" & "6"
mov.w   #03837h,&CRCDI      ; "7" & "8"
mov.b   #039h,   &CRCDI_L   ; "9"

cmp     #089F6h,&CRCINIRES  ; compare result
                               ; CRCRESR contains 06F91h
jeq     &Success            ; no error
br      &Error              ; to error handler
   ...
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.b   #00031h,&CRCDIRB_L  ; "1"
mov.b   #00032h,&CRCDIRB_L  ; "2"
mov.b   #00033h,&CRCDIRB_L  ; "3"
mov.b   #00034h,&CRCDIRB_L  ; "4"
mov.b   #00035h,&CRCDIRB_L  ; "5"
mov.b   #00036h,&CRCDIRB_L  ; "6"
mov.b   #00037h,&CRCDIRB_L  ; "7"
mov.b   #00038h,&CRCDIRB_L  ; "8"
mov.b   #00039h,&CRCDIRB_L  ; "9"

cmp     #029B1h,&CRCINIRES  ; compare result
                            ; CRCRESR contains 08D94h
jeq     &Success            ; no error
br      &Error              ; to error handler
...
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.w   #03231h,&CRCDIRB    ; "1" & "2"
mov.w   #03433h,&CRCDIRB    ; "3" & "4"
mov.w   #03635h,&CRCDIRB    ; "5" & "6"
mov.w   #03837h,&CRCDIRB    ; "7" & "8"
mov.b   #039h,   &CRCDIRB_L ; "9"

cmp     #029B1h,&CRCINIRES  ; compare result
                            ; CRCRESR contains 08D94h
jeq     &Success            ; no error
br      &Error              ; to error handler
```

10.3 CRC Module Registers

The CRC module registers are listed in Table 10-1. The base address can be found in the device-specific data sheet. The address offset is given in Table 10-1.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 10-1. CRC Module Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
CRC Data In	CRCDI	Read/write	Word	0000h	0000h
	CRCDI_L	Read/write	Byte	0000h	00h
	CRCDI_H	Read/write	Byte	0001h	00h
CRC Data In Reverse Byte[(1)]	CRCDIRB	Read/write	Word	0002h	0000h
	CRCDIRB_L	Read/write	Byte	0002h	00h
	CRCDIRB_H	Read/write	Byte	0003h	00h
CRC Initialization and Result	CRCINIRES	Read/write	Word	0004h	FFFFh
	CRCINIRES_L	Read/write	Byte	0004h	FFh
	CRCINIRES_H	Read/write	Byte	0005h	FFh
CRC Result Reverse[(1)]	CRCRESR	Read only	Word	0006h	FFFFh
	CRCRESR_L	Read/write	Byte	0006h	FFh
	CRCRESR_H	Read/write	Byte	0007h	FFh

(1) Not available on MSP430F543x and MSP430F541x non-A versions.

CRC Data In Register (CRCDI)

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

CRCDI	Bits 15-0	CRC data in. Data written to the CRCDI register is included to the present signature in the CRCINIRES register according to the CRC-CCITT standard.

CRC Data In Reverse Register (CRCDIRB)

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

CRCDIRB	Bits 15-0	CRC data in reverse byte. Data written to the CRCDIRB register is included to the present signature in the CRCINIRES and CRCRESR registers according to the CRC-CCITT standard. Reading the register returns the register CRCDI content.

CRC Initialization and Result Register (CRCINIRES)

15	14	13	12	11	10	9	8
CRCINIRES							
rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1

7	6	5	4	3	2	1	0
CRCINIRES							
rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1

CRCINIRES	Bits 15-0	CRC initialization and result. This register holds the current CRC result (according to the CRC-CCITT standard). Writing to this register initializes the CRC calculation with the value written to it. The value just written can be read from CRCINIRES register.

CRC Reverse Result Register (CRCRESR)

15	14	13	12	11	10	9	8
CRCRESR							
r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1

7	6	5	4	3	2	1	0
CRCRES R							
r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1

CRCRESR	Bits 15-0	CRC reverse result. This register holds the current CRC result (according to the CRC-CCITT standard). The order of bits is reverse (e.g., CRCINIRES[15] = CRCRESR[0]) to the order of bits in the CRCINIRES register (see example code).

Watchdog Timer (WDT_A)

The watchdog timer is a 32-bit timer that can be used as a watchdog or as an interval timer. This chapter describes the watchdog timer. The enhanced watchdog timer, WDT_A, is implemented in all devices.

Topic **Page**

11.1 WDT_A Introduction **336**
11.2 WDT_A Operation **338**
11.3 WDT_A Registers **340**

11.1 WDT_A Introduction

The primary function of the watchdog timer (WDT_A) module is to perform a controlled system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can be configured as an interval timer and can generate interrupts at selected time intervals.

Features of the watchdog timer module include:

- Eight software-selectable time intervals
- Watchdog mode
- Interval mode
- Password-protected access to Watchdog Timer Control (WDTCTL) register
- Selectable clock source
- Can be stopped to conserve power
- Clock fail-safe feature

The watchdog timer block diagram is shown in Figure 11-1.

NOTE: Watchdog timer powers up active.

After a PUC, the WDT_A module is automatically configured in the watchdog mode with an initial ~32-ms reset interval using the SMCLK. The user must setup or halt the WDT_A prior to the expiration of the initial reset interval.

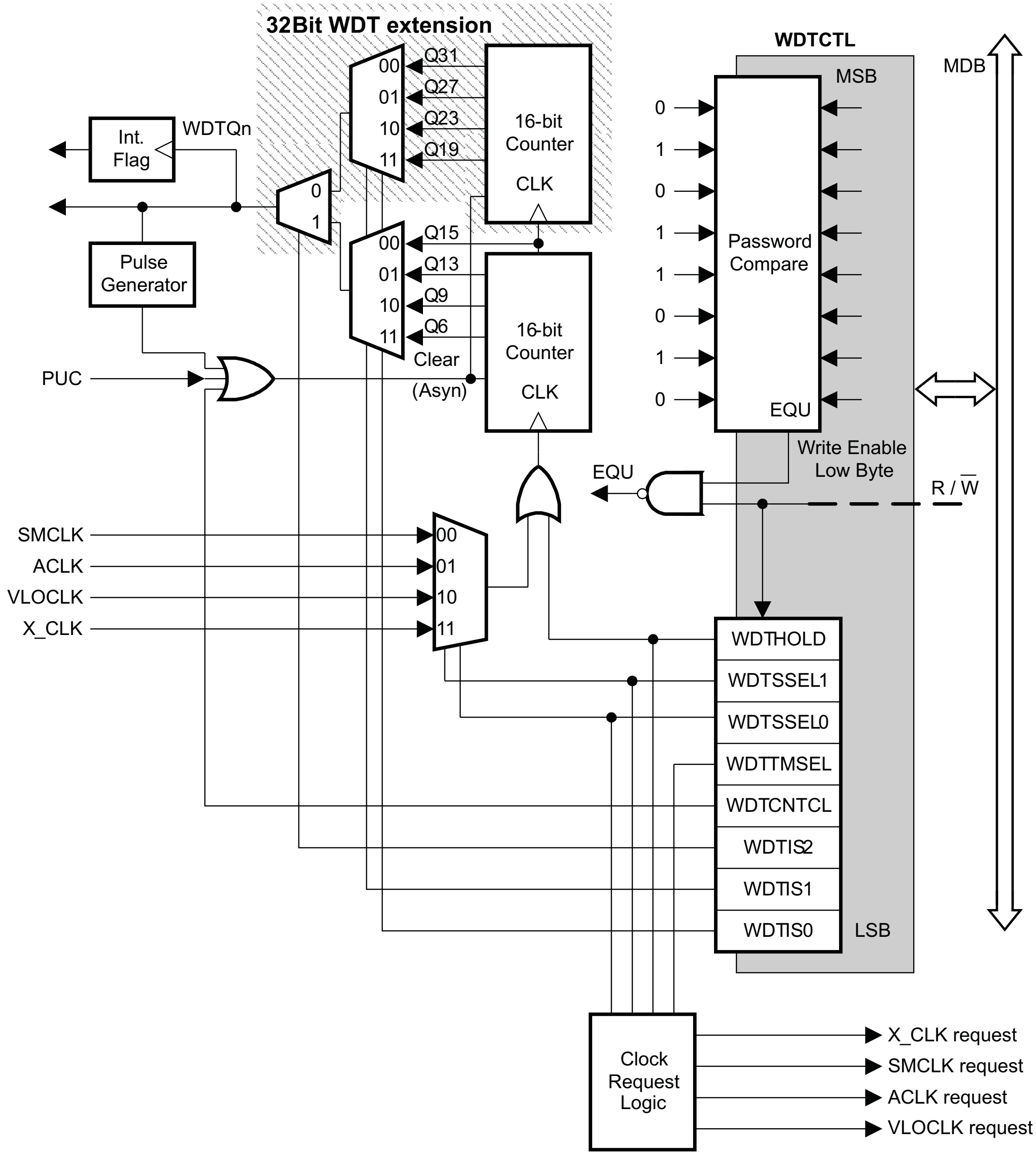


Figure 11-1. Watchdog Timer Block Diagram

11.2 WDT_A Operation

The watchdog timer module can be configured as either a watchdog or interval timer with the WDTCTL register. WDTCTL is a 16-bit password-protected read/write register. Any read or write access must use word instructions and write accesses must include the write password 05Ah in the upper byte. Any write to WDTCTL with any value other than 05Ah in the upper byte is a password violation and triggers a PUC system reset, regardless of timer mode. Any read of WDTCTL reads 069h in the upper byte. Byte reads on WDTCTL high or low part result in the value of the low byte. Writing byte wide to upper or lower parts of WDTCTL results in a PUC.

11.2.1 Watchdog Timer Counter (WDTCNT)

The WDTCNT is a 32-bit up counter that is not directly accessible by software. The WDTCNT is controlled and its time intervals are selected through the Watchdog Timer Control (WDTCTL) register. The WDTCNT can be sourced from SMCLK, ACLK, VLOCLK, and X_CLK on some devices. The clock source is selected with the WDTSSEL bits. The timer interval is selected with the WDTIS bits.

11.2.2 Watchdog Mode

After a PUC condition, the WDT module is configured in the watchdog mode with an initial ~32-ms reset interval using the SMCLK. The user must setup, halt, or clear the watchdog timer prior to the expiration of the initial reset interval or another PUC is generated. When the watchdog timer is configured to operate in watchdog mode, either writing to WDTCTL with an incorrect password, or expiration of the selected time interval triggers a PUC. A PUC resets the watchdog timer to its default condition.

11.2.3 Interval Timer Mode

Setting the WDTTMSEL bit to 1 selects the interval timer mode. This mode can be used to provide periodic interrupts. In interval timer mode, the WDTIFG flag is set at the expiration of the selected time interval. A PUC is not generated in interval timer mode at expiration of the selected timer interval, and the WDTIFG enable bit WDTIE remains unchanged

When the WDTIE bit and the GIE bit are set, the WDTIFG flag requests an interrupt. The WDTIFG interrupt flag is automatically reset when its interrupt request is serviced, or may be reset by software. The interrupt vector address in interval timer mode is different from that in watchdog mode.

NOTE: Modifying the watchdog timer

The watchdog timer interval should be changed together with WDTCNTCL = 1 in a single instruction to avoid an unexpected immediate PUC or interrupt. The watchdog timer should be halted before changing the clock source to avoid a possible incorrect interval.

11.2.4 Watchdog Timer Interrupts

The watchdog timer uses two bits in the SFRs for interrupt control:

- WDT interrupt flag, WDTIFG, located in SFRIFG1.0
- WDT interrupt enable, WDTIE, located in SFRIE1.0

When using the watchdog timer in the watchdog mode, the WDTIFG flag sources a reset vector interrupt. The WDTIFG can be used by the reset interrupt service routine to determine if the watchdog caused the device to reset. If the flag is set, the watchdog timer initiated the reset condition, either by timing out or by a password violation. If WDTIFG is cleared, the reset was caused by a different source.

When using the watchdog timer in interval timer mode, the WDTIFG flag is set after the selected time interval and requests a watchdog timer interval timer interrupt if the WDTIE and the GIE bits are set. The interval timer interrupt vector is different from the reset vector used in watchdog mode. In interval timer mode, the WDTIFG flag is reset automatically when the interrupt is serviced, or can be reset with software.

11.2.5 Clock Fail-Safe Feature

The WDT_A provides a fail-safe clocking feature, ensuring the clock to the WDT_A cannot be disabled while in watchdog mode. This means the low-power modes may be affected by the choice for the WDT_A clock.

If SMCLK or ACLK fails as the WDT_A clock source, VLOCLK is automatically selected as the WDT_A clock source.

When the WDT_A module is used in interval timer mode, there is no fail-safe feature within WDT_A for the clock source.

11.2.6 Operation in Low-Power Modes

The devices have several low-power modes. Different clock signals are available in different low-power modes. The requirements of the application and the type of clocking that is used determine how the WDT_A should be configured. For example, the WDT_A should not be configured in watchdog mode with a clock source that is originally sourced from DCO, XT1 in high-frequency mode, or XT2 via SMCLK or ACLK if the user wants to use low-power mode 3. In this case, SMCLK or ACLK would remain enabled, increasing the current consumption of LPM3. When the watchdog timer is not required, the WDTHOLD bit can be used to hold the WDTCNT, reducing power consumption.

11.2.7 Software Examples

Any write operation to WDTCTL must be a word operation with 05Ah (WDTPW) in the upper byte:

```
; Periodically clear an active watchdog
MOV #WDTPW+WDTIS2+WDTIS1+WDTCNTCL,&WDTCTL
;
; Change watchdog timer interval
MOV #WDTPW+WDTCNTCL+SSEL,&WDTCTL
;
; Stop the watchdog
MOV #WDTPW+WDTHOLD,&WDTCTL
;
; Change WDT to interval timer mode, clock/8192 interval
MOV #WDTPW+WDTCNTCL+WDTTMSEL+WDTIS2+WDTIS0,&WDTCTL
```

11.3 WDT_A Registers

The watchdog timer module registers are listed in Table 11-1. The base register or the watchdog timer module registers and special function registers (SFRs) can be found in device-specific data sheets. The address offset is given in Table 11-1.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 11-1. Watchdog Timer Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Watchdog Timer Control	WDTCTL	Read/write	Word	0Ch	6904h
	WDTCTL_L	Read/write	Byte	0Ch	04h
	WDTCTL_H	Read/write	Byte	0Dh	69h

Watchdog Timer Control Register (WDTCTL)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Read as 069h WDTPW, must be written as 05Ah							

7	6	5	4	3	2	1	0
WDTHOLD	WDTSSEL		WDTTMSEL	WDTCNTCL	WDTIS		
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-1	rw-0	rw-0

WDTPW	Bits 15-8	Watchdog timer password. Always read as 069h. Must be written as 05Ah, or a PUC is generated.
WDTHOLD	Bit 7	Watchdog timer hold. This bit stops the watchdog timer. Setting WDTHOLD = 1 when the WDT is not in use conserves power. 0 Watchdog timer is not stopped. 1 Watchdog timer is stopped.
WDTSSEL	Bits 6-5	Watchdog timer clock source select 00 SMCLK 01 ACLK 10 VLOCLK 11 X_CLK , same as VLOCLK if not defined differently in data sheet
WDTTMSEL	Bit 4	Watchdog timer mode select 0 Watchdog mode 1 Interval timer mode
WDTCNTCL	Bit 3	Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset. 0 No action 1 WDTCNT = 0000h
WDTIS	Bits 2-0	Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag and/or generate a PUC. 000 Watchdog clock source /2G (18:12:16 at 32 kHz) 001 Watchdog clock source /128M (01:08:16 at 32 kHz 010 Watchdog clock source /8192k (00:04:16 at 32 kHz) 011 Watchdog clock source /512k (00:00:16 at 32 kHz) 100 Watchdog clock source /32k (1 s at 32 kHz) 101 Watchdog clock source /8192 (250 ms at 32 kHz) 110 Watchdog clock source /512 (15,6 ms at 32 kHz) 111 Watchdog clock source /64 (1.95 ms at 32 kHz)

Chapter 12

SLAU208F–June 2008–Revised March 2010

Timer_A

Timer_A is a 16-bit timer/counter with multiple capture/compare registers. There can be multiple Timer_A modules on a given device (see the device-specific data sheet). This chapter describes the operation and use of the Timer_A module.

Topic **Page**

12.1 Timer_A Introduction **342**
12.2 Timer_A Operation **343**
12.3 Timer_A Registers **356**

12.1 Timer_A Introduction

Timer_A is a 16-bit timer/counter with up to seven capture/compare registers. Timer_A can support multiple capture/compares, PWM outputs, and interval timing. Timer_A also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Timer_A features include:

- Asynchronous 16-bit timer/counter with four operating modes
- Selectable and configurable clock source
- Up to seven configurable capture/compare registers
- Configurable outputs with pulse width modulation (PWM) capability
- Asynchronous input and output latching
- Interrupt vector register for fast decoding of all Timer_A interrupts

The block diagram of Timer_A is shown in Figure 12-1.

NOTE: Use of the word *count*

Count is used throughout this chapter. It means the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, an associated action does not take place.

NOTE: Nomenclature

There may be multiple instantiations of Timer_A on a given device. The prefix TAx is used, where x is a greater than equal to zero indicating the Timer_A instantiation. For devices with one instantiation, x = 0. The suffix n, where n = 0 to 6, represents the specific capture/compare registers associated with the Timer_A instantiation.

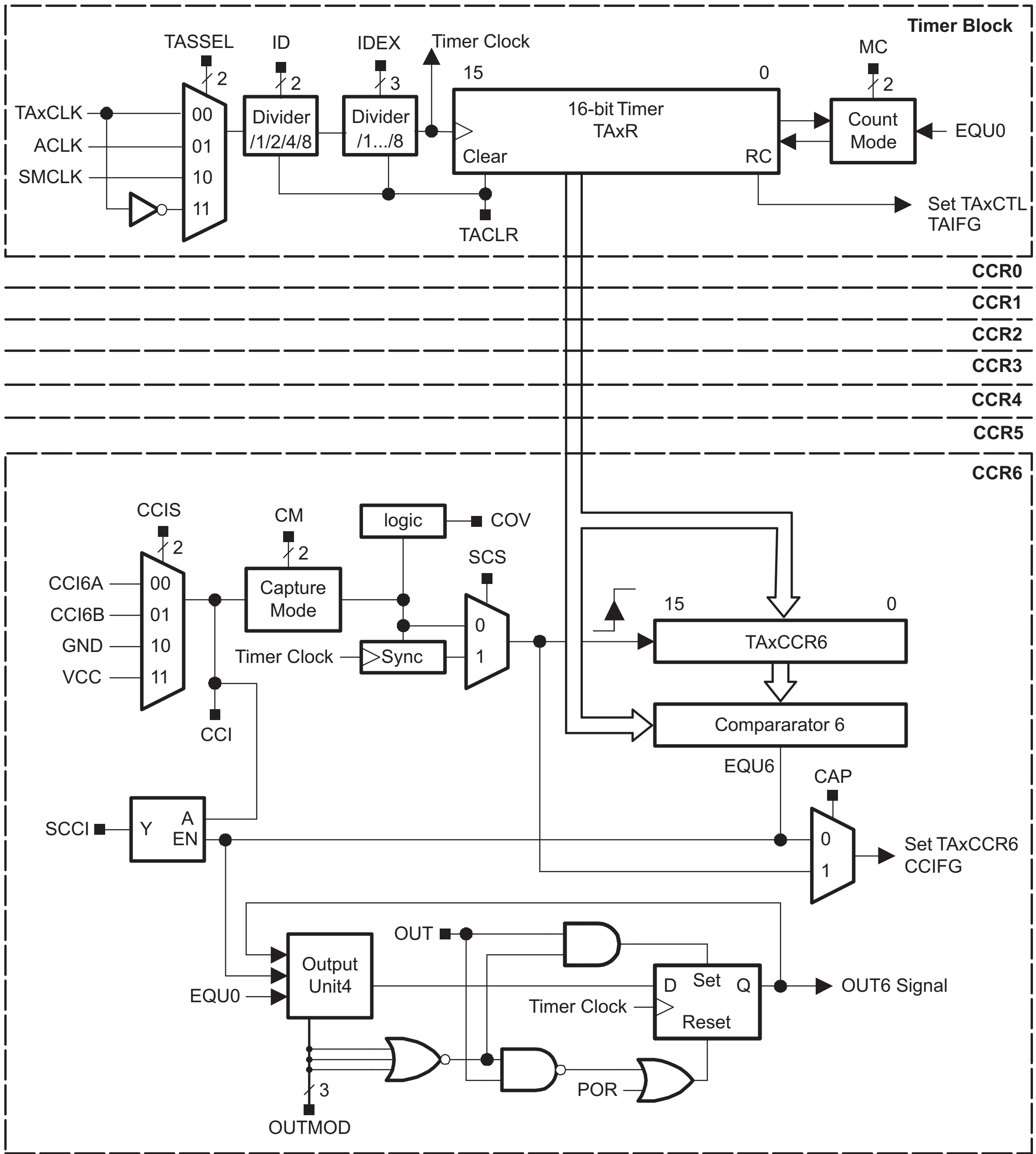


Figure 12-1. Timer_A Block Diagram

12.2 Timer_A Operation

The Timer_A module is configured with user software. The setup and operation of Timer_A are discussed in the following sections.

12.2.1 16-Bit Timer Counter

The 16-bit timer/counter register, TAxR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TAxR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.

TAxR may be cleared by setting the TACLR bit. Setting TACLR also clears the clock divider and count direction for up/down mode.

NOTE: Modifying Timer_A registers

It is recommended to stop the timer before modifying its operation (with exception of the interrupt enable, interrupt flag, and TACLR) to avoid errant operating conditions.

When the timer clock is asynchronous to the CPU clock, any read from TAxR should occur while the timer is not operating or the results may be unpredictable. Alternatively, the timer may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to TAxR takes effect immediately.

12.2.1.1 Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally via TAxCLK. The clock source is selected with the TASSEL bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the ID bits. The selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the IDEX bits. The timer clock dividers are reset when TACLR is set.

NOTE: Timer_A dividers

Setting the TACLR bit clears the contents of TAxR and the clock dividers. The clock dividers are implemented as down counters. Therefore, when the TACLR bit is cleared, the timer clock immediately begins clocking at the first rising edge of the Timer_A clock source selected with the TASSEL bits and continues clocking at the divider settings set by the ID and IDEX bits.

12.2.2 Starting the Timer

The timer may be started or restarted in the following ways:

- The timer counts when MC > { 0 } and the clock source is active.
- When the timer mode is either up or up/down, the timer may be stopped by writing 0 to TAxCCR0. The timer may then be restarted by writing a nonzero value to TAxCCR0. In this scenario, the timer starts incrementing in the up direction from zero.

12.2.3 Timer Mode Control

The timer has four modes of operation: stop, up, continuous, and up/down (see Table 12-1). The operating mode is selected with the MC bits.

Table 12-1. Timer Modes

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero.

12.2.3.1 Up Mode

The up mode is used if the timer period must be different from 0FFFFh counts. The timer repeatedly counts up to the value of compare register TAxCCR0, which defines the period (see Figure 12-2). The number of timer counts in the period is TAxCCR0 + 1. When the timer value equals TAxCCR0, the timer restarts counting from zero. If up mode is selected when the timer value is greater than TAxCCR0, the timer immediately restarts counting from zero.

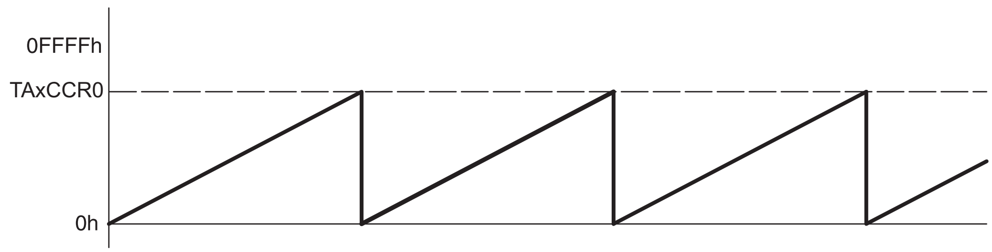


Figure 12-2. Up Mode

The TAxCCR0 CCIFG interrupt flag is set when the timer *counts* to the TAxCCR0 value. The TAIFG interrupt flag is set when the timer *counts* from TAxCCR0 to zero. Figure 12-3 shows the flag set cycle.

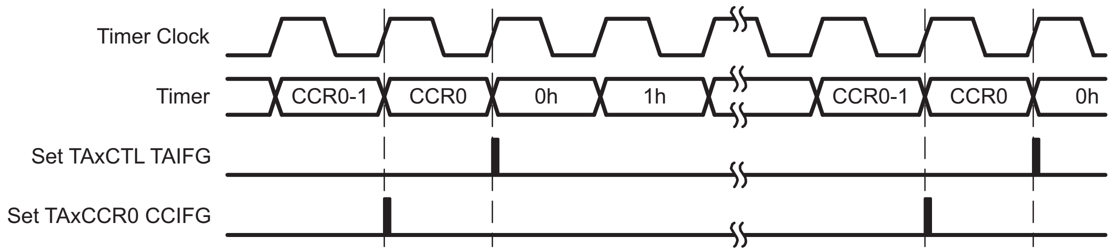


Figure 12-3. Up Mode Flag Setting

Changing Period Register TAxCCR0

When changing TAxCCR0 while the timer is running, if the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

12.2.3.2 Continuous Mode

In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in Figure 12-4. The capture/compare register TAxCCR0 works the same way as the other capture/compare registers.

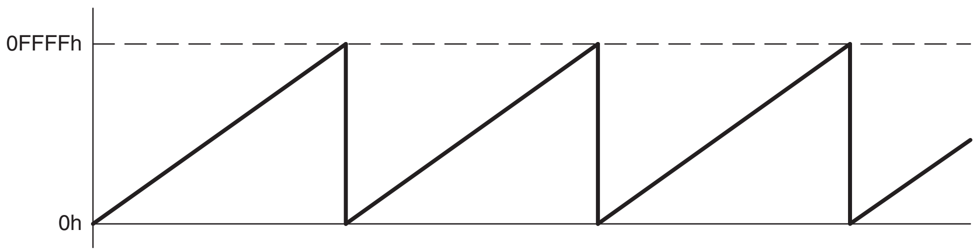


Figure 12-4. Continuous Mode

The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. Figure 12-5 shows the flag set cycle.

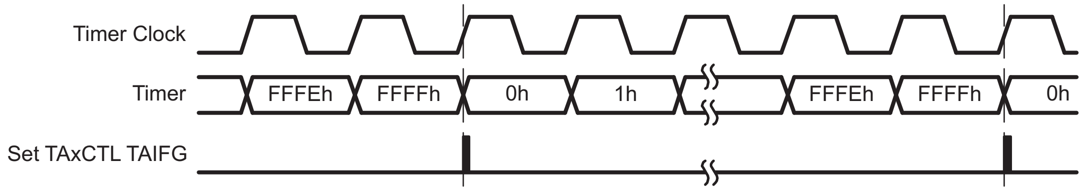


Figure 12-5. Continuous Mode Flag Setting

12.2.3.3 Use of Continuous Mode

The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TAxCCRn register in the interrupt service routine. Figure 12-6 shows two separate time intervals, t_0 and t_1, being added to the capture/compare registers. In this usage, the time interval is controlled by hardware, not software, without impact from interrupt latency. Up to n (where n = 0 to 6), independent time intervals or output frequencies can be generated using capture/compare registers.

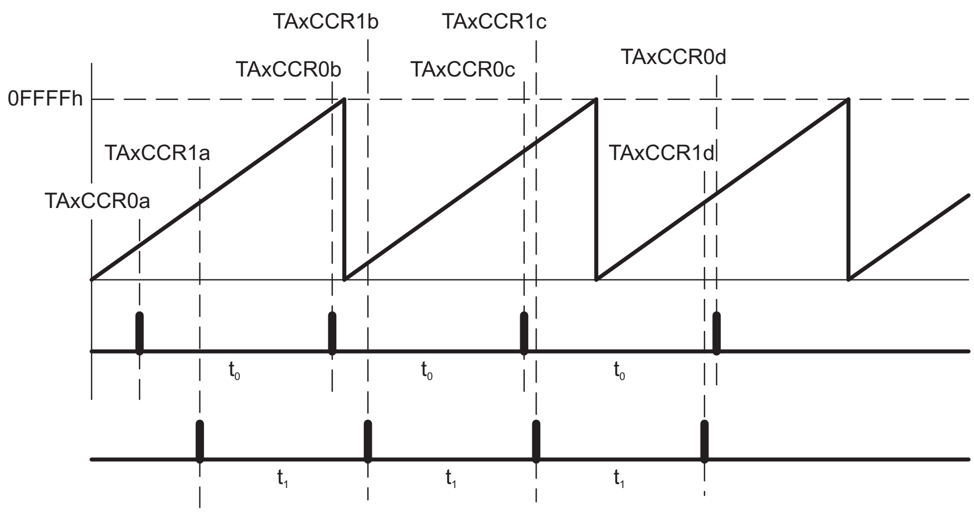


Figure 12-6. Continuous Mode Time Intervals

Time intervals can be produced with other modes as well, where TAxCCR0 is used as the period register. Their handling is more complex since the sum of the old TAxCCRn data and the new period can be higher than the TAxCCR0 value. When the previous TAxCCRn value plus t_x is greater than the TAxCCR0 data, the TAxCCR0 value must be subtracted to obtain the correct time interval.

12.2.3.4 Up/Down Mode

The up/down mode is used if the timer period must be different from 0FFFFh counts, and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare register TAxCCR0 and back down to zero (see Figure 12-7). The period is twice the value in TAxCCR0.

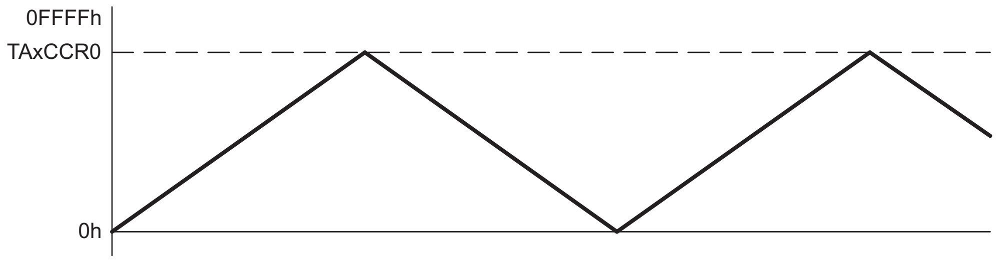


Figure 12-7. Up/Down Mode

The count direction is latched. This allows the timer to be stopped and then restarted in the same direction it was counting before it was stopped. If this is not desired, the TACLR bit must be set to clear the direction. The TACLR bit also clears the TAxR value and the timer clock divider.

In up/down mode, the TAxCCR0 CCIFG interrupt flag and the TAIFG interrupt flag are set only once during a period, separated by one-half the timer period. The TAxCCR0 CCIFG interrupt flag is set when the timer *counts* from TAxCCR0-1 to TAxCCR0, and TAIFG is set when the timer completes *counting* down from 0001h to 0000h. Figure 12-8 shows the flag set cycle.

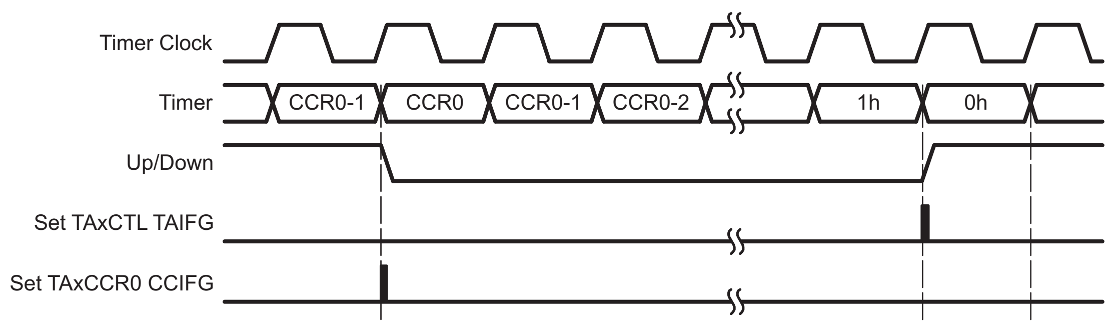


Figure 12-8. Up/Down Mode Flag Setting

Changing Period Register TAxCCR0

When changing TAxCCR0 while the timer is running and counting in the down direction, the timer continues its descent until it reaches zero. The new period takes affect after the counter counts down to zero.

When the timer is counting in the up direction, and the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period before counting down. When the timer is counting in the up direction and the new period is less than the current count value, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

12.2.3.5 Use of Up/Down Mode

The up/down mode supports applications that require dead times between output signals (see section *Timer_A Output Unit*). For example, to avoid overload conditions, two outputs driving an H-bridge must never be in a high state simultaneously. In the example shown in Figure 12-9, the t_{dead} is:

$t_{dead} = t_{timer} \times (\text{TAxCCR1} - \text{TAxCCR2})$

Where:

t_{dead} = Time during which both outputs need to be inactive
t_{timer} = Cycle time of the timer clock
TAxCCRn = Content of capture/compare register n

The TAxCCRn registers are not buffered. They update immediately when written to. Therefore, any required dead time is not maintained automatically.

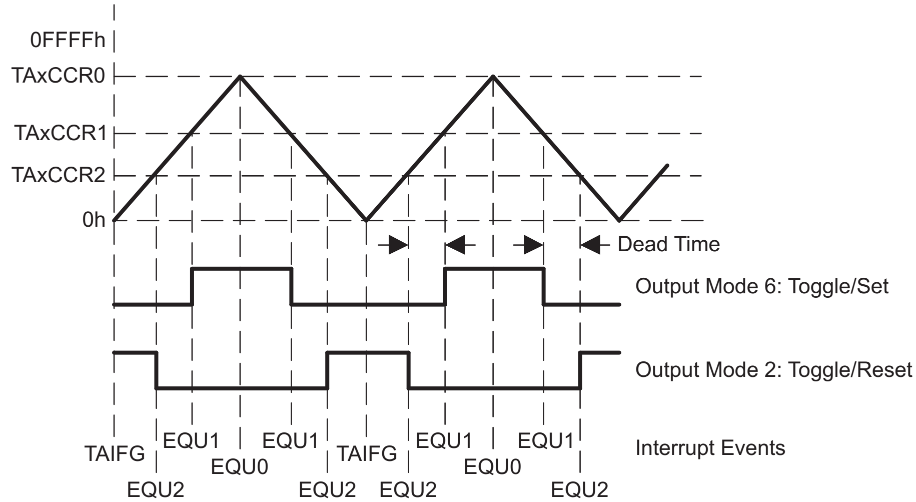


Figure 12-9. Output Unit in Up/Down Mode

12.2.4 Capture/Compare Blocks

Up to seven identical capture/compare blocks, TAxCCRn (where n = 0 to 7), are present in Timer_A. Any of the blocks may be used to capture the timer data or to generate time intervals.

12.2.4.1 Capture Mode

The capture mode is selected when CAP = 1. Capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CCIxA and CCIxB are connected to external pins or internal signals and are selected with the CCIS bits. The CM bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture occurs:

- The timer value is copied into the TAxCCRn register.
- The interrupt flag CCIFG is set.

The input signal level can be read at any time via the CCI bit. Devices may have different signals connected to CCIxA and CCIxB. See the device-specific data sheet for the connections of these signals.

The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit synchronizes the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended (see Figure 12-10).

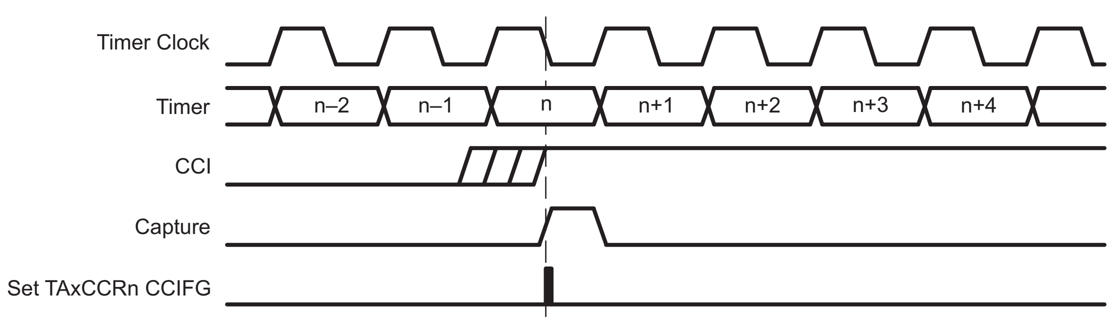


Figure 12-10. Capture Signal (SCS = 1)

NOTE: Changing Capture Inputs

Changing capture inputs while in capture mode may cause unintended capture events. To avoid this scenario, capture inputs should only be changed when capture mode is disabled (CM = {0} or CAP = 0).

Overflow logic is provided in each capture/compare register to indicate if a second capture was performed before the value from the first capture was read. Bit COV is set when this occurs as shown in Figure 12-11. COV must be reset with software.

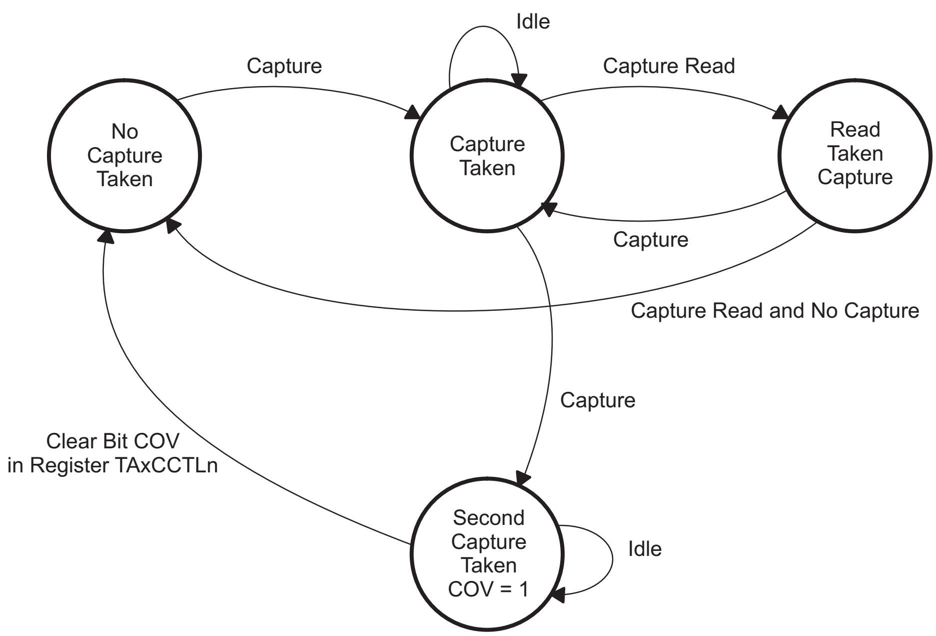


Figure 12-11. Capture Cycle

Capture Initiated by Software

Captures can be initiated by software. The CMx bits can be set for capture on both edges. Software then sets CCIS1 = 1 and toggles bit CCIS0 to switch the capture signal between V_{CC} and GND, initiating a capture each time CCIS0 changes state:

```
MOV   #CAP+SCS+CCIS1+CM_3,&TA0CCTL1  ; Setup TA0CCTL1, synch. capture mode
                                     ; Event trigger on both edges of capture input.
XOR   #CCIS0,&TA0CCTL1               ; TA0CCR1 = TA0R
```

NOTE: Capture Initiated by Software

In general, changing capture inputs while in capture mode may cause unintended capture events. For this scenario, switching the capture input between VCC and GND, disabling the capture mode is not required.

12.2.4.2 Compare Mode

The compare mode is selected when CAP = 0. The compare mode is used to generate PWM output signals or interrupts at specific time intervals. When TAxR *counts* to the value in a TAxCCRn, where n represents the specific capture/compare register.

- Interrupt flag CCIFG is set.
- Internal signal EQUn = 1.
- EQUn affects the output according to the output mode.
- The input signal CCI is latched into SCCI.

12.2.5 Output Unit

Each capture/compare block contains an output unit. The output unit is used to generate output signals, such as PWM signals. Each output unit has eight operating modes that generate signals based on the EQU0 and EQUn signals.

12.2.5.1 Output Modes

The output modes are defined by the OUTMOD bits and are described in Table 12-2. The OUTn signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0 because EQUn = EQU0.

Table 12-2. Output Modes

OUTMODx	Mode	Description
000	Output	The output signal OUTn is defined by the OUT bit. The OUTn signal updates immediately when OUT is updated.
001	Set	The output is set when the timer *counts* to the TAxCCRn value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer *counts* to the TAxCCRn value. It is reset when the timer *counts* to the TAxCCR0 value.
011	Set/Reset	The output is set when the timer *counts* to the TAxCCRn value. It is reset when the timer *counts* to the TAxCCR0 value.
100	Toggle	The output is toggled when the timer *counts* to the TAxCCRn value. The output period is double the timer period.
101	Reset	The output is reset when the timer *counts* to the TAxCCRn value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer *counts* to the TAxCCRn value. It is set when the timer *counts* to the TAxCCR0 value.
111	Reset/Set	The output is reset when the timer *counts* to the TAxCCRn value. It is set when the timer *counts* to the TAxCCR0 value.

Output Example—Timer in Up Mode

The OUTn signal is changed when the timer *counts* up to the TAxCCRn value and rolls from TAxCCR0 to zero, depending on the output mode. An example is shown in Figure 12-12 using TAxCCR0 and TAxCCR1.

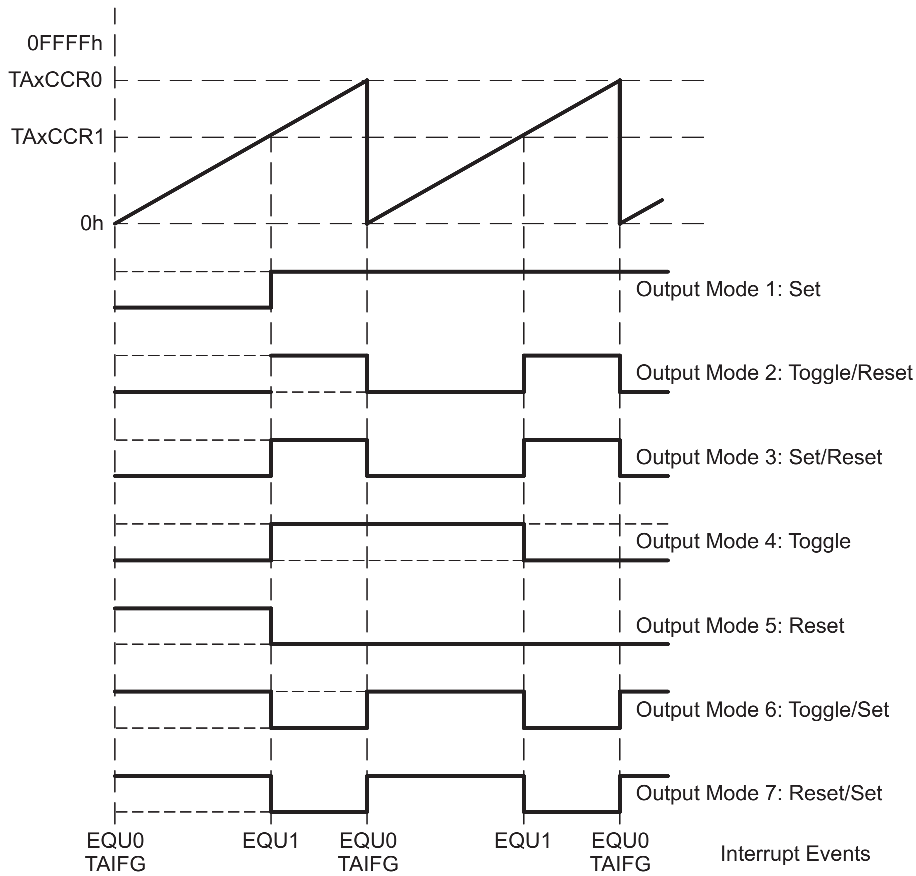


Figure 12-12. Output Example – Timer in Up Mode

Output Example – Timer in Continuous Mode

The OUTn signal is changed when the timer reaches the TAxCCRn and TAxCCR0 values, depending on the output mode. An example is shown in Figure 12-13 using TAxCCR0 and TAxCCR1.

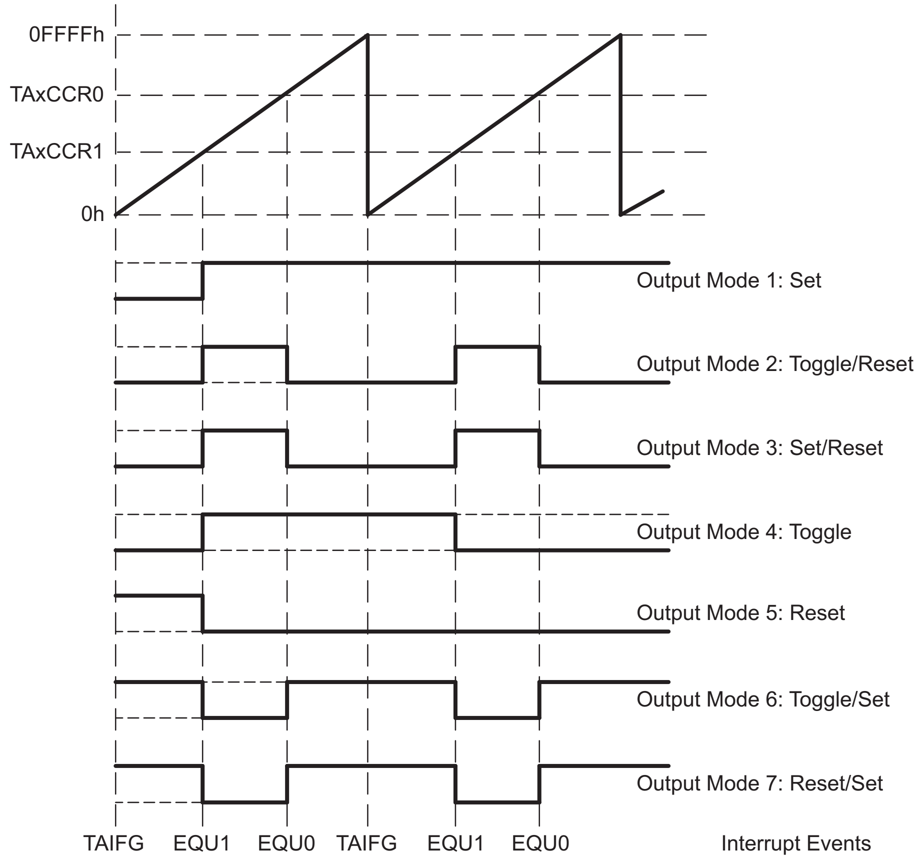


Figure 12-13. Output Example – Timer in Continuous Mode

Output Example – Timer in Up/Down Mode

The OUTn signal changes when the timer equals TAxCCRn in either count direction and when the timer equals TAxCCR0, depending on the output mode. An example is shown in Figure 12-14 using TAxCCR0 and TAxCCR2.

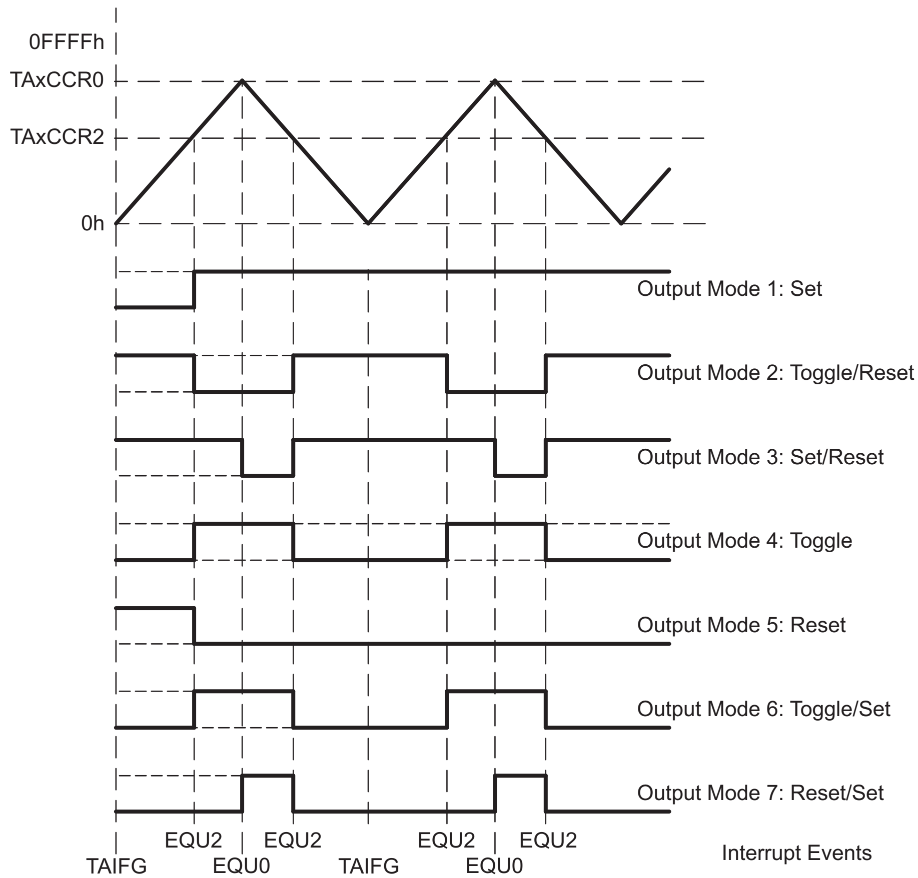


Figure 12-14. Output Example – Timer in Up/Down Mode

NOTE: Switching between output modes

When switching between output modes, one of the OUTMOD bits should remain set during the transition, unless switching to mode 0. Otherwise, output glitching can occur, because a NOR gate decodes output mode 0. A safe method for switching between output modes is to use output mode 7 as a transition state:

```
BIS    #OUTMOD_7,&TA0CCTL1      ; Set output mode=7
BIC    #OUTMOD,&TA0CCTL1       ; Clear unwanted bits
```

12.2.6 Timer_A Interrupts

Two interrupt vectors are associated with the 16-bit Timer_A module:

- TAxCCR0 interrupt vector for TAxCCR0 CCIFG
- TAxIV interrupt vector for all other CCIFG flags and TAIFG

In capture mode, any CCIFG flag is set when a timer value is captured in the associated TAxCCRn register. In compare mode, any CCIFG flag is set if TAxR *counts* to the associated TAxCCRn value. Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set.

12.2.6.1 TAxCCR0 Interrupt

The TAxCCR0 CCIFG flag has the highest Timer_A interrupt priority and has a dedicated interrupt vector as shown in Figure 12-15. The TAxCCR0 CCIFG flag is automatically reset when the TAxCCR0 interrupt request is serviced.

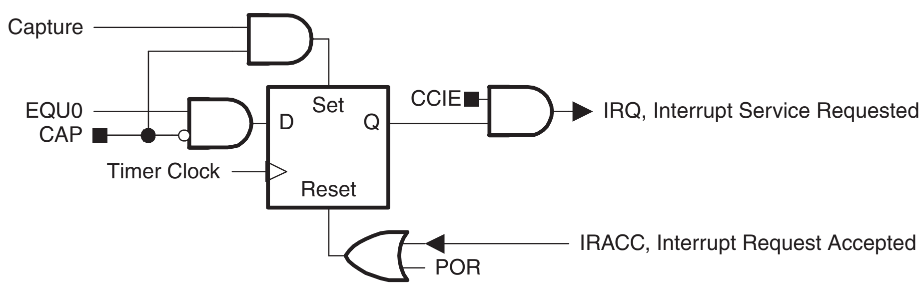


Figure 12-15. Capture/Compare TAxCCR0 Interrupt Flag

12.2.6.2 TAxIV, Interrupt Vector Generator

The TAxCCRy CCIFG flags and TAIFG flags are prioritized and combined to source a single interrupt vector. The interrupt vector register TAxIV is used to determine which flag requested an interrupt.

The highest-priority enabled interrupt generates a number in the TAxIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Timer_A interrupts do not affect the TAxIV value.

Any access, read or write, of the TAxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, if the TAxCCR1 and TAxCCR2 CCIFG flags are set when the interrupt service routine accesses the TAxIV register, TAxCCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TAxCCR2 CCIFG flag generates another interrupt.

TAxIV Software Example

The following software example shows the recommended use of TAxIV and the handling overhead. The TAxIV value is added to the PC to automatically jump to the appropriate routine. The example assumes a single instantiation of the largest timer configuration available.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- Capture/compare block TA0CCR0: 11 cycles
- Capture/compare blocks TA0CCR1, TA0CCR2, TA0CCR3, TA0CCR4, TA0CCR5, TA0CCR6: 16 cycles
- Timer overflow TA0IFG: 14 cycles

```
; Interrupt handler for TA0CCR0 CCIFG.                      Cycles
CCIFG_0_HND
;        ...         ; Start of handler Interrupt latency    6
         RETI                                                5

; Interrupt handler for TA0IFG, TA0CCR1 through TA0CCR6 CCIFG.

TA0_HND      ...               ; Interrupt latency           6
         ADD      &TA0IV,PC    ; Add offset to Jump table    3
         RETI                  ; Vector  0: No interrupt     5
         JMP      CCIFG_1_HND  ; Vector  2: TA0CCR1          2
         JMP      CCIFG_2_HND  ; Vector  4: TA0CCR2          2
         JMP      CCIFG_3_HND  ; Vector  6: TA0CCR3          2
         JMP      CCIFG_4_HND  ; Vector  8: TA0CCR4          2
         JMP      CCIFG_5_HND  ; Vector 10: TA0CCR5          2
         JMP      CCIFG_6_HND  ; Vector 12: TA0CCR6          2

TA0IFG_HND                     ; Vector 14: TA0IFG Flag
         ...                   ; Task starts here
         RETI                                                5

CCIFG_6_HND                    ; Vector 12: TA0CCR6
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_5_HND                    ; Vector 10: TA0CCR5
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_4_HND                    ; Vector 8: TA0CCR4
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_3_HND                    ; Vector 6: TA0CCR3
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_2_HND                    ; Vector 4: TA0CCR2
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_1_HND                    ; Vector 2: TA0CCR1
         ...                   ; Task starts here
         RETI                  ; Back to main program        5
```

12.3 Timer_A Registers

Timer_A registers are listed in Table 12-3 for the largest configuration available. The base address can be found in the device-specific data sheet. The address offsets are listed in Table 12-3.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 12-3. Timer_A Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Timer_A Control	TAxCTL	Read/write	Word	00h	0000h
	TAxCTL_L	Read/write	Byte	00h	00h
	TAxCTL_H	Read/write	Byte	01h	00h
Timer_A Capture/Compare Control 0	TAxCCTL0	Read/write	Word	02h	0000h
	TAxCCTL0_L	Read/write	Byte	02h	00h
	TAxCCTL0_H	Read/write	Byte	03h	00h
Timer_A Capture/Compare Control 1	TAxCCTL1	Read/write	Word	04h	0000h
	TAxCCTL1_L	Read/write	Byte	04h	00h
	TAxCCTL1_H	Read/write	Byte	05h	00h
Timer_A Capture/Compare Control 2	TAxCCTL2	Read/write	Word	06h	0000h
	TAxCCTL2_L	Read/write	Byte	06h	00h
	TAxCCTL2_H	Read/write	Byte	07h	00h
Timer_A Capture/Compare Control 3	TAxCCTL3	Read/write	Word	08h	0000h
	TAxCCTL3_L	Read/write	Byte	08h	00h
	TAxCCTL3_H	Read/write	Byte	09h	00h
Timer_A Capture/Compare Control 4	TAxCCTL4	Read/write	Word	0Ah	0000h
	TAxCCTL4_L	Read/write	Byte	0Ah	00h
	TAxCCTL4_H	Read/write	Byte	0Bh	00h
Timer_A Capture/Compare Control 5	TAxCCTL5	Read/write	Word	0Ch	0000h
	TAxCCTL5_L	Read/write	Byte	0Ch	00h
	TAxCCTL5_H	Read/write	Byte	0Dh	00h
Timer_A Capture/Compare Control 6	TAxCCTL6	Read/write	Word	0Eh	0000h
	TAxCCTL6_L	Read/write	Byte	0Eh	00h
	TAxCCTL6_H	Read/write	Byte	0Fh	00h
Timer_A Counter	TAxR	Read/write	Word	10h	0000h
	TAxR_L	Read/write	Byte	10h	00h
	TAxR_H	Read/write	Byte	11h	00h
Timer_A Capture/Compare 0	TAxCCR0	Read/write	Word	12h	0000h
	TAxCCR0_L	Read/write	Byte	12h	00h
	TAxCCR0_H	Read/write	Byte	13h	00h
Timer_A Capture/Compare 1	TAxCCR1	Read/write	Word	14h	0000h
	TAxCCR1_L	Read/write	Byte	14h	00h
	TAxCCR1_H	Read/write	Byte	15h	00h
Timer_A Capture/Compare 2	TAxCCR2	Read/write	Word	16h	0000h
	TAxCCR2_L	Read/write	Byte	16h	00h
	TAxCCR2_H	Read/write	Byte	17h	00h

Table 12-3. Timer_A Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Timer_A Capture/Compare 3	TAxCCR3	Read/write	Word	18h	0000h
	TAxCCR3_L	Read/write	Byte	18h	00h
	TAxCCR3_H	Read/write	Byte	19h	00h
Timer_A Capture/Compare 4	TAxCCR4	Read/write	Word	1Ah	0000h
	TAxCCR4_L	Read/write	Byte	1Ah	00h
	TAxCCR4_H	Read/write	Byte	1Bh	00h
Timer_A Capture/Compare 5	TAxCCR5	Read/write	Word	1Ch	0000h
	TAxCCR5_L	Read/write	Byte	1Ch	00h
	TAxCCR5_H	Read/write	Byte	1Dh	00h
Timer_A Capture/Compare 6	TAxCCR6	Read/write	Word	1Eh	0000h
	TAxCCR6_L	Read/write	Byte	1Eh	00h
	TAxCCR6_H	Read/write	Byte	1Fh	00h
Timer_A Interrupt Vector	TAxIV	Read only	Word	2Eh	0000h
	TAxIV_L	Read only	Byte	2Eh	00h
	TAxIV_H	Read only	Byte	2Fh	00h
Timer_A Expansion 0	TAxEX0	Read/write	Word	20h	0000h
	TAxEX0_L	Read/write	Byte	20h	00h
	TAxEX0_H	Read/write	Byte	21h	00h

Timer_A Control Register (TAxCTL)

15	14	13	12	11	10	9	8
Unused						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ID		MC		Unused	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Unused	Bits 15-10	Unused
TASSEL	Bits 9-8	Timer_A clock source select 00 TAxCLK 01 ACLK 10 SMCLK 11 Inverted TAxCLK
ID	Bits 7-6	Input divider. These bits along with the IDEX bits select the divider for the input clock. 00 /1 01 /2 10 /4 11 /8
MC	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00 Stop mode: Timer is halted 01 Up mode: Timer counts up to TAxCCR0 10 Continuous mode: Timer counts up to 0FFFFh 11 Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
Unused	Bit 3	Unused
TACLR	Bit 2	Timer_A clear. Setting this bit resets TAxR, the timer clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.
TAIE	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0 Interrupt disabled 1 Interrupt enabled
TAIFG	Bit 0	Timer_A interrupt flag 0 No interrupt pending 1 Interrupt pending

Timer_A Counter Register (TAxR)

15	14	13	12	11	10	9	8
TAxR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
TAxR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TAxR	Bits 15-0	Timer_A register. The TAxR register is the count of Timer_A.

Capture/Compare Control Register (TAxCCTLn)

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	r-(0)	rw-(0)

7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

CM Bits 15-14 Capture mode
- 00 No capture
- 01 Capture on rising edge
- 10 Capture on falling edge
- 11 Capture on both rising and falling edges

CCIS Bits 13-12 Capture/compare input select. These bits select the TAxCCRn input signal. See the device-specific data sheet for specific signal connections.
- 00 CCIxA
- 01 CCIxB
- 10 GND
- 11 V_{CC}

SCS Bit 11 Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.
- 0 Asynchronous capture
- 1 Synchronous capture

SCCI Bit 10 Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit.

Unused Bit 9 Unused. Read only. Always read as 0.

CAP Bit 8 Capture mode
- 0 Compare mode
- 1 Capture mode

OUTMOD Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 because EQUx = EQU0.
- 000 OUT bit value
- 001 Set
- 010 Toggle/reset
- 011 Set/reset
- 100 Toggle
- 101 Reset
- 110 Toggle/set
- 111 Reset/set

CCIE Bit 4 Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.
- 0 Interrupt disabled
- 1 Interrupt enabled

CCI Bit 3 Capture/compare input. The selected input signal can be read by this bit.

OUT Bit 2 Output. For output mode 0, this bit directly controls the state of the output.
- 0 Output low
- 1 Output high

COV Bit 1 Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.
- 0 No capture overflow occurred
- 1 Capture overflow occurred

(continued)

CCIFG	Bit 0	Capture/compare interrupt flag 0 No interrupt pending 1 Interrupt pending

Timer_A Interrupt Vector Register (TAxIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	0	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**		**TAIV**		**0**
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

TAIV Bits 15-0 Timer_A interrupt vector value

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Capture/compare 1	TAxCCR1 CCIFG	Highest
04h	Capture/compare 2	TAxCCR2 CCIFG	
06h	Capture/compare 3	TAxCCR3 CCIFG	
08h	Capture/compare 4	TAxCCR4 CCIFG	
0Ah	Capture/compare 5	TAxCCR5 CCIFG	
0Ch	Capture/compare 6	TAxCCR6 CCIFG	
0Eh	Timer overflow	TAxCTL TAIFG	Lowest

Timer_A Expansion 0 Register (TAxEX0)

15	14	13	12	11	10	9	8
Unused	**Unused**	**Unused**	**Unused**	**Unused**	**Unused**	**Unused**	**Unused**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**Unused**	**Unused**		**IDEX**	
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

Unused	Bits 15-3	Unused. Read only. Always read as 0.
IDEX	Bits 2-0	Input divider expansion. These bits along with the ID bits select the divider for the input clock. 000 /1 001 /2 010 /3 011 /4 100 /5 101 /6 110 /7 111 /8

Timer_B

Timer_B is a 16-bit timer/counter with multiple capture/compare registers. There can be multiple Timer_B modules on a given device (see the device-specific data sheet). This chapter describes the operation and use of the Timer_B module.

Topic **Page**

13.1 Timer_B Introduction **362**
13.2 Timer_B Operation **364**
13.3 Timer_B Registers **377**

13.1 Timer_B Introduction

Timer_B is a 16-bit timer/counter with up to seven capture/compare registers. Timer_B can support multiple capture/compares, PWM outputs, and interval timing. Timer_B also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Timer_B features include :

- Asynchronous 16-bit timer/counter with four operating modes and four selectable lengths
- Selectable and configurable clock source
- Up to seven configurable capture/compare registers
- Configurable outputs with PWM capability
- Double-buffered compare latches with synchronized loading
- Interrupt vector register for fast decoding of all Timer_B interrupts

The block diagram of Timer_B is shown in Figure 13-1.

NOTE: Use of the word *count*

Count is used throughout this chapter. It means the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, an associated action does not take place.

NOTE: Nomenclature

There may be multiple instantiations of Timer_B on a given device. The prefix TBx is used, where x is a greater than equal to zero indicating the Timer_B instantiation. For devices with one instantiation, x = 0. The suffix n, where n = 0 to 6, represents the specific capture/compare registers associated with the Timer_B instantiation.

13.1.1 Similarities and Differences From Timer_A

Timer_B is identical to Timer_A with the following exceptions:

- The length of Timer_B is programmable to be 8, 10, 12, or 16 bits.
- Timer_B TBxCCRn registers are double-buffered and can be grouped.
- All Timer_B outputs can be put into a high-impedance state.
- The SCCI bit function is not implemented in Timer_B.

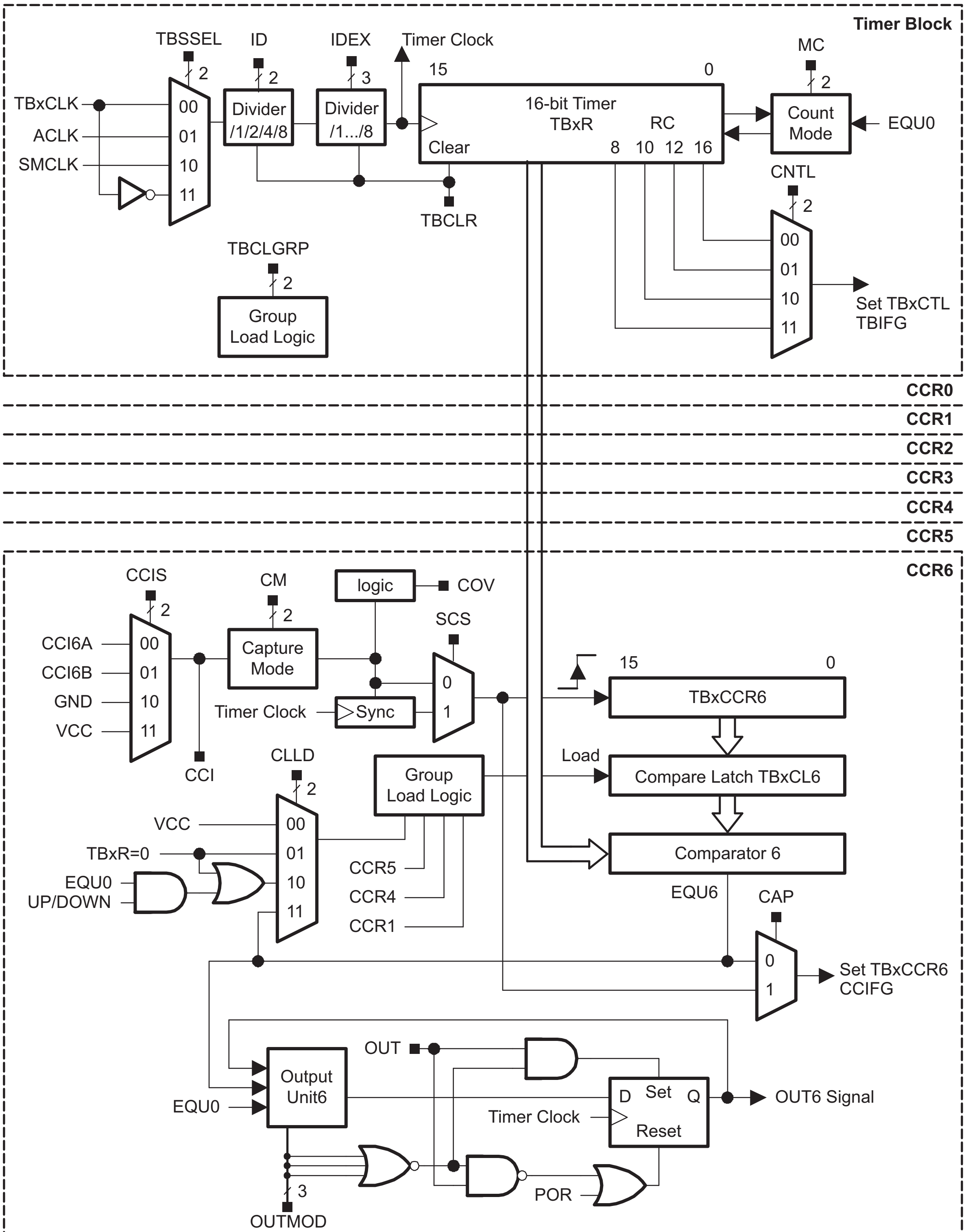


Figure 13-1. Timer_B Block Diagram

13.2 Timer_B Operation

The Timer_B module is configured with user software. The setup and operation of Timer_B is discussed in the following sections.

13.2.1 16-Bit Timer Counter

The 16-bit timer/counter register, TBxR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TBxR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.

TBxR may be cleared by setting the TBCLR bit. Setting TBCLR also clears the clock divider and count direction for up/down mode.

NOTE: Modifying Timer_B registers

It is recommended to stop the timer before modifying its operation (with exception of the interrupt enable, interrupt flag, and TBCLR) to avoid errant operating conditions.

When the timer clock is asynchronous to the CPU clock, any read from TBxR should occur while the timer is not operating or the results may be unpredictable. Alternatively, the timer may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to TBxR takes effect immediately.

13.2.1.1 TBxR Length

Timer_B is configurable to operate as an 8-, 10-, 12-, or 16-bit timer with the CNTL bits. The maximum count value, $TBxR_{(max)}$, for the selectable lengths is 0FFh, 03FFh, 0FFFh, and 0FFFFh, respectively. Data written to the TBxR register in 8-, 10-, and 12-bit mode is right justified with leading zeros.

13.2.1.2 Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally via TBxCLK. The clock source is selected with the TBSSEL bits. The selected clock source may be passed directly to the timer or divided by 2,4, or 8, using the ID bits. The selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the IDEX bits. The timer clock dividers are reset when TBCLR is set.

NOTE: Timer_B dividers

Setting the TBCLR bit clears the contents of TBxR and the clock dividers. The clock dividers are implemented as down counters. Therefore, when the TBCLR bit is cleared, the timer clock immediately begins clocking at the first rising edge of the Timer_B clock source selected with the TBSSEL bits and continues clocking at the divider settings set by the ID and IDEX bits.

13.2.2 Starting the Timer

The timer may be started or restarted in the following ways:

- The timer counts when MC > { 0 } and the clock source is active.
- When the timer mode is either up or up/down, the timer may be stopped by loading 0 to TBxCL0. The timer may then be restarted by loading a nonzero value to TBxCL0. In this scenario, the timer starts incrementing in the up direction from zero.

13.2.3 Timer Mode Control

The timer has four modes of operation: stop, up, continuous, and up/down (see Table 13-1). The operating mode is selected with the MC bits.

Table 13-1. Timer Modes

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of compare register TBxCL0.
10	Continuous	The timer repeatedly counts from zero to the value selected by the CNTL bits.
11	Up/down	The timer repeatedly counts from zero up to the value of TBxCL0 and then back down to zero.

13.2.3.1 Up Mode

The up mode is used if the timer period must be different from $TBxR_{(max)}$ counts. The timer repeatedly counts up to the value of compare latch TBxCL0, which defines the period (see Figure 13-2). The number of timer counts in the period is TBxCL0 + 1. When the timer value equals TBxCL0, the timer restarts counting from zero. If up mode is selected when the timer value is greater than TBxCL0, the timer immediately restarts counting from zero.

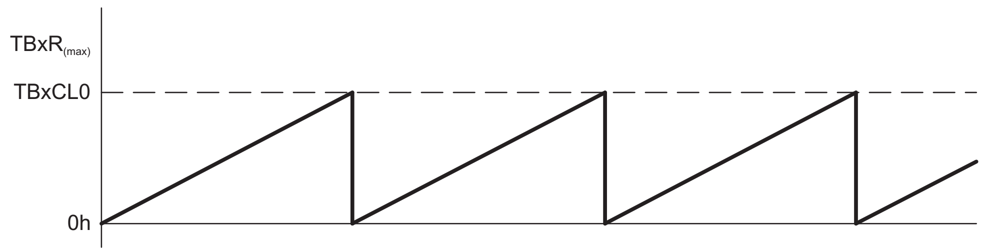


Figure 13-2. Up Mode

The TBxCCR0 CCIFG interrupt flag is set when the timer *counts* to the TBxCL0 value. The TBIFG interrupt flag is set when the timer *counts* from TBxCL0 to zero. Figure 13-3 shows the flag set cycle.

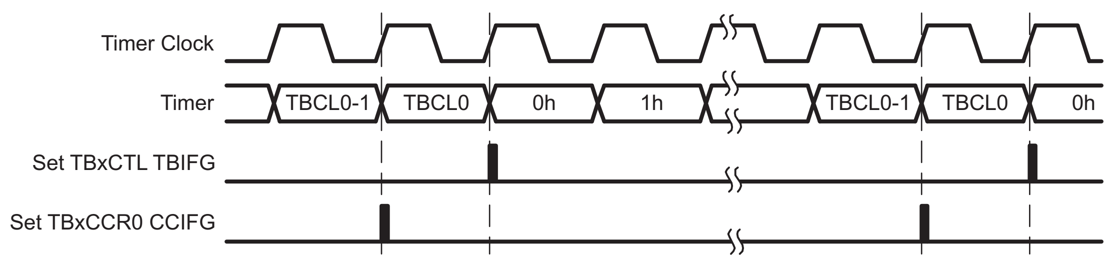


Figure 13-3. Up Mode Flag Setting

Changing Period Register TBxCL0

When changing TBxCL0 while the timer is running and when the TBxCL0 load mode is *immediate*, if the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

13.2.3.2 Continuous Mode

In continuous mode, the timer repeatedly counts up to $TBxR_{(max)}$ and restarts from zero (see Figure 13-4). The compare latch TBxCL0 works the same way as the other capture/compare registers.

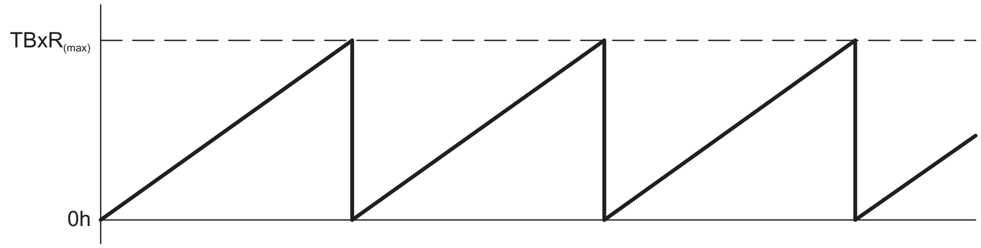


Figure 13-4. Continuous Mode

The TBIFG interrupt flag is set when the timer *counts* from $TBxR_{(max)}$ to zero. Figure 13-5 shows the flag set cycle.

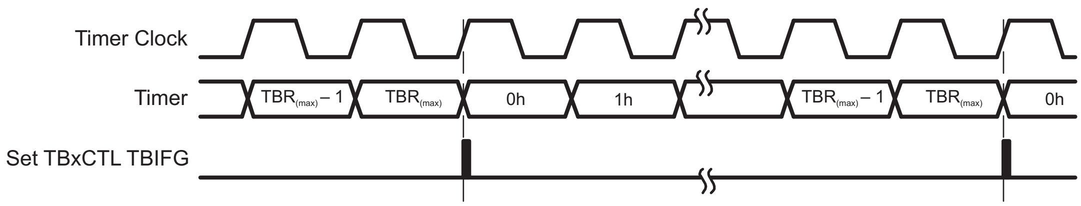


Figure 13-5. Continuous Mode Flag Setting

13.2.3.3 Use of Continuous Mode

The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TBxCLn latch in the interrupt service routine. Figure 13-6 shows two separate time intervals, t_0 and t_1, being added to the capture/compare registers. The time interval is controlled by hardware, not software, without impact from interrupt latency. Up to n (where n = 0 to 7), independent time intervals or output frequencies can be generated using capture/compare registers.

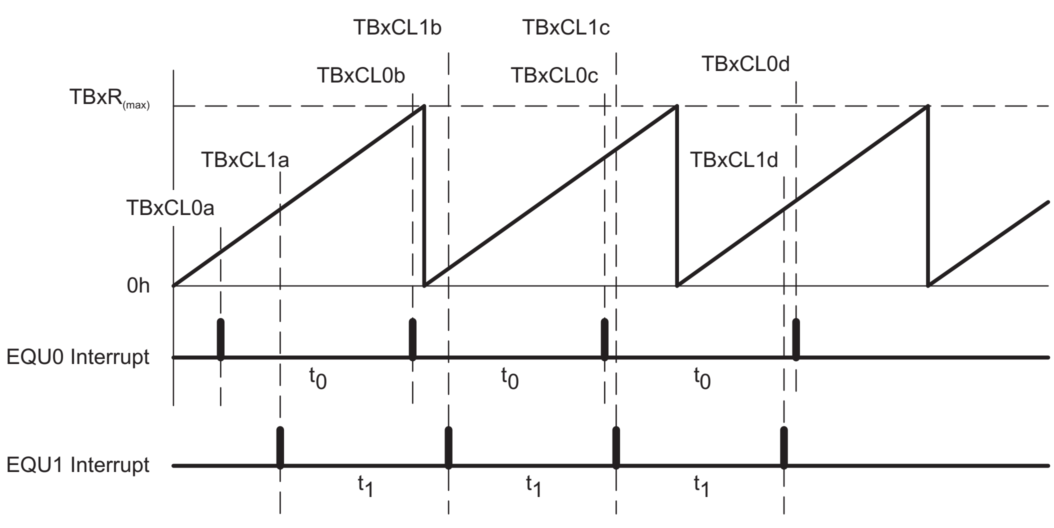


Figure 13-6. Continuous Mode Time Intervals

Time intervals can be produced with other modes as well, where TBxCL0 is used as the period register. Their handling is more complex, since the sum of the old TBxCLn data and the new period can be higher than the TBxCL0 value. When the sum of the previous TBxCLn value plus t_x is greater than the TBxCL0 data, the old TBxCL0 value must be subtracted to obtain the correct time interval.

13.2.3.4 Up/Down Mode

The up/down mode is used if the timer period must be different from $TBxR_{(max)}$ counts and, if symmetrical, pulse generation is needed. The timer repeatedly counts up to the value of compare latch TBxCL0, and back down to zero (see Figure 13-7). The period is twice the value in TBxCL0.

> **NOTE: TBxCL0 > $TBxR_{(max)}$**
>
> If TBxCL0 > $TBxR_{(max)}$, the counter operates as if it were configured for continuous mode. It does not count down from $TBxR_{(max)}$ to zero.

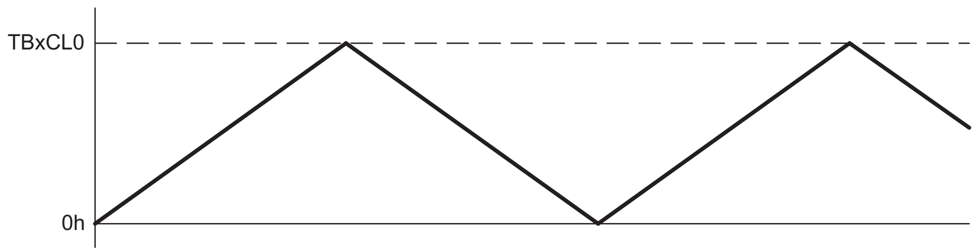


Figure 13-7. Up/Down Mode

The count direction is latched. This allows the timer to be stopped and then restarted in the same direction it was counting before it was stopped. If this is not desired, the TBCLR bit must be used to clear the direction. The TBCLR bit also clears the TBxR value and the timer clock divider.

In up/down mode, the TBxCCR0 CCIFG interrupt flag and the TBIFG interrupt flag are set only once during the period, separated by one-half the timer period. The TBxCCR0 CCIFG interrupt flag is set when the timer *counts* from TBxCL0-1 to TBxCL0, and TBIFG is set when the timer completes *counting* down from 0001h to 0000h. Figure 13-8 shows the flag set cycle.

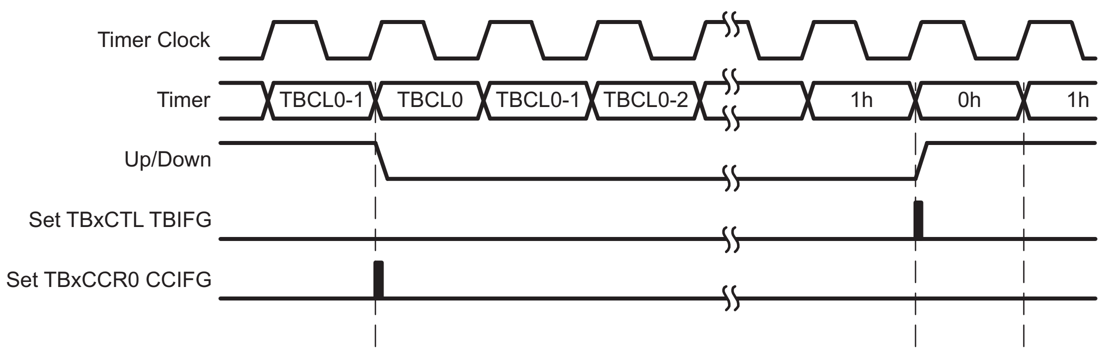


Figure 13-8. Up/Down Mode Flag Setting

Changing the Value of Period Register TBxCL0

When changing TBxCL0 while the timer is running and counting in the down direction, and when the TBxCL0 load mode is *immediate*, the timer continues its descent until it reaches zero. The new period takes effect after the counter counts down to zero.

If the timer is counting in the up direction when the new period is latched into TBxCL0, and the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period before counting down. When the timer is counting in the up direction, and the new period is less than the current count value when TBxCL0 is loaded, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

13.2.3.5 Use of Up/Down Mode

The up/down mode supports applications that require dead times between output signals (see section *Timer_B Output Unit*). For example, to avoid overload conditions, two outputs driving an H-bridge must never be in a high state simultaneously. In the example shown in Figure 13-9, the t_{dead} is:

$t_{dead} = t_{timer} \times (TBxCL1 - TBxCL3)$

Where:

t_{dead} = Time during which both outputs need to be inactive
t_{timer} = Cycle time of the timer clock
TBxCLn = Content of compare latch n

The ability to simultaneously load grouped compare latches ensures the dead times.

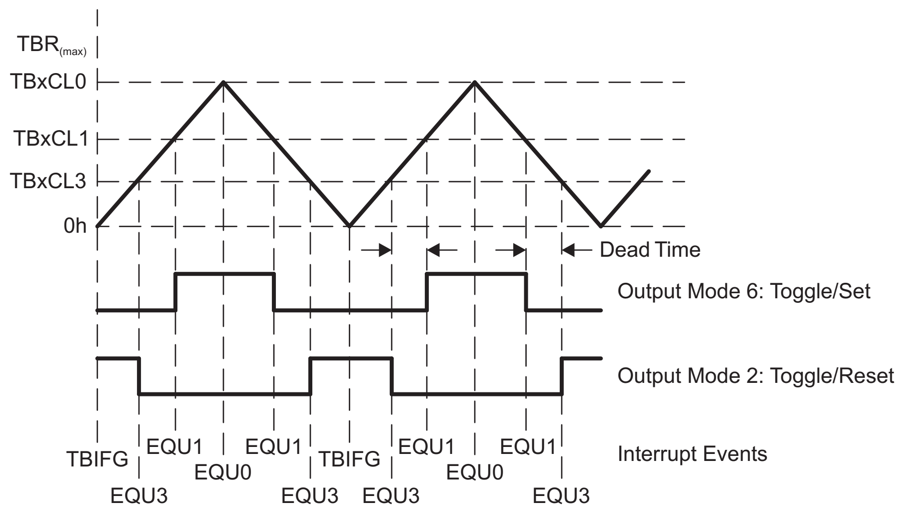


Figure 13-9. Output Unit in Up/Down Mode

13.2.4 Capture/Compare Blocks

Up to seven identical capture/compare blocks, TBxCCRn (where n = 0 to 6), are present in Timer_B. Any of the blocks may be used to capture the timer data or to generate time intervals.

13.2.4.1 Capture Mode

The capture mode is selected when CAP = 1. Capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CCIxA and CCIxB are connected to external pins or internal signals and are selected with the CCIS bits. The CM bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture is performed:

- The timer value is copied into the TBxCCRn register.
- The interrupt flag CCIFG is set.

The input signal level can be read at any time via the CCI bit. MSP430x5xx family devices may have different signals connected to CCIxA and CCIxB. See the device-specific data sheet for the connections of these signals.

The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit synchronizes the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended (see Figure 13-10).

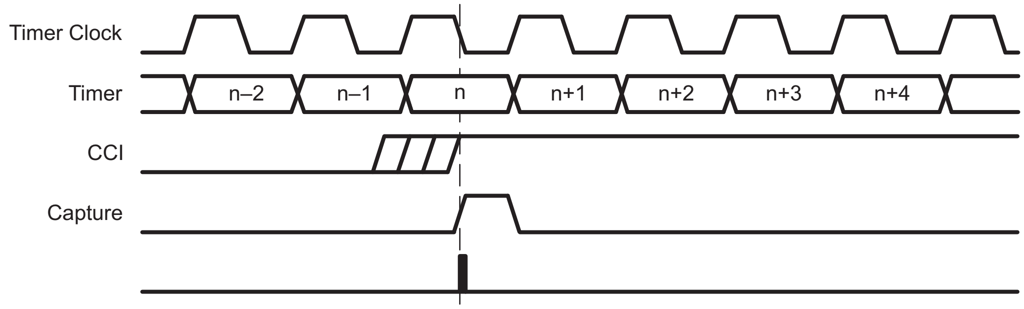


Figure 13-10. Capture Signal (SCS = 1)

NOTE: Changing Capture Inputs

Changing capture inputs while in capture mode may cause unintended capture events. To avoid this scenario, capture inputs should only be changed when capture mode is disabled (CM = {0} or CAP = 0).

Overflow logic is provided in each capture/compare register to indicate if a second capture was performed before the value from the first capture was read. Bit COV is set when this occurs (see Figure 13-11). COV must be reset with software.

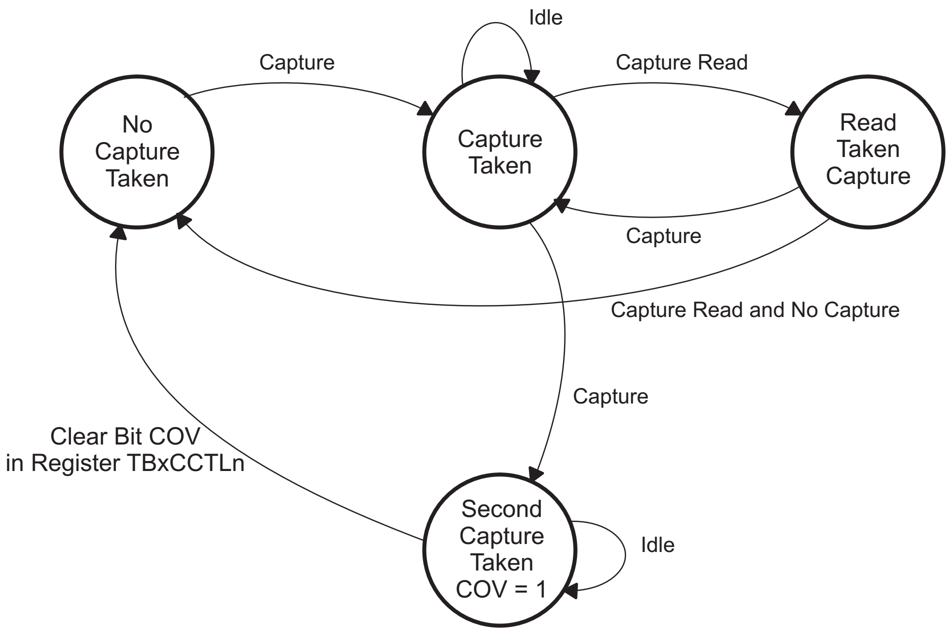


Figure 13-11. Capture Cycle

Capture Initiated by Software

Captures can be initiated by software. The CM bits can be set for capture on both edges. Software then sets bit CCIS1 = 1 and toggles bit CCIS0 to switch the capture signal between V_{CC} and GND, initiating a capture each time CCIS0 changes state:

```
MOV      #CAP+SCS+CCIS1+CM_3,&TB0CCTL1    ; Setup TB0CCTL1
XOR      #CCIS0,&TB0CCTL1                 ; TB0CCR1 = TB0R
```

NOTE: Capture Initiated by Software

In general, changing capture inputs while in capture mode may cause unintended capture events. For this scenario, switching the capture input between VCC and GND, disabling the capture mode is not required.

13.2.4.2 Compare Mode

The compare mode is selected when CAP = 0. Compare mode is used to generate PWM output signals or interrupts at specific time intervals. When TBxR *counts* to the value in a TBxCLn, where n represents the specific capture/compare latch:

- Interrupt flag CCIFG is set.
- Internal signal EQUn = 1.
- EQUn affects the output according to the output mode.

Compare Latch TBxCLn

The TBxCCRn compare latch, TBxCLn, holds the data for the comparison to the timer value in compare mode. TBxCLn is buffered by TBxCCRn. The buffered compare latch gives the user control over when a compare period updates. The user cannot directly access TBxCLn. Compare data is written to each TBxCCRn and automatically transferred to TxBCLn. The timing of the transfer from TBxCCRn to TBxCLn is user selectable, with the CLLD bits as described in Table 13-2.

Table 13-2. TBxCLn Load Events

CLLD	Description
00	New data is transferred from TBxCCRn to TBxCLn immediately when TBxCCRn is written to.
01	New data is transferred from TBxCCRn to TBxCLn when TBxR *counts* to 0.
10	New data is transferred from TBxCCRn to TBxCLn when TBxR *counts* to 0 for up and continuous modes. New data is transferred to from TBxCCRn to TBxCLn when TBxR *counts* to the old TBxCL0 value or to 0 for up/down mode.
11	New data is transferred from TBxCCRn to TBxCLn when TBxR *counts* to the old TBxCLn value.

Grouping Compare Latches

Multiple compare latches may be grouped together for simultaneous updates with the TBCLGRPx bits. When using groups, the CLLD bits of the lowest numbered TBxCCRn in the group determine the load event for each compare latch of the group, except when TBCLGRP = 3 (see Table 13-3). The CLLD bits of the controlling TBxCCRn must not be set to zero. When the CLLD bits of the controlling TBxCCRn are set to zero, all compare latches update immediately when their corresponding TBxCCRn is written; no compare latches are grouped.

Two conditions must exist for the compare latches to be loaded when grouped. First, all TBxCCRn registers of the group must be updated, even when new TBxCCRn data = old TBxCCRn data. Second, the load event must occur.

Table 13-3. Compare Latch Operating Modes

TBCLGRPx	Grouping	Update Control
00	None	Individual
01	TBxCL1+TBxCL2TBxCL3+TBxCL4+TBxCL5+TBxCL6	TBxCCR1 TBxCCR3 TBxCCR5
10	TBxCL1+TBxCL2+TBxCL3TBxCL4+TBxCL5+TBxCL6	TBxCCR1 TBxCCR4
11	TBxCL0+TBxCL1+TBxCL2+TBxCL3+TBxCL4+TBxCL5+TBxCL6	TBxCCR1

13.2.5 Output Unit

Each capture/compare block contains an output unit. The output unit is used to generate output signals, such as PWM signals. Each output unit has eight operating modes that generate signals based on the EQU0 and EQUn signals. The TBOUTH pin function can be used to put all Timer_B outputs into a high-impedance state. When the TBOUTH pin function is selected for the pin (corresponding PSEL bit is set, and port configured as input) and when the pin is pulled high, all Timer_B outputs are in a high-impedance state.

13.2.5.1 Output Modes

The output modes are defined by the OUTMOD bits and are described in Table 13-4. The OUTn signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0 because EQUn = EQU0.

Table 13-4. Output Modes

OUTMOD	Mode	Description
000	Output	The output signal OUTn is defined by the OUT bit. The OUTn signal updates immediately when OUT is updated.
001	Set	The output is set when the timer *counts* to the TBxCLn value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer *counts* to the TBxCLn value. It is reset when the timer *counts* to the TBxCL0 value.
011	Set/Reset	The output is set when the timer *counts* to the TBxCLn value. It is reset when the timer *counts* to the TBxCL0 value.
100	Toggle	The output is toggled when the timer *counts* to the TBxCLn value. The output period is double the timer period.
101	Reset	The output is reset when the timer *counts* to the TBxCLn value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer *counts* to the TBxCLn value. It is set when the timer *counts* to the TBxCL0 value.
111	Reset/Set	The output is reset when the timer *counts* to the TBxCLn value. It is set when the timer *counts* to the TBxCL0 value.

Output Example – Timer in Up Mode

The OUTn signal is changed when the timer *counts* up to the TBxCLn value, and rolls from TBxCL0 to zero, depending on the output mode. An example is shown in Figure 13-12 using TBxCL0 and TBxCL1.

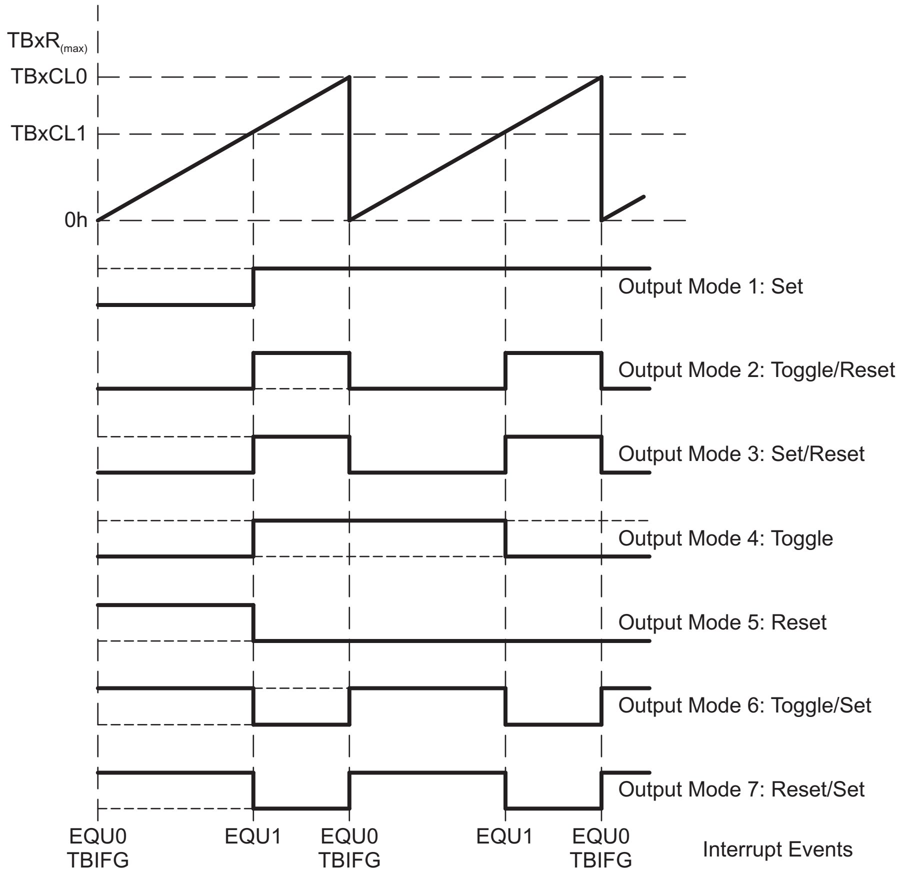


Figure 13-12. Output Example – Timer in Up Mode

Output Example – Timer in Continuous Mode

The OUTn signal is changed when the timer reaches the TBxCLn and TBxCL0 values, depending on the output mode. An example is shown in Figure 13-13 using TBxCL0 and TBxCL1.

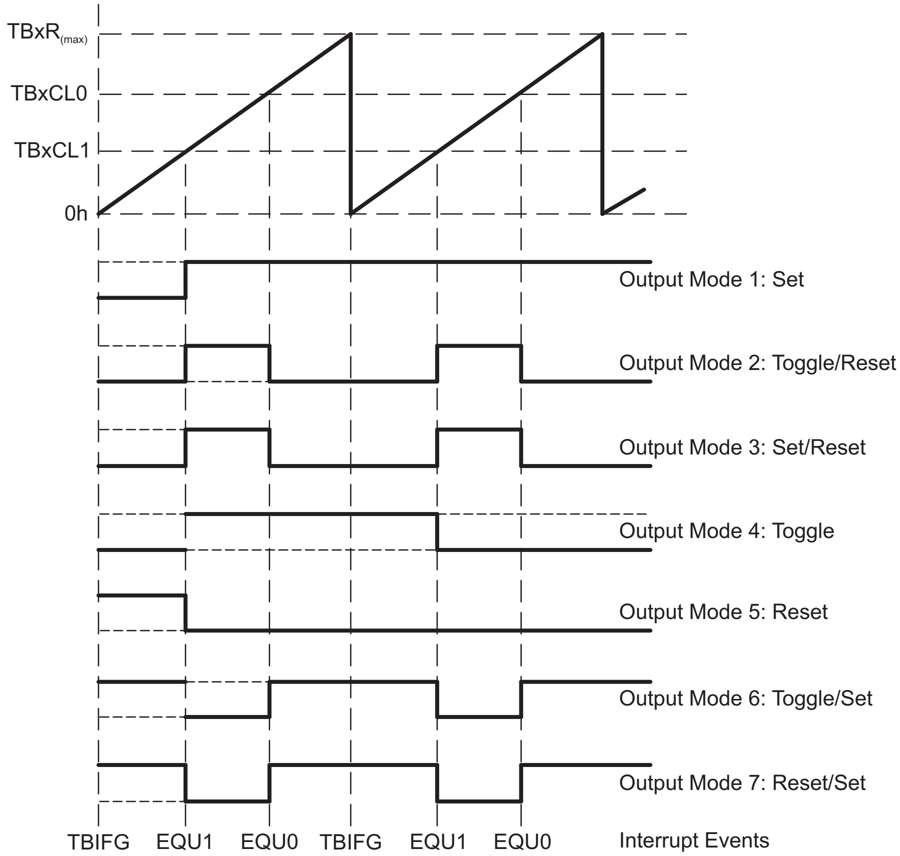


Figure 13-13. Output Example – Timer in Continuous Mode

Output Example – Timer in Up/Down Mode

The OUTn signal changes when the timer equals TBxCLn in either count direction and when the timer equals TBxCL0, depending on the output mode. An example is shown in Figure 13-14 using TBxCL0 and TBxCL3.

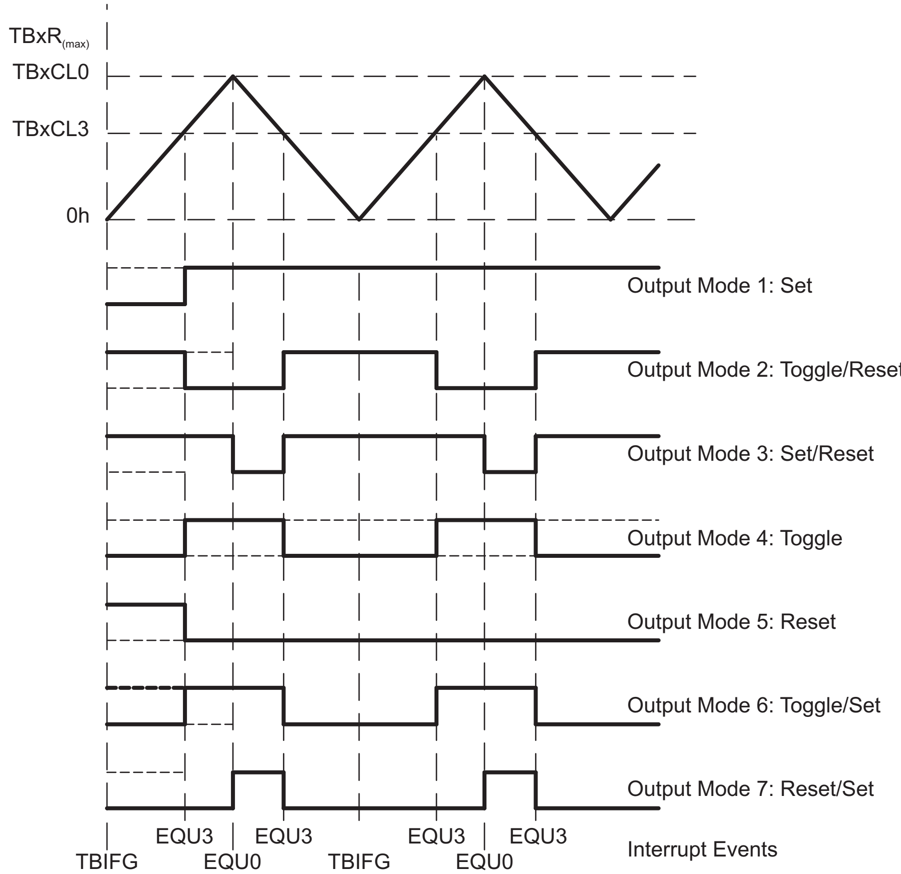


Figure 13-14. Output Example – Timer in Up/Down Mode

NOTE: Switching between output modes

When switching between output modes, one of the OUTMOD bits should remain set during the transition, unless switching to mode 0. Otherwise, output glitching can occur because a NOR gate decodes output mode 0. A safe method for switching between output modes is to use output mode 7 as a transition state:

```
BIS  #OUTMOD_7,&TBCCTLx  ; Set output mode=7
BIC  #OUTMOD,&TBCCTLx   ; Clear unwanted bits
```

13.2.6 *Timer_B Interrupts*

Two interrupt vectors are associated with the 16-bit Timer_B module:

- TBxCCR0 interrupt vector for TBxCCR0 CCIFG
- TBIV interrupt vector for all other CCIFG flags and TBIFG

In capture mode, any CCIFG flag is set when a timer value is captured in the associated TBxCCRn register. In compare mode, any CCIFG flag is set when TBxR *counts* to the associated TBxCLn value. Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set.

13.2.6.1 TBxCCR0 Interrupt Vector

The TBxCCR0 CCIFG flag has the highest Timer_B interrupt priority and has a dedicated interrupt vector (see Figure 13-15). The TBxCCR0 CCIFG flag is automatically reset when the TBxCCR0 interrupt request is serviced.

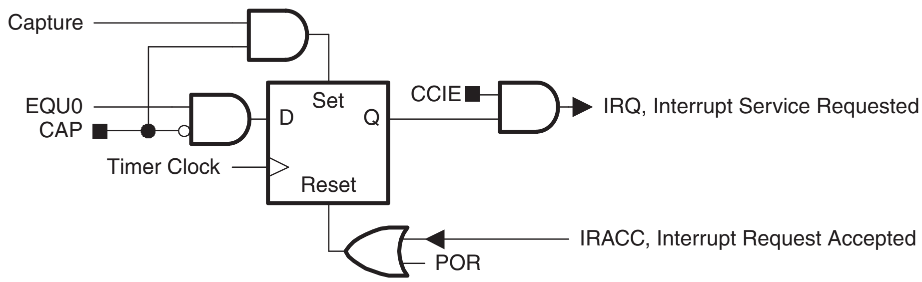


Figure 13-15. Capture/Compare TBxCCR0 Interrupt Flag

13.2.6.2 TBxIV, Interrupt Vector Generator

The TBIFG flag and TBxCCRn CCIFG flags (excluding TBxCCR0 CCIFG) are prioritized and combined to source a single interrupt vector. The interrupt vector register TBxIV is used to determine which flag requested an interrupt.

The highest-priority enabled interrupt (excluding TBxCCR0 CCIFG) generates a number in the TBxIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Timer_B interrupts do not affect the TBxIV value.

Any access, read or write, of the TBxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, if the TBxCCR1 and TBxCCR2 CCIFG flags are set when the interrupt service routine accesses the TBxIV register, TBxCCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TBxCCR2 CCIFG flag generates another interrupt.

13.2.6.3 TBxIV, Interrupt Handler Examples

The following software example shows the recommended use of TBxIV and the handling overhead. The TBxIV value is added to the PC to automatically jump to the appropriate routine. The example assumes a single instantiation of the largest timer configuration available.

The numbers at the right margin show the necessary CPU clock cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- Capture/compare block CCR0: 11 cycles
- Capture/compare blocks CCR1 to CCR6: 16 cycles
- Timer overflow TBIFG: 14 cycles

The following software example shows the recommended use of TBxIV for Timer_B3.

```
; Interrupt handler for TB0CCR0 CCIFG.                      Cycles
CCIFG_0_HND
;        ...         ; Start of handler Interrupt latency    6
         RETI                                                5

; Interrupt handler for TB0IFG, TB0CCR1 through TB0CCR6 CCIFG.

TB0_HND      ...               ; Interrupt latency           6
         ADD      &TB0IV,PC    ; Add offset to Jump table    3
         RETI                  ; Vector  0: No interrupt     5
         JMP      CCIFG_1_HND  ; Vector  2: TB0CCR1          2
         JMP      CCIFG_2_HND  ; Vector  4: TB0CCR2          2
         JMP      CCIFG_3_HND  ; Vector  6: TB0CCR3          2
         JMP      CCIFG_4_HND  ; Vector  8: TB0CCR4          2
         JMP      CCIFG_5_HND  ; Vector 10: TB0CCR5          2
         JMP      CCIFG_6_HND  ; Vector 12: TB0CCR6          2

TB0IFG_HND                     ; Vector 14: TB0IFG Flag
         ...                   ; Task starts here
         RETI                                                5

CCIFG_6_HND                    ; Vector 12: TB0CCR6
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_5_HND                    ; Vector 10: TB0CCR5
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_4_HND                    ; Vector 8: TB0CCR4
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_3_HND                    ; Vector 6: TB0CCR3
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_2_HND                    ; Vector 4: TB0CCR2
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_1_HND                    ; Vector 2: TB0CCR1
         ...                   ; Task starts here
         RETI                  ; Back to main program        5
```

13.3 Timer_B Registers

The Timer_B registers are listed in Table 13-5. The base address can be found in the device-specific data sheet. The address offset is listed in Table 13-5.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 13-5. Timer_B Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Timer_B Control	TBxCTL	Read/write	Word	00h	0000h
	TBxCTL_L	Read/write	Byte	00h	00h
	TBxCTL_H	Read/write	Byte	01h	00h
Timer_B Capture/Compare Control 0	TBxCCTL0	Read/write	Word	02h	0000h
	TBxCCTL0_L	Read/write	Byte	02h	00h
	TBxCCTL0_H	Read/write	Byte	03h	00h
Timer_B Capture/Compare Control 1	TBxCCTL1	Read/write	Word	04h	0000h
	TBxCCTL1_L	Read/write	Byte	04h	00h
	TBxCCTL1_H	Read/write	Byte	05h	00h
Timer_B Capture/Compare Control 2	TBxCCTL2	Read/write	Word	06h	0000h
	TBxCCTL2_L	Read/write	Byte	06h	00h
	TBxCCTL2_H	Read/write	Byte	07h	00h
Timer_B Capture/Compare Control 3	TBxCCTL3	Read/write	Word	08h	0000h
	TBxCCTL3_L	Read/write	Byte	08h	00h
	TBxCCTL3_H	Read/write	Byte	09h	00h
Timer_B Capture/Compare Control 4	TBxCCTL4	Read/write	Word	0Ah	0000h
	TBxCCTL4_L	Read/write	Byte	0Ah	00h
	TBxCCTL4_H	Read/write	Byte	0Bh	00h
Timer_B Capture/Compare Control 5	TBxCCTL5	Read/write	Word	0Ch	0000h
	TBxCCTL5_L	Read/write	Byte	0Ch	00h
	TBxCCTL5_H	Read/write	Byte	0Dh	00h
Timer_B Capture/Compare Control 6	TBxCCTL6	Read/write	Word	0Eh	0000h
	TBxCCTL6_L	Read/write	Byte	0Eh	00h
	TBxCCTL6_H	Read/write	Byte	0Fh	00h
Timer_B Counter	TBxR	Read/write	Word	10h	0000h
	TBxR_L	Read/write	Byte	10h	00h
	TBxR_H	Read/write	Byte	11h	00h
Timer_B Capture/Compare 0	TBxCCR0	Read/write	Word	12h	0000h
	TBxCCR0_L	Read/write	Byte	12h	00h
	TBxCCR0_H	Read/write	Byte	13h	00h
Timer_B Capture/Compare 1	TBxCCR1	Read/write	Word	14h	0000h
	TBxCCR1_L	Read/write	Byte	14h	00h
	TBxCCR1_H	Read/write	Byte	15h	00h
Timer_B Capture/Compare 2	TBxCCR2	Read/write	Word	16h	0000h
	TBxCCR2_L	Read/write	Byte	16h	00h
	TBxCCR2_H	Read/write	Byte	17h	00h

Table 13-5. Timer_B Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Timer_B Capture/Compare 3	TBxCCR3	Read/write	Word	18h	0000h
	TBxCCR3_L	Read/write	Byte	18h	00h
	TBxCCR3_H	Read/write	Byte	19h	00h
Timer_B Capture/Compare 4	TBxCCR4	Read/write	Word	1Ah	0000h
	TBxCCR4_L	Read/write	Byte	1Ah	00h
	TBxCCR4_H	Read/write	Byte	1Bh	00h
Timer_B Capture/Compare 5	TBxCCR5	Read/write	Word	1Ch	0000h
	TBxCCR5_L	Read/write	Byte	1Ch	00h
	TBxCCR5_H	Read/write	Byte	1Dh	00h
Timer_B Capture/Compare 6	TBxCCR6	Read/write	Word	1Eh	0000h
	TBxCCR6_L	Read/write	Byte	1Eh	00h
	TBxCCR6_H	Read/write	Byte	1Fh	00h
Timer_B Interrupt Vector	TBxIV	Read only	Word	2Eh	0000h
	TBxIV_L	Read only	Byte	2Eh	00h
	TBxIV_H	Read only	Byte	2Fh	00h
Timer_B Expansion 0	TBxEX0	Read/write	Word	20h	0000h
	TBxEX0_L	Read/write	Byte	20h	00h
	TBxEX0_H	Read/write	Byte	21h	00h

Timer_B Control Register (TBxCTL)

15	14	13	12	11	10	9	8
Unused	**TBCLGRPx**		**CNTL**		**Unused**	**TBSSEL**	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ID		**MC**		**Unused**	**TBCLR**	**TBIE**	**TBIFG**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Unused Bit 15 Unused

TBCLGRP Bits 14-13 TBxCLn group

- 00 Each TBxCLn latch loads independently.
- 01 TBxCL1+TBxCL2 (TBxCCR1 CLLD bits control the update)
  TBxCL3+TBxCL4 (TBxCCR3 CLLD bits control the update)
  TBxCL5+TBxCL6 (TBxCCR5 CLLD bits control the update)
  TBxCL0 independent
- 10 TBxCL1+TBxCL2+TBxCL3 (TBxCCR1 CLLD bits control the update)
  TBxCL4+TBxCL5+TBxCL6 (TBxCCR4 CLLD bits control the update)
  TBxCL0 independent
- 11 TBxCL0+TBxCL1+TBxCL2+TBxCL3+TBxCL4+TBxCL5+TBxCL6 (TBxCCR1 CLLD bits control the update)

CNTL Bits 12-11 Counter length

- 00 16-bit, $TBxR_{(max)}$ = 0FFFFh
- 01 12-bit, $TBxR_{(max)}$ = 0FFFh
- 10 10-bit, $TBxR_{(max)}$ = 03FFh
- 11 8-bit, $TBxR_{(max)}$ = 0FFh

Unused Bit 10 Unused

TBSSEL Bits 9-8 Timer_B clock source select

- 00 TBxCLK
- 01 ACLK
- 10 SMCLK
- 11 Inverted TBxCLK

ID Bits 7-6 Input divider. These bits, along with the IDEX bits, select the divider for the input clock.

- 00 /1
- 01 /2
- 10 /4
- 11 /8

MC Bits 5-4 Mode control. Setting MC = 00h when Timer_B is not in use conserves power.

- 00 Stop mode: Timer is halted
- 01 Up mode: Timer counts up to TBxCL0
- 10 Continuous mode: Timer counts up to the value set by CNTL
- 11 Up/down mode: Timer counts up to TBxCL0 and down to 0000h

Unused Bit 3 Unused

TBCLR Bit 2 Timer_B clear. Setting this bit resets TBxR, the timer clock divider, and the count direction. The TBCLR bit is automatically reset and is always read as zero.

TBIE Bit 1 Timer_B interrupt enable. This bit enables the TBIFG interrupt request.

- 0 Interrupt disabled
- 1 Interrupt enabled

TBIFG Bit 0 Timer_B interrupt flag

- 0 No interrupt pending
- 1 Interrupt pending

Timer_B Counter Register (TBxR)

15	14	13	12	11	10	9	8
TBxR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TBxR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TBxR	Bits 15-0	Timer_B register. The TBxR register is the count of Timer_B.

Capture/Compare Control Register (TBxCCTLn)

15	14	13	12	11	10	9	8
CM		**CCIS**		**SCS**	**CLLD**		**CAP**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
OUTMOD			**CCIE**	**CCI**	**OUT**	**COV**	**CCIFG**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

CM Bits 15-14 Capture mode
- 00 No capture
- 01 Capture on rising edge
- 10 Capture on falling edge
- 11 Capture on both rising and falling edges

CCIS Bits 13-12 Capture/compare input select. These bits select the TBxCCRn input signal. See the device-specific data sheet for specific signal connections.
- 00 CCIxA
- 01 CCIxB
- 10 GND
- 11 V_{CC}

SCS Bit 11 Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.
- 0 Asynchronous capture
- 1 Synchronous capture

CLLD Bits 10-9 Compare latch load. These bits select the compare latch load event.
- 00 TBxCLn loads on write to TBxCCRn
- 01 TBxCLn loads when TBxR *counts* to 0
- 10 TBxCLn loads when TBxR *counts* to 0 (up or continuous mode)
  TBxCLn loads when TBxR *counts* to TBxCL0 or to 0 (up/down mode)
- 11 TBxCLn loads when TBxR *counts* to TBxCLn

CAP Bit 8 Capture mode
- 0 Compare mode
- 1 Capture mode

OUTMOD Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TBxCL0 because EQUn = EQU0.
- 000 OUT bit value
- 001 Set
- 010 Toggle/reset
- 011 Set/reset
- 100 Toggle
- 101 Reset
- 110 Toggle/set
- 111 Reset/set

CCIE Bit 4 Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.
- 0 Interrupt disabled
- 1 Interrupt enabled

CCI Bit 3 Capture/compare input. The selected input signal can be read by this bit.

OUT Bit 2 Output. For output mode 0, this bit directly controls the state of the output.
- 0 Output low
- 1 Output high

COV Bit 1 Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.
- 0 No capture overflow occurred
- 1 Capture overflow occurred

CCIFG Bit 0 Capture/compare interrupt flag
- 0 No interrupt pending
- 1 Interrupt pending

Timer_B Interrupt Vector Register (TBxIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**		**TBIV**		**0**
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

TBIV Bits 15-0 Timer_B interrupt vector value

TBIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Capture/compare 1	TBxCCR1 CCIFG	Highest
04h	Capture/compare 2	TBxCCR2 CCIFG	
06h	Capture/compare 3	TBxCCR3 CCIFG	
08h	Capture/compare 4	TBxCCR4 CCIFG	
0Ah	Capture/compare 5	TBxCCR5 CCIFG	
0Ch	Capture/compare 6	TBxCCR6 CCIFG	
0Eh	Timer overflow	TBxCTL TBIFG	Lowest

Timer_B Expansion Register 0 (TBxEX0)

15	14	13	12	11	10	9	8
Unused	**Unused**	**Unused**	**Unused**	**Unused**	**Unused**	**Unused**	**Unused**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**Unused**	**Unused**		**IDEX**	
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

Unused Bits 15-3 Unused. Read only. Always read as 0.

IDEX Bits 2-0 Input divider expansion. These bits along with the ID bits select the divider for the input clock.

000	/1
001	/2
010	/3
011	/4
100	/5
101	/6
110	/7
111	/8

Real-Time Clock (RTC_A)

The Real-Time Clock (RTC_A) module provides clock counters with a calendar, a flexible programmable alarm, and calibration. This chapter describes the RTC_A module.

Topic **Page**

14.1 RTC_A Introduction **384**
14.2 RTC_A Operation **386**
14.3 Real-Time Clock Registers **391**

14.1 RTC_A Introduction

The RTC_A module provides a real-time clock and calendar function that can also be configured as a general-purpose counter.

RTC_A features include:

- Configurable for real-time clock with calendar function or general-purpose counter
- Provides seconds, minutes, hours, day of week, day of month, month, and year in real-time clock with calendar function
- Interrupt capability
- Selectable BCD or binary format in real-time clock mode
- Programmable alarms in real-time clock mode
- Calibration logic for time offset correction in real-time clock mode

The RTC_A block diagram is shown in Figure 14-1.

NOTE: Real-time clock initialization

Most RTC_A module registers have no initial condition. These registers must be configured by user software before use.

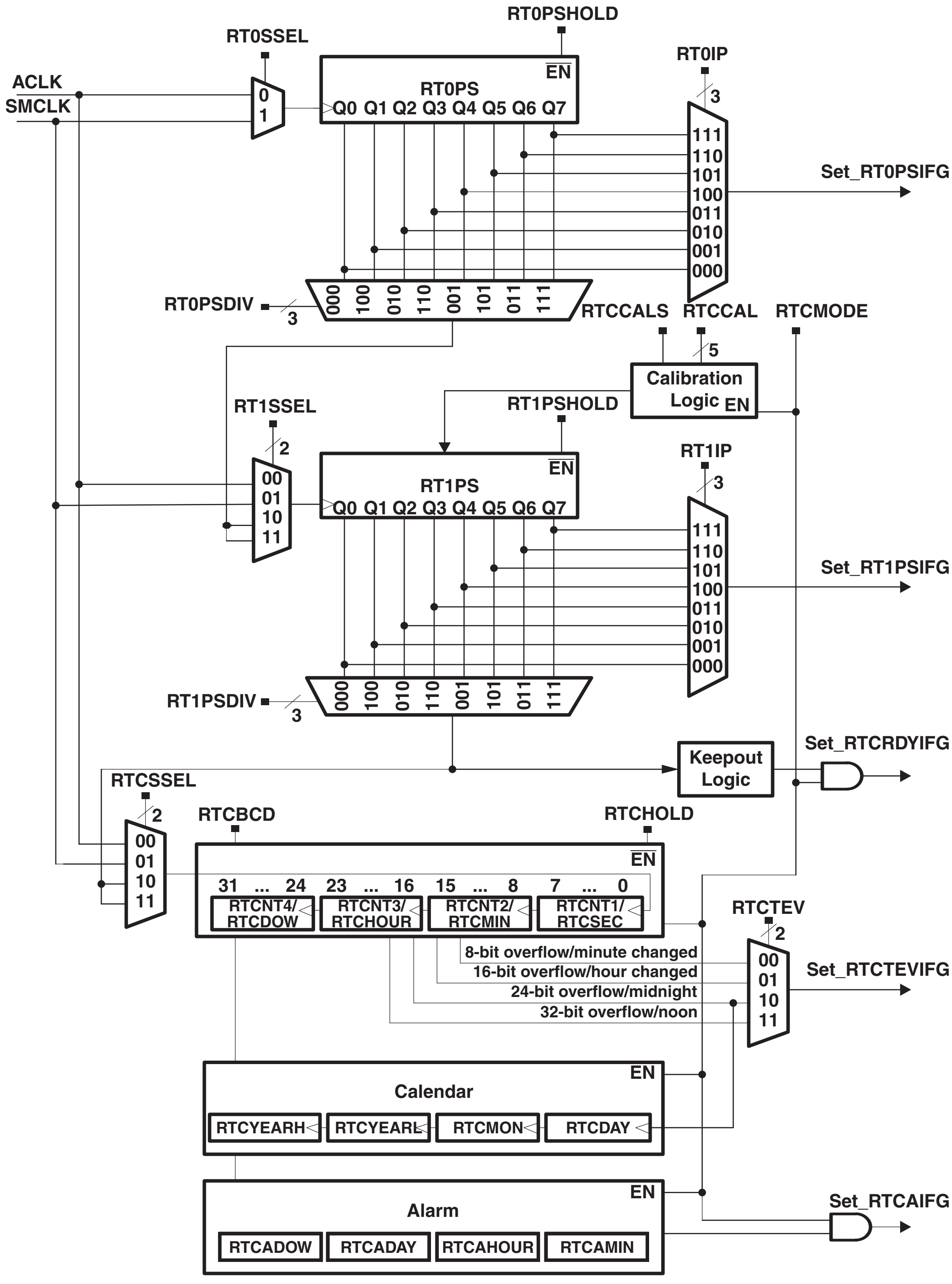


Figure 14-1. RTC_A

14.2 RTC_A Operation

The RTC_A module can be configured as a real-time clock with calendar function (calendar mode) or as a 32-bit general purpose counter (counter mode) with the RTCMODE bit.

14.2.1 Counter Mode

Counter mode is selected when RTCMODE is reset. In this mode, a 32-bit counter is provided that is directly accessible by software. Switching from calendar mode to counter mode resets the count value (RTCNT1, RTCNT2, RTCNT3, RTCNT4), as well as the prescale counters (RT0PS, RT1PS).

The clock to increment the counter can be sourced from ACLK, SMCLK, or prescaled versions of ACLK or SMCLK. Prescaled versions of ACLK or SMCLK are sourced from the prescale dividers (RT0PS and RT1PS). RT0PS and RT1PS output /2, /4, /8, 16, /32, /64, /128, and /256 versions of ACLK and SMCLK, respectively. The output of RT0PS can be cascaded with RT1PS. The cascaded output can be used as a clock source input to the 32-bit counter.

Four individual 8-bit counters are cascaded to provide the 32-bit counter. This provides 8-bit, 16-bit, 24-bit, or 32-bit overflow intervals of the counter clock. The RTCTEV bits select the respective trigger event. An RTCTEV event can trigger an interrupt by setting the RTCTEVIE bit. Each counter, RTCNT1 through RTCNT4, is individually accessible and may be written to.

RT0PS and RT1PS can be configured as two 8-bit counters or cascaded into a single 16-bit counter. RT0PS and RT1PS can be halted on an individual basis by setting their respective RT0PSHOLD and RT1PSHOLD bits. When RT0PS is cascaded with RT1PS, setting RT0PSHOLD causes both RT0PS and RT1PS to be halted. The 32-bit counter can be halted several ways depending on the configuration. If the 32-bit counter is sourced directly from ACLK or SMCLK, it can be halted by setting RTCHOLD. If it is sourced from the output of RT1PS, it can be halted by setting RT1PSHOLD or RTCHOLD. Finally, if it is sourced from the cascaded outputs of RT0PS and RT1PS, it can be halted by setting RT0PSHOLD, RT1PSHOLD, or RTCHOLD.

NOTE: Accessing the RTCNT1, RTCNT2, RTCNT3, RTCNT4, RT0PS, RT1PS registers

When the counter clock is asynchronous to the CPU clock, any read from any RTCNT1, RTCNT2, RTCNT3, RTCNT4, RT0PS, or RT1PS register should occur while the counter is not operating. Otherwise, the results may be unpredictable. Alternatively, the counter may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to these registers takes effect immediately.

14.2.2 Calendar Mode

Calendar mode is selected when RTCMODE is set. In calendar mode, the RTC_A module provides seconds, minutes, hours, day of week, day of month, month, and year in selectable BCD or hexadecimal format. The calendar includes a leap-year algorithm that considers all years evenly divisible by four as leap years. This algorithm is accurate from the year 1901 through 2099.

14.2.2.1 Real-Time Clock and Prescale Dividers

The prescale dividers, RT0PS and RT1PS, are automatically configured to provide a 1-s clock interval for the RTC_A. RT0PS is sourced from ACLK. ACLK must be set to 32768 Hz (nominal) for proper RTC_A calendar operation. RT1PS is cascaded with the output ACLK/256 of RT0PS. The RTC_A is sourced with the /128 output of RT1PS, thereby providing the required 1-s interval. Switching from counter to calendar mode clears the seconds, minutes, hours, day-of-week, and year counts and sets day-of-month and month counts to 1. In addition, RT0PS and RT1PS are cleared.

When RTCBCD = 1, BCD format is selected for the calendar registers. The format must be selected before the time is set. Changing the state of RTCBCD clears the seconds, minutes, hours, day-of-week, and year counts and sets day-of-month and month counts to 1. In addition, RT0PS and RT1PS are cleared.

In calendar mode, the RT0SSEL, RT1SSEL, RT0PSDIV, RT1PSDIV, RT0PSHOLD, RT1PSHOLD, and RTCSSEL bits are don't care. Setting RTCHOLD halts the real-time counters and prescale counters, RT0PS and RT1PS.

14.2.2.2 Real-Time Clock Alarm Function

The RTC_A module provides for a flexible alarm system. There is a single user-programmable alarm that can be programmed based on the settings contained in the alarm registers for minutes, hours, day of week, and day of month. The user-programmable alarm function is only available in the calendar mode of operation.

Each alarm register contains an alarm enable (AE) bit that can be used to enable the respective alarm register. By setting AE bits of the various alarm registers, a variety of alarm events can be generated.

- Example 1: A user wishes to set an alarm every hour at 15 minutes past the hour; i.e., 00:15:00, 01:15:00, 02:15:00, etc. This is possible by setting RTCAMIN to 15. By setting the AE bit of the RTCAMIN and clearing all other AE bits of the alarm registers, the alarm is enabled. When enabled, the AF is set when the count transitions from 00:14:59 to 00:15:00, 01:14:59 to 01:15:00, 02:14:59 to 02:15:00, etc.
- Example 2: A user wishes to set an alarm every day at 04:00:00. This is possible by setting RTCAHOUR to 4. By setting the AE bit of the RTCHOUR and clearing all other AE bits of the alarm registers, the alarm is enabled. When enabled, the AF is set when the count transitions from 03:59:59 to 04:00:00.
- Example 3: A user wishes to set an alarm for 06:30:00. RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCAHOUR and RTCAMIN, the alarm is enabled. Once enabled, the AF is set when the the time count transitions from 06:29:59 to 06:30:00. In this case, the alarm event occurs every day at 06:30:00.
- Example 4: A user wishes to set an alarm every Tuesday at 06:30:00. RTCADOW would be set to 2, RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCADOW, RTCAHOUR and RTCAMIN, the alarm is enabled. Once enabled, the AF is set when the the time count transitions from 06:29:59 to 06:30:00 and the RTCDOW transitions from 1 to 2.
- Example 5: A user wishes to set an alarm the fifth day of each month at 06:30:00. RTCADAY would be set to 5, RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCADAY, RTCAHOUR and RTCAMIN, the alarm is enabled. Once enabled, the AF is set when the the time count transitions from 06:29:59 to 06:30:00 and the RTCDAY equals 5.

NOTE: Invalid alarm settings

Invalid alarm settings are not checked via hardware. It is the user's responsibility to ensure that valid alarm settings are entered.

NOTE: Invalid time and date values

Writing of invalid date and/or time information or data values outside the legal ranges specified in the RTCSEC, RTCMIN, RTCHOUR, RTCDAY, RTCDOW, RTCYEARH, RTCYEARL, RTCAMIN, RTCAHOUR, RTCADAY, and RTCADOW registers can result in unpredictable behavior.

NOTE: Setting the alarm

To prevent potential erroneous alarm conditions from occurring, the alarms should be disabled by clearing the RTCAIE, RTCAIFG, and AE bits prior to writing new time values to the RTC time registers.

14.2.2.3 Reading or Writing Real-Time Clock Registers in Calendar Mode

Because the system clock may be asynchronous to the RTC_A clock source, special care must be taken when accessing the real-time clock registers.

In calendar mode, the real-time clock registers are updated once per second. To prevent reading any real-time clock register at the time of an update, which could result in an invalid time being read, a keepout window is provided. The keepout window is centered approximately -128/32768 s around the update transition. The read-only RTCRDY bit is reset during the keepout window period and set outside the keepout the window period. Any read of the clock registers while RTCRDY is reset is considered to be potentially invalid, and the time read should be ignored.

An easy way to safely read the real-time clock registers is to use the RTCRDYIFG interrupt flag. Setting RTCRDYIE enables the RTCRDYIFG interrupt. Once enabled, an interrupt is generated based on the rising edge of the RTCRDY bit, causing the RTCRDYIFG to be set. At this point, the application has nearly a complete second to safely read any or all of the real-time clock registers. This synchronization process prevents reading the time value during transition. The RTCRDYIFG flag is reset automatically when the interrupt is serviced, or can be reset with software.

In counter mode, the RTCRDY bit remains reset. RTCRDYIE is a don't care and RTCRDYIFG remains reset.

NOTE: Reading or writing real-time clock registers

When the counter clock is asynchronous to the CPU clock, any read from any RTCSEC, RTCMIN, RTCHOUR, RTCDOW, RTCDAY, RTCMON, RTCYEARL, or RTCYEARH register while the RTCRDY is reset may result in invalid data being read. To safely read the counting registers, either polling of the RTCRDY bit or the synchronization procedure previously described can be used. Alternatively, the counter register can be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Reading the RT0PS and RT1PS can only be handled by reading the registers multiple times and a majority vote taken in software to determine the correct reading or by halting the counters.

Any write to any counting register takes effect immediately. However, the clock is stopped during the write. In addition, RT0PS and RT1PS registers are reset. This could result in losing up to 1 s during a write. Writing of data outside the legal ranges or invalid time stamp combinations results in unpredictable behavior.

14.2.3 Real-Time Clock Interrupts

The RTC_A module has five interrupt sources available, each with independent enables and flags.

14.2.3.1 Real-Time Clock Interrupts in Calendar Mode

In calendar mode, five sources for interrupts are available, namely RT0PSIFG, RT1PSIFG, RTCRDYIFG, RTCTEVIFG, and RTCAIFG. These flags are prioritized and combined to source a single interrupt vector. The interrupt vector register (RTCIV) is used to determine which flag requested an interrupt.

The highest-priority enabled interrupt generates a number in the RTCIV register (see register description). This number can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled RTC interrupts do not affect the RTCIV value.

Any access, read or write, of the RTCIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. In addition, all flags can be cleared via software.

The user-programmable alarm event sources the real-time clock interrupt, RTCAIFG. Setting RTCAIE enables the interrupt. In addition to the user-programmable alarm, the RTC_A module provides for an interval alarm that sources real-time clock interrupt, RTCTEVIFG. The interval alarm can be selected to cause an alarm event when RTCMIN changed or RTCHOUR changed, every day at midnight (00:00:00) or every day at noon (12:00:00). The event is selectable with the RTCTEV bits. Setting the RTCTEVIE bit enables the interrupt.

The RTCRDY bit sources the real-time clock interrupt, RTCRDYIFG, and is useful in synchronizing the read of time registers with the system clock. Setting the RTCRDYIE bit enables the interrupt.

RT0PSIFG can be used to generate interrupt intervals selectable by the RT0IP bits. In calendar mode, RT0PS is sourced with ACLK at 32768 Hz, so intervals of 16384 Hz, 8192 Hz, 4096 Hz, 2048 Hz, 1024 Hz, 512 Hz, 256 Hz, or 128 Hz are possible. Setting the RT0PSIE bit enables the interrupt.

RT1PSIFG can generate interrupt intervals selectable by the RT1IP bits. In calendar mode, RT1PS is sourced with the output of RT0PS, which is 128 Hz (32768/256 Hz). Therefore, intervals of 64 Hz, 32 Hz, 16 Hz, 8 Hz, 4 Hz, 2 Hz, 1 Hz, or 0.5 Hz are possible. Setting the RT1PSIE bit enables the interrupt.

14.2.3.2 Real-Time Clock Interrupts in Counter Mode

In counter mode, three interrupt sources are available: RT0PSIFG, RT1PSIFG, and RTCTEVIFG. RTCAIFG and RTCRDYIFG are cleared. RTCRDYIE and RTCAIE are don't care.

RT0PSIFG can be used to generate interrupt intervals selectable by the RT0IP bits. In counter mode, RT0PS is sourced with ACLK or SMCLK, so divide ratios of /2, /4, /8, /16, /32, /64, /128, and /256 of the respective clock source are possible. Setting the RT0PSIE bit enables the interrupt.

RT1PSIFG can be used to generate interrupt intervals selectable by the RT1IP bits. In counter mode, RT1PS is sourced with ACLK, SMCLK, or the output of RT0PS, so divide ratios of /2, /4, /8, /16, /32, /64, /128, and /256 of the respective clock source are possible. Setting the RT1PSIE bit enables the interrupt.

The RTC_A module provides for an interval timer that sources real-time clock interrupt, RTCTEVIFG. The interval timer can be selected to cause an interrupt event when an 8-bit, 16-bit, 24-bit, or 32-bit overflow occurs within the 32-bit counter. The event is selectable with the RTCTEV bits. Setting the RTCTEVIE bit enables the interrupt.

RTCIV Software Example

The following software example shows the recommended use of RTCIV and the handling overhead. The RTCIV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself.

```
; Interrupt handler for RTC interrupt flags.
RTC_HND                     ; Interrupt latency        6
         ADD   &RTCIV,PC    ; Add offset to Jump table 3
         RETI               ; Vector 0: No interrupt   5
         JMP   RTCRDYIFG_HND  ; Vector 2: RTCRDYIFG    2
         JMP   RTCTEVIFG_HND  ; Vector 4: RTCTEVIFG    2
         JMP   RTCAIFG      ; Vector 6: RTCAIFG        5
         JMP   RT0PSIFG     ; Vector 8: RT0PSIFG       5
         JMP   RT1PSIFG     ; Vector A: RT1PSIFG       5
         RETI               ; Vector C: Reserved       5
RTCRDYIFG_HND               ; Vector 2: RTCRDYIFG Flag
          to                 ; Task starts here
         RETI                                          5
RTCTEVIFG_HND               ; Vector 4: RTCTEVIFG
          to                 ; Task starts here
         RETI               ; Back to main program     5
RTCAIFG_HND                 ; Vector 6: RTCAIFG
          to                 ; Task starts here
RT0PSIFG_HND                ; Vector 8: RT0PSIFG
          to                 ; Task starts here
RT1PSIFG_HND                ; Vector A: RT1PSIFG
          to                 ; Task starts here
```

14.2.4 Real-Time Clock Calibration

The RTC_A module has calibration logic that allows for adjusting the crystal frequency in +4-ppm or –2-ppm steps, allowing for higher time keeping accuracy from standard crystals.The RTCCAL bits are used to adjust the frequency. When RTCCALS is set, each RTCCAL LSB causes a +4-ppm adjustment. When RTCCALS is cleared, each RTCCAL LSB causes a –2-ppm adjustment. Calibration is only available in calendar mode. In counter mode (RTCMODE = 0), the calibration logic is disabled.

Calibration is accomplished by periodically adjusting the RT1PS counter based on the RTCCALS and RTCCALx settings. In calendar mode, the RT0PS divides the nominial 37268 Hz clock input by 256. A 64 minute period has 32768 cycles/sec * 60 sec/min * 64 min = 125829120 cycles. Therefore a -2 ppm reduction in frequency (down calibration) equates to adding an additional 256 cycles every 125829120 cycles (256/125829120 = 2.035 ppm). This is accomplished by holding the RT1PS counter for one additional clock of the RT0PS output within a 64 minute period. Similary, a +4 ppm increase in frequency (up calibration) equates to removing 512 cycles every 125829120 cycle (512/125829120 = 4.069ppm). This is accomplished by incrementing the RT1PS counter for two additional clocks of the RT0PS output within a 64 minute period. Each RTCCALx calibration bit causes either 256 clock cycles to be added every 64 minutes or 512 clock cycles to be subtracted every 64 minutes, giving a frequency adjustment of approximately -2 ppm or +4 ppm, respectively.

To calibrate the frequency, the RTCCLK output signal is available at a pin by setting the respective PxSEL bit (secondary function) along with PxDIR bit (output mode). The RTCCALF bits can be used to select the frequency rate of the RTCCLK output signal, either no signal, 512 Hz, 256 Hz, or 1 Hz. The basic flow is as follows:

1. Configure the RTCCLK pin.
2. Measure the RTCCLK output signal with an appropriate resolution frequency counter i.e. within the resolution required.
3. Compute the absolute error in ppm: Absolute Error (ppm) = $|10^6 \times (f_{MEASURED} - f_{RTCCLK}) / f_{RTCCLK}|$
4. Adjust the frequency, by performing the following:
   (a) If the frequency is too low, set RTCALS and apply the appropriate RTCCALx bits, where RTCCALx = (Absolute Error) / 4.069 rounded to the nearest integer.
   (b) If the frequency is too high, clear RTCALS and apply the appropriate RTCCALx bits, where RTCCALx = (Absolute Error) / 2.035 rounded to the nearest integer.

For example, say RTCCLK is output at a frequency of 512 Hz. The measured RTCCLK is 511.9658 Hz. The frequency error is approximately 66.8 ppm too low. To increase the frequency by 66.8 ppm, RTCCALS would be set, and RTCCAL would be set to 16 (66.8/4.069). Similarly, say the measured RTCCLK is 512.0125 Hz. The frequency error is approximately 24.4 ppm too high. To decrease the frequency by 24.4 ppm, RTCCALS would be cleared, and RTCCAL would be set to 12 (24.4/2.035).

The calibration will only correct initial offsets and does not adjust for temperature and aging effects. This can be handled by periodically measuring temperature and using the crystal's charateristic curve to adjust the ppm based on temperature as required. In counter mode (RTCMODE = 0), the calibration logic is disabled.

NOTE: Calibration output frequency

The 512-Hz and 256-Hz output frequencies observed at the RTCCLK pin are not affected by changes in the calibration settings since these output frequencies are generated prior to the calibration logic. The 1-Hz output frequency is affected by changes in the calibration settings. Since the frequency change is small and infrequent over a very long time interval, it can be difficult to observe.

14.3 Real-Time Clock Registers

The RTC_A module registers are listed in and Table 14-1. The base register for the RTC_A module registers can be found in the device-specific data sheet. The address offsets are given in Table 14-1.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 14-1. Real-Time Clock Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Real-Time Clock Control 0, 1	RTCCTL01	Read/write	Word	00h	4000h
Real-Time Clock Control 0	RTCCTL0 or RTCCTL01_L	Read/write	Byte	00h	00h
Real-Time Clock Control 1	RTCCTL1 or RTCCTL01_H	Read/write	Byte	01h	40h
Real-Time Clock Control 2, 3	RTCCTL23	Read/write	Word	02h	0000h
Real-Time Clock Control 2	RTCCTL2 or RTCCTL23_L	Read/write	Byte	02h	00h
Real-Time Clock Control 3	RTCCTL3 or RTCCTL23_H	Read/write	Byte	03h	00h
Real-Time Prescale Timer 0 Control	RTCPS0CTL	Read/write	Word	08h	0100h
	RTCPS0CTLL or RTCPS0CTL_L	Read/write	Byte	08h	00h
	RTCPS0CTLH or RTCPS0CTL_H	Read/write	Byte	09h	01h
Real-Time Prescale Timer 1 Control	RTCPS1CTL	Read/write	Word	0Ah	0100h
	RTCPS1CTLL or RTCPS1CTL_L	Read/write	Byte	0Ah	00h
	RTCPS0CTLH or RTCPS0CTL_H	Read/write	Byte	0Bh	01h
Real-Time Prescale Timer 0, 1 Counter	RTCPS	Read/write	Word	0Ch	undefined
Real-Time Prescale Timer 0 Counter	RT0PS or RTCPS_L	Read/write	Byte	0Ch	undefined
Real-Time Prescale Timer 1 Counter	RT1PS or RTCPS_H	Read/write	Byte	0Dh	undefined
Real Time Clock Interrupt Vector	RTCIV	Read	Word	0Eh	0000h
	RTCIV_L	Read	Byte	0Eh	00h
	RTCIV_H	Read	Byte	0Fh	00h
Real-Time Clock Seconds, Minutes/ Real-Time Counter 1, 2	RTCTIM0 or RTCNT12	Read/write	Word	10h	undefined
Real-Time Clock Seconds/ Real-Time Counter 1	RTCSEC /RTCNT1 or RTCTIM0_L	Read/write	Byte	10h	undefined
Real-Time Clock Minutes/ Real-Time Counter 2	RTCMIN/RTCNT2 or RTCTIM0_H	Read/write	Byte	11h	undefined
Real-Time Clock Hour, Day of Week/ Real-Time Counter 3, 4	RTCTIM1 or RTCNT34	Read/write	Word	12h	undefined
Real-Time Clock Hour/ Real-Time Counter 3	RTCHOUR/RTCNT3 or RTCTIM1_L	Read/write	Byte	12h	undefined
Real-Time Clock Day of Week/ Real-Time Counter 4	RTCDOWRTCNT4 or RTCTIM1_H	Read/write	Byte	13h	undefined

Table 14-1. Real-Time Clock Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Real-Time Clock Date	RTCDATE	Read/write	Word	14h	undefined
Real-Time Clock Day of Month	RTCDAY or RTCDATE_L	Read/write	Byte	14h	undefined
Real-Time Clock Month	RTCMON or RTCDATE_H	Read/write	Byte	15h	undefined
Real-Time Clock Year	RTCYEAR	Read/write	Word	16h	undefined
	RTCYEARL or RTCYEAR_L	Read/write	Byte	16h	undefined
	RTCYEARH or RTCYEAR_H	Read/write	Byte	17h	undefined
Real-Time Clock Minutes, Hour Alarm	RTCAMINHR	Read/write	Word	18h	undefined
Real-Time Clock Minutes Alarm	RTCAMIN or RTCAMINHR_L	Read/write	Byte	18h	undefined
Real-Time Clock Hours Alarm	RTCAHOUR or RTCAMINHR_H	Read/write	Byte	19h	undefined
Real-Time Clock Day of Week, Day of Month Alarm	RTCADOWDAY	Read/write	Word	1Ah	undefined
Real-Time Clock Day of Week Alarm	RTCADOW or RTCADOWDAY_L	Read/write	Byte	1Ah	undefined
Real-Time Clock Day of Month Alarm	RTCADAY or RTCADOWDAY_H	Read/write	Byte	1Bh	undefined

Real-Time Clock Control 0 Register (RTCCTL0)

7	6	5	4	3	2	1	0
Reserved	**RTCTEVIE**	**RTCAIE**	**RTCRDYIE**	**Reserved**	**RTCTEVIFG**	**RTCAIFG**	**RTCRDYIFG**
r0	rw-0	rw-0	rw-0	r0	rw-(0)	rw-(0)	rw-(0)

Reserved	Bit 7	Reserved. Always read as 0.
RTCTEVIE	Bit 6	Real-time clock time event interrupt enable 0 Interrupt not enabled 1 Interrupt enabled
RTCAIE	Bit 5	Real-time clock alarm interrupt enable. This bit remains cleared when in counter mode (RTCMODE = 0). 0 Interrupt not enabled 1 Interrupt enabled
RTCRDYIE	Bit 4	Real-time clock read ready interrupt enable 0 Interrupt not enabled 1 Interrupt enabled
Reserved	Bit 3	Reserved. Always read as 0.
RTCTEVIFG	Bit 2	Real-time clock time event flag 0 No time event occurred. 1 Time event occurred.
RTCAIFG	Bit 1	Real-time clock alarm flag. This bit remains cleared when in counter mode (RTCMODE = 0). 0 No time event occurred. 1 Time event occurred.
RTCRDYIFG	Bit 0	Real-time clock read ready flag 0 RTC cannot be read safely. 1 RTC can be read safely.

RTCCTL1, Real-Time Clock Control Register 1

7	6	5	4	3	2	1	0
RTCBCD	**RTCHOLD**	**RTCMODE**	**RTCRDY**	**RTCSSEL**		**RTCTEV**	
rw-(0)	rw-(1)	rw-(0)	r-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

RTCBCD Bit 7 Real-time clock BCD select. Selects BCD counting for real-time clock. Applies to calendar mode (RTCMODE = 1) only; setting is ignored in counter mode. Changing this bit clears seconds, minutes, hours, day of week, and year to 0 and sets day of month and month to 1. The real-time clock registers must be set by software afterwards.

- 0 Binary/hexadecimal code selected
- 1 BCD Binary coded decimal (BCD) code selected

RTCHOLD Bit 6 Real-time clock hold

- 0 Real-time clock (32-bit counter or calendar mode) is operational.
- 1 In counter mode (RTCMODE = 0), only the 32-bit counter is stopped. In calendar mode (RTCMODE = 1), the calendar is stopped as well as the prescale counters, RT0PS and RT1PS. RT0PSHOLD and RT1PSHOLD are don't care.

RTCMODE Bit 5 Real-time clock mode

- 0 32-bit counter mode
- 1 Calendar mode. Switching between counter and calendar mode resets the real-time clock/counter registers. Switching to calendar mode clears seconds, minutes, hours, day of week, and year to 0 and sets day of month and month to 1. The real-time clock registers must be set by software afterwards. RT0PS and RT1PS are also cleared.

RTCRDY Bit 4 Real-time clock ready

- 0 RTC time values in transition (calendar mode only)
- 1 RTC time values safe for reading (calendar mode only). This bit indicates when the real-time clock time values are safe for reading (calendar mode only). In counter mode, RTCRDY signal remains cleared.

RTCSSEL Bits 3-2 Real-time clock source select. Selects clock input source to the RTC/32-bit counter. In calendar mode, these bits are don't care. The clock input is automatically set to the output of RT1PS.

- 00 ACLK
- 01 SMCLK
- 10 Output from RT1PS
- 11 Output from RT1PS

RTCTEV Bits 1-0 Real-time clock time event

RTC Mode	RTCTEV	Interrupt Interval
Counter mode (RTCMODE = 0)	00	8-bit overflow
	01	16-bit overflow
	10	24-bit overflow
	11	32-bit overflow
Calendar mode (RTCMODE = 1)	00	Minute changed
	01	Hour changed
	10	Every day at midnight (00:00)
	11	Every day at noon (12:00)

Real-Time Clock Control 2 Register (RTCCTL2)

7	6	5	4	3	2	1	0
RTCCALS	**Reserved**	**RTCCAL**					
rw-(0)	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

RTCCALS	Bit 7	Real-time clock calibration sign 0 Frequency adjusted down 1 Frequency adjusted up
Reserved	Bit 6	Reserved. Always read as 0.
RTCCAL	Bits 5-0	Real-time clock calibration. Each LSB represents approximately +4-ppm (RTCCALS = 1) or a –2-ppm (RTCCALS = 0) adjustment in frequency.

Real-Time Clock Control 3 Register (RTCCTL3)

7	6	5	4	3	2	1	0
Reserved						**RTCCALF**	
r0	r0	r0	r0	r0	r0	rw-(0)	rw-(0)

Reserved	Bits 7-2	Reserved. Always read as 0.
RTCCALF	Bits 1-0	Real-time clock calibration frequency. Selects frequency output to RTCCLK pin for calibration measurement. The corresponding port must be configured for the peripheral module function. The RTCCLK is not available in counter mode and remains low, and the RTCCALF bits are don't care. 00 No frequency output to RTCCLK pin 01 512 Hz 10 256 Hz 11 1 Hz

Real-Time Clock Counter 1 Register (RTCNT1) – Counter Mode

7	6	5	4	3	2	1	0
RTCNT1							
rw	rw	rw	rw	rw	rw	rw	rw

RTCNT1	Bits 7-0	The RTCNT1 register is the count of RTCNT1.

Real-Time Clock Counter 2 Register (RTCNT2) – Counter Mode

7	6	5	4	3	2	1	0
RTCNT2							
rw	rw	rw	rw	rw	rw	rw	rw

RTCNT2	Bits 7-0	The RTCNT2 register is the count of RTCNT2.

Real-Time Clock Counter 3 Register (RTCNT3) – Counter Mode

7	6	5	4	3	2	1	0
RTCNT3							
rw	rw	rw	rw	rw	rw	rw	rw

RTCNT3	Bits 7-0	The RTCNT3 register is the count of RTCNT3.

Real-Time Clock Counter 4 Register (RTCNT4) – Counter Mode

7	6	5	4	3	2	1	0
RTCNT4							
rw	rw	rw	rw	rw	rw	rw	rw

RTCNT4	Bits 7-0	The RTCNT4 register is the count of RTCNT4.

Real-Time Clock Seconds Register (RTCSEC) – Calendar Mode With Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**Seconds (0 to 59)**					
r-0	r-0	rw	rw	rw	rw	rw	rw

Real-Time Clock Seconds Register (RTCSEC) – Calendar Mode With BCD Format

7	6	5	4	3	2	1	0
0	**Seconds – high digit (0 to 5)**			**Seconds – low digit (0 to 9)**			
r-0	rw	rw	rw	rw	rw	rw	rw

Real-Time Clock Minutes Register (RTCMIN) – Calendar Mode With Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**Minutes (0 to 59)**					
r-0	r-0	rw	rw	rw	rw	rw	rw

Real-Time Clock Minutes Register (RTCMIN) – Calendar Mode With BCD Format

7	6	5	4	3	2	1	0
0	**Minutes – high digit (0 to 5)**			**Minutes – low digit (0 to 9)**			
r-0	rw	rw	rw	rw	rw	rw	rw

Real-Time Clock Hours Register (RTCHOUR) – Calendar Mode With Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**0**	**Hours (0 to 24)**				
r-0	r-0	r-0	rw	rw	rw	rw	rw

Real-Time Clock Hours Register (RTCHOUR) – Calendar Mode With BCD Format

7	6	5	4	3	2	1	0
0	**0**	**Hours – high digit (0 to 2)**		**Hours – low digit (0 to 9)**			
r-0	r-0	rw	rw	rw	rw	rw	rw

Real-Time Clock Day of Week Register (RTCDOW) – Calendar Mode

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**Day of week (0 to 6)**		
r-0	r-0	r-0	r-0	r-0	rw	rw	rw

Real-Time Clock Day of Month Register (RTCDAY) – Calendar Mode With Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**0**	**Day of month (1 to 28, 29, 30, 31)**				
r-0	r-0	r-0	rw	rw	rw	rw	rw

Real-Time Clock Day of Month Register (RTCDAY) – Calendar Mode With BCD Format

7	6	5	4	3	2	1	0
0	**0**	**Day of month – high digit (0 to 3)**		**Day of month – low digit (0 to 9)**			
r-0	r-0	rw	rw	rw	rw	rw	rw

Real-Time Clock Month Register (RTCMON) – Calendar Mode With Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**Month (1 to 12)**			
r-0	r-0	r-0	r-0	rw	rw	rw	rw

Real-Time Clock Month Register (RTCMON) – Calendar Mode With BCD Format

7	6	5	4	3	2	1	0
0	**0**	**0**	**Month – high digit (0 to 3)**	**Month – low digit (0 to 9)**			
r-0	r-0	r-0	rw	rw	rw	rw	rw

Real-Time Clock Year Low-Byte Register (RTCYEARL) – Calendar Mode With Hexadecimal Format

7	6	5	4	3	2	1	0
Year – low byte of 0 to 4095							
rw	rw	rw	rw	rw	rw	rw	rw

Real-Time Clock Year Low-Byte Register (RTCYEARL) – Calendar Mode With BCD Format

7	6	5	4	3	2	1	0
Decade (0 to 9)				**Year – lowest digit (0 to 9)**			
rw	rw	rw	rw	rw	rw	rw	rw

Real-Time Clock Year High-Byte Register (RTCYEARH) – Calendar Mode With Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**Year – high byte of 0 to 4095**			
r-0	r-0	r-0	r-0	rw	rw	rw	rw

Real-Time Clock Year High-Byte Register (RTCYEARH) – Calendar Mode With BCD Format

7	6	5	4	3	2	1	0
0	**Century – high digit (0 to 4)**			**Century – low digit (0 to 9)**			
r-0	rw	rw	rw	rw	rw	rw	rw

Real-Time Clock Minutes Alarm Register (RTCAMIN) – Calendar Mode With Hexadecimal Format

7	6	5	4	3	2	1	0
AE	**0**	**Minutes (0 to 59)**					
rw-0	r-0	rw	rw	rw	rw	rw	rw

Real-Time Clock Minutes Alarm Register (RTCAMIN) – Calendar Mode With BCD Format

7	6	5	4	3	2	1	0
AE	**Minutes – high digit (0 to 5)**			**Minutes – low digit (0 to 9)**			
rw-0	rw	rw	rw	rw	rw	rw	rw

Real-Time Clock Hours Alarm Register (RTCAHOUR) – Calendar Mode With Hexadecimal Format

7	6	5	4	3	2	1	0
AE	**0**	**0**	**Hours (0 to 24)**				
rw-0	r-0	r-0	rw	rw	rw	rw	rw

Real-Time Clock Hours Alarm Register (RTCAHOUR) – Calendar Mode With BCD Format

7	6	5	4	3	2	1	0
AE	**0**	**Hours – high digit (0 to 2)**		**Hours – low digit (0 to 9)**			
rw-0	r-0	rw	rw	rw	rw	rw	rw

Real-Time Clock Day of Week Alarm Register (RTCADOW) – Calendar Mode

7	6	5	4	3	2	1	0
AE	**0**	**0**	**0**	**0**	**Day of week (0 to 6)**		
rw-0	r-0	r-0	r-0	r-0	rw	rw	rw

Real-Time Clock Day of Month Alarm Register (RTCADAY) – Calendar Mode With Hexadecimal Format

7	6	5	4	3	2	1	0
AE	**0**	**0**	**Day of month (1 to 28, 29, 30, 31)**				
rw-0	r-0	r-0	rw	rw	rw	rw	rw

Real-Time Clock Day of Month Alarm Register (RTCADAY) – Calendar Mode With BCD Format

7	6	5	4	3	2	1	0
AE	**0**	**Day of month – high digit (0 to 3)**		**Day of month – low digit (0 to 9)**			
rw-0	r-0	rw	rw	rw	rw	rw	rw

Real-Time Clock Prescale Timer 0 Control Register (RTCPS0CTL)

15	14	13	12	11	10	9	8
Reserved	**RT0SSEL**	**RT0PSDIV**			**Reserved**	**Reserved**	**RT0PSHOLD**
r0	rw-0	rw-0	rw-0	rw-0	r0	r0	rw-1

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**RT0IP**			**RT0PSIE**	**RT0PSIFG**
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-(0)

Reserved Bit 15 Reserved. Always read as 0.

RT0SSEL Bit 14 Prescale timer 0 clock source select. Selects clock input source to the RT0PS counter. In real-time clock calendar mode, these bits are don't care. RT0PS clock input is automatically set to ACLK. RT1PS clock input is automatically set to the output of RT0PS.

- 0 ACLK
- 1 SMCLK

RT0PSDIV Bits 13-11 Prescale timer 0 clock divide. These bits control the divide ratio of the RT0PS counter. In real-time clock calendar mode, these bits are don't care for RT0PS and RT1PS. RT0PS clock output is automatically set to /256. RT1PS clock output is automatically set to /128.

- 000 /2
- 001 /4
- 010 /8
- 011 /16
- 100 /32
- 101 /64
- 110 /128
- 111 /256

Reserved Bits 10-9 Reserved. Always read as 0.

RT0PSHOLD Bit 8 Prescale timer 0 hold. In real-time clock calendar mode, this bit is don't care. RT0PS is stopped via the RTCHOLD bit.

- 0 RT0PS is operational.
- 1 RT0PS is held.

Reserved Bits 7-5 Reserved. Always read as 0.

RT0IP Bits 4-2 Prescale timer 0 interrupt interval

- 000 /2
- 001 /4
- 010 /8
- 011 /16
- 100 /32
- 101 /64
- 110 /128
- 111 /256

RT0PSIE Bit 1 Prescale timer 0 interrupt enable

- 0 Interrupt not enabled
- 1 Interrupt enabled

RT0PSIFG Bit 0 Prescale timer 0 interrupt flag

- 0 No time event occurred.
- 1 Time event occurred.

Real-Time Clock Prescale Timer 1 Control Register (RTCPS1CTL)

15	14	13	12	11	10	9	8
RT1SSEL		**RT1PSDIV**			**Reserved**	**Reserved**	**RT1PSHOLD**
rw-0	rw-0	rw-0	rw-0	rw-0	r0	r0	rw-1

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**RT1IP**			**RT1PSIE**	**RT1PSIFG**
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-(0)

RT1SSEL Bits 15-14 Prescale timer 1 clock source select. Selects clock input source to the RT1PS counter. In real-time clock calendar mode, these bits are do not care. RT1PS clock input is automatically set to the output of RT0PS.

- 00 ACLK
- 01 SMCLK
- 10 Output from RT0PS
- 11 Output from RT0PS

RT1PSDIV Bits 13-11 Prescale timer 1 clock divide. These bits control the divide ratio of the RT0PS counter. In real-time clock calendar mode, these bits are don't care for RT0PS and RT1PS. RT0PS clock output is automatically set to /256. RT1PS clock output is automatically set to /128.

- 000 /2
- 001 /4
- 010 /8
- 011 /16
- 100 /32
- 101 /64
- 110 /128
- 111 /256

Reserved Bits 10-9 Reserved. Always read as 0.

RT1PSHOLD Bit 8 Prescale timer 1 hold. In real-time clock calendar mode, this bit is don't care. RT1PS is stopped via the RTCHOLD bit.

- 0 RT1PS is operational.
- 1 RT1PS is held.

Reserved Bits 7-5 Reserved. Always read as 0.

RT1IP Bits 4-2 Prescale timer 1 interrupt interval

- 000 /2
- 001 /4
- 010 /8
- 011 /16
- 100 /32
- 101 /64
- 110 /128
- 111 /256

RT1PSIE Bit 1 Prescale timer 1 interrupt enable

- 0 Interrupt not enabled
- 1 Interrupt enabled

RT1PSIFG Bit 0 Prescale timer 1 interrupt flag

- 0 No time event occurred.
- 1 Time event occurred.

Real-Time Clock Prescale Timer 0 Counter Register (RT0PS)

7	6	5	4	3	2	1	0
RT0PS							
rw	rw	rw	rw	rw	rw	rw	rw

RT0PS Bits 7-0 Prescale timer 0 counter value

Real-Time Clock Prescale Timer 1 Counter Register (RT1PS)

7	6	5	4	3	2	1	0
RT1PS							
rw	rw	rw	rw	rw	rw	rw	rw

RT1PS Bits 7-0 Prescale timer 1 counter value

Real-Time Clock Interrupt Vector Register (RTCIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**		**RTCIV**				0
r0	r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r0

RTCIV Bits 15-0 Real-time clock interrupt vector value

RTCIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	RTC ready	RTCRDYIFG	Highest
04h	RTC interval timer	RTCTEVIFG	
06h	RTC user alarm	RTCAIFG	
08h	RTC prescaler 0	RT0PSIFG	
0Ah	RTC prescaler 1	RT1PSIFG	
0Ch	Reserved		
0Eh	Reserved		
10h	Reserved		Lowest

32-Bit Hardware Multiplier (MPY32)

This chapter describes the 32-bit hardware multiplier (MPY32). The MPY32 module is implemented in all devices.

Topic **Page**

15.1 32-Bit Hardware Multiplier (MPY32) Introduction 404
15.2 MPY32 Operation 405
15.3 MPY32 Registers 418

15.1 32-Bit Hardware Multiplier (MPY32) Introduction

The MPY32 is a peripheral and is not part of the CPU. This means its activities do not interfere with the CPU activities. The multiplier registers are peripheral registers that are loaded and read with CPU instructions.

The MPY32 supports:

- Unsigned multiply
- Signed multiply
- Unsigned multiply accumulate
- Signed multiply accumulate
- 8-bit, 16-bit, 24-bit, and 32-bit operands
- Saturation
- Fractional numbers
- 8-bit and 16-bit operation compatible with 16-bit hardware multiplier
- 8-bit and 24-bit multiplications without requiring a "sign extend" instruction

The MPY32 block diagram is shown in Figure 15-1.

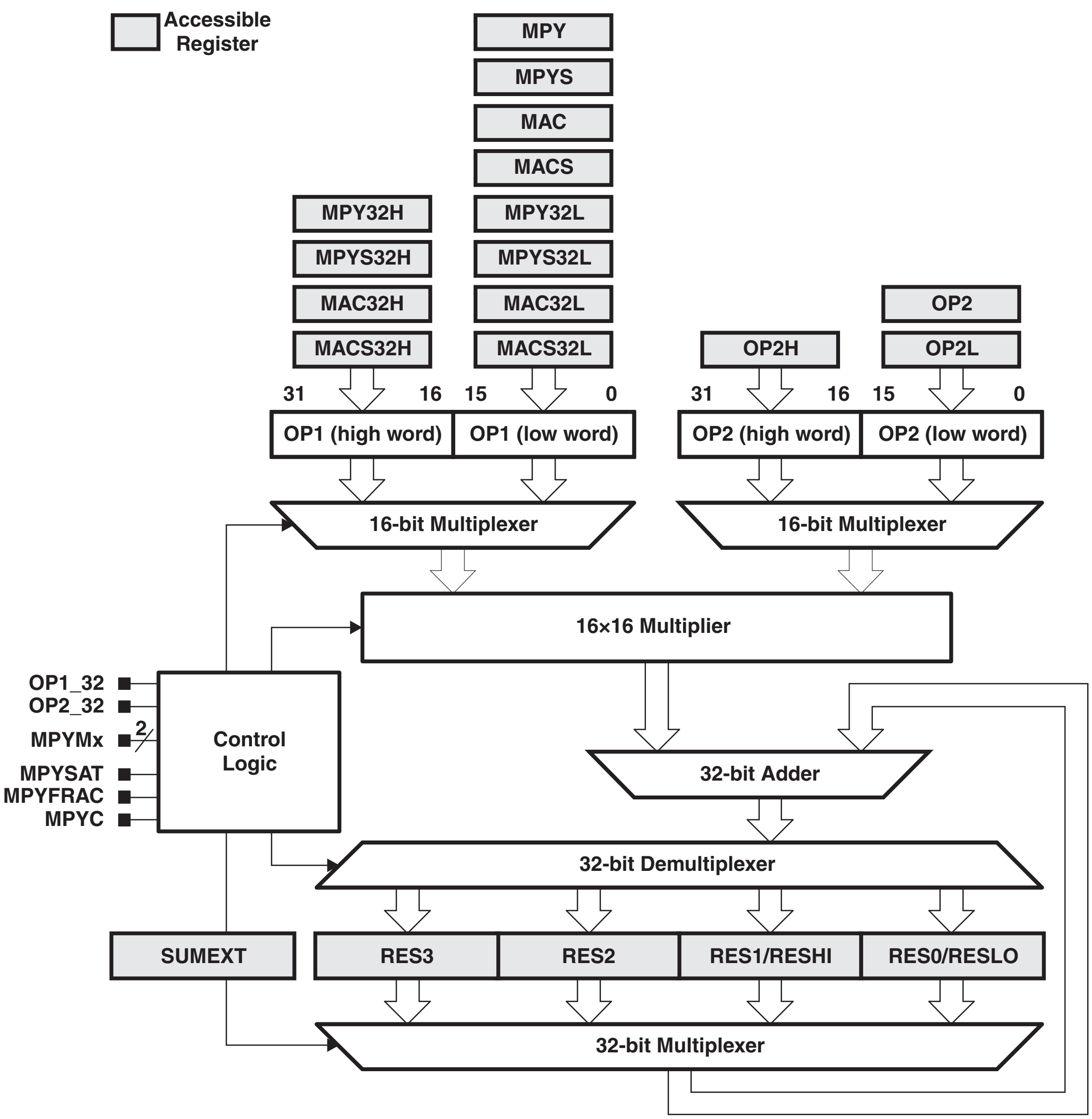


Figure 15-1. MPY32 Block Diagram

15.2 MPY32 Operation

The MPY32 supports 8-bit, 16-bit, 24-bit, and 32-bit operands with unsigned multiply, signed multiply, unsigned multiply-accumulate, and signed multiply-accumulate operations. The size of the operands are defined by the address the operand is written to and if it is written as word or byte. The type of operation is selected by the address the first operand is written to.

The hardware multiplier has two 32-bit operand registers – operand one (OP1) and operand two (OP2), and a 64-bit result register accessible via registers RES0 to RES3. For compatibility with the 16×16 hardware multiplier, the result of a 8-bit or 16-bit operation is accessible via RESLO, RESHI, and SUMEXT, as well. RESLO stores the low word of the 16×16-bit result, RESHI stores the high word of the result, and SUMEXT stores information about the result.

The result of a 8-bit or 16-bit operation is ready in three MCLK cycles and can be read with the next instruction after writing to OP2, except when using an indirect addressing mode to access the result. When using indirect addressing for the result, a `NOP` is required before the result is ready.

The result of a 24-bit or 32-bit operation can be read with successive instructions after writing OP2 or OP2H starting with RES0, except when using an indirect addressing mode to access the result. When using indirect addressing for the result, a `NOP` is required before the result is ready.

Table 15-1 summarizes when each word of the 64-bit result is available for the various combinations of operand sizes. With a 32-bit-wide second operand, OP2L and OP2H must be written. Depending on when the two 16-bit parts are written, the result availability may vary; thus, the table shows two entries, one for OP2L written and one for OP2H written. The worst case defines the actual result availability.

Table 15-1. Result Availability (MPYFRAC = 0, MPYSAT = 0)

Operation (OP1 × OP2)	Result Ready in MCLK Cycles					After
	RES0	RES1	RES2	RES3	MPYC Bit	
8/16 × 8/16	3	3	4	4	3	OP2 written
24/32 × 8/16	3	5	6	7	7	OP2 written
8/16 × 24/32	3	5	6	7	7	OP2L written
	N/A	3	4	4	4	OP2H written
24/32 × 24/32	3	8	10	11	11	OP2L written
	N/A	3	5	6	6	OP2H written

15.2.1 Operand Registers

Operand one (OP1) has 12 registers (see Table 15-2) used to load data into the multiplier and also select the multiply mode. Writing the low word of the first operand to a given address selects the type of multiply operation to be performed, but does not start any operation. When writing a second word to a high-word register with suffix 32H, the multiplier assumes a 32-bit-wide OP1, otherwise, 16 bits are assumed. The last address written prior to writing OP2 defines the width of the first operand. For example, if MPY32L is written first followed by MPY32H, all 32 bits are used and the data width of OP1 is set to 32 bits. If MPY32H is written first followed by MPY32L, the multiplication ignores MPY32H and assumes a 16-bit-wide OP1 using the data written into MPY32L.

Repeated multiply operations may be performed without reloading OP1 if the OP1 value is used for successive operations. It is not necessary to rewrite the OP1 value to perform the operations.

Table 15-2. OP1 Registers

OP1 Register	Operation
MPY	Unsigned multiply – operand bits 0 up to 15
MPYS	Signed multiply – operand bits 0 up to 15
MAC	Unsigned multiply accumulate –operand bits 0 up to 15
MACS	Signed multiply accumulate – operand bits 0 up to 15
MPY32L	Unsigned multiply – operand bits 0 up to 15
MPY32H	Unsigned multiply – operand bits 16 up to 31
MPYS32L	Signed multiply – operand bits 0 up to 15
MPYS32H	Signed multiply – operand bits 16 up to 31
MAC32L	Unsigned multiply accumulate – operand bits 0 up to 15
MAC32H	Unsigned multiply accumulate – operand bits 16 up to 31
MACS32L	Signed multiply accumulate – operand bits 0 up to 15
MACS32H	Signed multiply accumulate – operand bits 16 up to 31

Writing the second operand to the OP2 initiates the multiply operation. Writing OP2 starts the selected operation with a 16-bit-wide second operand together with the values stored in OP1. Writing OP2L starts the selected operation with a 32-bit-wide second operand and the multiplier expects a the high word to be written to OP2H. Writing to OP2H without a preceding write to OP2L is ignored.

Table 15-3. OP2 Registers

OP2 Register	Operation
OP2	Start multiplication with 16-bit-wide OP2 – operand bits 0 up to 15
OP2L	Start multiplication with 32-bit-wide OP2 – operand bits 0 up to 15
OP2H	Continue multiplication with 32-bit-wide OP2 – operand bits 16 up to 31

For 8-bit or 24-bit operands, the operand registers can be accessed with byte instructions. Accessing the multiplier with a byte instruction during a signed operation automatically causes a sign extension of the byte within the multiplier module. For 24-bit operands, only the high word should be written as byte. If the 24-bit operands are sign-extended as defined by the register, that is used to write the low word to, because this register defines if the operation is unsigned or signed.

The high-word of a 32-bit operand remains unchanged when changing the size of the operand to 16 bit, either by modifying the operand size bits or by writing to the respective operand register. During the execution of the 16-bit operation, the content of the high-word is ignored.

NOTE: Changing of first or second operand during multiplication

By default, changing OP1 or OP2 while the selected multiply operation is being calculated renders any results invalid that are not ready at the time the new operand(s) are changed. Writing OP2 or OP2L aborts any ongoing calculation and starts a new operation. Results that are not ready at that time are also invalid for following MAC or MACS operations.

To avoid this behavior, the MPYDLYWRTEN bit can be set to 1. Then, all writes to any MPY32 registers are delayed with MPYDLY32 = 0 until the 64-bit result is ready or with MPYDLY32 = 1 until the 32-bit result is ready. For MAC and MACS operations, the complete 64-bit result should always be ready.

See Table 15-1 for how many CPU cycles are needed until a certain result register is ready and valid for each of the different modes.

15.2.2 Result Registers

The multiplication result is always 64 bits wide. It is accessible via registers RES0 to RES3. Used with a signed operation, MPYS or MACS, the results are appropriately sign extended. If the result registers are loaded with initial values before a MACS operation, the user software must take care that the written value is properly sign extended to 64 bits.

NOTE: Changing of result registers during multiplication

The result registers must not be modified by the user software after writing the second operand into OP2 or OP2L until the initiated operation is completed.

In addition to RES0 to RES3, for compatibility with the 16×16 hardware multiplier, the 32-bit result of a 8-bit or 16-bit operation is accessible via RESLO, RESHI, and SUMEXT. In this case, the result low register RESLO holds the lower 16 bits of the calculation result and the result high register RESHI holds the upper 16 bits. RES0 and RES1 are identical to RESLO and RESHI, respectively, in usage and access of calculated results.

The sum extension register SUMEXT contents depend on the multiply operation and are listed in Table 15-4. If all operands are 16 bits wide or less, the 32-bit result is used to determine sign and carry. If one of the operands is larger than 16 bits, the 64-bit result is used.

The MPYC bit reflects the multiplier's carry as listed in Table 15-4 and, thus, can be used as 33rd or 65th bit of the result, if fractional or saturation mode is not selected. With MAC or MACS operations, the MPYC bit reflects the carry of the 32-bit or 64-bit accumulation and is not taken into account for successive MAC and MACS operations as the 33rd or 65th bit.

Table 15-4. SUMEXT and MPYC Contents

Mode	SUMEXT		MPYC	
MPY	SUMEXT is always 0000h.		MPYC is always 0.	
MPYS	SUMEXT contains the extended sign of the result.		MPYC contains the sign of the result.	
	00000h	Result was positive or zero	0	Result was positive or zero
	0FFFFh	Result was negative	1	Result was negative
MAC	SUMEXT contains the carry of the result.		MPYC contains the carry of the result.	
	0000h	No carry for result	0	No carry for result
	0001h	Result has a carry	1	Result has a carry
MACS	SUMEXT contains the extended sign of the result.		MPYC contains the carry of the result.	
	00000h	Result was positive or zero	0	No carry for result
	0FFFFh	Result was negative	1	Result has a carry

15.2.2.1 MACS Underflow and Overflow

The multiplier does not automatically detect underflow or overflow in MACS mode. For example, working with 16-bit input data and 32-bit results (i.e., using only RESLO and RESHI), the available range for positive numbers is 0 to 07FFF FFFFh and for negative numbers is 0FFFF FFFFh to 08000 0000h. An underflow occurs when the sum of two negative numbers yields a result that is in the range for a positive number. An overflow occurs when the sum of two positive numbers yields a result that is in the range for a negative number.

The SUMEXT register contains the sign of the result in both cases described above, 0FFFFh for a 32-bit overflow and 0000h for a 32-bit underflow. The MPYC bit in MPY32CTL0 can be used to detect the overflow condition. If the carry is different from the sign reflected by the SUMEXT register, an overflow or underflow occurred. User software must handle these conditions appropriately.

15.2.3 Software Examples

Examples for all multiplier modes follow. All 8×8 modes use the absolute address for the registers, because the assembler does not allow .B access to word registers when using the labels from the standard definitions file.

There is no sign extension necessary in software. Accessing the multiplier with a byte instruction during a signed operation automatically causes a sign extension of the byte within the multiplier module.

```
; 32x32 Unsigned Multiply
    MOV     #01234h,&MPY32L    ; Load low  word of 1st operand
    MOV     #01234h,&MPY32H    ; Load high word of 1st operand
```

```
    MOV     #05678h,&OP2L     ; Load low  word of 2nd operand
    MOV     #05678h,&OP2H     ; Load high word of 2nd operand
;   ...                       ; Process results

; 16x16 Unsigned Multiply
    MOV     #01234h,&MPY      ; Load 1st operand
    MOV     #05678h,&OP2      ; Load 2nd operand
;   ...                       ; Process results

; 8x8 Unsigned Multiply. Absolute addressing.
    MOV.B   #012h,&MPY_B      ; Load 1st operand
    MOV.B   #034h,&OP2_B      ; Load 2nd operand
;   ...                       ; Process results

; 32x32 Signed Multiply
    MOV     #01234h,&MPYS32L  ; Load low  word of 1st operand
    MOV     #01234h,&MPYS32H  ; Load high word of 1st operand
    MOV     #05678h,&OP2L     ; Load low  word of 2nd operand
    MOV     #05678h,&OP2H     ; Load high word of 2nd operand
;   ...                       ; Process results

; 16x16 Signed Multiply
    MOV     #01234h,&MPYS     ; Load 1st operand
    MOV     #05678h,&OP2      ; Load 2nd operand
;   ...                       ; Process results

; 8x8 Signed Multiply. Absolute addressing.
    MOV.B   #012h,&MPYS_B     ; Load 1st operand
    MOV.B   #034h,&OP2_B      ; Load 2nd operand
;   ...                       ; Process results
```

15.2.4 Fractional Numbers

The MPY32 provides support for fixed-point signal processing. In fixed-point signal processing, fractional number are represented by using a fixed decimal point. To classify different ranges of decimal numbers, a Q-format is used. Different Q-formats represent different locations of the decimal point. Figure 15-2 shows the format of a signed Q15 number using 16 bits. Every bit after the decimal point has a resolution of 1/2, the most significant bit (MSB) is used as the sign bit. The most negative number is 08000h and the maximum positive number is 07FFFh. This gives a range from –1.0 to 0.999969482 X 1.0 for the signed Q15 format with 16 bits.

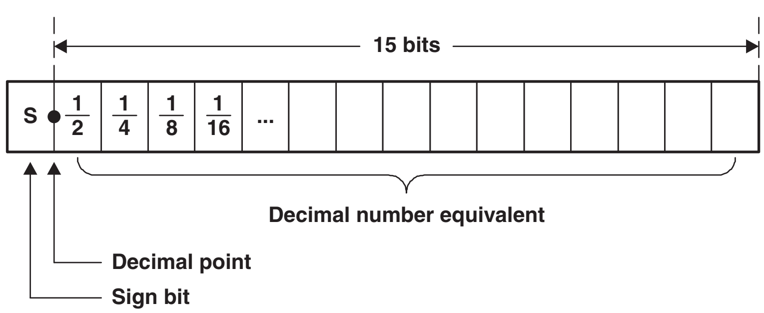


Figure 15-2. Q15 Format Representation

The range can be increased by shifting the decimal point to the right as shown in Figure 15-3. The signed Q14 format with 16 bits gives a range from –2.0 to 1.999938965 X 2.0.

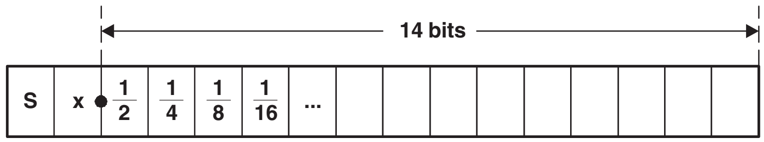


Figure 15-3. Q14 Format Representation

The benefit of using 16-bit signed Q15 or 32-bit signed Q31 numbers with multiplication is that the product of two number in the range from –1.0 to 1.0 is always in that same range.

15.2.4.1 Fractional Number Mode

Multiplying two fractional numbers using the default multiplication mode with MPYFRAC = 0 and MPYSAT = 0 gives a result with two sign bits. For example, if two 16-bit Q15 numbers are multiplied, a 32-bit result in Q30 format is obtained. To convert the result into Q15 format manually, the first 15 trailing bits and the extended sign bit must be removed. However, when the fractional mode of the multiplier is used, the redundant sign bit is automatically removed, yielding a result in Q31 format for the multiplication of two 16-bit Q15 numbers. Reading the result register RES1 gives the result as 16-bit Q15 number. The 32-bit Q31 result of a multiplication of two 32-bit Q31 numbers is accessed by reading registers RES2 and RES3.

The fractional mode is enabled with MPYFRAC = 1 in register MPY32CTL0. The actual content of the result register(s) is not modified when MPYFRAC = 1. When the result is accessed using software, the value is left shifted one bit, resulting in the final Q formatted result. This allows user software to switch between reading both the shifted (fractional) and the unshifted result. The fractional mode should only be enabled when required and disabled after use.

In fractional mode, the SUMEXT register contains the sign extended bits 32 and 33 of the shifted result for 16×16-bit operations and bits 64 and 65 for 32×32-bit operations – not only bits 32 or 64, respectively.

The MPYC bit is not affected by the fractional mode. It always reads the carry of the nonfractional result.

```
; Example using
; Fractional 16x16 multiplication
   BIS      #MPYFRAC,&MPY32CTL0   ; Turn on fractional mode
   MOV      &FRACT1,&MPYS         ; Load 1st operand as Q15
   MOV      &FRACT2,&OP2          ; Load 2nd operand as Q15
   MOV      &RES1,&PROD           ; Save result as Q15
   BIC      #MPYFRAC,&MPY32CTL0   ; Back to normal mode
```

Table 15-5. Result Availability in Fractional Mode (MPYFRAC = 1, MPYSAT = 0)

Operation (OP1 × OP2)	Result Ready in MCLK Cycles					After
	RES0	RES1	RES2	RES3	MPYC Bit	
8/16 × 8/16	3	3	4	4	3	OP2 written
24/32 × 8/16	3	5	6	7	7	OP2 written
8/16 × 24/32	3	5	6	7	7	OP2L written
	N/A	3	4	4	4	OP2H written
24/32 × 24/32	3	8	10	11	11	OP2L written
	N/A	3	5	6	6	OP2H written

15.2.4.2 Saturation Mode

The multiplier prevents overflow and underflow of signed operations in saturation mode. The saturation mode is enabled with MPYSAT = 1 in register MPY32CTL0. If an overflow occurs, the result is set to the most-positive value available. If an underflow occurs, the result is set to the most-negative value available. This is useful to reduce mathematical artifacts in control systems on overflow and underflow conditions. The saturation mode should only be enabled when required and disabled after use.

The actual content of the result register(s) is not modified when MPYSAT = 1. When the result is accessed using software, the value is automatically adjusted providing the most-positive or most-negative result when an overflow or underflow has occurred. The adjusted result is also used for successive multiply-and-accumulate operations. This allows user software to switch between reading the saturated and the nonsaturated result.

With 16×16 operations, the saturation mode only applies to the least significant 32 bits, i.e., the result registers RES0 and RES1. Using the saturation mode in MAC or MACS operations that mix 16×16 operations with 32×32, 16×32, or 32×16 operations leads to unpredictable results.

With 32×32, 16×32, and 32×16 operations, the saturated result can only be calculated when RES3 is ready. In non-5xx devices, reading RES0 to RES2 prior to the complete result being ready delivers the nonsaturated results independent of the MPYSAT bit setting.

Enabling the saturation mode does not affect the content of the SUMEXT register nor the content of the MPYC bit.

```
; Example using
; Fractional 16x16 multiply accumulate with Saturation
   ; Turn on fractional and saturation mode:
   BIS      #MPYSAT+MPYFRAC,&MPY32CTL0
   MOV      &A1,&MPYS                   ; Load A1 for 1st term
   MOV      &K1,&OP2                    ; Load K1 to get A1*K1
   MOV      &A2,&MACS                   ; Load A2 for 2nd term
   MOV      &K2,&OP2                    ; Load K2 to get A2*K2
   MOV      &RES1,&PROD                 ; Save A1*K1+A2*K2 as result
   BIC      #MPYSAT+MPYFRAC,&MPY32CTL0  ; turn back to normal
```

Table 15-6. Result Availability in Saturation Mode (MPYSAT = 1)

Operation (OP1 × OP2)	Result Ready in MCLK Cycles					After
	RES0	RES1	RES2	RES3	MPYC Bit	
8/16 × 8/16	3	3	N/A	N/A	3	OP2 written
24/32 × 8/16	7	7	7	7	7	OP2 written
8/16 × 24/32	7	7	7	7	7	OP2L written
	4	4	4	4	4	OP2H written
24/32 × 24/32	11	11	11	11	11	OP2L written
	6	6	6	6	6	OP2H written

Figure 15-4 shows the flow for 32-bit saturation used for 16×16 bit multiplications and the flow for 64-bit saturation used in all other cases. Primarily, the saturated results depends on the carry bit MPYC and the MSB of the result. Secondly, if the fractional mode is enabled, it depends also on the two MSBs of the unshift result, i.e., the result that is read with fractional mode disabled.

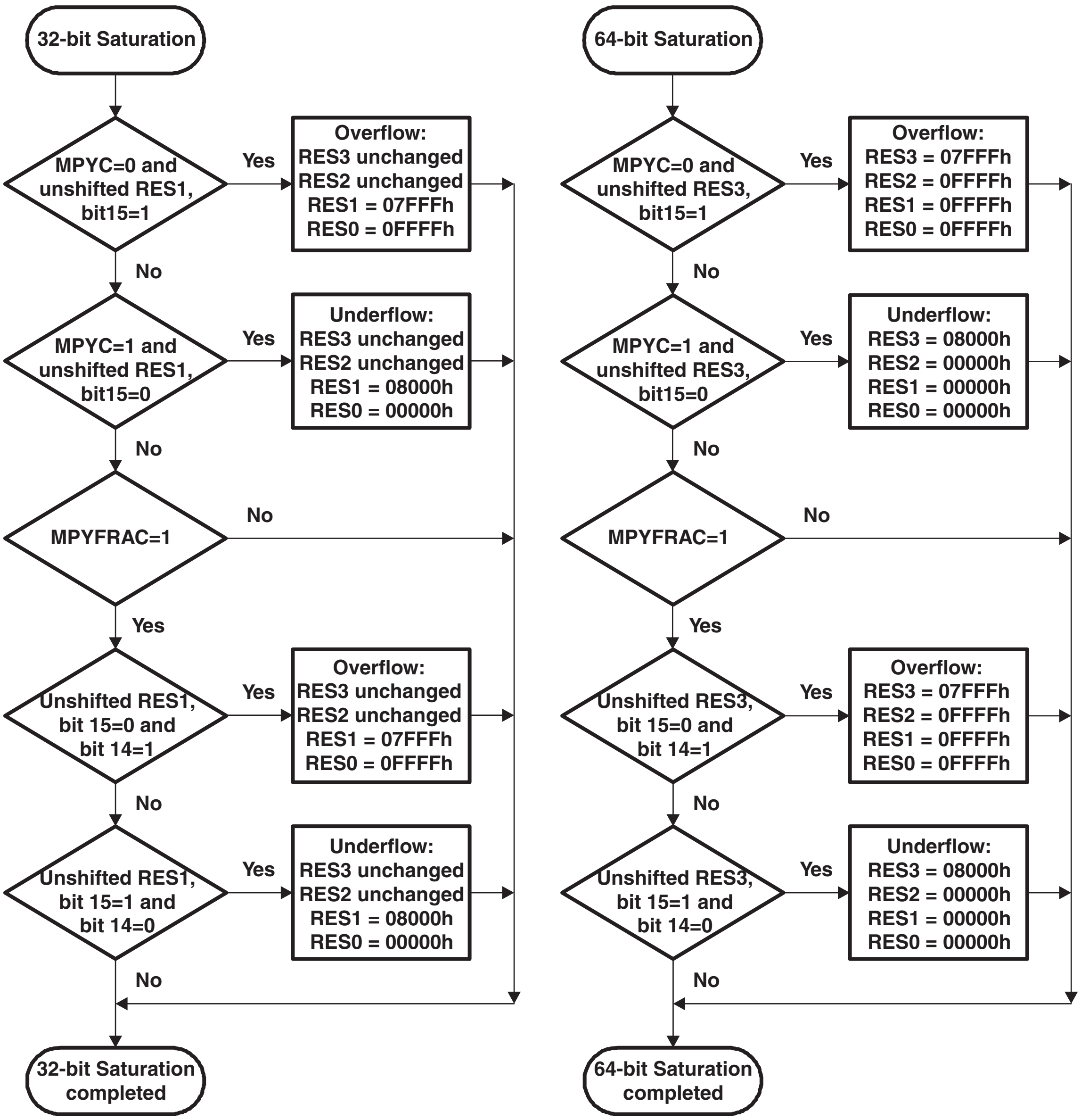


Figure 15-4. Saturation Flow Chart

NOTE: Saturation in fractional mode

In case of multiplying –1.0 × –1.0 in fractional mode, the result of +1.0 is out of range, thus, the saturated result gives the most positive result.

When using multiply-and-accumulate operations, the accumulated values are saturated as if MPYFRAC = 0 – only during read accesses to the result registers the values are saturated taking the fractional mode into account. This provides additional dynamic range during the calculation and only the end result is then saturated if needed.

The following example illustrates a special case showing the saturation function in fractional mode. It also uses the 8-bit functionality of the MPY32 module.

```
; Turn on fractional and saturation mode,
; clear all other bits in MPY32CTL0:
MOV      #MPYSAT+MPYFRAC,&MPY32CTL0
;Pre-load result registers to demonstrate overflow
MOV      #0,&RES3           ;
MOV      #0,&RES2           ;
MOV      #07FFFh,&RES1      ;
MOV      #0FA60h,&RES0      ;
MOV.B    #050h,&MACS_B      ; 8-bit signed MAC operation
MOV.B    #012h,&OP2_B       ; Start 16x16 bit operation
MOV      &RES0,R6           ; R6 = 0FFFFh
MOV      &RES1,R7           ; R7 = 07FFFh
```

The result is saturated because already the result not converted into a fractional number shows an overflow. The multiplication of the two positive numbers 00050h and 00012h gives 005A0h. 005A0h added to 07FFF FA60h results in 8000 059Fh, without MPYC being set. Because the MSB of the unmodified result RES1 is 1 and MPYC = 0, the result is saturated according Figure 15-4.

NOTE: Validity of saturated result

The saturated result is only valid if the registers RES0 to RES3, the size of OP! and OP2, and MPYC are not modified.

If the saturation mode is used with a preloaded result, user software must ensure that MPYC in the MPY32CTL0 register is loaded with the sign bit of the written result, otherwise, the saturation mode erroneously saturates the result.

15.2.5 Putting It All Together

Figure 15-5 shows the complete multiplication flow, depending on the various selectable modes for the MPY32 module.

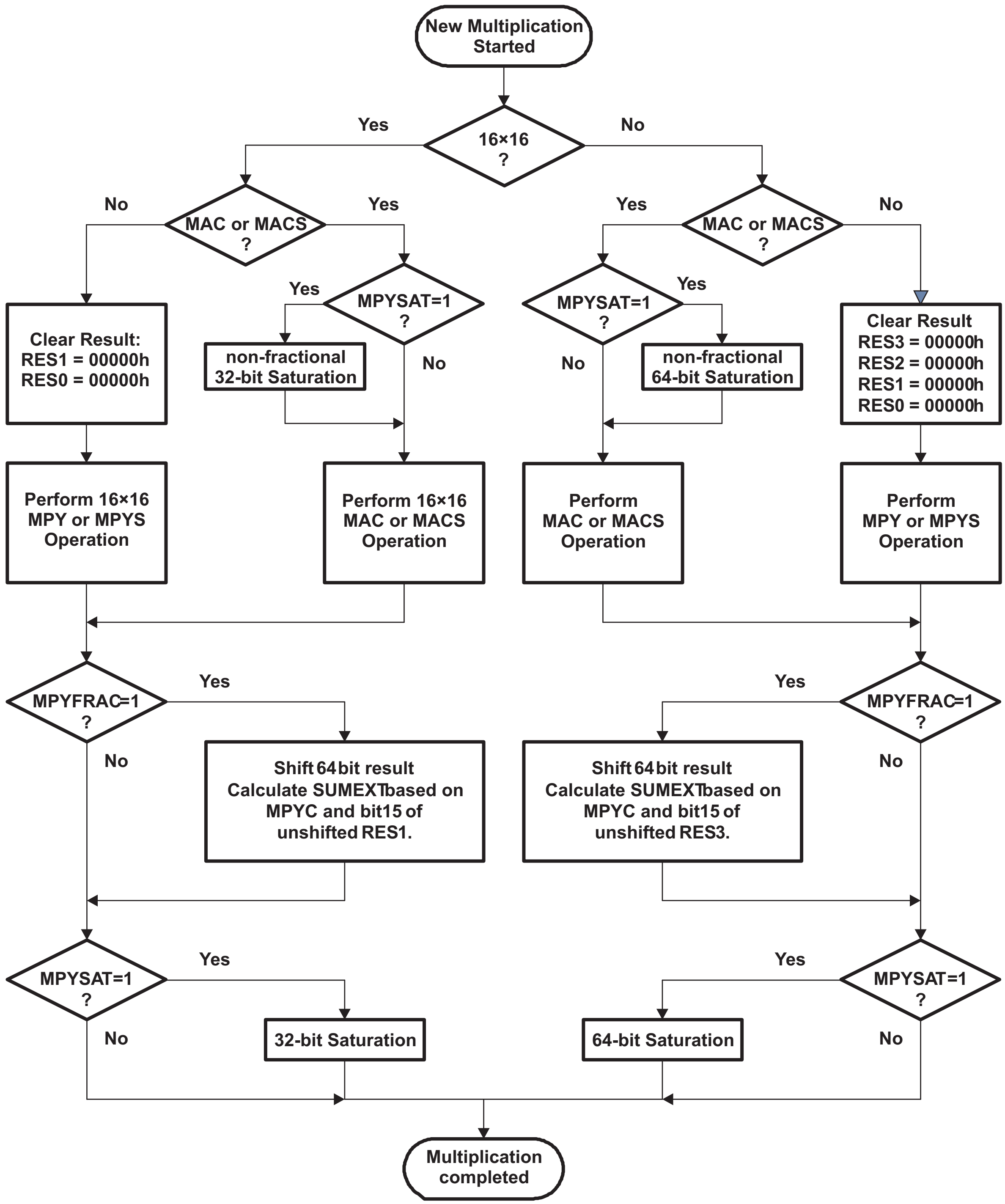


Figure 15-5. Multiplication Flow Chart

Given the separation in processing of 16-bit operations (32-bit results) and 32-bit operations (64-bit results) by the module, it is important to understand the implications when using MAC/MACS operations and mixing 16-bit operands/results with 32-bit operands/results. User software must address these points during usage when mixing these operations. The following code snippet illustrates the issue.

```
; Mixing 32x24 multiplication with 16x16 MACS operation
   MOV      #MPYSAT,&MPY32CTL0   ; Saturation mode
   MOV      #052C5h,&MPY32L      ; Load low word of 1st operand
   MOV      #06153h,&MPY32H      ; Load high word of 1st operand
   MOV      #001ABh,&OP2L        ; Load low word of 2nd operand
   MOV.B    #023h,&OP2H_B        ; Load high word of 2nd operand
                                 ;... 5 NOPs required
   MOV      &RES0,R6             ; R6 = 00E97h
   MOV      &RES1,R7             ; R7 = 0A6EAh
   MOV      &RES2,R8             ; R8 = 04F06h
   MOV      &RES3,R9             ; R9 = 0000Dh
                                 ; Note that MPYC = 0!
   MOV      #0CCC3h,&MACS        ; Signed MAC operation
   MOV      #0FFB6h,&OP2         ; 16x16 bit operation
   MOV      &RESLO,R6            ; R6 = 0FFFFh
   MOV      &RESHI,R7            ; R7 = 07FFFh
```

The second operation gives a saturated result because the 32-bit value used for the 16×16-bit MACS operation was already saturated when the operation was started; the carry bit MPYC was 0 from the previous operation, but the MSB in result register RES1 is set. As one can see in the flow chart, the content of the result registers are saturated for multiply-and-accumulate operations after starting a new operation based on the previous results, but depending on the size of the result (32 bit or 64 bit) of the newly initiated operation.

The saturation before the multiplication can cause issues if the MPYC bit is not properly set as the following code example illustrates.

```
   ;Pre-load result registers to demonstrate overflow
   MOV      #0,&RES3          ;
   MOV      #0,&RES2          ;
   MOV      #0,&RES1          ;
   MOV      #0,&RES0          ;
   ; Saturation mode and set MPYC:
   MOV      #MPYSAT+MPYC,&MPY32CTL0
   MOV.B    #082h,&MACS_B     ; 8-bit signed MAC operation
   MOV.B    #04Fh,&OP2_B      ; Start 16x16 bit operation
   MOV      &RES0,R6          ; R6 = 00000h
   MOV      &RES1,R7          ; R7 = 08000h
```

Even though the result registers were loaded with all zeros, the final result is saturated. This is because the MPYC bit was set causing the result used for the multiply-and-accumulate to be saturated to 08000 0000h. Adding a negative number to it would again cause an underflow, thus, the final result is also saturated to 08000 0000h.

15.2.6 Indirect Addressing of Result Registers

When using indirect or indirect autoincrement addressing mode to access the result registers and the multiplier requires three cycles until result availability according to Table 15-1, at least one instruction is needed between loading the second operand and accessing the result registers:

```
; Access multiplier 16x16 results with indirect addressing
   MOV      #RES0,R5          ; RES0 address in R5 for indirect
   MOV      &OPER1,&MPY       ; Load 1st operand
   MOV      &OPER2,&OP2       ; Load 2nd operand
   NOP                        ; Need one cycle
   MOV      @R5+,&xxx         ; Move RES0
   MOV      @R5,&xxx          ; Move RES1
```

In case of a 32×16 multiplication, there is also one instruction required between reading the first result register RES0 and the second result register RES1:

```
; Access multiplier 32x16 results with indirect addressing
   MOV   #RES0,R5           ; RES0 address in R5 for indirect
   MOV   &OPER1L,&MPY32L    ; Load low word of 1st operand
   MOV   &OPER1H,&MPY32H    ; Load high word of 1st operand
   MOV   &OPER2,&OP2        ; Load 2nd operand (16 bits)
   NOP                      ; Need one cycle
   MOV   @R5+,&xxx          ; Move RES0
   NOP                      ; Need one additional cycle
   MOV   @R5,&xxx           ; Move RES1
                            ; No additional cycles required!
   MOV   @R5,&xxx           ; Move RES2
```

15.2.7 Using Interrupts

If an interrupt occurs after writing OP, but before writing OP2, and the multiplier is used in servicing that interrupt, the original multiplier mode selection is lost and the results are unpredictable. To avoid this, disable interrupts before using the MPY32, do not use the MPY32 in interrupt service routines, or use the save and restore functionality of the MPY32.

```
; Disable interrupts before using the hardware multiplier
   DINT                  ; Disable interrupts
   NOP                   ; Required for DINT
   MOV    #xxh,&MPY      ; Load 1st operand
   MOV    #xxh,&OP2      ; Load 2nd operand
   EINT                  ; Interrupts may be enabled before
                         ; processing results if result
                         ; registers are stored and restored in
                         ; interrupt service routines
```

15.2.7.1 Save and Restore

If the multiplier is used in interrupt service routines, its state can be saved and restored using the MPY32CTL0 register. The following code example shows how the complete multiplier status can be saved and restored to allow interruptible multiplications together with the usage of the multiplier in interrupt service routines. Because the state of the MPYSAT and MPYFRAC bits are unknown, they should be cleared before the registers are saved as shown in the code example.

```
; Interrupt service routine using multiplier
MPY_USING_ISR
   PUSH   &MPY32CTL0     ; Save multiplier mode, etc.
   BIC    #MPYSAT+MPYFRAC,&MPY32CTL0
                         ; Clear MPYSAT+MPYFRAC
   PUSH   &RES3          ; Save result 3
   PUSH   &RES2          ; Save result 2
   PUSH   &RES1          ; Save result 1
   PUSH   &RES0          ; Save result 0
   PUSH   &MPY32H        ; Save operand 1, high word
   PUSH   &MPY32L        ; Save operand 1, low word
   PUSH   &OP2H          ; Save operand 2, high word
   PUSH   &OP2L          ; Save operand 2, low word
                         ;
   ...                   ; Main part of ISR
                         ; Using standard MPY routines
                         ;
   POP    &OP2L          ; Restore operand 2, low word
   POP    &OP2H          ; Restore operand 2, high word
                         ; Starts dummy multiplication but
                         ; result is overwritten by
                         ; following restore operations:
   POP    &MPY32L        ; Restore operand 1, low word
   POP    &MPY32H        ; Restore operand 1, high word
   POP    &RES0          ; Restore result 0
   POP    &RES1          ; Restore result 1
   POP    &RES2          ; Restore result 2
   POP    &RES3          ; Restore result 3
   POP    &MPY32CTL0     ; Restore multiplier mode, etc.
   reti                  ; End of interrupt service routine
```

15.2.8 Using DMA

In devices with a DMA controller, the multiplier can trigger a transfer when the complete result is available. The DMA controller needs to start reading the result with MPY32RES0 successively up to MPY32RES3. Not all registers need to be read. The trigger timing is such that the DMA controller starts reading MPY32RES0 when its ready, and that the MPY32RES3 can be read exactly in the clock cycle when it is available to allow fastest access via DMA. The signal into the DMA controller is 'Multiplier ready' (see the *DMA Controller* chapter for details).

15.3 MPY32 Registers

MPY32 registers are listed inTable 15-7. The base address can be found in the device-specific data sheet. The address offsets are listed inTable 15-7.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 15-7. MPY32 Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
16-bit operand one – multiply	MPY	Read/write	Word	00h	Undefined
	MPY_L	Read/write	Byte	00h	Undefined
	MPY_H	Read/write	Byte	01h	Undefined
8-bit operand one – multiply	MPY_B	Read/write	Byte	00h	Undefined
16-bit operand one – signed multiply	MPYS	Read/write	Word	02h	Undefined
	MPYS_L	Read/write	Byte	02h	Undefined
	MPYS_H	Read/write	Byte	03h	Undefined
8-bit operand one – signed multiply	MPYS_B	Read/write	Byte	02h	Undefined
16-bit operand one – multiply accumulate	MAC	Read/write	Word	04h	Undefined
	MAC_L	Read/write	Byte	04h	Undefined
	MAC_H	Read/write	Byte	05h	Undefined
8-bit operand one – multiply accumulate	MAC_B	Read/write	Byte	04h	Undefined
16-bit operand one – signed multiply accumulate	MACS	Read/write	Word	06h	Undefined
	MACS_L	Read/write	Byte	06h	Undefined
	MACS_H	Read/write	Byte	07h	Undefined
8-bit operand one – signed multiply accumulate	MACS_B	Read/write	Byte	06h	Undefined
16-bit operand two	OP2	Read/write	Word	08h	Undefined
	OP2_L	Read/write	Byte	08h	Undefined
	OP2_H	Read/write	Byte	09h	Undefined
8-bit operand two	OP2_B	Read/write	Byte	08h	Undefined
16x16-bit result low word	RESLO	Read/write	Word	0Ah	Undefined
	RESLO_L	Read/write	Byte	0Ah	Undefined
	RESLO_H	Read/write	Byte	0Bh	Undefined
16x16-bit result high word	RESHI	Read/write	Word	0Ch	Undefined
	RESHI_L	Read/write	Byte	0Ch	Undefined
	RESHI_H	Read/write	Byte	0Dh	Undefined
16x16-bit sum extension register	SUMEXT	Read	Word	0Eh	Undefined
	SUMEXT_L	Read	Byte	0Eh	Undefined
	SUMEXT_H	Read	Byte	0Fh	Undefined
32-bit operand 1 – multiply – low word	MPY32L	Read/write	Word	10h	Undefined
	MPY32L_L	Read/write	Byte	10h	Undefined
	MPY32L_H	Read/write	Byte	11h	Undefined
32-bit operand 1 – multiply – high word	MPY32H	Read/write	Word	12h	Undefined
	MPY32H_L	Read/write	Byte	12h	Undefined
	MPY32H_H	Read/write	Byte	13h	Undefined
24-bit operand 1 – multiply – high byte	MPY32H_B	Read/write	Byte	12h	Undefined
32-bit operand 1 – signed multiply – low word	MPYS32L	Read/write	Word	14h	Undefined
	MPYS32L_L	Read/write	Byte	14h	Undefined
	MPYS32L_H	Read/write	Byte	15h	Undefined

Table 15-7. MPY32 Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
32-bit operand 1 – signed multiply – high word	MPYS32H	Read/write	Word	16h	Undefined
	MPYS32H_L	Read/write	Byte	16h	Undefined
	MPYS32H_H	Read/write	Byte	17h	Undefined
24-bit operand 1 – signed multiply – high byte	MPYS32H_B	Read/write	Byte	16h	Undefined
32-bit operand 1 – multiply accumulate – low word	MAC32L	Read/write	Word	18h	Undefined
	MAC32L_L	Read/write	Byte	18h	Undefined
	MAC32L_H	Read/write	Byte	19h	Undefined
32-bit operand 1 – multiply accumulate – high word	MAC32H	Read/write	Word	1Ah	Undefined
	MAC32H_L	Read/write	Byte	1Ah	Undefined
	MAC32H_H	Read/write	Byte	1Bh	Undefined
24-bit operand 1 – multiply accumulate – high byte	MAC32H_B	Read/write	Byte	1Ah	Undefined
32-bit operand 1 – signed multiply accumulate – low word	MACS32L	Read/write	Word	1Ch	Undefined
	MACS32L_L	Read/write	Byte	1Ch	Undefined
	MACS32L_H	Read/write	Byte	1Dh	Undefined
32-bit operand 1 – signed multiply accumulate – high word	MACS32H	Read/write	Word	1Eh	Undefined
	MACS32H_L	Read/write	Byte	1Eh	Undefined
	MACS32H_H	Read/write	Byte	1Fh	Undefined
24-bit operand 1 – signed multiply accumulate – high byte	MACS32H_B	Read/write	Byte	1Eh	Undefined
32-bit operand 2 – low word	OP2L	Read/write	Word	20h	Undefined
	OP2L_L	Read/write	Byte	20h	Undefined
	OP2L_H	Read/write	Byte	21h	Undefined
32-bit operand 2 – high word	OP2H	Read/write	Word	22h	Undefined
	OP2H_L	Read/write	Byte	22h	Undefined
	OP2H_H	Read/write	Byte	23h	Undefined
24-bit operand 2 – high byte	OP2H_B	Read/write	Byte	22h	Undefined
32x32-bit result 0 – least significant word	RES0	Read/write	Word	24h	Undefined
	RES0_L	Read/write	Byte	24h	Undefined
	RES0_H	Read/write	Byte	25h	Undefined
32x32-bit result 1	RES1	Read/write	Word	26h	Undefined
	RES1_L	Read/write	Byte	26h	Undefined
	RES1_H	Read/write	Byte	27h	Undefined
32x32-bit result 2	RES2	Read/write	Word	28h	Undefined
	RES2_L	Read/write	Byte	28h	Undefined
	RES2_H	Read/write	Byte	29h	Undefined
32x32-bit result 3 – most significant word	RES3	Read/write	Word	2Ah	Undefined
	RES3_L	Read/write	Byte	2Ah	Undefined
	RES3_H	Read/write	Byte	2Bh	Undefined
MPY32 control register 0	MPY32CTL0	Read/write	Word	2Ch	Undefined
	MPY32CTL0_L	Read/write	Byte	2Ch	Undefined
	MPY32CTL0_H	Read/write	Byte	2Dh	00h

The registers listed in Table 15-8 are treated equally.

Table 15-8. Alternative Registers

Register	Alternative 1	Alternative 2
16-bit operand one – multiply	MPY	MPY32L
8-bit operand one – multiply	MPY_B or MPY_L	MPY32L_B or MPY32L_L
16-bit operand one – signed multiply	MPYS	MPYS32L
8-bit operand one – signed multiply	MPYS_B or MPYS_L	MPYS32L_B or MPYS32L_L
16-bit operand one – multiply accumulate	MAC	MAC32L
8-bit operand one – multiply accumulate	MAC_B or MAC_L	MAC32L_B or MAC32L_L
16-bit operand one – signed multiply accumulate	MACS	MACS32L
8-bit operand one – signed multiply accumulate	MACS_B or MACS_L	MACS32L_B or MACS32L_L
16x16-bit result low word	RESLO	RES0
16x16-bit result high word	RESHI	RES1

32-Bit Hardware Multiplier Control 0 Register (MPY32CTL0)

15	14	13	12	11	10	9	8
Reserved						**MPYDLY32**	**MPYDLY WRTEN**
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

7	6	5	4	3	2	1	0
MPYOP2_32	**MPYOP1_32**	**MPYMx**		**MPYSAT**	**MPYFRAC**	**Reserved**	**MPYC**
rw	rw	rw	rw	rw-0	rw-0	rw-0	rw

Reserved	Bits 15-10	Reserved
MPYDLY32	Bit 9	Delayed write mode 0 Writes are delayed until 64-bit result (RES0 to RES3) is available. 1 Writes are delayed until 32-bit result (RES0 to RES1) is available.
MPYDLYWRTEN	Bit 8	Delayed write enable All writes to any MPY32 register are delayed until the 64-bit (MPYDLY32 = 0) or 32-bit (MPYDLY32 = 1) result is ready. 0 Writes are not delayed. 1 Writes are delayed.
MPYOP2_32	Bit 7	Multiplier bit width of operand 2 0 16 bits 1 32 bits
MPYOP1_32	Bit 6	Multiplier bit width of operand 1 0 16 bits 1 32 bits
MPYMx	Bits 5-4	Multiplier mode 00 MPY – Multiply 01 MPYS – Signed multiply 10 MAC – Multiply accumulate 11 MACS – Signed multiply accumulate
MPYSAT	Bit 3	Saturation mode 0 Saturation mode disabled 1 Saturation mode enabled
MPYFRAC	Bit 2	Fractional mode 0 Fractional mode disabled 1 Fractional mode enabled
Reserved	Bit 1	Reserved
MPYC	Bit 0	Carry of the multiplier. It can be considered as 33rd or 65th bit of the result if fractional or saturation mode is not selected, because the MPYC bit does not change when switching to saturation or fractional mode. It is used to restore the SUMEXT content in MAC mode. 0 No carry for result 1 Result has a carry

REF

The REF module is a general purpose reference system that is used to generate voltage references required for other subsystems available on a given device such as digital-to-analog converters, analog-to-digital converters, comparators, etc. This chapter describes the REF module.

16.1 REF Introduction

The reference module (REF) is responsible for generation of all critical reference voltages that can be used by various analog peripherals in a given device. These include, but are not necessarily limited to, the ADC10_A, ADC12_A, DAC12_A, LCD_B, and COMP_B modules dependent upon the particular device. The heart of the reference system is the bandgap from which all other references are derived by unity or non-inverting gain stages. The REFGEN sub-system consists of the bandgap, the bandgap bias, and the non-inverting buffer stage which generates the three primary voltage reference available in the system, namely 1.5 V, 2.0 V, and 2.5 V. In addition, when enabled, a buffered bandgap voltage is also available.

Features of the REF include:

- Centralized, factory trimmed bandgap with excellent PSRR, temperature coefficient, and accuracy
- 1.5-V, 2.0-V, or 2.5-V user selectable internal references
- Buffered bandgap voltage available to rest of system
- Power saving features
- Backward compatibility to existing reference system

The block diagram of the REF module (example of a device with ADC12_A) is shown in Figure 16-1. Devices with ADC10_A do not include the reference voltage output to the external pad.

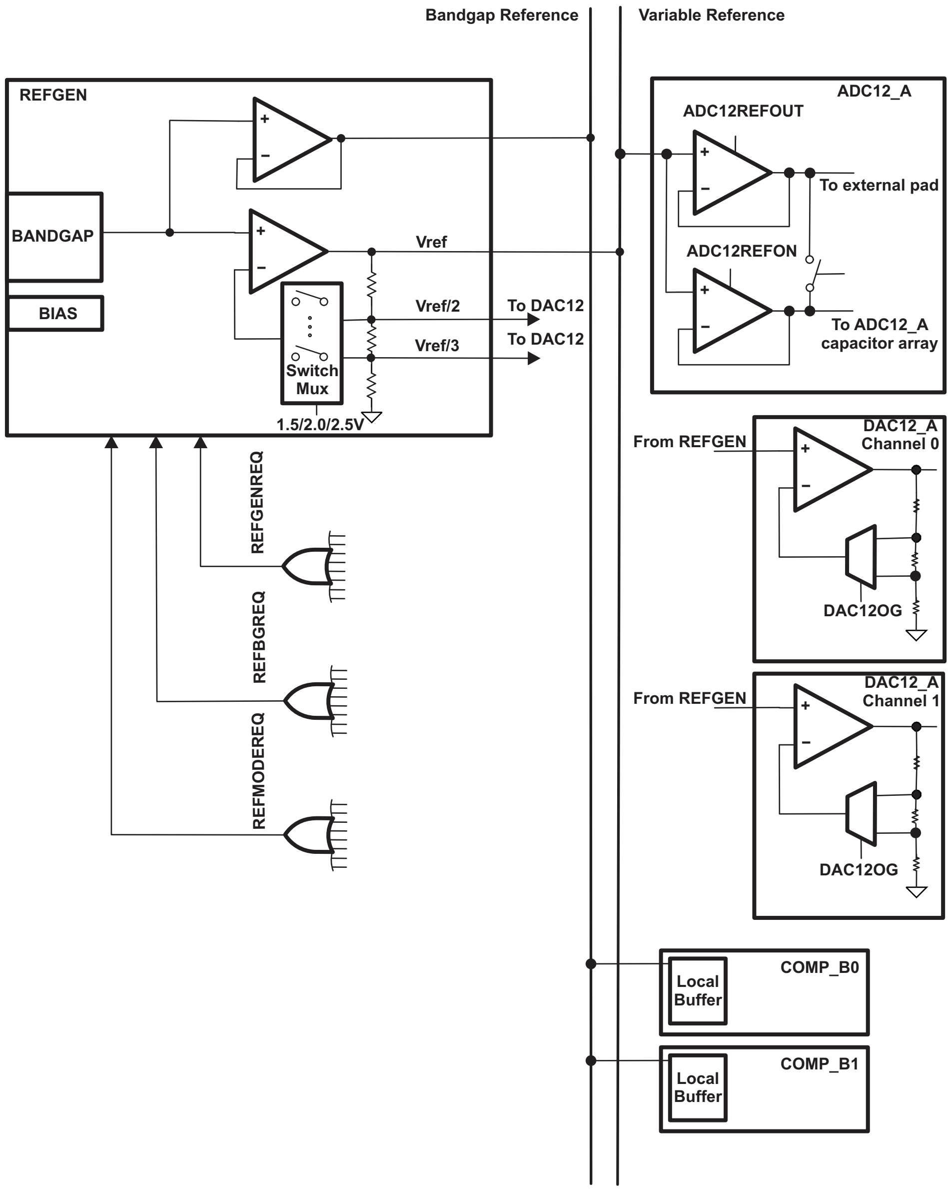


Figure 16-1. REF Block Diagram

16.2 Principle of Operation

The REF module provides all the necessary voltage references to be used by various peripheral modules throughout the system. These may include, but are not limited to, devices that contain an ADC10_A, ADC12_A, DAC12_A, LCD_B, or COMP_B.

The REFGEN subsystem contains a high-performance bandgap. This bandgap has very good accuracy (factory trimmed), low temperature coefficient, and high PSRR while operating at low power. The bandgap voltage is used to generate three voltages via a non-inverting amplifier stage, namely 1.5 V, 2.0 V, and 2.5 V. One voltage can be selected at a time. One output of the REFGEN subsystem is the variable reference line. The variable reference line provides either 1.5 V, 2.0 V, or 2.5 V to the rest of the system. A second output of the REFGEN subsystem provides a buffered bandgap reference line that can also be used by modules throughout the system. Additionally, the REFGEN supports voltage references required for the DAC12_A module, when available. Lastly, the REFGEN subsystem also includes the temperature sensor circuitry since this is derived from the bandgap. The temperature sensor is used by an ADC to measure a voltage proportional to temperature.

16.2.1 Low-Power Operation

The REF module is capable of supporting low-power applications such as LCD generation. Many of these applications do not require a very accurate reference, compared to data conversion, yet power is of prime concern. To support these kinds of applications, the bandgap is capable of being used in a sampled mode. In sampled mode, the bandgap circuitry is clocked via the VLO at an appropriate duty cycle. This reduces the average power of the bandgap circuitry significantly, at the cost of accuracy. When not in sampled mode, the bandgap is in static mode. Its power is at its highest, but so is its accuracy.

Modules automatically can request static mode or sampled mode via their own individual request lines. In this way, the particular module determines what mode is appropriate for its proper operation and performance. Any one active module that requests static mode will cause all other modules to use static mode, regardless if another module is requesting sampled mode. In other words, static mode always has higher priority over sampled mode.

16.2.2 REFCTL

The REFCTL registers provide a way to control the reference system from one centralized set of registers. By default, REFCTL is used as the primary control of the reference system. On legacy devices, the ADC12_A provided the control bits necessary to configure the reference system, namely ADC12REFON, ADC12REF2_5, ADC12TCOFF, ADC12REFOUT, ADC12SR, and ADC12REFBURST. The ADC12SR and ADC12REFBURST bits are very specific to the ADC12 operation and therefore are not included in REFCTL. All legacy control bits can still be used to configure the reference system allowing for backward compatibility by clearing REFMSTR. In this case, the REFCTL register bits are a 'do not care'.

Setting the reference master bit (REFMSTR = 1), allows the reference system to be controlled via the REFCTL register. This is the default setting. In this mode, the legacy control bits ADC12REFON, ADC12REF2_5, ADC12TCOFF, and ADC12REFOUT are do not care. The ADC12SR and ADC12REFBURST are still controlled via the ADC12_A since these are very specific to the ADC12_A module. If REFMSTR set is cleared, all settings in the REFCTL are do not care and the reference system is controlled completely by the legacy control bits inside the ADC12_A module. Table Table 16-1summarizes the REFCTL bits and their effect on the REF module.

Table 16-1. REF Control of Reference System (REFMSTR = 1) (Default)

REF Register Setting	Function
REFON	Setting this bit enables the REFGEN subsystem which includes the bandgap, the bandgap bias circuitry, and the 1.5-V/2.0-V/2.5-V buffer. Setting this bit will cause the REFGEN subsystem to remain enabled regardless if any module has requested it. Clearing this bit will disable the REFGEN subsystem only when there are no pending requests for REFGEN from all modules.
REFVSEL	Selects 1.5 V, 2.0 V, or 2.5 V to be present on the variable reference line when REFON = 1 or REFGEN is requested by any module.
REFOUT	Setting this bits enables the variable reference line voltage to be present external to the device via a buffer (external reference buffer).
REFTCOFF	Setting this bit disables the temperature sensor (when available) to conserve power.

Table 16-2 summarizes the ADC12_A control bits and their effect on the REF module. Please see the ADC12_A module description for further details.

NOTE: Although the REF module supports using the ADC12_A bits as control for the reference system, it is recommended that the usage of the new REFCTL register be used and older code migrated to this methodology. This allows the logical partitioning of the reference system to be separate from the ADC12_A system and forms a more natural partitioning for future products.

Table 16-2. Table 2. ADC Control of Reference System (REFMSTR = 0)

ADC12_A Register Setting	Function
ADC12REFON	Setting this bit enables the REFGEN subsystem which includes the bandgap, the bandgap bias circuitry, and the 1.5-V/2.0-V/2.5-V buffer. Setting this bit will cause the REFGEN subsystem to remain enabled regardless if any module has requested it. Clearing this bit will disable the REFGEN subsystem only when there are no pending requests for REFGEN from all modules.
ADC12REF2_5	Setting this bits causes 2.5 V to be present on the variable reference line when ADC12REFON = 1. Clearing this bit causes 1.5 V to be present on the variable reference line when ADC12REFON = 1.
ADC12REFOUT	Setting this bits enables the variable reference line voltage to be present external to the device via a buffer (external reference buffer).
ADC12TCOFF	Setting this bit disables the temperature sensor to conserve power.

As stated previously, the ADC12REFBURST does have an effect on the reference system and can be controlled via the ADC12_A. This bit is in effect regardless if REFCTL or the ADC12_A is controlling the reference system. Setting ADC12REFBURST = 1 enables burst mode when REFON = 1 and REFMSTR = 1 or when ADC12REFON = 1 and REFMSTR = 0. In burst mode, the internal buffer (ADC12REFOUT = 0) or the external buffer (ADC12REFOUT = 1) is enabled only during a conversion and disabled automatically to conserve power.

NOTE: The legacy ADC12_A bit ADC12REF2_5 only allows for selecting either 1.5 V or 2.5 V. To select 2.0 V, the REFVSEL control bits must be used (REFMSTR = 1).

16.2.3 Reference System Requests

There are three basic reference system requests that are used by the reference system. Each module can utilize these requests to obtain the proper response from the reference system. The three basic requests are REFGENREQ, REFBGREQ, and REFMODEREQ. No interaction is required by the user code. The modules select the proper requests automatically.

A reference request signal, REFGENREQ, is available as an input into the REFGEN subsystem. This signal represents a logical OR of individual requests coming from the various modules in the system that require a voltage reference to be available on the variable reference line. When a module requires a voltage reference, it asserts its corresponding REFGENREQ signal. Once the REFGENREQ is asserted, the REFGEN subsystem will be enabled. After the specified settling time, the variable reference line voltage will be stable and ready for use. The REFVSEL settings determine which voltage will be generated on the variable reference line.

In addition to the REFGENREQ, a second reference request signal, REFBGREQ is available. The REFBGREQ signal represents a logical OR of requests coming from the various modules that require the bandgap reference line. Once the REFBGREQ is asserted, the bandgap, along with its bias circuitry and local buffer, will be enabled if it is not already enabled by a prior request.

The REFMODEREQ request signal is available that configures the bandgap and its bias circuitry to operate in a sampled or static mode of operation. The REFMODEREQ signal basically represents a logical AND of individual requests coming from the various analog modules. In reality, a REFMODEREQ occurs only if a module's REFGENREQ or REFBGQ is also asserted, otherwise it is a do not care. When REFMODEREQ = 1, the bandgap operates in sampled mode. When a module asserts its corresponding REFMODEREQ signal, it is requesting that the bandgap operate in sampled mode. Since REMODEREQ is a logical AND of all individual requests, any modules requesting static mode will cause the bandgap to operate in static mode. The BGMODE bit can be used as an indicator of static or sampled mode of operation.

16.2.3.1 REFBGACT, REFGENACT, REFGENBUSY

Any module that is using the variable reference line will cause REFGENACT to be set inside the REFCTL register. This bit is read only and indicates to the user that the REFGEN is active or off. Similarly, the REFBGACT is active any time one or more modules is actively utilizing the bandgap reference line and indicates to the user that the REFBG is active or off.

The REFGENBUSY signal, when asserted, indicates that a module is using the reference and cannot have any of it settings changed. For example, during an active ADC12_A conversion, the reference voltage level should not be changed. REFGENBUSY is asserted when there is an active ADC12_A conversion (ENC = 1) or when the DAC12_A is actively converting (DAC12AMPx > 1 and DAC12SREFx = 0). REFGENBUSY when asserted, write protects the REFCTL register. This prevents the reference from being disabled or its level changed during any active conversion. Please note that there is no such protection for the DAC12_A if the ADC12_A legacy control bits are used for the reference control. If the user changes the ADC12_A settings and the DAC12_A is using the reference, the DAC12_A conversion will be effected.

16.2.3.2 ADC10_A

For devices that contain an ADC10_A module, the ADC10_A module contains one local buffer. REFOUT must be written 0. When ADC10REFBURST = 1, the buffer is enabled only during an ADC conversion, shutting down automatically upon completion of a conversion to save power. In this case, the output of the large buffer is connected to the capacitor array via an internal analog switch. This ensures the same reference is used throughout the system.

16.2.3.3 ADC12_A

For devices that contain an ADC12_A module, the ADC12_A module contains two local buffers. The larger buffer can be used to drive the reference voltage, present on the variable reference line, external to the device. This buffer has larger power consumption due to a selectable burst mode, as well as, its need to drive larger DC loads that may be present outside the device. The large buffer is enabled continuously when REFON = 1, REFOUT =1, and ADC12REFBURST = 0. When ADC12REFBURST = 1, the buffer is enabled only during an ADC conversion, shutting down automatically upon completion of a conversion to save power. In addition, when REFON = 1 and REFOUT = 1, the second smaller buffer is automatically disabled. In this case, the output of the large buffer is connected to the capacitor array via an internal analog switch. This ensures the same reference is used throughout the system. If REFON = 1 and REFOUT = 0, the internal buffer is used for ADC conversion and the large buffer remains disabled. The small internal buffer can operate in burst mode as well by setting ADC12REFBURST = 1

16.2.3.4 DAC12_A

Some devices may contain a DAC12_A module. The DAC12_A can use the 1.5 V, 2.0 V, or 2.5 V from the variable reference line for its reference. The DAC12_A can request its reference directly by the settings within the DAC12_A module itself. Basically, if the DAC is enabled and the internal reference is selected, it will request it from the REF module. In addition, as before, setting REFON = 1 (REFMSTR = 1) or ADC12REFON = 1 (REFMSTR = 0) can enable the variable reference line independent of the DAC12_A control bits.

The REGEN subsystem will provide divided versions of the variable reference line for usage in the DAC12_A module. The DAC12_A module requires either /2 or /3 of the variable reference. The selection of these depends on the control bits inside the DAC12_A module (DAC12IR, DAC12OG) and is handled automatically by the REF module.

When the DAC12_A selects AVcc or VeREF+ as its reference, the DAC12_A has its own /2 and /3 resistor string available that scales the input reference appropriately based on the DAC12IR and DAC12OG settings.

16.2.3.5 LCD_B

Devices that contain an LCD will utilize the LCD_B module. The LCD_B module requires a reference to generate the proper LCD voltages. The bandgap reference line from the REFGEN sub-system is used for this purpose. The LCD is enabled when LCDON = 1 of the LCD_B module. This causes a REFBGREQ from the LCD module to be asserted. The buffered bandgap will be made available on the bandgap reference line for usage inside the LCD_B module.

16.3 REF Registers

The REF registers are listed in Table 16-3. The base address can be found in the device specific datasheet. The address offset is listed in Table 16-3.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 16-3. REF Registers

Register	Short Form	Register Type	Access	Address Offset	Initial State
REFCTL0	REFCTL0	Read/write	Word	00h	0080h
	REFCTL0_L	Read/write	Byte	00h	80h
	REFCTL0_H	Read/write	Byte	01h	00h

REFCTL0, REF Control Register 0

15	14	13	12	11	10	9	8
Reserved	**Reserved**	**Reserved**	**Reserved**	**BGMODE**	**REFGENBUSY**	**REFBGACT**	**REFGENACT**
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r-(0)

7	6	5	4	3	2	1	0
REFMSTR	**Reserved**	**REFVSEL**		**REFTCOFF**	**Reserved**	**REFOUT**	**REFON**
rw-(1)	r0	rw-(0)	rw-(0)	rw-(0)	r0	rw-(0)	rw-(0)

Modifiable only when REFGENBUSY = 0

Reserved	Bits 15-12	Reserved. Always reads back 0.
BGMODE	Bit 11	Bandgap mode. Read only. 0 Static mode. 1 Sampled mode.
REFGENBUSY	Bit 10	Reference generator busy. Read only. 0 Reference generator not busy. 1 Reference generator busy.
REFBGACT	Bit 9	Reference bandgap active. Read only. 0 Reference bandgap buffer not active. 1 Reference bandgap buffer active.
REFGENACT	Bit 8	Reference generator active. Read only. 0 Reference generator not active. 1 Reference generator active.
REFMSTR	Bit 7	REF master control. ADC10_A devices: Must be written 1. 0 Reference system controlled by legacy control bits inside the ADC12_A module when available. 1 Reference system controlled by REFCTL register. Common settings inside the ADC12_A module (if exists) are do not care.
Reserved	Bit 6	Reserved. Always reads back 0.
REFVSEL	Bits 5-4	Reference voltage level select 0 0 1.5 V available when reference requested or REFON = 1 0 1 2.0 V available when reference requested or REFON = 1 1 x 2.5 V available when reference requested or REFON = 1
REFTCOFF	Bit 3	Temperature sensor disabled 0 Temperature sensor enabled. 1 Temperature sensor disabled to save power.
Reserved	Bit 2	Reserved. Always reads back 0.
REFOUT	Bit 1	Reference output buffer. ADC10_A devices: Must be written 0. 0 Reference output not available externally. 1 Reference output available externally. If ADC12REFBURST = 0, or DAC12_A is enabled, output is available continuously. If ADC12REFBURST = 1, output is available only during an ADC12_A conversion.
REFON	Bit 0	Reference enable 0 Disables reference if no other reference requests are pending. 1 Enables reference.

ADC12_A

The ADC12_A module is a high-performance 12-bit analog-to-digital converter (ADC). This chapter describes the operation of the ADC12_A module.

Topic	Page
17.1 ADC12_A Introduction	**432**
17.2 ADC12_A Operation	**434**
17.3 ADC12_A Registers	**448**

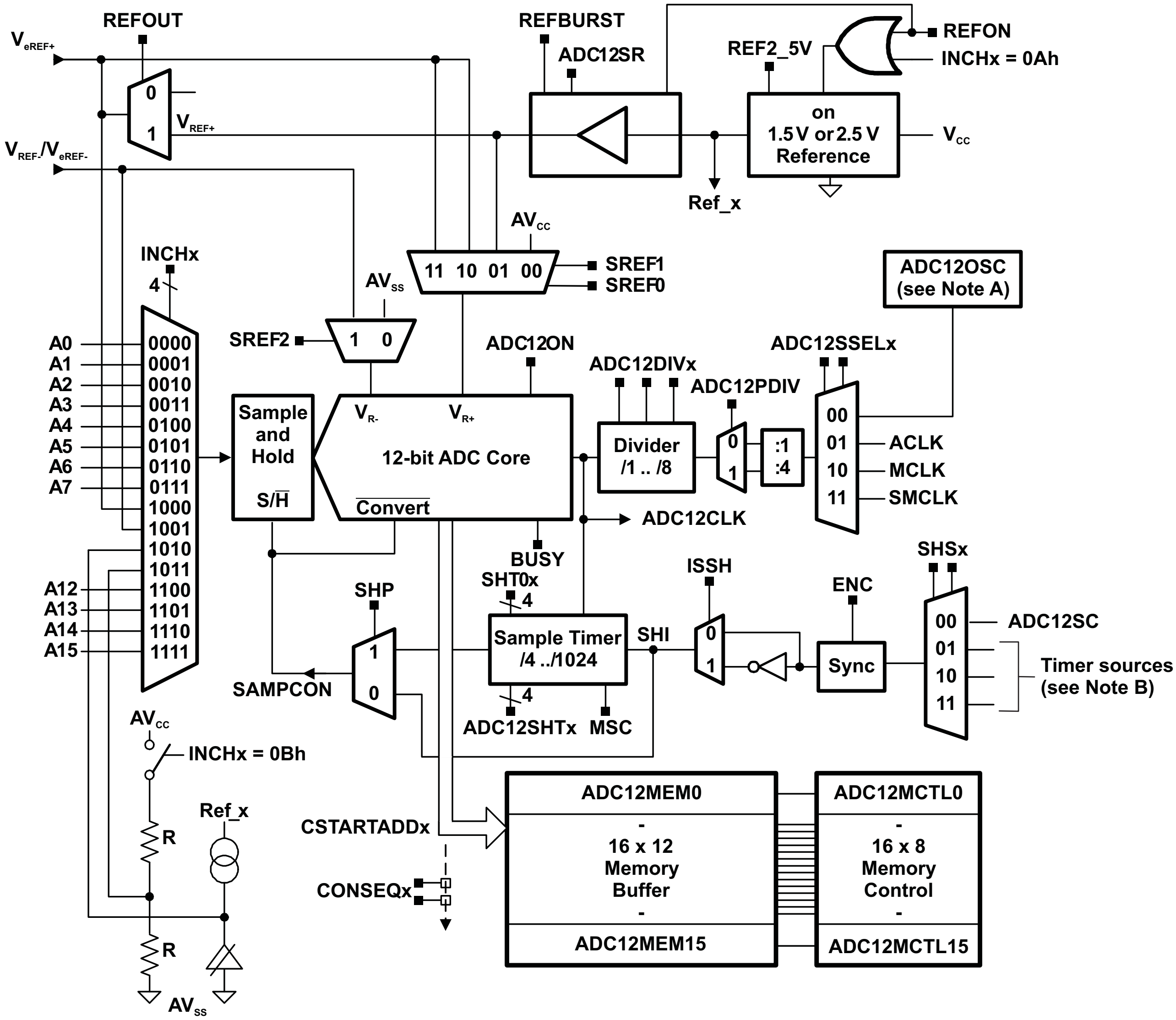
17.1 ADC12_A Introduction

The ADC12_A module supports fast 12-bit analog-to-digital conversions. The module implements a 12-bit SAR core, sample select control, reference generator (MSP430F54xx only – in other devices, separate REF module), and a 16-word conversion-and-control buffer. The conversion-and-control buffer allows up to 16 independent analog-to-digital converter (ADC) samples to be converted and stored without any CPU intervention.

ADC12_A features include:

- Greater than 200-ksps maximum conversion rate
- Monotonic 12-bit converter with no missing codes
- Sample-and-hold with programmable sampling periods controlled by software or timers.
- Conversion initiation by software or timers.
- Software-selectable on-chip reference voltage generation (MSP430F54xx: 1.5 V or 2.5 V, other devices: 1.5 V, 2.0 V, or 2.5 V)
- Software-selectable internal or external reference
- Up to 12 individually configurable external input channels
- Conversion channels for internal temperature sensor, AV_{CC}, and external references
- Independent channel-selectable reference sources for both positive and negative references
- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence (autoscan), and repeat-sequence (repeated autoscan) conversion modes
- ADC core and reference voltage can be powered down separately (MSP430F54xx only, other devices see REF module specification for details)
- Interrupt vector register for fast decoding of 18 ADC interrupts
- 16 conversion-result storage registers

The block diagram of ADC12_A is shown in Figure 17-1. The reference generation is in MSP430F54xx devices located in the ADC12_A module. In other devices, the reference generator is located in the reference module (see the device-specific data sheet).



A The MODOSC is part of the UCS. See the UCS chapter for more information.

B See the device-specific data sheet for timer sources available.

Figure 17-1. ADC12_A Block Diagram

17.2 ADC12_A Operation

The ADC12_A module is configured with user software. The setup and operation of the ADC12_A is discussed in the following sections.

17.2.1 12-Bit ADC Core

The ADC core converts an analog input to its 12-bit digital representation and stores the result in conversion memory. The core uses two programmable/selectable voltage levels (V_{R+} and V_{R-}) to define the upper and lower limits of the conversion. The digital output (N_{ADC}) is full scale (0FFFh) when the input signal is equal to or higher than V_{R+}, and zero when the input signal is equal to or lower than V_{R-}. The input channel and the reference voltage levels (V_{R+} and V_{R-}) are defined in the conversion-control memory. The conversion formula for the ADC result N_{ADC} is:

$$N_{ADC} = 4095 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

The ADC12_A core is configured by two control registers, ADC12CTL0 and ADC12CTL1. The core is enabled with the ADC12ON bit. The ADC12_A can be turned off when not in use to save power. With few exceptions, the ADC12_A control bits can only be modified when ADC12ENC = 0. ADC12ENC must be set to 1 before any conversion can take place.

17.2.1.1 Conversion Clock Selection

The ADC12CLK is used both as the conversion clock and to generate the sampling period when the pulse sampling mode is selected. The ADC12_A source clock is selected using the predivider controlled by the ADC12PDIV bit and the divider using the ADC12SSELx bits. The input clock can be divided from 1–32 using both the ADC12DIVx bits and the ADC12PDIV bit. Possible ADC12CLK sources are SMCLK, MCLK, ACLK, and the ADC12OSC.

The ADC12OSC in the block diagram refers to the MODOSC 5 MHz oscillator from the UCS (see the UCS module for more information) which can vary with individual devices, supply voltage, and temperature. See the device-specific data sheet for the ADC12OSC specification.

The user must ensure that the clock chosen for ADC12CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation does not complete and any result is invalid.

17.2.2 ADC12_A Inputs and Multiplexer

The 12 external and 4 internal analog signals are selected as the channel for conversion by the analog input multiplexer. The input multiplexer is a break-before-make type to reduce input-to-input noise injection resulting from channel switching (see Figure 17-2). The input multiplexer is also a T-switch to minimize the coupling between channels. Channels that are not selected are isolated from the A/D and the intermediate node is connected to analog ground (AV_{SS}), so that the stray capacitance is grounded to eliminate crosstalk.

The ADC12_A uses the charge redistribution method. When the inputs are internally switched, the switching action may cause transients on the input signal. These transients decay and settle before causing errant conversion.

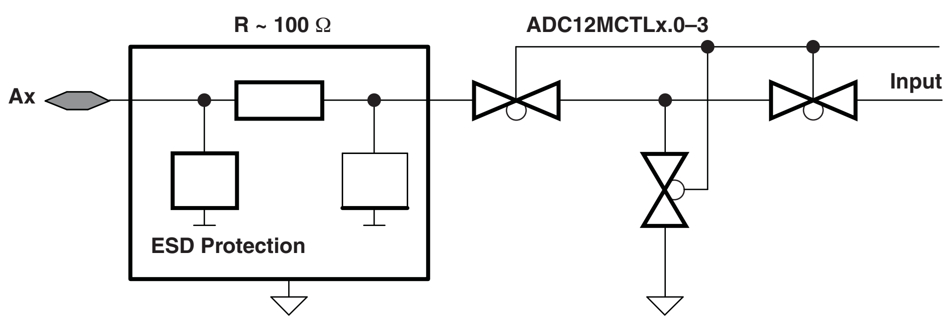


Figure 17-2. Analog Multiplexer

17.2.2.1 Analog Port Selection

The ADC12_A inputs are multiplexed with digital port pins. When analog signals are applied to digital gates, parasitic current can flow from V_{CC} to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the digital pat of the port pin eliminates the parasitic current flow and, therefore, reduces overall current consumption. The PySELx bits provide the ability to disable the port pin input and output buffers.

```
; Py.0 and Py.1 configured for analog input
      BIS.B    #3h,&PySEL   ; Py.1 and Py.0 ADC12_A function
```

17.2.3 Voltage Reference Generator

The ADC12_A module of the MSP430F54xx contains a built-in voltage reference with two selectable voltage levels, 1.5 V and 2.5 V. Either of these reference voltages may be used internally and externally on pin V_{REF+}.

The ADC12_A modules of other devices have a separate reference module that supplies three selectable voltage levels, 1.5 V, 2.0 V, and 2.5 V to the ADC12_A. Either of these voltages may be used internally and externally on pin V_{REF+}.

Setting ADC12REFON = 1 enables the reference voltage of the ADC12_A module. When ADC12REF2_5V = 1, the internal reference is 2.5 V; when ADC12REF2_5V = 0, the reference is 1.5 V . The reference can be turned off to save power when not in use. Devices with the REF module can use the control bits located in the ADC12_A module, or the control registers located in the REF module to control the reference voltage supplied to the ADC. Per default, the register settings of the REF module define the reference voltage settings. The control bit REFMSTR in the REF module is used to hand over control to the ADC12_A reference control register settings. If the register bit REFMSTR is set to 1 (default), the REF module registers control the reference settings. If REFMSTR is set to 0, the ADC12_A reference setting define the reference voltage of the ADC12_A module.

External references may be supplied for V_{R+} and V_{R-} through pins V_{REF+}/Ve_{REF+} and V_{REF-}/Ve_{REF-} , respectively.

External storage capacitors are only required if REFOUT = 1 and the reference voltage is made available at the pins.

17.2.3.1 Internal Reference Low-Power Features

The ADC12_A internal reference generator is designed for low-power applications. The reference generator includes a band-gap voltage source and a separate buffer. The current consumption and settling time of each is specified separately in the device-specific data sheet. When ADC12REFON = 1, both are enabled, and if ADC12REFON = 0, both are disabled.

When ADC12REFON = 1 and REFBURST = 1 but no conversion is active, the buffer is automatically disabled and automatically reenabled when needed. When the buffer is disabled, it consumes no current. In this case, the band-gap voltage source remains enabled.

The REFBURST bit controls the operation of the reference buffer. When REFBURST = 1, the buffer is automatically disabled when the ADC12_A is not actively converting, and automatically reenabled when needed. When REFBURST = 0, the buffer is on continuously. This allows the reference voltage to be present outside the device continuously if REFOUT = 1.

The internal reference buffer also has selectable speed versus power settings. When the maximum conversion rate is below 50 ksps, setting ADC12SR = 1 reduces the current consumption of the buffer approximately 50%.

17.2.4 Auto Power Down

The ADC12_A is designed for low-power applications. When the ADC12_A is not actively converting, the core is automatically disabled and automatically reenabled when needed. The MODOSC is also automatically enabled when needed and disabled when not needed.

17.2.5 Sample and Conversion Timing

An analog-to-digital conversion is initiated with a rising edge of the sample input signal SHI. The source for SHI is selected with the SHSx bits and includes the following:

- ADC12SC bit
- Up to three timer outputs (see to the device-specific data sheet for available timer sources).

The ADC12_A supports 8-bit, 10-bit, and 12-bit resolution modes selectable by the ADC12RES bits. The analog-to-digital conversion requires 9, 11, and 13 ADC12CLK cycles, respectively. The polarity of the SHI signal source can be inverted with the ADC12ISSH bit. The SAMPCON signal controls the sample period and start of conversion. When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the analog-to-digital conversion. Two different sample-timing methods are defined by control bit ADC12SHP, extended sample mode, and pulse mode. See the device-specific data sheet for available timers for SHI sources.

17.2.5.1 Extended Sample Mode

The extended sample mode is selected when ADC12SHP = 0. The SHI signal directly controls SAMPCON and defines the length of the sample period $t_{sample.}$ When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the conversion after synchronization with ADC12CLK (see Figure 17-3).

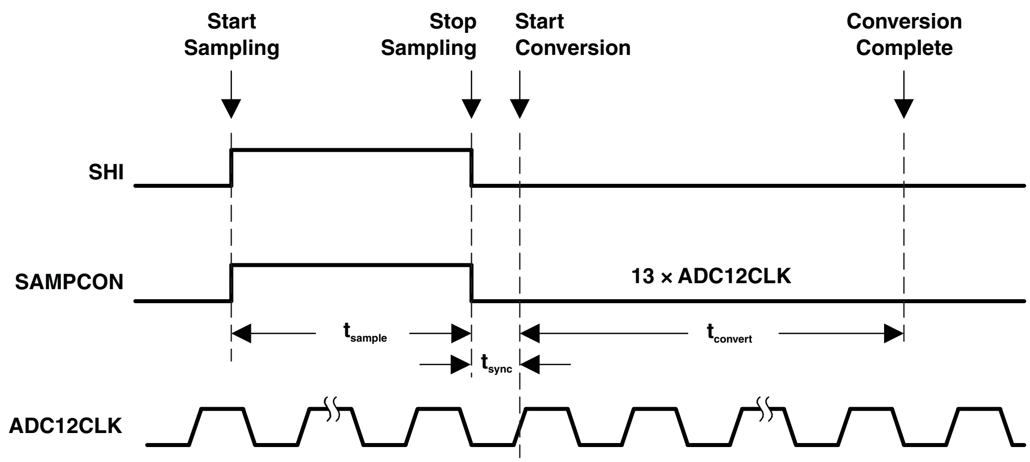


Figure 17-3. Extended Sample Mode

17.2.5.2 Pulse Sample Mode

The pulse sample mode is selected when ADC12SHP = 1. The SHI signal is used to trigger the sampling timer. The ADC12SHT0x and ADC12SHT1x bits in ADC12CTL0 control the interval of the sampling timer that defines the SAMPCON sample period $t_{sample.}$ The sampling timer keeps SAMPCON high after synchronization with AD12CLK for a programmed interval t_{sample}. The total sampling time is t_{sample} plus t_{sync} (see Figure 17-4).

The ADC12SHTx bits select the sampling time in 4× multiples of ADC12CLK. ADC12SHT0x selects the sampling time for ADC12MCTL0 to ADC12MCTL7, and ADC12SHT1x selects the sampling time for ADC12MCTL8 to ADC12MCTL15.

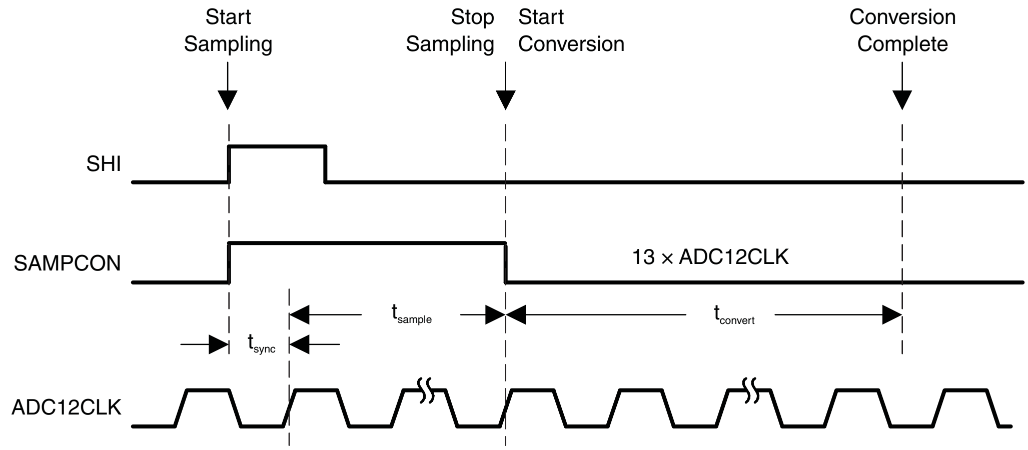


Figure 17-4. Pulse Sample Mode

17.2.5.3 Sample Timing Considerations

When SAMPCON = 0, all Ax inputs are high impedance. When SAMPCON = 1, the selected Ax input can be modeled as an RC low-pass filter during the sampling time t_{sample} (see Figure 17-5). An internal MUX-on input resistance R_I (maximum 1.8 kΩ) in series with capacitor C_I (25 pF maximum) is seen by the source. The capacitor C_I voltage V_C must be charged to within one-half LSB of the source voltage V_S for an accurate n-bit conversion, where n is the bits of resolution required.

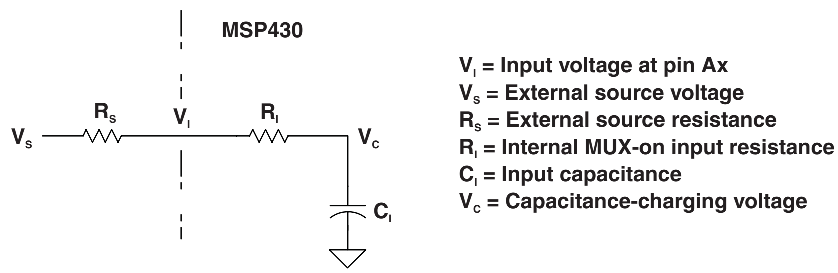


Figure 17-5. Analog Input Equivalent Circuit

The resistance of the source R_S and R_I affect t_{sample}. The following equation can be used to calculate the minimum sampling time t_{sample} for a n-bit conversion, where n equals the bits of resolution:

$$t_{sample} > (R_S + R_I) \times \ln(2^{n+1}) \times C_I + 800 \text{ ns}$$

Substituting the values for R_I and C_I given above, the equation becomes:

$$t_{sample} > (R_S + 1.8 \text{ k}\Omega) \times \ln(2^{n+1}) \times 25 \text{ pF} + 800 \text{ ns}$$

For example, for 12-bit resolution, if R_S is 10 kΩ, t_{sample} must be greater than 3.46 µs.

17.2.6 Conversion Memory

There are 16 ADC12MEMx conversion memory registers to store conversion results. Each ADC12MEMx is configured with an associated ADC12MCTLx control register. The SREFx bits define the voltage reference and the INCHx bits select the input channel. The ADC12EOS bit defines the end of sequence when a sequential conversion mode is used. A sequence rolls over from ADC12MEM15 to ADC12MEM0 when the ADC12EOS bit in ADC12MCTL15 is not set.

The CSTARTADDx bits define the first ADC12MCTLx used for any conversion. If the conversion mode is single-channel or repeat-single-channel, the CSTARTADDx points to the single ADC12MCTLx to be used.

If the conversion mode selected is either sequence-of-channels or repeat-sequence-of-channels, CSTARTADDx points to the first ADC12MCTLx location to be used in a sequence. A pointer, not visible to software, is incremented automatically to the next ADC12MCTLx in a sequence when each conversion completes. The sequence continues until an ADC12EOS bit in ADC12MCTLx is processed; this is the last control byte processed.

When conversion results are written to a selected ADC12MEMx, the corresponding flag in the ADC12IFGx register is set.

There are two formats available to store the conversion result, ADC12MEMx. When ADC12DF = 0, the conversion is right justified, unsigned. For 8-bit, 10-bit, and 12-bit resolutions, the upper 8, 6, and 4 bits of ADC12MEMx are always zeros, respectively. When ADC12DF = 1, the conversion result is left justified, two's complement. For 8-bit, 10-bit, and 12-bit resolutions, the lower 8, 6, and 4 bits of ADC12MEMx are always zeros, respectively. This is summarized in Table 17-1.

Table 17-1. ADC12_A Conversion Result Formats

Analog Input Voltage	ADC12DF	ADC12RES	Ideal Conversion Results	ADC12MEMx
$-V_{REF}$ to $+V_{REF}$	0	00	0 to 255	0000h - 00FFh
	0	01	0 to 1023	0000h - 03FFh
	0	10	0 to 4095	0000h - 0FFFh
	1	00	-128 to 127	8000h - 7F00h
	1	01	-512 to 511	8000h - 7FC0h
	1	10	-2048 to 2047	8000h - 7FF0h

17.2.7 ADC12_A Conversion Modes

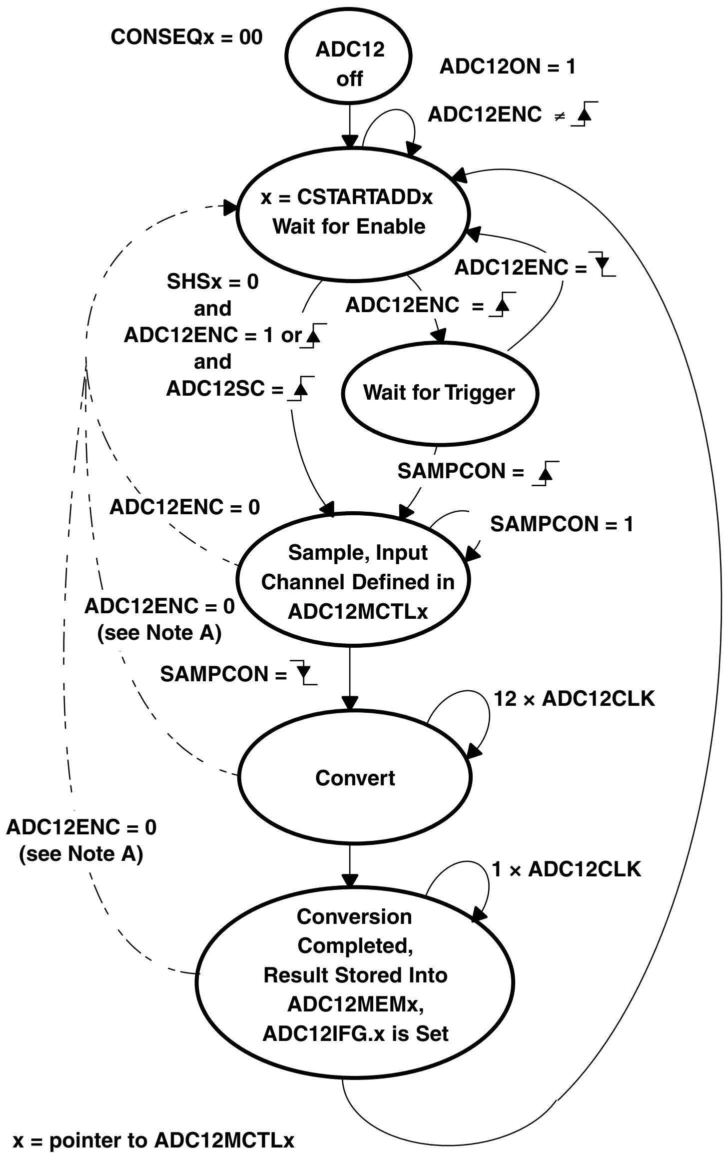
The ADC12_A has four operating modes selected by the CONSEQx bits as listed in Table 17-2. All state diagrams assume a 12-bit resolution setting.

Table 17-2. Conversion Mode Summary

ADC12CONSEQx	Mode	Operation
00	Single-channel single-conversion	A single channel is converted once.
01	Sequence-of-channels (autoscan)	A sequence of channels is converted once.
10	Repeat-single-channel	A single channel is converted repeatedly.
11	Repeat-sequence-of-channels (repeated autoscan)	A sequence of channels is converted repeatedly.

17.2.7.1 Single-Channel Single-Conversion Mode

A single channel is sampled and converted once. The ADC result is written to the ADC12MEMx defined by the CSTARTADDx bits. Figure 17-6 shows the flow of the single-channel single-conversion mode. When ADC12SC triggers a conversion, successive conversions can be triggered by the ADC12SC bit. When any other trigger source is used, ADC12ENC must be toggled between each conversion.



A Conversion result is unpredictable.

Figure 17-6. Single-Channel Single-Conversion Mode

17.2.7.2 Sequence-of-Channels Mode (Autoscan Mode)

In sequence-of-channels mode, also referred to as autoscan mode, a sequence of channels is sampled and converted once. The ADC results are written to the conversion memories starting with the ADCMEMx defined by the CSTARTADDx bits. The sequence stops after the measurement of the channel with a set ADC12EOS bit. Figure 17-7 shows the sequence-of-channels mode. When ADC12SC triggers a sequence, successive sequences can be triggered by the ADC12SC bit. When any other trigger source is used, ADC12ENC must be toggled between each sequence.

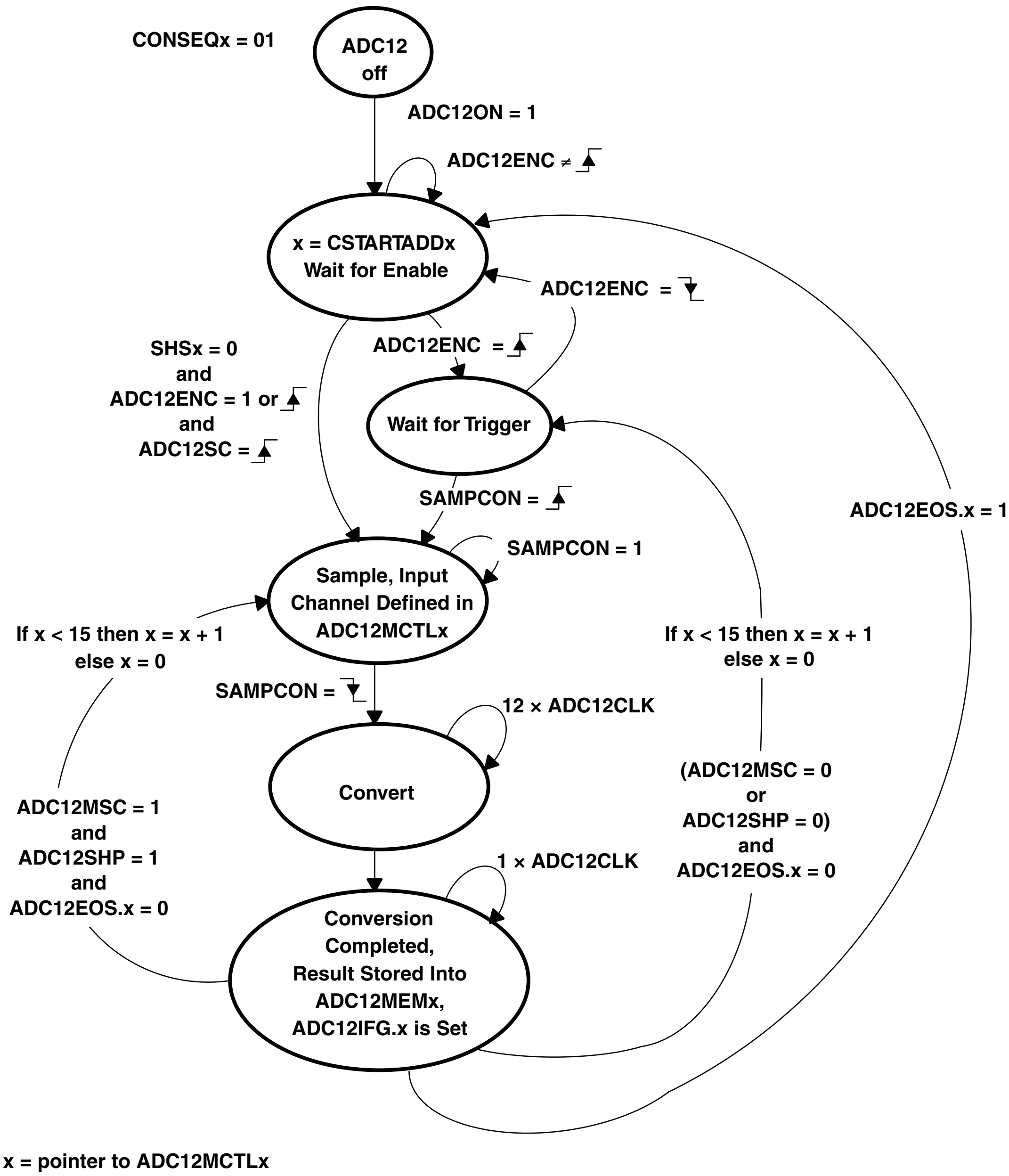


Figure 17-7. Sequence-of-Channels Mode

17.2.7.3 Repeat-Single-Channel Mode

A single channel is sampled and converted continuously. The ADC results are written to the ADC12MEMx defined by the CSTARTADDx bits. It is necessary to read the result after the completed conversion because only one ADC12MEMx memory is used and is overwritten by the next conversion. Figure 17-8 shows the repeat-single-channel mode.

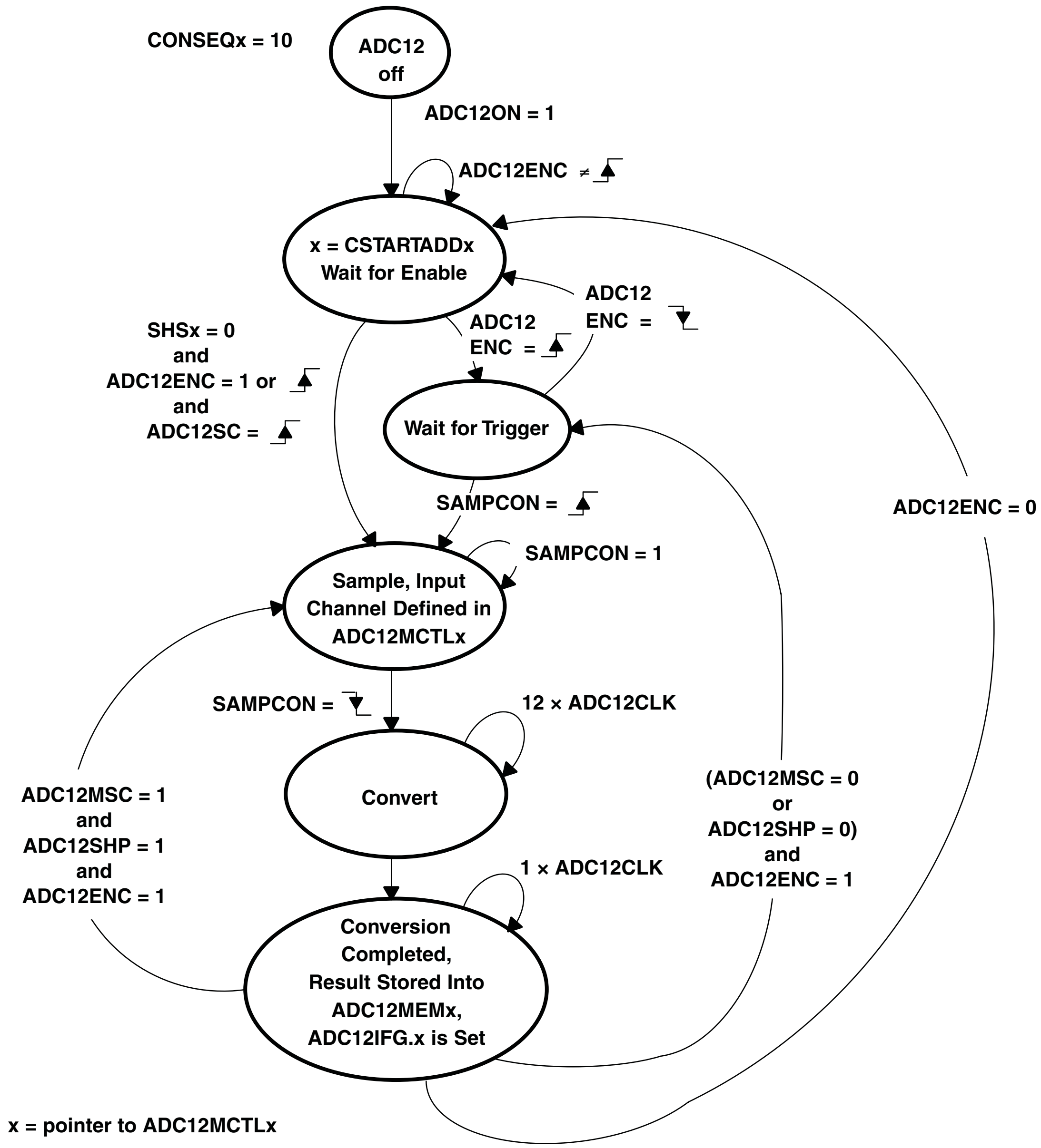


Figure 17-8. Repeat-Single-Channel Mode

17.2.7.4 Repeat-Sequence-of-Channels Mode (Repeated Autoscan Mode)

In this mode, a sequence of channels is sampled and converted repeatedly. This mode is also referred to as repeated autoscan mode. The ADC results are written to the conversion memories starting with the ADC12MEMx defined by the CSTARTADDx bits. The sequence ends after the measurement of the channel with a set ADC12EOS bit and the next trigger signal restarts the sequence. Figure 17-9 shows the repeat-sequence-of-channels mode.

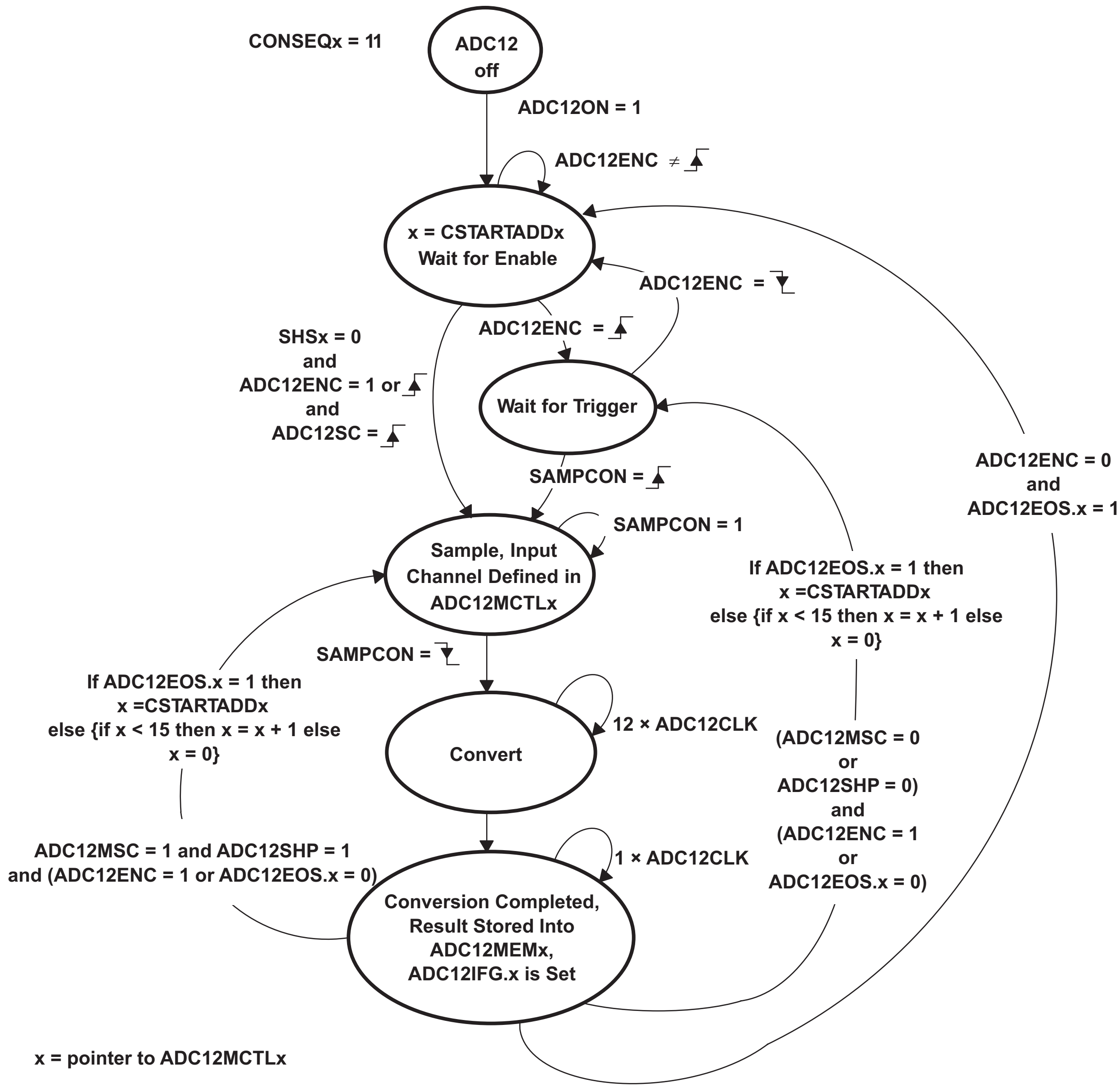


Figure 17-9. Repeat-Sequence-of-Channels Mode

17.2.7.5 Using the Multiple Sample and Convert (ADC12MSC) Bit

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is available. When ADC12MSC = 1, CONSEQx > 0, and the sample timer is used, the first rising edge of the SHI signal triggers the first conversion. Successive conversions are triggered automatically as soon as the prior conversion is completed. Additional rising edges on SHI are ignored until the sequence is completed in the single-sequence mode, or until the ADC12ENC bit is toggled in repeat-single-channel or repeated-sequence modes. The function of the ADC12ENC bit is unchanged when using the ADC12MSC bit.

17.2.7.6 Stopping Conversions

Stopping ADC12_A activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Resetting ADC12ENC in single-channel single-conversion mode stops a conversion immediately and the results are unpredictable. For correct results, poll the busy bit until reset before clearing ADC12ENC.
- Resetting ADC12ENC during repeat-single-channel operation stops the converter at the end of the current conversion.
- Resetting ADC12ENC during a sequence or repeat-sequence mode stops the converter at the end of the sequence.
- Any conversion mode may be stopped immediately by setting the CONSEQx = 0 and resetting the ADC12ENC bit. Conversion data are unreliable.

NOTE: No ADC12EOS bit set for sequence

If no ADC12EOS bit is set and a sequence mode is selected, resetting the ADC12ENC bit does not stop the sequence. To stop the sequence, first select a single-channel mode and then reset ADC12ENC.

17.2.8 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel INCHx = 1010. Any other configuration is done as if an external channel was selected, including reference selection, conversion-memory selection, etc. The temperature sensor is in the ADC12_A in the MSP430F54xx devices, while it is part of the REF module in other devices.

A typical temperature sensor transfer function is shown in Figure 17-10 . The transfer function shown below is only an example. The device-specific data sheet contains the actual parameters for a given device. When using the temperature sensor, the sample period must be greater than 30 µs. The temperature sensor offset error can be large and may need to be calibrated for most applications. Temperature calibration values are available for use in the TLV descriptors (please see the device-specific data sheet for locations).

Selecting the temperature sensor automatically turns on the on-chip reference generator as a voltage source for the temperature sensor. However, it does not enable the V_{REF+} output or affect the reference selections for the conversion. The reference choices for converting the temperature sensor are the same as with any other channel.

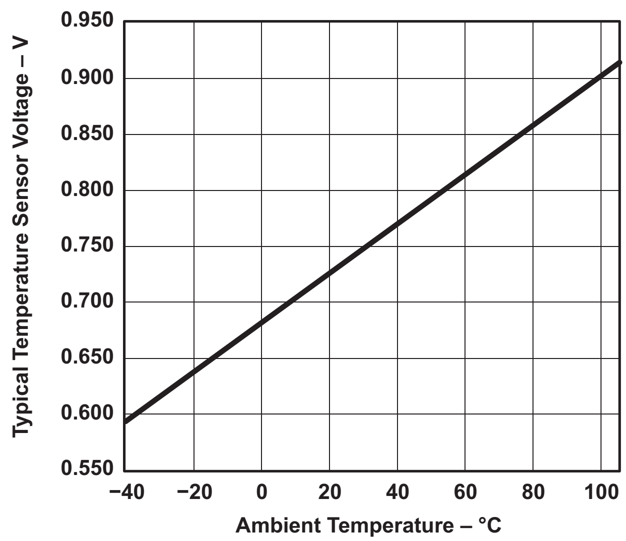


Figure 17-10. Typical Temperature Sensor Transfer Function

17.2.9 ADC12_A Grounding and Noise Considerations

As with any high-resolution ADC, appropriate printed-circuit-board layout and grounding techniques should be followed to eliminate ground loops, unwanted parasitic effects, and noise.

Ground loops are formed when return current from the A/D flows through paths that are common with other analog or digital circuitry. If care is not taken, this current can generate small, unwanted offset voltages that can add to or subtract from the reference or input voltages of the ADC. The connections shown in Figure 17-11 prevent this.

In addition to grounding, ripple and noise spikes on the power-supply lines due to digital switching or switching power supplies can corrupt the conversion result. A noise-free design using separate analog and digital ground planes with a single-point connection is recommend to achieve high accuracy.

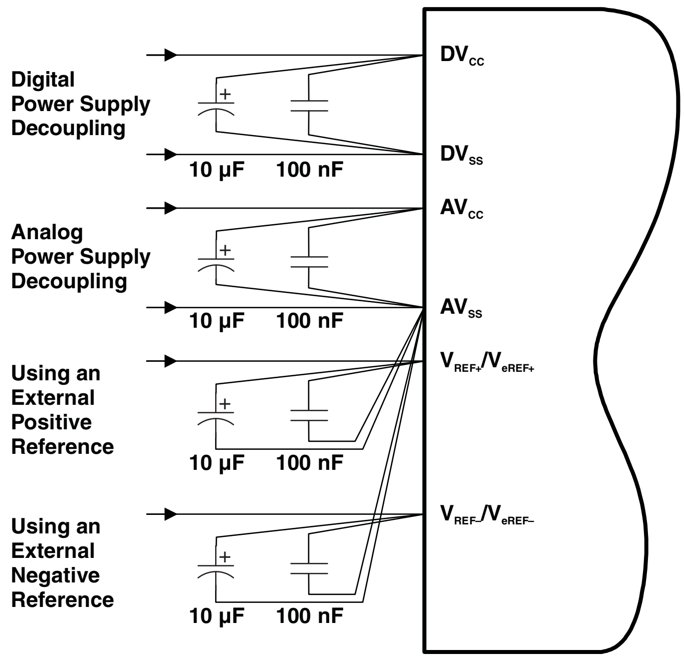


Figure 17-11. ADC12_A Grounding and Noise Considerations

17.2.10 ADC12_A Interrupts

The ADC12_A has 18 interrupt sources:

- ADC12IFG0-ADC12IFG15
- ADC12OV, ADC12MEMx overflow
- ADC12TOV, ADC12_A conversion time overflow

The ADC12IFGx bits are set when their corresponding ADC12MEMx memory register is loaded with a conversion result. An interrupt request is generated if the corresponding ADC12IEx bit and the GIE bit are set. The ADC12OV condition occurs when a conversion result is written to any ADC12MEMx before its previous conversion result was read. The ADC12TOV condition is generated when another sample-and-conversion is requested before the current conversion is completed. The DMA is triggered after the conversion in single-channel conversion mode or after the completion of a sequence of channel conversions in sequence-of-channels conversion mode.

17.2.10.1 ADC12IV, Interrupt Vector Generator

All ADC12_A interrupt sources are prioritized and combined to source a single interrupt vector. The interrupt vector register ADC12IV is used to determine which enabled ADC12_A interrupt source requested an interrupt.

The highest-priority enabled ADC12_A interrupt generates a number in the ADC12IV register (see register description). This number can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled ADC12_A interrupts do not affect the ADC12IV value.

Any access, read or write, of the ADC12IV register automatically resets the ADC12OV condition or the ADC12TOV condition, if either was the highest-pending interrupt. Neither interrupt condition has an accessible interrupt flag. The ADC12IFGx flags are not reset by an ADC12IV access. ADC12IFGx bits are reset automatically by accessing their associated ADC12MEMx register or may be reset with software.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if the ADC12OV and ADC12IFG3 interrupts are pending when the interrupt service routine accesses the ADC12IV register, the ADC12OV interrupt condition is reset automatically. After the RETI instruction of the interrupt service routine is executed, the ADC12IFG3 generates another interrupt.

17.2.10.2 ADC12_A Interrupt Handling Software Example

The following software example shows the recommended use of the ADC12IV and handling overhead. The ADC12IV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- ADC12IFG0–ADC12IFG14, ADC12TOV, and ADC12OV: 16 cycles
- ADC12IFG15: 14 cycles

The interrupt handler for ADC12IFG15 shows a way to check immediately if a higher-prioritized interrupt occurred during the processing of ADC12IFG15. This saves nine cycles if another ADC12_A interrupt is pending.

```
; Interrupt handler for ADC12.
INT_ADC12                ; Enter Interrupt Service Routine
   ADD      &ADC12IV,PC  ; Add offset to PC
   RETI                  ; Vector 0: No interrupt
   JMP      ADOV         ; Vector 2: ADC overflow
   JMP      ADTOV        ; Vector 4: ADC timing overflow
   JMP      ADM0         ; Vector 6: ADC12IFG0
      ...                ; Vectors 8-32
   JMP      ADM14        ; Vector 34: ADC12IFG14
;
; Handler for ADC12IFG15 starts here. No JMP required.
;
ADM15     MOV   &ADC12MEM15,xxx     ; Move result, flag is reset
          ...                       ; Other instruction needed?
          JMP   INT_ADC12           ; Check other int pending
;
; ADC12IFG14-ADC12IFG1 handlers go here
;
ADM0      MOV   &ADC12MEM0,xxx      ; Move result, flag is reset
          ...                       ; Other instruction needed?
RETI                                ; Return
;
ADTOV     ...                       ; Handle Conv. time overflow
          RETI                      ; Return
;
ADOV      ...                       ; Handle ADCMEMx overflow
          RETI                      ; Return
```

17.3 ADC12_A Registers

The ADC12_A registers are listed in Table 17-3. The base address of the ADC12_A can be found in the device-specific data sheet. The address offset of each ADC12_A register is given in Table 17-3.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 17-3. ADC12_A Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
ADC12_A Control 0	ADC12CTL0	Read/write	Word	00h	0000h
	ADC12CTL0_L	Read/write	Byte	00h	00h
	ADC12CTL0_H	Read/write	Byte	01h	00h
ADC12_A Control 1	ADC12CTL1	Read/write	Word	02h	0000h
	ADC12CTL1_L	Read/write	Byte	02h	00h
	ADC12CTL1_H	Read/write	Byte	03h	00h
ADC12_A Control 2	ADC12CTL2	Read/write	Word	04h	0020h
	ADC12CTL2_L	Read/write	Byte	04h	20h
	ADC12CTL2_H	Read/write	Byte	05h	00h
ADC12_A Interrupt Flag	ADC12IFG	Read/write	Word	0Ah	0000h
	ADC12IFG_L	Read/write	Byte	0Ah	00h
	ADC12IFG_H	Read/write	Byte	0Bh	00h
ADC12_A Interrupt Enable	ADC12IE	Read/write	Word	0Ch	0000h
	ADC12IE_L	Read/write	Byte	0Ch	00h
	ADC12IE_H	Read/write	Byte	0Dh	00h
ADC12_A Interrupt Vector	ADC12IV	Read	Word	0Eh	0000h
	ADC12IV_L	Read	Byte	0Eh	00h
	ADC12IV_H	Read	Byte	0Fh	00h
ADC12_A Memory 0	ADC12MEM0	Read/write	Word	20h	undefined
	ADC12MEM0_L	Read/write	Byte	20h	undefined
	ADC12MEM0_H	Read/write	Byte	21h	undefined
ADC12_A Memory 1	ADC12MEM1	Read/write	Word	22h	undefined
	ADC12MEM1_L	Read/write	Byte	22h	undefined
	ADC12MEM1_H	Read/write	Byte	23h	undefined
ADC12_A Memory 2	ADC12MEM2	Read/write	Word	24h	undefined
	ADC12MEM2_L	Read/write	Byte	24h	undefined
	ADC12MEM2_H	Read/write	Byte	25h	undefined
ADC12_A Memory 3	ADC12MEM3	Read/write	Word	26h	undefined
	ADC12MEM3_L	Read/write	Byte	26h	undefined
	ADC12MEM3_H	Read/write	Byte	27h	undefined
ADC12_A Memory 4	ADC12MEM4	Read/write	Word	28h	undefined
	ADC12MEM4_L	Read/write	Byte	28h	undefined
	ADC12MEM4_H	Read/write	Byte	29h	undefined
ADC12_A Memory 5	ADC12MEM5	Read/write	Word	2Ah	undefined
	ADC12MEM5_L	Read/write	Byte	2Ah	undefined
	ADC12MEM5_H	Read/write	Byte	2Bh	undefined

Table 17-3. ADC12_A Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
ADC12_A Memory 6	ADC12MEM6	Read/write	Word	2Ch	undefined
	ADC12MEM6_L	Read/write	Byte	2Ch	undefined
	ADC12MEM6_H	Read/write	Byte	2Dh	undefined
ADC12_A Memory 7	ADC12MEM7	Read/write	Word	2Eh	undefined
	ADC12MEM7_L	Read/write	Byte	2Eh	undefined
	ADC12MEM7_H	Read/write	Byte	2Fh	undefined
ADC12_A Memory 8	ADC12MEM8	Read/write	Word	30h	undefined
	ADC12MEM8_L	Read/write	Byte	30h	undefined
	ADC12MEM8_H	Read/write	Byte	31h	undefined
ADC12_A Memory 9	ADC12MEM9	Read/write	Word	32h	undefined
	ADC12MEM9_L	Read/write	Byte	32h	undefined
	ADC12MEM9_H	Read/write	Byte	33h	undefined
ADC12_A Memory 10	ADC12MEM10	Read/write	Word	34h	undefined
	ADC12MEM10_L	Read/write	Byte	34h	undefined
	ADC12MEM10_H	Read/write	Byte	35h	undefined
ADC12_A Memory 11	ADC12MEM11	Read/write	Word	36h	undefined
	ADC12MEM11_L	Read/write	Byte	36h	undefined
	ADC12MEM11_H	Read/write	Byte	37h	undefined
ADC12_A Memory 12	ADC12MEM12	Read/write	Word	38h	undefined
	ADC12MEM12_L	Read/write	Byte	38h	undefined
	ADC12MEM12_H	Read/write	Byte	39h	undefined
ADC12_A Memory 13	ADC12MEM13	Read/write	Word	3Ah	undefined
	ADC12MEM13_L	Read/write	Byte	3Ah	undefined
	ADC12MEM13_H	Read/write	Byte	3Bh	undefined
ADC12_A Memory 14	ADC12MEM14	Read/write	Word	3Ch	undefined
	ADC12MEM14_L	Read/write	Byte	3Ch	undefined
	ADC12MEM14_H	Read/write	Byte	3Dh	undefined
ADC12_A Memory 15	ADC12MEM15	Read/write	Word	3Dh	undefined
	ADC12MEM15_L	Read/write	Byte	3Dh	undefined
	ADC12MEM15_H	Read/write	Byte	3Eh	undefined
ADC12_A Memory Control 0	ADC12MCTL0	Read/write	Byte	10h	undefined
ADC12_A Memory Control 1	ADC12MCTL1	Read/write	Byte	11h	undefined
ADC12_A Memory Control 2	ADC12MCTL2	Read/write	Byte	12h	undefined
ADC12_A Memory Control 3	ADC12MCTL3	Read/write	Byte	13h	undefined
ADC12_A Memory Control 4	ADC12MCTL4	Read/write	Byte	14h	undefined
ADC12_A Memory Control 5	ADC12MCTL5	Read/write	Byte	15h	undefined
ADC12_A Memory Control 6	ADC12MCTL6	Read/write	Byte	16h	undefined
ADC12_A Memory Control 7	ADC12MCTL7	Read/write	Byte	17h	undefined
ADC12_A Memory Control 8	ADC12MCTL8	Read/write	Byte	18h	undefined
ADC12_A Memory Control 9	ADC12MCTL9	Read/write	Byte	19h	undefined
ADC12_A Memory Control 10	ADC12MCTL10	Read/write	Byte	1Ah	undefined
ADC12_A Memory Control 11	ADC12MCTL11	Read/write	Byte	1Bh	undefined
ADC12_A Memory Control 12	ADC12MCTL12	Read/write	Byte	1Ch	undefined
ADC12_A Memory Control 13	ADC12MCTL13	Read/write	Byte	1Dh	undefined
ADC12_A Memory Control 14	ADC12MCTL14	Read/write	Byte	1Eh	undefined
ADC12_A Memory Control 15	ADC12MCTL15	Read/write	Byte	1Fh	undefined

ADC12_A Control Register 0 (ADC12CTL0)

15	14	13	12	11	10	9	8
ADC12SHT1x				ADC12SHT0x			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC12MSC	ADC12 REF2_5V	ADC12 REFON	ADC12ON	ADC12OVIE	ADC12TOVIE	ADC12ENC	ADC12SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Modifiable only when ADC12ENC = 0

ADC12SHT1x Bits 15-12 ADC12_A sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM8 to ADC12MEM15.

ADC12SHT0x Bits 11-8 ADC12_A sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM0 to ADC12MEM7.

ADC12SHTx Bits	ADC12CLK Cycles
0000	4
0001	8
0010	16
0011	32
0100	64
0101	96
0110	128
0111	192
1000	256
1001	384
1010	512
1011	768
1100	1024
1101	1024
1110	1024
1111	1024

ADC12MSC Bit 7 ADC12_A multiple sample and conversion. Valid only for sequence or repeated modes.
- 0 The sampling timer requires a rising edge of the SHI signal to trigger each sample-and-convert.
- 1 The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed.

ADC12REF2_5V Bit 6 ADC12_A reference generator voltage. ADC12REFON must also be set.
- 0 1.5 V
- 1 2.5 V

ADC12REFON Bit 5 ADC12_A reference generator on. In devices with the REF module, this bit is only valid if the REFMSTR bit of the REF module is set to 0. In the 'F54xx device, the REF module is not available.
- 0 Reference off
- 1 Reference on

ADC12ON Bit 4 ADC12_A on
- 0 ADC12_A off
- 1 ADC12_A on

(continued)

ADC12OVIE	Bit 3	ADC12MEMx overflow-interrupt enable. The GIE bit must also be set to enable the interrupt. 0 Overflow interrupt disabled 1 Overflow interrupt enabled
ADC12TOVIE	Bit 2	ADC12_A conversion-time-overflow interrupt enable. The GIE bit must also be set to enable the interrupt. 0 Conversion time overflow interrupt disabled 1 Conversion time overflow interrupt enabled
ADC12ENC	Bit 1	ADC12_A enable conversion 0 ADC12_A disabled 1 ADC12_A enabled
ADC12SC	Bit 0	ADC12_A start conversion. Software-controlled sample-and-conversion start. ADC12SC and ADC12ENC may be set together with one instruction. ADC12SC is reset automatically. 0 No sample-and-conversion-start 1 Start sample-and-conversion

ADC12_A Control Register 1 (ADC12CTL1)

15	14	13	12	11	10	9	8
ADC12CSTARTADDx				**ADC12SHSx**		**ADC12SHP**	**ADC12ISSH**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC12DIVx			**ADC12SSELx**		**ADC12CONSEQx**		**ADC12BUSY**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

(shaded) Modifiable only when ADC12ENC = 0

ADC12CSTARTADDx	Bits 15-12	ADC12_A conversion start address. These bits select which ADC12_A conversion-memory register is used for a single conversion or for the first conversion in a sequence. The value of CSTARTADDx is 0 to 0Fh, corresponding to ADC12MEM0 to ADC12MEM15.
ADC12SHSx	Bits 11-10	ADC12_A sample-and-hold source select 00 ADC12SC bit 01 Timer source (see device-specific data sheet for exact timer and locations) 10 Timer source (see device-specific data sheet for exact timer and locations) 11 Timer source (see device-specific data sheet for exact timer and locations)
ADC12SHP	Bit 9	ADC12_A sample-and-hold pulse-mode select. This bit selects the source of the sampling signal (SAMPCON) to be either the output of the sampling timer or the sample-input signal directly. 0 SAMPCON signal is sourced from the sample-input signal. 1 SAMPCON signal is sourced from the sampling timer.
ADC12ISSH	Bit 8	ADC12_A invert signal sample-and-hold 0 The sample-input signal is not inverted. 1 The sample-input signal is inverted.
ADC12DIVx	Bits 7-5	ADC12_A clock divider 000 /1 001 /2 010 /3 011 /4 100 /5 101 /6 110 /7 111 /8
ADC12SSELx	Bits 4-3	ADC12_A clock source select 00 ADC12OSC (MODOSC) 01 ACLK 10 MCLK 11 SMCLK
ADC12CONSEQx	Bits 2-1	ADC12_A conversion sequence mode select 00 Single-channel, single-conversion 01 Sequence-of-channels 10 Repeat-single-channel 11 Repeat-sequence-of-channels
ADC12BUSY	Bit 0	ADC12_A busy. This bit indicates an active sample or conversion operation. 0 No operation is active. 1 A sequence, sample, or conversion is active.

ADC12_A Control Register 2 (ADC12CTL2)

15	14	13	12	11	10	9	8
Reserved							ADC12PDIV
r-0	r-0	r-0	r-0	r-0	r-0	r-0	rw-0

7	6	5	4	3	2	1	0
ADC12TCOFF	Reserved	ADC12RES		ADC12DF	ADC12SR	ADC12 REFOUT	ADC12 REFBURST
rw-(0)	r-0	rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Modifiable only when ADC12ENC = 0

Reserved	Bits 15-9	Reserved. Read back as 0.
ADC12PDIV	Bit 8	ADC12_A predivider. This bit predivides the selected ADC12_A clock source. 0 Predivide by 1 1 Predivide by 4
ADC12TCOFF	Bit 7	ADC12_A temperature sensor off. If the bit is set, the temperature sensor turned off. This is used to save power.
Reserved	Bit 6	Reserved. Read back as 0.
ADC12RES	Bits 5-4	ADC12_A resolution. This bit defines the conversion result resolution. 00 8 bit (9 clock cycle conversion time) 01 10 bit (11 clock cycle conversion time) 10 12 bit (13 clock cycle conversion time) 11 Reserved
ADC12DF	Bit 3	ADC12_A data read-back format. Data is always stored in the binary unsigned format. 0 Binary unsigned. Theoretically the analog input voltage – V_{REF} results in 0000h, the analog input voltage + V_{REF} results in 0FFFh. 1 Signed binary (2s complement), left aligned. Theoretically the analog input voltage – V_{REF} results in 8000h, the analog input voltage + V_{REF} results in 7FF0h.
ADC12SR	Bit 2	ADC12_A sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC12SR reduces the current consumption of the reference buffer. 0 Reference buffer supports up to ~200 ksps. 1 Reference buffer supports up to ~50 ksps.
ADC12REFOUT	Bit 1	Reference output 0 Reference output off 1 Reference output on
ADC12REFBURST	Bit 0	Reference burst. ADC12REFOUT must also be set. 0 Reference buffer on continuously 1 Reference buffer on only during sample-and-conversion

ADC12_A Conversion Memory Register (ADC12MEMx)

15	14	13	12	11	10	9	8
0	0	0	0	Conversion Results			
r0	r0	r0	r0	rw	rw	rw	rw

7	6	5	4	3	2	1	0
Conversion Results							
rw	rw	rw	rw	rw	rw	rw	rw

Conversion Results	Bits 15-0	The 12-bit conversion results are right justified. Bit 11 is the MSB. Bits 15–12 are 0 in 12-bit mode, bits 15–10 are 0 in 10-bit mode, and bits 15–8 are 0 in 8-bit mode. Writing to the conversion memory registers corrupts the results. This data format is used if ADC12DF = 0.

ADC12_A Conversion Memory Register (ADC12MEMx), 2s-Complement Format

15	14	13	12	11	10	9	8
Conversion Results							
rw	rw	rw	rw	rw	rw	rw	rw

7	6	5	4	3	2	1	0
Conversion Results				**0**	**0**	**0**	**0**
rw	rw	rw	rw	r0	r0	r0	r0

Conversion Results	Bits 15-0	The 12-bit conversion results are left justified, 2s-complement format. Bit 15 is the MSB. Bits 3–0 are 0 in 12-bit mode, bits 5–0 are 0 in 10-bit mode, and bits 7–0 are 0 in 8-bit mode. This data format is used if ADC12DF = 1. The data is stored in the right-justified format and is converted to the left-justified 2s-complement format during read back.

ADC12_A Conversion Memory Control Register (ADC12MCTLx)

7	6	5	4	3	2	1	0
ADC12EOS	ADC12SREFx			ADC12INCHx			
rw	rw	rw	rw	rw	rw	rw	rw

Modifiable only when ADC12ENC = 0

ADC12EOS	Bit 7	End of sequence. Indicates the last conversion in a sequence.	
		0	Not end of sequence
		1	End of sequence
ADC12SREFx	Bits 6-4	Select reference	
		000	V_{R+} = AV_{CC} and V_{R-} = AV_{SS}
		001	V_{R+} = V_{REF+} and V_{R-} = AV_{SS}
		010	V_{R+} = Ve_{REF+} and V_{R-} = AV_{SS}
		011	V_{R+} = Ve_{REF+} and V_{R-} = AV_{SS}
		100	V_{R+} = AV_{CC} and V_{R-} = V_{REF-}/ Ve_{REF-}
		101	V_{R+} = V_{REF+} and V_{R-} = V_{REF-}/ Ve_{REF-}
		110	V_{R+} = Ve_{REF+} and V_{R-} = V_{REF-}/ Ve_{REF-}
		111	V_{R+} = Ve_{REF+} and V_{R-} = V_{REF-}/ Ve_{REF-}
ADC12INCHx	Bits 3-0	Input channel select	
		0000	A0
		0001	A1
		0010	A2
		0011	A3
		0100	A4
		0101	A5
		0110	A6
		0111	A7
		1000	Ve_{REF+}
		1001	V_{REF-}/Ve_{REF-}
		1010	Temperature diode
		1011	(AV_{CC} – AV_{SS}) / 2
		1100	A12
		1101	A13
		1110	A14
		1111	A15

ADC12_A Interrupt Enable Register (ADC12IE)

15	14	13	12	11	10	9	8
ADC12IE15	**ADC12IE14**	**ADC12IE13**	**ADC12IE12**	**ADC12IE11**	**ADC12IE10**	**ADC12IFG9**	**ADC12IE8**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC12IE7	**ADC12IE6**	**ADC12IE5**	**ADC12IE4**	**ADC12IE3**	**ADC12IE2**	**ADC12IE1**	**ADC12IE0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC12IEx Bits 15-0 Interrupt enable. These bits enable or disable the interrupt request for the ADC12IFGx bits.

- 0 Interrupt disabled
- 1 Interrupt enabled

ADC12_A Interrupt Flag Register (ADC12IFG)

15	14	13	12	11	10	9	8
ADC12IFG15	**ADC12IFG14**	**ADC12IFG13**	**ADC12IFG12**	**ADC12IFG11**	**ADC12IFG10**	**ADC12IFG9**	**ADC12IFG8**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC12IFG7	**ADC12IFG6**	**ADC12IFG5**	**ADC12IFG4**	**ADC12IFG3**	**ADC12IFG2**	**ADC12IFG1**	**ADC12IFG0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC12IFGx Bits 15-0 ADC12MEMx interrupt flag. These bits are set when corresponding ADC12MEMx is loaded with a conversion result. The ADC12IFGx bits are reset if the corresponding ADC12MEMx is accessed, or may be reset with software.

- 0 No interrupt pending
- 1 Interrupt pending

ADC12_A Interrupt Vector Register (ADC12IV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**ADC12IVx**					**0**
r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)	r0

ADC12IVx Bits 15-0 ADC12_A interrupt vector value

ADC12IV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	ADC12MEMx overflow	–	Highest
004h	Conversion time overflow	–	
006h	ADC12MEM0 interrupt flag	ADC12IFG0	
008h	ADC12MEM1 interrupt flag	ADC12IFG1	
00Ah	ADC12MEM2 interrupt flag	ADC12IFG2	
00Ch	ADC12MEM3 interrupt flag	ADC12IFG3	
00Eh	ADC12MEM4 interrupt flag	ADC12IFG4	
010h	ADC12MEM5 interrupt flag	ADC12IFG5	
012h	ADC12MEM6 interrupt flag	ADC12IFG6	
014h	ADC12MEM7 interrupt flag	ADC12IFG7	
016h	ADC12MEM8 interrupt flag	ADC12IFG8	
018h	ADC12MEM9 interrupt flag	ADC12IFG9	
01Ah	ADC12MEM10 interrupt flag	ADC12IFG10	
01Ch	ADC12MEM11 interrupt flag	ADC12IFG11	
01Eh	ADC12MEM12 interrupt flag	ADC12IFG12	
020h	ADC12MEM13 interrupt flag	ADC12IFG13	
022h	ADC12MEM14 interrupt flag	ADC12IFG14	
024h	ADC12MEM15 interrupt flag	ADC12IFG15	Lowest

Comp_B

Comp_B is an analog voltage comparator. This chapter describes the Comp_B. Comp_B covers general comparator functionality for up to 16 channels.

Topic **Page**

18.1 Comp_B Introduction **458**
18.2 Comp_B Operation **459**
18.3 Comp_B Registers **464**

18.1 Comp_B Introduction

The Comp_B module supports precision slope analog-to-digital conversions, supply voltage supervision, and monitoring of external analog signals.

Features of Comp_B include:

- Inverting and noninverting terminal input multiplexer
- Software-selectable RC filter for the comparator output
- Output provided to Timer_A capture input
- Software control of the port input buffer
- Interrupt capability
- Selectable reference voltage generator, voltage hysteresis generator
- Reference voltage input from shared reference
- Ultra-low-power comparator mode
- Interrupt driven measurement system – low-power operation support

The Comp_B block diagram is shown in Figure 18-1.

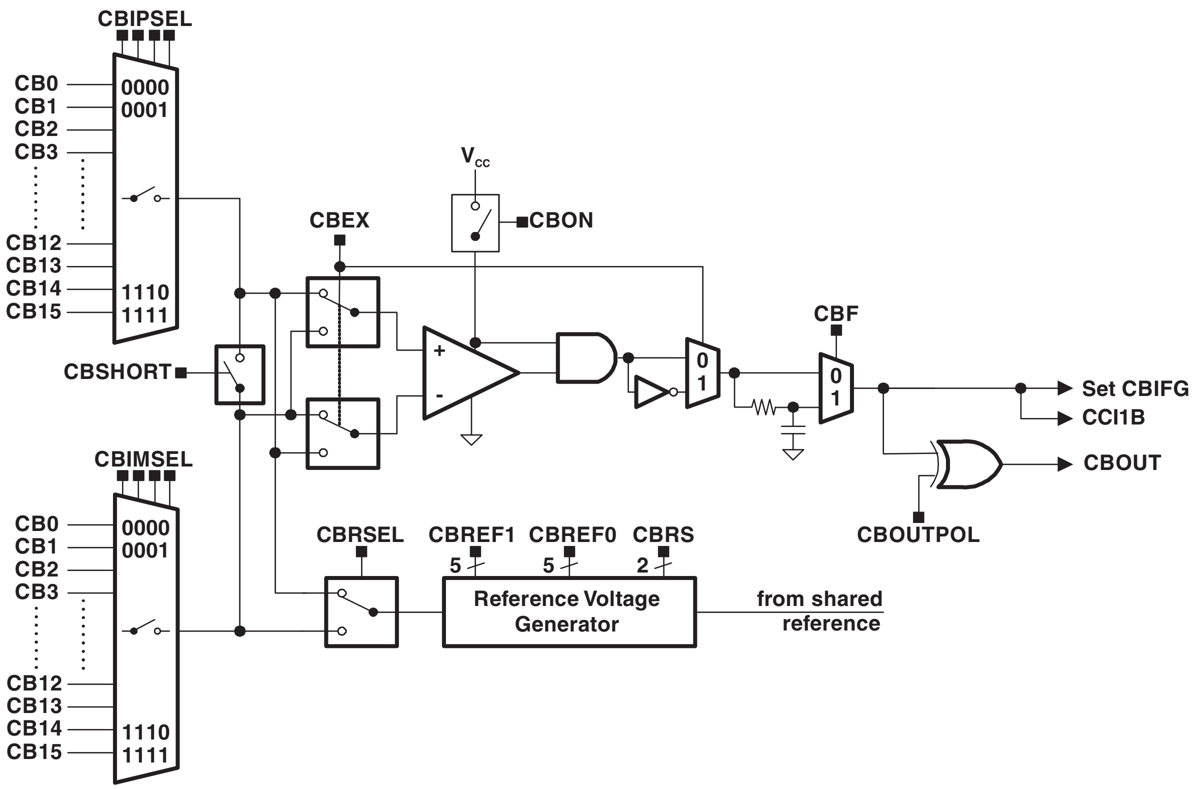


Figure 18-1. Comp_B Block Diagram

18.2 Comp_B Operation

The Comp_B module is configured by user software. The setup and operation of Comp_B is discussed in the following sections.

18.2.1 Comparator

The comparator compares the analog voltages at the + and – input terminals. If the + terminal is more positive than the – terminal, the comparator output CBOUT is high. The comparator can be switched on or off using control bit CBON. The comparator should be switched off when not in use to reduce current consumption. When the comparator is switched off, CBOUT is always low. The bias current of the comparator is programmable.

18.2.2 Analog Input Switches

The analog input switches connect or disconnect the two comparator input terminals to associated port pins using the CBIPSELx and CBIMSELx bits. The comparator terminal inputs can be controlled individually. The CBIPSELx/CBIMSELx bits allow:

- Application of an external signal to the + and – terminals of the comparator
- Routing of an internal reference voltage to an associated output port pin
- Application of an external current source (e.g., resistor) to the + or – terminal of the comparator
- The mapping of both terminals of the internal multiplexer to the outside

Internally, the input switch is constructed as a T-switch to suppress distortion in the signal path.

NOTE: Comparator Input Connection

When the comparator is on, the input terminals should be connected to a signal, power, or ground. Otherwise, floating levels may cause unexpected interrupts and increased current consumption.

The CBEX bit controls the input multiplexer, permuting the input signals of the comparator's + and – terminals. Additionally, when the comparator terminals are permuted, the output signal from the comparator is inverted too. This allows the user to determine or compensate for the comparator input offset voltage.

18.2.3 Port Logic

The Px.y pins associated with a comparator channel are enabled by the CBIPSELx or CBIMSELx bits to disable its digital components while used as comparator input. Only one of the comparator input pins is selected as input to the comparator by the input multiplexer at a time.

18.2.4 Input Short Switch

The CBSHORT bit shorts the Comp_B inputs. This can be used to build a simple sample-and-hold for the comparator as shown in Figure 18-2.

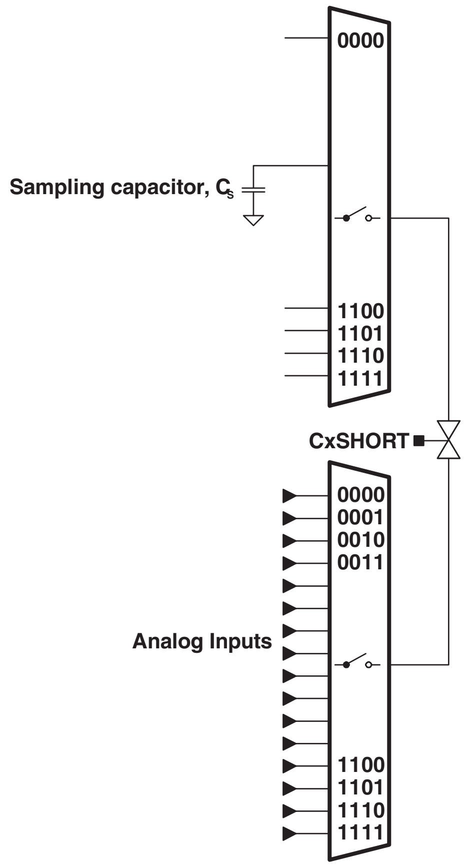


Figure 18-2. Comp_B Sample-And-Hold

The required sampling time is proportional to the size of the sampling capacitor (C_S), the resistance of the input switches in series with the short switch (R_i), and the resistance of the external source (R_S). The total internal resistance (R_I) is typically in the range of 1 kΩ. The sampling capacitor C_S should be greater than 100 pF. The time constant, Tau, to charge the sampling capacitor C_S can be calculated with the following equation:

$$Tau = (R_I + R_S) \times C_S$$

Depending on the required accuracy, 3 to 10 Tau should be used as a sampling time. With 3 Tau the sampling capacitor is charged to approximately 95% of the input signals voltage level, with 5 Tau it is charged to more than 99%, and with 10 Tau the sampled voltage is sufficient for 12-bit accuracy.

18.2.5 Output Filter

The output of the comparator can be used with or without internal filtering. When control bit CBF is set, the output is filtered with an on-chip RC filter. The delay of the filter can be adjusted in four different steps.

All comparator outputs are oscillating if the voltage difference across the input terminals is small. Internal and external parasitic effects and cross coupling on and between signal lines, power supply lines, and other parts of the system are responsible for this behavior as shown in Figure 18-3. The comparator output oscillation reduces the accuracy and resolution of the comparison result. Selecting the output filter can reduce errors associated with comparator oscillation.

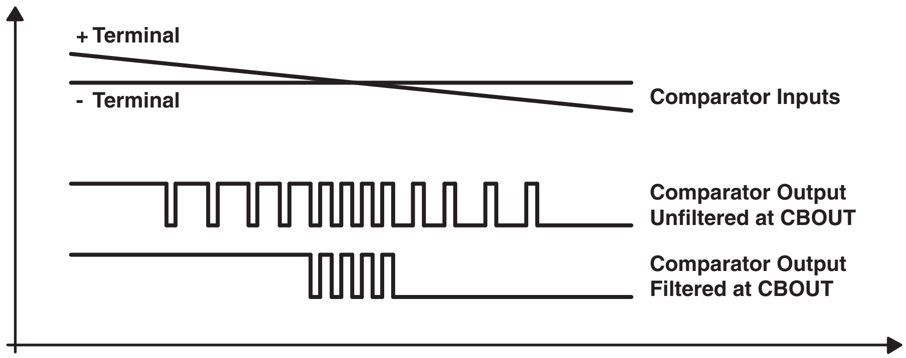


Figure 18-3. RC-Filter Response at the Output of the Comparator

18.2.6 Reference Voltage Generator

The Comp_B reference block diagram is shown in Figure 18-4.

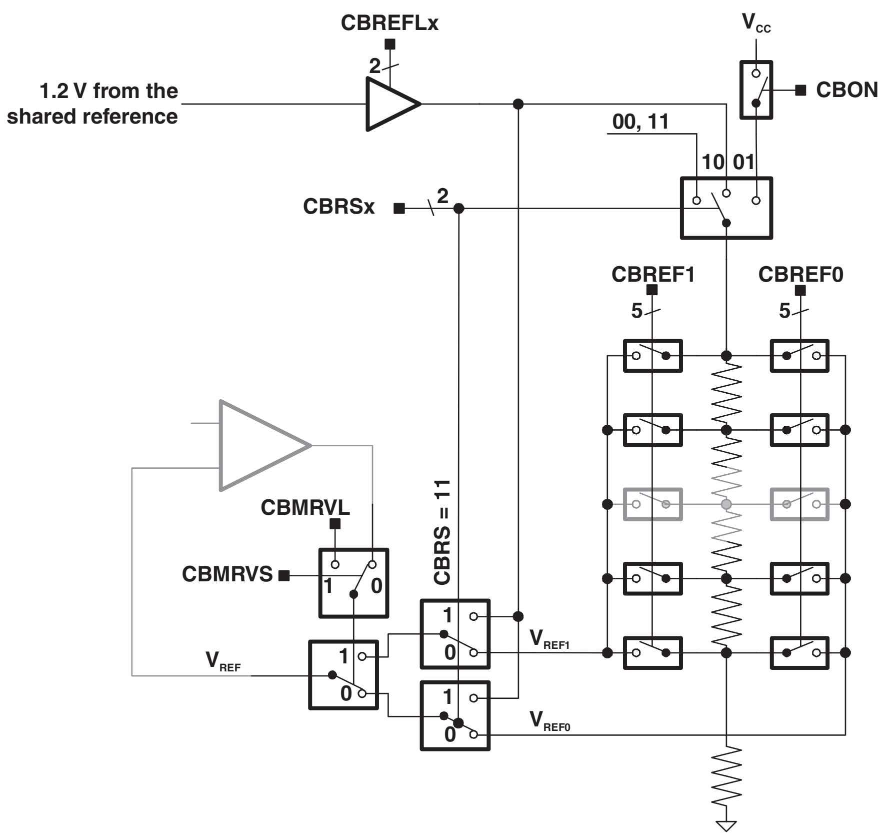


Figure 18-4. Reference Generator Block Diagram

The voltage reference generator is used to generate VREF, which can be applied to either comparator input terminal. The CBREF1x (VREF1) and CBREF0x (VREF0) bits control the output of the voltage generator. The CBRSEL bit selects the comparator terminal to which VREF is applied. If external signals are applied to both comparator input terminals, the internal reference generator should be turned off to reduce current consumption. The voltage reference generator can generate a fraction of the device's V_{CC} or of the voltage reference of the integrated precision voltage reference source. Vref1 is used while CBOUT is 1 and Vref0 is used while CBOUT is 0. This allows the generation of a hysteresis without using external components.

18.2.7 Comp_B, Port Disable Register CBPD

The comparator input and output functions are multiplexed with the associated I/O port pins, which are digital CMOS gates. When analog signals are applied to digital CMOS gates, parasitic current can flow from V_{CC} to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the port pin buffer eliminates the parasitic current flow and therefore reduces overall current consumption.

The CBPDx bits, when set, disable the corresponding Px.y input buffer as shown in Figure 18-5. When current consumption is critical, any Px.y pin connected to analog signals should be disabled with their associated CBPDx bits.

Selecting an input pin to the comparator multiplexer with the CBIPSEL or CBIMSEL bits automatically disables the input buffer for that pin, regardless of the state of the associated CBPDx bit.

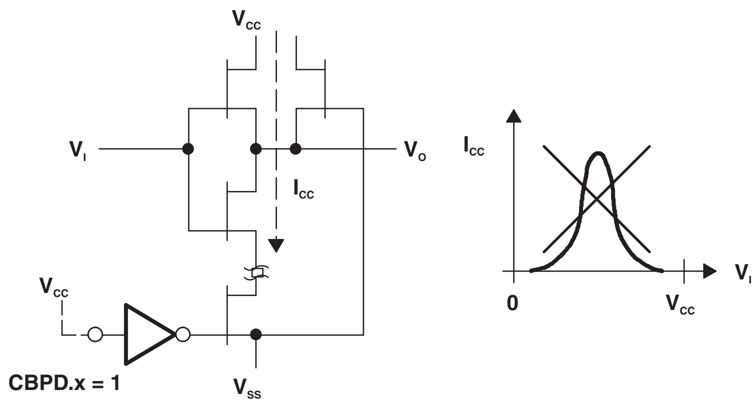


Figure 18-5. Transfer Characteristic and Power Dissipation in a CMOS Inverter/Buffer

18.2.8 Comp_B Interrupts

One interrupt flag and one interrupt vector is associated with the Comp_B.

The interrupt flag CBIFG is set on either the rising or falling edge of the comparator output, selected by the CBIES bit. If both the CBIE and the GIE bits are set, then the CBIFG interrupt flag generates an interrupt request.

18.2.9 Comp_B Used to Measure Resistive Elements

The Comp_B can be optimized to precisely measure resistive elements using single slope analog-to-digital conversion. For example, temperature can be converted into digital data using a thermistor, by comparing the thermistor's capacitor discharge time to that of a reference resistor as shown in Figure 18-6. A reference resister Rref is compared to Rmeas.

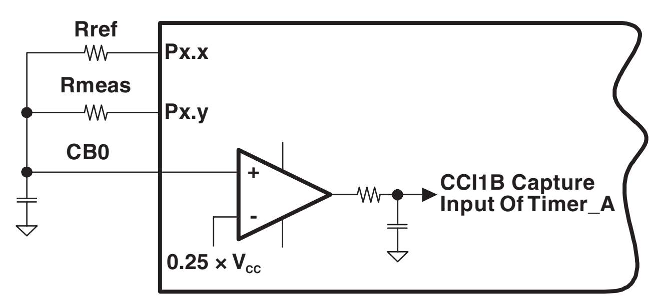


Figure 18-6. Temperature Measurement System

The resources used to calculate the temperature sensed by Rmeas are:

- Two digital I/O pins charge and discharge the capacitor.
- I/O is set to output high (V_{CC}) to charge capacitor, reset to discharge.
- I/O is switched to high-impedance input with CBPDx set when not in use.
- One output charges and discharges the capacitor via Rref.
- One output discharges capacitor via Rmeas.
- The + terminal is connected to the positive terminal of the capacitor.
- The – terminal is connected to a reference level, for example $0.25 \times V_{CC}$.
- The output filter should be used to minimize switching noise.
- CBOUT is used to gate Timer_A CCI1B, capturing capacitor discharge time.

More than one resistive element can be measured. Additional elements are connected to CB0 with available I/O pins and switched to high impedance when not being measured.

The thermistor measurement is based on a ratiometric conversion principle. The ratio of two capacitor discharge times is calculated as shown in Figure 18-7.

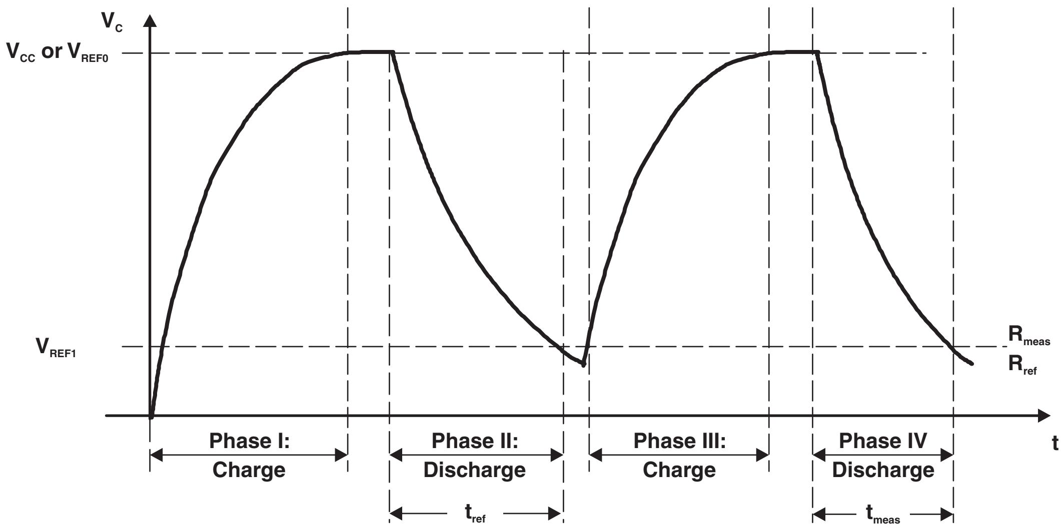


Figure 18-7. Timing for Temperature Measurement Systems

The V_{CC} voltage and the capacitor value should remain constant during the conversion, but are not critical since they cancel in the ratio:

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times \ln\frac{V_{ref1}}{V_{CC}}}{-R_{ref} \times C \times \ln\frac{V_{ref1}}{V_{CC}}}$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}$$

18.3 Comp_B Registers

The Comp_B registers are listed in Table 18-1. The base address of the Comp_B module can be found in the device-specific data sheet.

Table 18-1. Comp_B Registers

Register	Short Form	Register Type	Address Offset	Initial State
Comp_B control register 0	CBCTL0	Read/write	0x0000	Reset with PUC
Comp_B control register 1	CBCTL1	Read/write	0x0002	Reset with PUC
Comp_B control register 2	CBCTL2	Read/write	0x0004	Reset with PUC
Comp_B control register 3	CBCTL3	Read/write	0x0006	Reset with POR
Comp_B interrupt register	CBINT	Read/write	0x000C	Reset with PUC
Comp_B interrupt vector word	CBIV	Read	0x000E	Reset with PUC

Comp_B Control Register 0 (CBCTL0)

15	14	13	12	11	10	9	8
CBIMEN	**Reserved**			**CBIMSEL**			
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
CBIPEN	**Reserved**			**CBIPSEL**			
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

CBIMEN	Bit 15	Channel input enable for the V– terminal of the comparator. 0 Selected analog input channel for V– terminal is disabled. 1 Selected analog input channel for V– terminal is enabled.
Reserved	Bits 14-12	Reserved
CBIMSEL	Bits 11-8	Channel input selected for the V– terminal of the comparator if CBIMEN is set to 1.
CBIPEN	Bit 7	Channel input enable for the V+ terminal of the comparator. 0 Selected analog input channel for V+ terminal is disabled. 1 Selected analog input channel for V+ terminal is enabled.
Reserved	Bits 6-4	Reserved
CBIPSEL	Bits 3-0	Channel input selected for the V+ terminal of the comparator if CBIPEN is set to 1.

Comp_B, Control Register 1 (CBCTL1)

15	14	13	12	11	10	9	8
Reserved			**CBMRVS**	**CBMRVL**	**CBON**	**CBPWRMD**	
r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
CBFDLY		**CBEX**	**CBSHORT**	**CBIES**	**CBF**	**CBOUTPOL**	**CBOUT**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

Reserved	Bits 15-13	Reserved
CBMRVS	Bit 12	This bit defines if the comparator output selects between VREF0 or VREF1 if CBRS = 00, 01, or 10.
		0 Comparator output state selects between VREF0 or VREF1.
		1 CBMRVL selects between VREF0 or VREF1.
CBMRVL	Bit 11	This bit is valid of CBMRVS is set to 1.
		0 VREF0 is selected if CBRS = 00, 01, or 10.
		1 VREF1 is selected if CBRS = 00, 01, or 10.
CBON	Bit 10	On. This bit turns the comparator on. When the comparator is turned off the Comp_B consumes no power.
		0 Off
		1 On
CBPWRMD	Bits 9-8	Power mode. Not all modes are supported in all products. See devices specific data sheet for details.
		00 High-speed mode (optional)
		01 Normal mode (optional)
		10 Ultra-low-power mode (optional)
		11 Reserved
CBFDLY	Bits 7-6	Filter delay. The filter delay can be selected in 4 steps. See the device-specific data sheet for details.
		00 Typical filter delay of 450 ns
		01 Typical filter delay of 900 ns
		10 Typical filter delay of 1800 ns
		11 Typical filter delay of 3600 ns
CBEX	Bit 5	Exchange. This bit permutes the comparator 0 inputs and inverts the comparator 0 output.
CBSHORT	Bit 4	Input short. This bit shorts the + and – input terminals.
		0 Inputs not shorted
		1 Inputs shorted
CBIES	Bit 3	Interrupt edge select for CBIIFG and CBIFG
		0 Rising edge for CBIFG, falling edge for CBIIFG
		1 Falling edge for CBIFG, rising edge for CBIIFG
CBF	Bit 2	Output filter
		0 Comp_B output is not filtered
		1 Comp_B output is filtered
CBOUTPOL	Bit 1	Output polarity. This bit defines the CBOUT polarity.
		0 Noninverted
		1 Inverted
CBOUT	Bit 0	Output value. This bit reflects the value of the Comp_B output. Writing this bit has no effect on the comparator output.

Comp_B, Control Register 2 (CBCTL2)

15	14	13	12	11	10	9	8
CBREFACC	**CBREFL**		**CBREF1**				
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
CBRS		**CBRSEL**	**CBREF0**				
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

CBREFACC Bit 15 Reference accuracy. A reference voltage is requested only if CBREFL > 0.
- 0 Static mode
- 1 Clocked (low-power, low-accuracy) mode

CBREFL Bits 14-13 Reference voltage level
- 00 Reference voltage is disabled. No reference voltage is requested.
- 01 1.5 V
- 10 2.0 V
- 11 2.5 V

CBREF1 Bits 12-8 Reference resistor tap 1. This register defines the tap of the resistor string while CBOUT = 1.

CBRS Bits 7-6 Reference source. This bit define if the reference voltage is derived from V_{CC} or from the precise shared reference.
- 00 No current is drawn by the reference curcuitry.
- 01 V_{CC} applied to the resistor ladder
- 10 Shared reference voltage applied to the resistor ladder.
- 11 Shared reference voltage supplied to V_{CREF}. Resistor ladder is off.

CBRSEL Bit 5 Reference select. This bit selects which terminal the V_{CCREF} is applied to.
When CBEX = 0:
- 0 V_{REF} is applied to the + terminal
- 1 V_{REF} is applied to the – terminal

When CBEX = 1:
- 0 V_{REF} is applied to the – terminal
- 1 V_{REF} is applied to the + terminal

CBREF0 Bits 4-0 Reference resistor tap 0. This register defines the tap of the resistor string while CBOUT = 0.

Comp_B, Control Register 3 (CBCTL3)

15	14	13	12	11	10	9	8
CBPD15	**CBPD14**	**CBPD13**	**CBPD12**	**CBPD11**	**CBPD10**	**CBPD9**	**CBPD8**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
CBPD7	**CBPD6**	**CBPD5**	**CBPD4**	**CBPD3**	**CBPD2**	**CBPD1**	**CBPD0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

CBPDx Bit 15-0 Port disable. These bits individually disable the input buffer for the pins of the port associated with Comp_B. The bit CBPDx disabled the port of the comparator channel x.
- 0 The input buffer is enabled.
- 1 The input buffer is disabled.

Comp_B, Interrupt Control Register (CBINT)

15	14	13	12	11	10	9	8
Reserved						**CBIIE**	**CBIE**
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved						**CBIIFG**	**CBIFG**
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Reserved	Bits 15-10	Reserved. Always read back 0.
CBIIE	Bit 9	Comp_B output interrupt enable inverted polarity 0 Interrupt is disabled 1 Interrupt is enabled
CBIE	Bit 8	Comp_B output interrupt enable 0 Interrupt is disabled 1 Interrupt is enabled
Reserved	Bits 7-2	Reserved. Always read back 0.
CBIIFG	Bit 1	Comp_B output inverted interrupt flag. The bit CBIES defines the transition of the output setting this bit. 0 No interrupt pending 1 Output interrupt pending
CBIFG	Bit 0	Comp_B output interrupt flag. The bit CBIES defines the transition of the output setting this bit. 0 No interrupt pending 1 Output interrupt pending

Comp_B, Interrupt Vector Word Register (CBIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**CBIV**		**0**
r0	r0	r0	r0	r0	r-(0)	r-(0)	r0

CBIV Bits 15-0 Comp_B interrupt vector word register. The interrupt vector register reflects only interrupt flags whose interrupt enable bit are set. Reading the CBIV register clears the pending interrupt flag with the highest priority.

CBIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	–	–
02h	CBOUT interrupt	CBIFG	Highest
04h	CBOUT interrupt inverted polarity	CBIIFG	Lowest

Chapter 19

SLAU208F–June 2008–Revised March 2010

Universal Serial Communication Interface – UART Mode

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the asynchronous UART mode.

Topic **Page**

19.1 Universal Serial Communication Interface (USCI) Overview 470
19.2 USCI Introduction – UART Mode 471
19.3 USCI Operation – UART Mode 473
19.4 USCI Registers – UART Mode 489

19.1 Universal Serial Communication Interface (USCI) Overview

The USCI modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI_A is different from USCI_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. See the device-specific data sheet to determine which USCI modules, if any, are implemented on which devices.

USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud-rate detection for LIN communications
- SPI mode

USCI_Bx modules support:

- I^2C mode
- SPI mode

19.2 USCI Introduction – UART Mode

In asynchronous mode, the USCI_Ax modules connect the device to an external system via two external pins, UCAxRXD and UCAxTXD. UART mode is selected when the UCSYNC bit is cleared.

UART mode features include:

- 7- or 8-bit data with odd, even, or non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first or MSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto wake up from LPMx modes
- Programmable baud rate with modulation for fractional baud-rate support
- Status flags for error detection and suppression
- Status flags for address detection
- Independent interrupt capability for receive and transmit

Figure 19-1 shows the USCI_Ax when configured for UART mode.

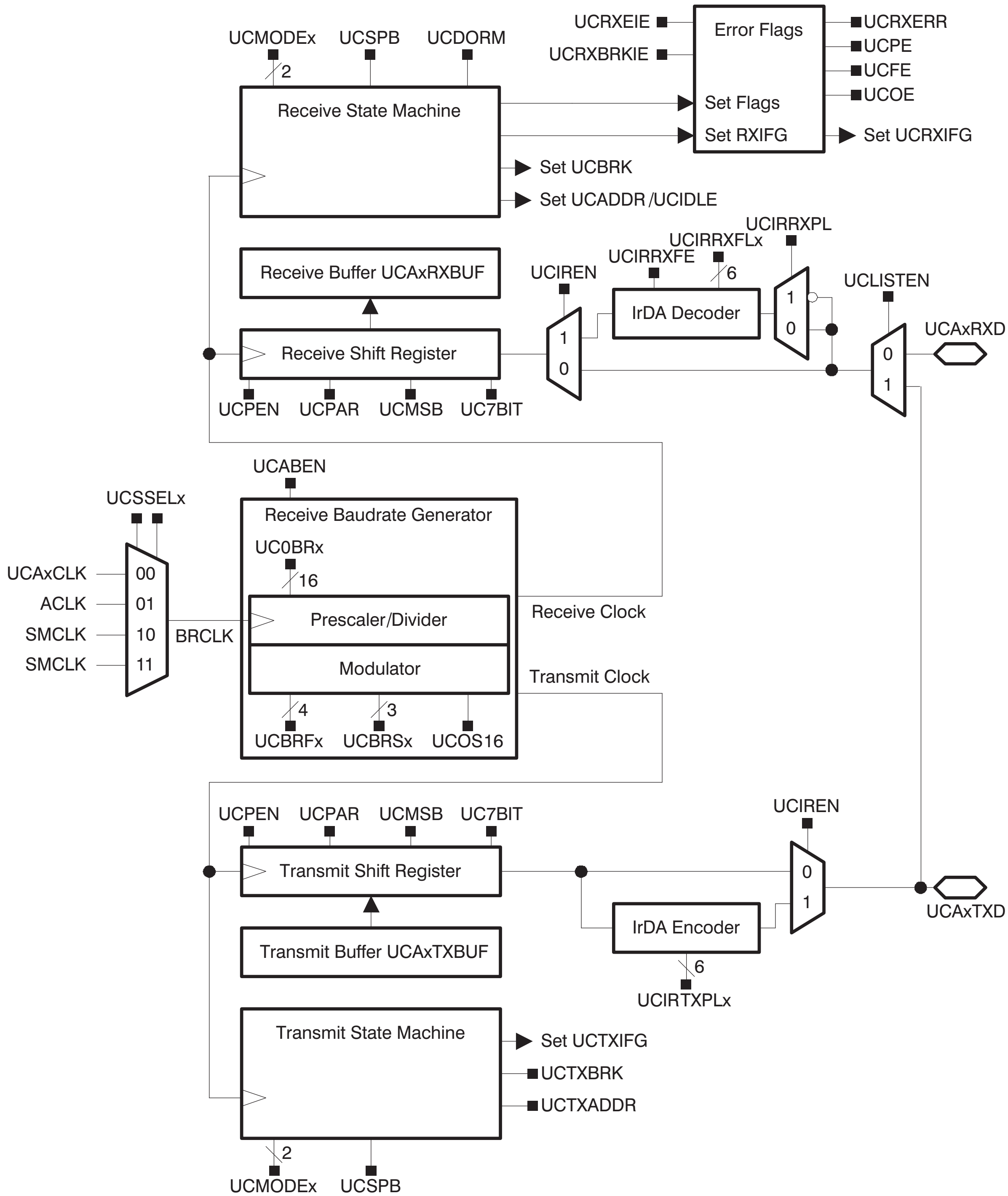


Figure 19-1. USCI_Ax Block Diagram – UART Mode (UCSYNC = 0)

19.3 USCI Operation – UART Mode

In UART mode, the USCI transmits and receives characters at a bit rate asynchronous to another device. Timing for each character is based on the selected baud rate of the USCI. The transmit and receive functions use the same baud-rate frequency.

19.3.1 USCI Initialization and Reset

The USCI is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the USCI in a reset condition. When set, the UCSWRST bit resets the UCRXIE, UCTXIE, UCRXIFG, UCRXERR, UCBRK, UCPE, UCOE, UCFE, UCSTOE, and UCBTOE bits, and sets the UCTXIFG bit. Clearing UCSWRST releases the USCI for operation.

NOTE: Initializing or reconfiguring the USCI module

The recommended USCI initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST,&UCAxCTL1`).
2. Initialize all USCI registers with UCSWRST = 1 (including UCAxCTL1).
3. Configure ports.
4. Clear UCSWRST via software (`BIC.B #UCSWRST,&UCAxCTL1`).
5. Enable interrupts (optional) via UCRXIE and/or UCTXIE.

19.3.2 Character Format

The UART character format (see Figure 19-2) consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit (address-bit mode), and one or two stop bits. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first. LSB first is typically required for UART communication.

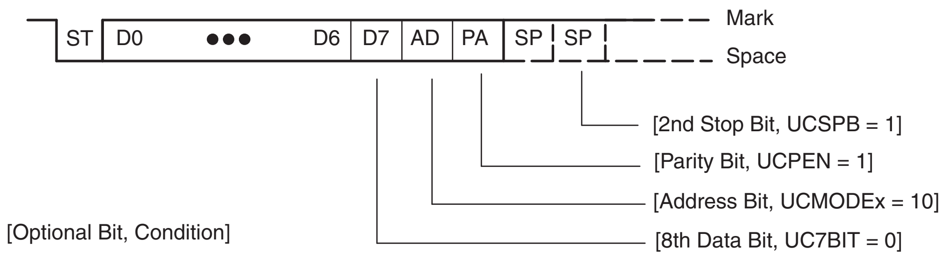


Figure 19-2. Character Format

19.3.3 Asynchronous Communication Format

When two devices communicate asynchronously, no multiprocessor format is required for the protocol. When three or more devices communicate, the USCI supports the idle-line and address-bit multiprocessor communication formats.

19.3.3.1 Idle-Line Multiprocessor Format

When UCMODEx = 01, the idle-line multiprocessor format is selected. Blocks of data are separated by an idle time on the transmit or receive lines (see Figure 19-3). An idle receive line is detected when ten or more continuous ones (marks) are received after the one or two stop bits of a character. The baud-rate generator is switched off after reception of an idle line until the next start edge is detected. When an idle line is detected, the UCIDLE bit is set.

The first character received after an idle period is an address character. The UCIDLE bit is used as an address tag for each block of characters. In idle-line multiprocessor format, this bit is set when a received character is an address.

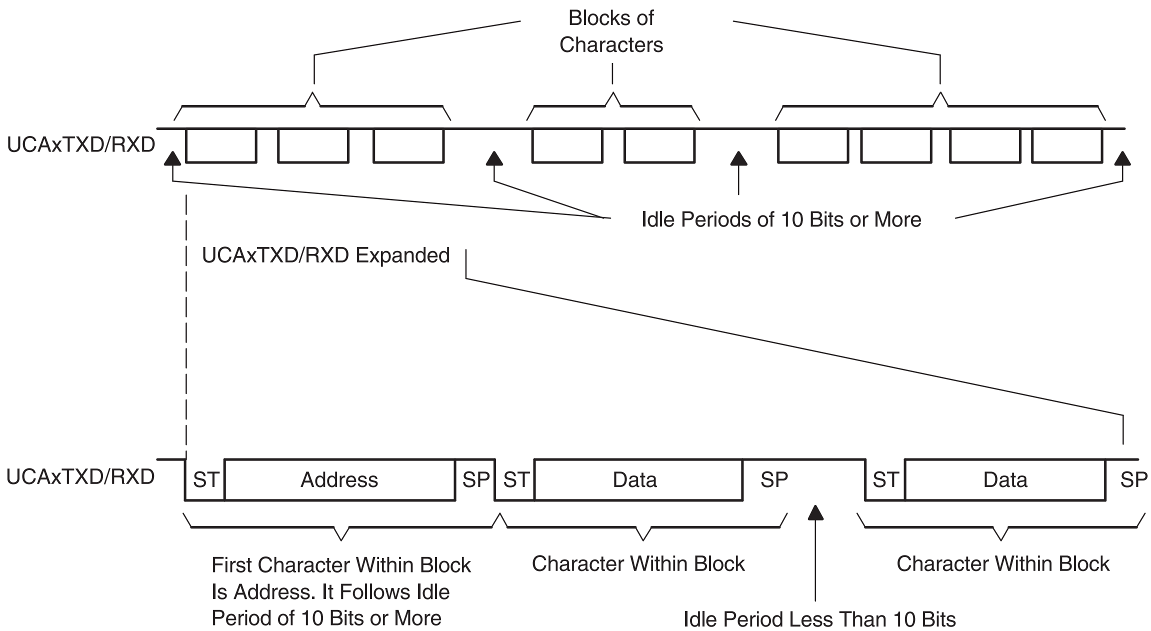


Figure 19-3. Idle-Line Format

The UCDORM bit is used to control data reception in the idle-line multiprocessor format. When UCDORM = 1, all non-address characters are assembled but not transferred into the UCAxRXBUF, and interrupts are not generated. When an address character is received, the character is transferred into UCAxRXBUF, UCRXIFG is set, and any applicable error flag is set when UCRXEIE = 1. When UCRXEIE = 0 and an address character is received but has a framing error or parity error, the character is not transferred into UCAxRXBUF and UCRXIFG is not set.

If an address is received, user software can validate the address and must reset UCDORM to continue receiving data. If UCDORM remains set, only address characters are received. When UCDORM is cleared during the reception of a character, the receive interrupt flag is set after the reception completed. The UCDORM bit is not modified by the USCI hardware automatically.

For address transmission in idle-line multiprocessor format, a precise idle period can be generated by the USCI to generate address character identifiers on UCAxTXD. The double-buffered UCTXADDR flag indicates if the next character loaded into UCAxTXBUF is preceded by an idle line of 11 bits. UCTXADDR is automatically cleared when the start bit is generated.

Transmitting an Idle Frame

The following procedure sends out an idle frame to indicate an address character followed by associated data:

1. Set UCTXADDR, then write the address character to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCTXIFG = 1).

   This generates an idle period of exactly 11 bits followed by the address character. UCTXADDR is reset automatically when the address character is transferred from UCAxTXBUF into the shift register.
2. Write desired data characters to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCTXIFG = 1).

   The data written to UCAxTXBUF is transferred to the shift register and transmitted as soon as the shift register is ready for new data.

   The idle-line time must not be exceeded between address and data transmission or between data transmissions. Otherwise, the transmitted data is misinterpreted as an address.

19.3.3.2 Address-Bit Multiprocessor Format

When UCMODEx = 10, the address-bit multiprocessor format is selected. Each processed character contains an extra bit used as an address indicator (see Figure 19-4). The first character in a block of characters carries a set address bit that indicates that the character is an address. The USCI UCADDR bit is set when a received character has its address bit set and is transferred to UCAxRXBUF.

The UCDORM bit is used to control data reception in the address-bit multiprocessor format. When UCDORM is set, data characters with address bit = 0 are assembled by the receiver but are not transferred to UCAxRXBUF and no interrupts are generated. When a character containing a set address bit is received, the character is transferred into UCAxRXBUF, UCRXIFG is set, and any applicable error flag is set when UCRXEIE = 1. When UCRXEIE = 0 and a character containing a set address bit is received but has a framing error or parity error, the character is not transferred into UCAxRXBUF and UCRXIFG is not set.

If an address is received, user software can validate the address and must reset UCDORM to continue receiving data. If UCDORM remains set, only address characters with address bit = 1 are received. The UCDORM bit is not modified by the USCI hardware automatically.

When UCDORM = 0, all received characters set the receive interrupt flag UCRXIFG. If UCDORM is cleared during the reception of a character, the receive interrupt flag is set after the reception is completed.

For address transmission in address-bit multiprocessor mode, the address bit of a character is controlled by the UCTXADDR bit. The value of the UCTXADDR bit is loaded into the address bit of the character transferred from UCAxTXBUF to the transmit shift register. UCTXADDR is automatically cleared when the start bit is generated.

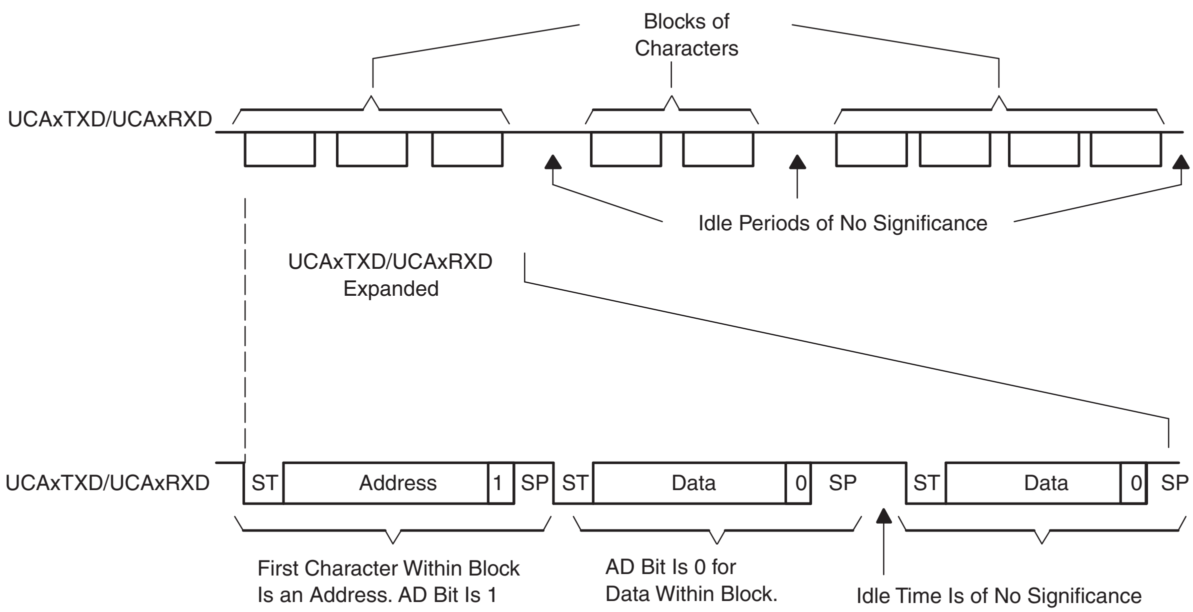


Figure 19-4. Address-Bit Multiprocessor Format

Break Reception and Generation

When UCMODEx = 00, 01, or 10, the receiver detects a break when all data, parity, and stop bits are low, regardless of the parity, address mode, or other character settings. When a break is detected, the UCBRK bit is set. If the break interrupt enable bit (UCBRKIE) is set, the receive interrupt flag UCRXIFG is also set. In this case, the value in UCAxRXBUF is 0h, because all data bits were zero.

To transmit a break, set the UCTXBRK bit, then write 0h to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCTXIFG = 1). This generates a break with all bits low. UCTXBRK is automatically cleared when the start bit is generated.

19.3.4 Automatic Baud-Rate Detection

When UCMODEx = 11, UART mode with automatic baud-rate detection is selected. For automatic baud-rate detection, a data frame is preceded by a synchronization sequence that consists of a break and a synch field. A break is detected when 11 or more continuous zeros (spaces) are received. If the length of the break exceeds 21 bit times the break timeout error flag UCBTOE is set. The USCI can not transmit data while receiving the break/sync field. The synch field follows the break as shown in Figure 19-5.

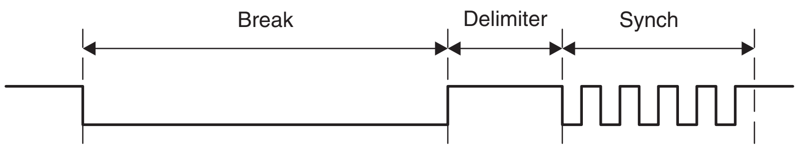


Figure 19-5. Auto Baud-Rate Detection – Break/Synch Sequence

For LIN conformance, the character format should be set to eight data bits, LSB first, no parity, and one stop bit. No address bit is available.

The synch field consists of the data 055h inside a byte field (see Figure 19-6). The synchronization is based on the time measurement between the first falling edge and the last falling edge of the pattern. The transmit baud-rate generator is used for the measurement if automatic baud-rate detection is enabled by setting UCABDEN. Otherwise, the pattern is received but not measured. The result of the measurement is transferred into the baud-rate control registers (UCAxBR0, UCAxBR1, and UCAxMCTL). If the length of the synch field exceeds the measurable time, the synch timeout error flag UCSTOE is set.

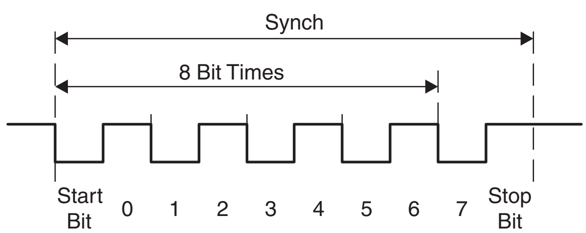


Figure 19-6. Auto Baud-Rate Detection – Synch Field

The UCDORM bit is used to control data reception in this mode. When UCDORM is set, all characters are received but not transferred into the UCAxRXBUF, and interrupts are not generated. When a break/synch field is detected, the UCBRK flag is set. The character following the break/synch field is transferred into UCAxRXBUF and the UCRXIFG interrupt flag is set. Any applicable error flag is also set. If the UCBRKIE bit is set, reception of the break/synch sets the UCRXIFG. The UCBRK bit is reset by user software or by reading the receive buffer UCAxRXBUF.

When a break/synch field is received, user software must reset UCDORM to continue receiving data. If UCDORM remains set, only the character after the next reception of a break/synch field is received. The UCDORM bit is not modified by the USCI hardware automatically.

When UCDORM = 0, all received characters set the receive interrupt flag UCRXIFG. If UCDORM is cleared during the reception of a character, the receive interrupt flag is set after the reception is complete.

The counter used to detect the baud rate is limited to 07FFFh (32767) counts. This means the minimum baud rate detectable is 488 baud in oversampling mode and 30 baud in low-frequency mode.

The automatic baud-rate detection mode can be used in a full-duplex communication system with some restrictions. The USCI can not transmit data while receiving the break/sync field and, if a 0h byte with framing error is received, any data transmitted during this time gets corrupted. The latter case can be discovered by checking the received data and the UCFE bit.

19.3.4.1 Transmitting a Break/Synch Field

The following procedure transmits a break/synch field:

1. Set UCTXBRK with UMODEx = 11.
2. Write 055h to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCTXIFG = 1).

   This generates a break field of 13 bits followed by a break delimiter and the synch character. The length of the break delimiter is controlled with the UCDELIMx bits. UCTXBRK is reset automatically when the synch character is transferred from UCAxTXBUF into the shift register.
3. Write desired data characters to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCTXIFG = 1).

   The data written to UCAxTXBUF is transferred to the shift register and transmitted as soon as the shift register is ready for new data.

19.3.5 IrDA Encoding and Decoding

When UCIREN is set, the IrDA encoder and decoder are enabled and provide hardware bit shaping for IrDA communication.

19.3.5.1 IrDA Encoding

The encoder sends a pulse for every zero bit in the transmit bit stream coming from the UART (see Figure 19-7). The pulse duration is defined by UCIRTXPLx bits specifying the number of one-half clock periods of the clock selected by UCIRTXCLK.

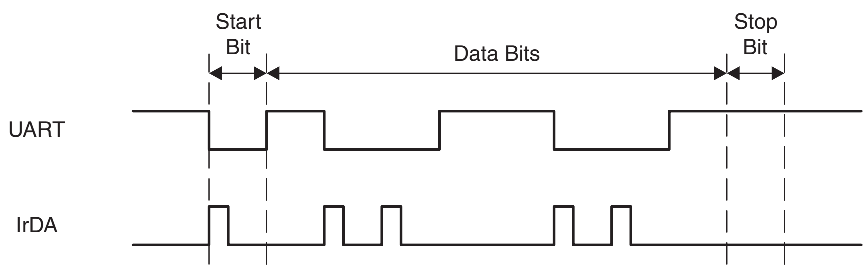


Figure 19-7. UART vs IrDA Data Format

To set the pulse time of 3/16 bit period required by the IrDA standard, the BITCLK16 clock is selected with UCIRTXCLK = 1 ,and the pulse length is set to six one-half clock cycles with UCIRTXPLx = 6 – 1 = 5.

When UCIRTXCLK = 0, the pulse length t_{PULSE} is based on BRCLK and is calculated as:

$$UCIRTXPLx = t_{PULSE} \times 2 \times f_{BRCLK} - 1$$

When UCIRTXCLK = 0 ,the prescaler UCBRx must to be set to a value greater or equal to 5.

19.3.5.2 IrDA Decoding

The decoder detects high pulses when UCIRRXPL = 0. Otherwise, it detects low pulses. In addition to the analog deglitch filter, an additional programmable digital filter stage can be enabled by setting UCIRRXFE. When UCIRRXFE is set, only pulses longer than the programmed filter length are passed. Shorter pulses are discarded. The equation to program the filter length UCIRRXFLx is:

$$UCIRRXFLx = (t_{PULSE} - t_{WAKE}) \times 2 \times f_{BRCLK} - 4$$

Where:

t_{PULSE} = Minimum receive pulse width

t_{WAKE} = Wake time from any low-power mode. Zero when the device is in active mode.

19.3.6 Automatic Error Detection

Glitch suppression prevents the USCI from being accidentally started. Any pulse on UCAxRXD shorter than the deglitch time t_t (approximately 150 ns) is ignored (see the device-specific data sheet for parameters).

When a low period on UCAxRXD exceeds t_t, a majority vote is taken for the start bit. If the majority vote fails to detect a valid start bit, the USCI halts character reception and waits for the next low period on UCAxRXD. The majority vote is also used for each bit in a character to prevent bit errors.

The USCI module automatically detects framing errors, parity errors, overrun errors, and break conditions when receiving characters. The bits UCFE, UCPE, UCOE, and UCBRK are set when their respective condition is detected. When the error flags UCFE, UCPE, or UCOE are set, UCRXERR is also set. The error conditions are described in Table 19-1.

Table 19-1. Receive Error Conditions

Error Condition	Error Flag	Description
Framing error	UCFE	A framing error occurs when a low stop bit is detected. When two stop bits are used, both stop bits are checked for framing error. When a framing error is detected, the UCFE bit is set.
Parity error	UCPE	A parity error is a mismatch between the number of 1s in a character and the value of the parity bit. When an address bit is included in the character, it is included in the parity calculation. When a parity error is detected, the UCPE bit is set.
Receive overrun	UCOE	An overrun error occurs when a character is loaded into UCAxRXBUF before the prior character has been read. When an overrun occurs, the UCOE bit is set.
Break condition	UCBRK	When not using automatic baud-rate detection, a break is detected when all data, parity, and stop bits are low. When a break condition is detected, the UCBRK bit is set. A break condition can also set the interrupt flag UCRXIFG if the break interrupt enable UCBRKIE bit is set.

When UCRXEIE = 0 and a framing error or parity error is detected, no character is received into UCAxRXBUF. When UCRXEIE = 1, characters are received into UCAxRXBUF and any applicable error bit is set.

When any of the UCFE, UCPE, UCOE, UCBRK, or UCRXERR bit is set, the bit remains set until user software resets it or UCAxRXBUF is read. UCOE must be reset by reading UCAxRXBUF. Otherwise, it does not function properly. To detect overflows reliably the following flow is recommended. After a character was received and UCAxRXIFG is set, first read UCAxSTAT to check the error flags including the overflow flag UCOE. Read UCAxRXBUF next. This clears all error flags except UCOE, if UCAxRXBUF was overwritten between the read access to UCAxSTAT and to UCAxRXBUF. Therefore, the UCOE flag should be checked after reading UCAxRXBUF to detect this condition. Note that, in this case, the UCRXERR flag is not set.

19.3.7 USCI Receive Enable

The USCI module is enabled by clearing the UCSWRST bit and the receiver is ready and in an idle state. The receive baud rate generator is in a ready state but is not clocked nor producing any clocks.

The falling edge of the start bit enables the baud rate generator and the UART state machine checks for a valid start bit. If no valid start bit is detected the UART state machine returns to its idle state and the baud rate generator is turned off again. If a valid start bit is detected, a character is received.

When the idle-line multiprocessor mode is selected with UCMODEx = 01 the UART state machine checks for an idle line after receiving a character. If a start bit is detected another character is received. Otherwise the UCIDLE flag is set after 10 ones are received and the UART state machine returns to its idle state and the baud rate generator is turned off.

19.3.7.1 Receive Data Glitch Suppression

Glitch suppression prevents the USCI from being accidentally started. Any glitch on UCAxRXD shorter than the deglitch time t_t (approximately 150 ns) is ignored by the USCI, and further action is initiated as shown in Figure 19-8 (see the device-specific data sheet for parameters).

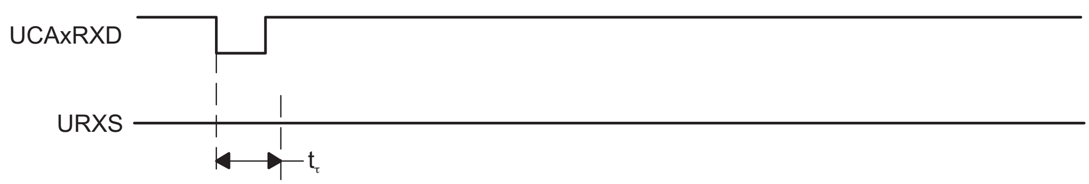


Figure 19-8. Glitch Suppression, USCI Receive Not Started

When a glitch is longer than t_t, or a valid start bit occurs on UCAxRXD, the USCI receive operation is started and a majority vote is taken (see Figure 19-9). If the majority vote fails to detect a start bit, the USCI halts character reception.

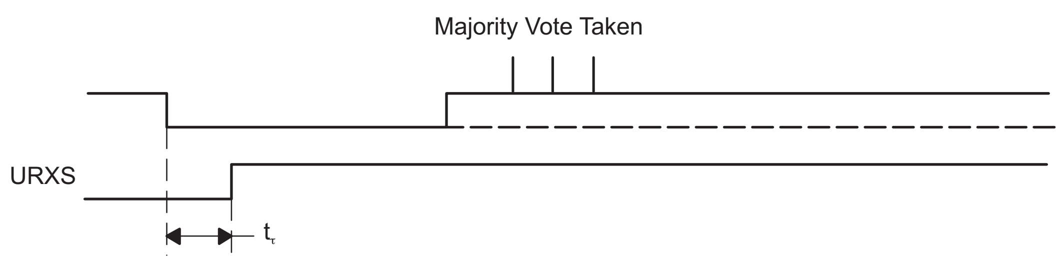


Figure 19-9. Glitch Suppression, USCI Activated

19.3.8 USCI Transmit Enable

The USCI module is enabled by clearing the UCSWRST bit and the transmitter is ready and in an idle state. The transmit baud-rate generator is ready but is not clocked nor producing any clocks.

A transmission is initiated by writing data to UCAxTXBUF. When this occurs, the baud-rate generator is enabled, and the data in UCAxTXBUF is moved to the transmit shift register on the next BITCLK after the transmit shift register is empty. UCTXIFG is set when new data can be written into UCAxTXBUF.

Transmission continues as long as new data is available in UCAxTXBUF at the end of the previous byte transmission. If new data is not in UCAxTXBUF when the previous byte has transmitted, the transmitter returns to its idle state and the baud-rate generator is turned off.

19.3.9 UART Baud-Rate Generation

The USCI baud-rate generator is capable of producing standard baud rates from nonstandard source frequencies. It provides two modes of operation selected by the UCOS16 bit. The baud-rate is generate using the BRCLK that can be sourced by the external clock UCAxCLK, or the internal clocks ACLK or SMCLK depending on the UCSSELx settings.

19.3.9.1 Low-Frequency Baud-Rate Generation

The low-frequency mode is selected when UCOS16 = 0. This mode allows generation of baud rates from low frequency clock sources (e.g., 9600 baud from a 32768-Hz crystal). By using a lower input frequency, the power consumption of the module is reduced. Using this mode with higher frequencies and higher prescaler settings causes the majority votes to be taken in an increasingly smaller window and, thus, decrease the benefit of the majority vote.

In low-frequency mode, the baud-rate generator uses one prescaler and one modulator to generate bit clock timing. This combination supports fractional divisors for baud-rate generation. In this mode, the maximum USCI baud rate is one-third the UART source clock frequency BRCLK.

Timing for each bit is shown in Figure 19-10. For each bit received, a majority vote is taken to determine the bit value. These samples occur at the N/2 – 1/2, N/2, and N/2 + 1/2 BRCLK periods, where N is the number of BRCLKs per BITCLK.

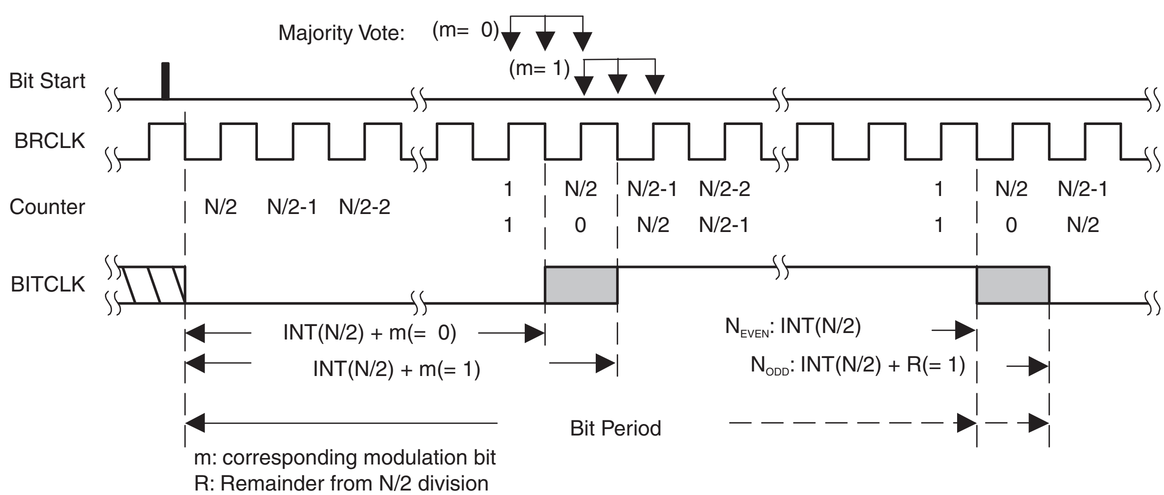


Figure 19-10. BITCLK Baud-Rate Timing With UCOS16 = 0

Modulation is based on the UCBRSx setting (see Table 19-2). A 1 in the table indicates that m = 1 and the corresponding BITCLK period is one BRCLK period longer than a BITCLK period with m = 0. The modulation wraps around after eight bits but restarts with each new start bit.

Table 19-2. BITCLK Modulation Pattern

UCBRSx	Bit 0 (Start Bit)	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0
3	0	1	0	1	0	1	0	0
4	0	1	0	1	0	1	0	1
5	0	1	1	1	0	1	0	1
6	0	1	1	1	0	1	1	1
7	0	1	1	1	1	1	1	1

19.3.9.2 Oversampling Baud-Rate Generation

The oversampling mode is selected when UCOS16 = 1. This mode supports sampling a UART bit stream with higher input clock frequencies. This results in majority votes that are always 1/16 of a bit clock period apart. This mode also easily supports IrDA pulses with a 3/16 bit time when the IrDA encoder and decoder are enabled.

This mode uses one prescaler and one modulator to generate the BITCLK16 clock that is 16 times faster than the BITCLK. An additional divider and modulator stage generates BITCLK from BITCLK16. This combination supports fractional divisions of both BITCLK16 and BITCLK for baud-rate generation. In this mode, the maximum USCI baud rate is 1/16 the UART source clock frequency BRCLK. When UCBRx is set to 0 or 1, the first prescaler and modulator stage is bypassed and BRCLK is equal to BITCLK16 – in this case, no modulation for the BITCLK16 is possible and, thus, the UCBRFx bits are ignored.

Modulation for BITCLK16 is based on the UCBRFx setting (see Table 19-3). A 1 in the table indicates that the corresponding BITCLK16 period is one BRCLK period longer than the periods m = 0. The modulation restarts with each new bit timing.

Modulation for BITCLK is based on the UCBRSx setting (see Table 19-2) as previously described.

Table 19-3. BITCLK16 Modulation Pattern

UCBRFx	No. of BITCLK16 Clocks After Last Falling BITCLK Edge															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
00h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01h	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
02h	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
03h	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
04h	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
05h	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1
06h	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
07h	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1
08h	0	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1
09h	0	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1
0Ah	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
0Bh	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1
0Ch	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
0Dh	0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
0Eh	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0Fh	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

19.3.10 Setting a Baud Rate

For a given BRCLK clock source, the baud rate used determines the required division factor N:

$N = f_{BRCLK}/\text{Baudrate}$

The division factor N is often a noninteger value, thus, at least one divider and one modulator stage is used to meet the factor as closely as possible.

If N is equal or greater than 16, the oversampling baud-rate generation mode can be chosen by setting UCOS16.

19.3.10.1 Low-Frequency Baud-Rate Mode Setting

In low-frequency mode, the integer portion of the divisor is realized by the prescaler:

$UCBRx = INT(N)$

and the fractional portion is realized by the modulator with the following nominal formula:

$UCBRSx = round[(N – INT(N)) \times 8]$

Incrementing or decrementing the UCBRSx setting by one count may give a lower maximum bit error for any given bit. To determine if this is the case, a detailed error calculation must be performed for each bit for each UCBRSx setting.

19.3.10.2 Oversampling Baud-Rate Mode Setting

In the oversampling mode, the prescaler is set to:

$UCBRx = INT(N/16)$

and the first stage modulator is set to:

$UCBRFx = round([(N/16) – INT(N/16)] \times 16)$

When greater accuracy is required, the UCBRSx modulator can also be implemented with values from 0 to 7. To find the setting that gives the lowest maximum bit error rate for any given bit, a detailed error calculation must be performed for all settings of UCBRSx from 0 to 7 with the initial UCBRFx setting, and with the UCBRFx setting incremented and decremented by one.

19.3.11 Transmit Bit Timing

The timing for each character is the sum of the individual bit timings. Using the modulation features of the baud-rate generator reduces the cumulative bit error. The individual bit error can be calculated using the following steps.

19.3.11.1 Low-Frequency Baud-Rate Mode Bit Timing

In low-frequency mode, calculate the length of bit i $T_{bit,TX}[i]$ based on the UCBRx and UCBRSx settings:

$T_{bit,TX}[i] = (1/f_{BRCLK})(UCBRx + m_{UCBRSx}[i])$

Where:

$m_{UCBRSx}[i]$ = Modulation of bit i from Table 19-2

19.3.11.2 Oversampling Baud-Rate Mode Bit Timing

In oversampling baud-rate mode, calculate the length of bit i $T_{bit,TX}[i]$ based on the baud-rate generator UCBRx, UCBRFx and UCBRSx settings:

$$T_{bit,TX}[i] = \frac{1}{f_{BRCLK}}\left((16 + m_{UCBRSx}[i]) \times UCBRx + \sum_{j=0}^{15} m_{UCBRFx}[j]\right)$$

Where:

$\sum_{j=0}^{15} m_{UCBRFx}[j]$ = Sum of ones from the corresponding row in Table 19-3

$m_{UCBRSx}[i]$ = Modulation of bit i from Table 19-2

This results in an end-of-bit time $t_{bit,TX}[i]$ equal to the sum of all previous and the current bit times:

$$T_{bit,TX}[i] = \sum_{j=0}^{i} T_{bit,TX}[j]$$

To calculate bit error, this time is compared to the ideal bit time $t_{bit,ideal,TX}[i]$:

$t_{bit,ideal,TX}[i] = (1/Baudrate)(i + 1)$

This results in an error normalized to one ideal bit time (1/baudrate):

$Error_{TX}[i] = (t_{bit,TX}[i] - t_{bit,ideal,TX}[i]) \times Baudrate \times 100\%$

19.3.12 Receive Bit Timing

Receive timing error consists of two error sources. The first is the bit-to-bit timing error similar to the transmit bit timing error. The second is the error between a start edge occurring and the start edge being accepted by the USCI module. Figure 19-11 shows the asynchronous timing errors between data on the UCAxRXD pin and the internal baud-rate clock. This results in an additional synchronization error. The synchronization error t_{SYNC} is between –0.5 BRCLKs and +0.5 RCLKs, independent of the selected baud-rate generation mode.

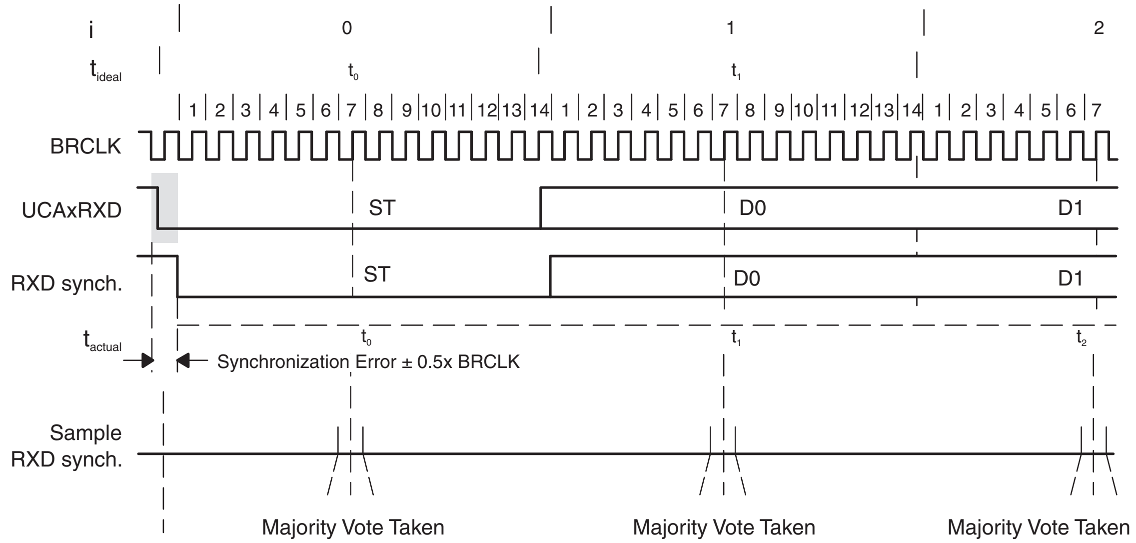


Figure 19-11. Receive Error

The ideal sampling time $t_{bit,ideal,RX}[i]$ is in the middle of a bit period:

$t_{bit,ideal,RX}[i] = (1/Baudrate)(i + 0.5)$

The real sampling time, $t_{bit,RX}[i]$, is equal to the sum of all previous bits according to the formulas shown in the transmit timing section, plus one-half BITCLK for the current bit i, plus the synchronization error t_{SYNC}.

This results in the following $t_{bit,RX}[i]$ for the low-frequency baud-rate mode:

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}}\left(INT(½UCBRx) + m_{UCBRSx}[i]\right)$$

Where:

$T_{bit,RX}[i] = (1/f_{BRCLK})(UCBRx + m_{UCBRSx}[i])$

$m_{UCBRSx}[i]$ = Modulation of bit i from Table 19-2

For the oversampling baud-rate mode, the sampling time $t_{bit,RX}[i]$ of bit i is calculated by:

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}}\left((8 + m_{UCBRSx}[i]) \times UCBRx + \sum_{j=0}^{7+m_{UCBRSx}[i]} m_{UCBRFx}[j]\right)$$

Where:

$$T_{bit,RX}[i] = \frac{1}{f_{BRCLK}}\left((16 + m_{UCBRSx}[i]) \times UCBRx + \sum_{j=0}^{15} m_{UCBRFx}[j]\right)$$

$\sum_{j=0}^{7+m_{UCBRSx}[i]} m_{UCBRFx}[j]$ = Sum of ones from columns 0 to $(7 + m_{UCBRSx}[i])$ from the corresponding row in Table 19-3.

$m_{UCBRSx}[i]$ = Modulation of bit i from Table 19-2

This results in an error normalized to one ideal bit time (1/baudrate) according to the following formula:

$$Error_{RX}[i] = (t_{bit,RX}[i] - t_{bit,ideal,RX}[i]) \times Baudrate \times 100\%$$

19.3.13 Typical Baud Rates and Errors

Standard baud-rate data for UCBRx, UCBRSx, and UCBRFx are listed in Table 19-4 and Table 19-5 for a 32,768-Hz crystal sourcing ACLK and typical SMCLK frequencies. Please ensure that the selected BRCLK frequency does not exceed the device specific maximum USCI input frequency (see the device-specific data sheet).

The receive error is the accumulated time versus the ideal scanning time in the middle of each bit. The worst-case error is given for the reception of an 8-bit character with parity and one stop bit including synchronization error.

The transmit error is the accumulated timing error versus the ideal time of the bit period. The worst-case error is given for the transmission of an 8-bit character with parity and stop bit.

Table 19-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
32,768	1200	27	2	0	-2.8	1.4	-5.9	2.0
32,768	2400	13	6	0	-4.8	6.0	-9.7	8.3
32,768	4800	6	7	0	-12.1	5.7	-13.4	19.0
32,768	9600	3	3	0	-21.1	15.2	-44.3	21.3
1,000,000	9600	104	1	0	-0.5	0.6	-0.9	1.2
1,000,000	19200	52	0	0	-1.8	0	-2.6	0.9
1,000,000	38400	26	0	0	-1.8	0	-3.6	1.8
1,000,000	57600	17	3	0	-2.1	4.8	-6.8	5.8
1,000,000	115200	8	6	0	-7.8	6.4	-9.7	16.1
1,048,576	9600	109	2	0	-0.2	0.7	-1.0	0.8
1,048,576	19200	54	5	0	-1.1	1.0	-1.5	2.5
1,048,576	38400	27	2	0	-2.8	1.4	-5.9	2.0
1,048,576	57600	18	1	0	-4.6	3.3	-6.8	6.6
1,048,576	115200	9	1	0	-1.1	10.7	-11.5	11.3
4,000,000	9600	416	6	0	-0.2	0.2	-0.2	0.4
4,000,000	19200	208	3	0	-0.2	0.5	-0.3	0.8
4,000,000	38400	104	1	0	-0.5	0.6	-0.9	1.2
4,000,000	57600	69	4	0	-0.6	0.8	-1.8	1.1
4,000,000	115200	34	6	0	-2.1	0.6	-2.5	3.1
4,000,000	230400	17	3	0	-2.1	4.8	-6.8	5.8
4,194,304	9600	436	7	0	-0.3	0	-0.3	0.2

Table 19-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0 (continued)

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
4,194,304	19200	218	4	0	-0.2	0.2	-0.3	0.6
4,194,304	57600	72	7	0	-1.1	0.6	-1.3	1.9
4,194,304	115200	36	3	0	-1.9	1.5	-2.7	3.4
8,000,000	9600	833	2	0	-0.1	0	-0.2	0.1
8,000,000	19200	416	6	0	-0.2	0.2	-0.2	0.4
8,000,000	38400	208	3	0	-0.2	0.5	-0.3	0.8
8,000,000	57600	138	7	0	-0.7	0	-0.8	0.6
8,000,000	115200	69	4	0	-0.6	0.8	-1.8	1.1
8,000,000	230400	34	6	0	-2.1	0.6	-2.5	3.1
8,000,000	460800	17	3	0	-2.1	4.8	-6.8	5.8
8,388,608	9600	873	7	0	-0.1	0.06	-0.2	0,1
8,388,608	19200	436	7	0	-0.3	0	-0.3	0.2
8,388,608	57600	145	5	0	-0.5	0.3	-1.0	0.5
8,388,608	115200	72	7	0	-1.1	0.6	-1.3	1.9
12,000,000	9600	1250	0	0	0	0	-0.05	0.05
12,000,000	19200	625	0	0	0	0	-0.2	0
12,000,000	38400	312	4	0	-0.2	0	-0.2	0.2
12,000,000	57600	208	2	0	-0.5	0.2	-0.6	0.5
12,000,000	115200	104	1	0	-0.5	0.6	-0.9	1.2
12,000,000	230400	52	0	0	-1.8	0	-2.6	0.9
12,000,000	460800	26	0	0	-1.8	0	-3.6	1.8
16,000,000	9600	1666	6	0	-0.05	0.05	-0.05	0.1
16,000,000	19200	833	2	0	-0.1	0.05	-0.2	0.1
16,000,000	38400	416	6	0	-0.2	0.2	-0.2	0.4
16,000,000	57600	277	7	0	-0.3	0.3	-0.5	0.4
16,000,000	115200	138	7	0	-0.7	0	-0.8	0.6
16,000,000	230400	69	4	0	-0.6	0.8	-1.8	1.1
16,000,000	460800	34	6	0	-2.1	0.6	-2.5	3.1
16,777,216	9600	1747	5	0	-0.04	0.03	-0.08	0.05
16,777,216	19200	873	7	0	-0.09	0.06	-0.2	0.1
16,777,216	57600	291	2	0	-0.2	0.2	-0.5	0.2
16,777,216	115200	145	5	0	-0.5	0.3	-1.0	0.5
20,000,000	9600	2083	2	0	-0.05	0.02	-0.09	0.02
20,000,000	19200	1041	6	0	-0.06	0.06	-0.1	0.1
20,000,000	38400	520	7	0	-0.2	0.06	-0.2	0.2
20,000,000	57600	347	2	0	-0.06	0.2	-0.3	0.3
20,000,000	115200	173	5	0	-0.4	0.3	-0.8	0.5
20,000,000	230400	86	7	0	-1.0	0.6	-1.0	1.7
20,000,000	460800	43	3	0	-1.4	1.3	-3.3	1.8

Table 19-5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
1,000,000	9600	6	0	8	-1.8	0	-2.2	0.4
1,000,000	19200	3	0	4	-1.8	0	-2.6	0.9
1,048,576	9600	6	0	13	-2.3	0	-2.2	0.8
1,048,576	19200	3	1	6	-4.6	3.2	-5.0	4.7
4,000,000	9600	26	0	1	0	0.9	0	1.1
4,000,000	19200	13	0	0	-1.8	0	-1.9	0.2
4,000,000	38400	6	0	8	-1.8	0	-2.2	0.4
4,000,000	57600	4	5	3	-3.5	3.2	-1.8	6.4
4,000,000	115200	2	3	2	-2.1	4.8	-2.5	7.3
4,194,304	9600	27	0	5	0	0.2	0	0.5
4,194,304	19200	13	0	10	-2.3	0	-2.4	0.1
4,194,304	57600	4	4	7	-2.5	2.5	-1.3	5.1
4,194,304	115200	2	6	3	-3.9	2.0	-1.9	6.7
8,000,000	9600	52	0	1	-0.4	0	-0.4	0.1
8,000,000	19200	26	0	1	0	0.9	0	1.1
8,000,000	38400	13	0	0	-1.8	0	-1.9	0.2
8,000,000	57600	8	0	11	0	0.88	0	1.6
8,000,000	115200	4	5	3	-3.5	3.2	-1.8	6.4
8,000,000	230400	2	3	2	-2.1	4.8	-2.5	7.3
8,388,608	9600	54	0	10	0	0.2	-0.05	0.3
8,388,608	19200	27	0	5	0	0.2	0	0.5
8,388,608	57600	9	0	2	0	2.8	-0.2	3.0
8,388,608	115200	4	4	7	-2.5	2.5	-1.3	5.1
12,000,000	9600	78	0	2	0	0	-0.05	0.05
12,000,000	19200	39	0	1	0	0	0	0.2
12,000,000	38400	19	0	8	-1.8	0	-1.8	0.1
12,000,000	57600	13	0	0	-1.8	0	-1.9	0.2
12,000,000	115200	6	0	8	-1.8	0	-2.2	0.4
12,000,000	230400	3	0	4	-1.8	0	-2.6	0.9
16,000,000	9600	104	0	3	0	0.2	0	0.3
16,000,000	19200	52	0	1	-0.4	0	-0.4	0.1
16,000,000	38400	26	0	1	0	0.9	0	1.1
16,000,000	57600	17	0	6	0	0.9	-0.1	1.0
16,000,000	115200	8	0	11	0	0.9	0	1.6
16,000,000	230400	4	5	3	-3.5	3.2	-1.8	6.4
16,000,000	460800	2	3	2	-2.1	4.8	-2.5	7.3
16,777,216	9600	109	0	4	0	0.2	-0.02	0.3
16,777,216	19200	54	0	10	0	0.2	-0.05	0.3
16,777,216	57600	18	0	3	-1.0	0	-1.0	0.3
16,777,216	115200	9	0	2	0	2.8	-0.2	3.0
20,000,000	9600	130	0	3	-0.2	0	-0.2	0.04
20,000,000	19200	65	0	2	0	0.4	-0.03	0.4
20,000,000	38400	32	0	9	0	0.4	0	0.5
20,000,000	57600	21	0	11	-0.7	0	-0.7	0.3
20,000,000	115200	10	0	14	0	2.5	-0.2	2.6
20,000,000	230400	5	0	7	0	2.5	0	3.5

Table 19-5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1 (continued)

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
20,000,000	460800	2	6	10	-3.2	1.8	-2.8	4.6

19.3.14 Using the USCI Module in UART Mode With Low-Power Modes

The USCI module provides automatic clock activation for use with low-power modes. When the USCI clock source is inactive because the device is in a low-power mode, the USCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the USCI module returns to its idle condition. After the USCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

19.3.15 USCI Interrupts

The USCI has only one interrupt vector that is shared for transmission and for reception. USCI_Ax and USC_Bx do not share the same interrupt vector.

19.3.15.1 USCI Transmit Interrupt Operation

The UCTXIFG interrupt flag is set by the transmitter to indicate that UCAxTXBUF is ready to accept another character. An interrupt request is generated if UCTXIE and GIE are also set. UCTXIFG is automatically reset if a character is written to UCAxTXBUF.

UCTXIFG is set after a PUC or when UCSWRST = 1. UCTXIE is reset after a PUC or when UCSWRST = 1.

19.3.15.2 USCI Receive Interrupt Operation

The UCRXIFG interrupt flag is set each time a character is received and loaded into UCAxRXBUF. An interrupt request is generated if UCRXIE and GIE are also set. UCRXIFG and UCRXIE are reset by a system reset PUC signal or when UCSWRST = 1. UCRXIFG is automatically reset when UCAxRXBUF is read.

Additional interrupt control features include:

- When UCAxRXEIE = 0, erroneous characters do not set UCRXIFG.
- When UCDORM = 1, nonaddress characters do not set UCRXIFG in multiprocessor modes. In plain UART mode, no characters are set UCRXIFG.
- When UCBRKIE = 1, a break condition sets the UCBRK bit and the UCRXIFG flag.

19.3.15.3 UCAxIV, Interrupt Vector Generator

The USCI interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCAxIV is used to determine which flag requested an interrupt. The highest-priority enabled interrupt generates a number in the UCAxIV register that can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCAxIV value.

Any access, read or write, of the UCAxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

UCAxIV Software Example

The following software example shows the recommended use of UCAxIV. The UCAxIV value is added to the PC to automatically jump to the appropriate routine. The following example is given for USCI_A0.

```
USCI_UART_ISR
         ADD    &UCA0IV, PC    ; Add offset to jump table
         RETI                  ; Vector 0: No interrupt
         JMP    RXIFG_ISR      ; Vector 2: RXIFG
TXIFG_ISR                      ; Vector 4: TXIFG
         ...                   ; Task starts here
         RETI                  ; Return
RXIFG_ISR                      ; Vector 2
         ...                   ; Task starts here
         RETI                  ; Return
```

19.4 USCI Registers – UART Mode

The USCI registers applicable in UART mode listed in Table 19-6. The base address can be found in the device-specific data sheet. The address offsets are listed in Table 19-6.

Table 19-6. USCI_Ax Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
USCI_Ax Control Word 0	UCAxCTLW0	Read/write	Word	00h	0001h
USCI_Ax Control 1	UCAxCTL1	Read/write	Byte	00h	01h
USCI_Ax Control 0	UCAxCTL0	Read/write	Byte	01h	00h
USCI_Ax Baud Rate Control Word	UCAxBRW	Read/write	Word	06h	0000h
USCI_Ax Baud Rate Control 0	UCAxBR0	Read/write	Byte	06h	00h
USCI_Ax Baud Rate Control 1	UCAxBR1	Read/write	Byte	07h	00h
USCI_Ax Modulation Control	UCAxMCTL	Read/write	Byte	08h	00h
Reserved - reads zero		Read	Byte	09h	00h
USCI_Ax Status	UCAxSTAT	Read/write	Byte	0Ah	00h
Reserved - reads zero		Read	Byte	0Bh	00h
USCI_Ax Receive Buffer	UCAxRXBUF	Read/write	Byte	0Ch	00h
Reserved - reads zero		Read	Byte	0Dh	00h
USCI_Ax Transmit Buffer	UCAxTXBUF	Read/write	Byte	0Eh	00h
Reserved - reads zero		Read	Byte	0Fh	00h
USCI_Ax Auto Baud Rate Control	UCAxABCTL	Read/write	Byte	10h	00h
Reserved - reads zero		Read	Byte	11h	00h
USCI_Ax IrDA Control	UCAxIRCTL	Read/write	Word	12h	0000h
USCI_Ax IrDA Transmit Control	UCAxIRTCTL	Read/write	Byte	12h	00h
USCI_Ax IrDA Receive Control	UCAxIRRCTL	Read/write	Byte	13h	00h
USCI_Ax Interrupt Control	UCAxICTL	Read/write	Word	1Ch	0000h
USCI_Ax Interrupt Enable	UCAxIE	Read/write	Byte	1Ch	00h
USCI_Ax Interrupt Flag	UCAxIFG	Read/write	Byte	1Dh	00h
USCI_Ax Interrupt Vector	UCAxIV	Read	Word	1Eh	0000h

USCI_Ax Control Register 0 (UCAxCTL0)

7	6	5	4	3	2	1	0
UCPEN	**UCPAR**	**UCMSB**	**UC7BIT**	**UCSPB**	**UCMODEx**		**UCSYNC=0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCPEN	Bit 7	Parity enable	
		0	Parity disabled
		1	Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.
UCPAR	Bit 6	Parity select. UCPAR is not used when parity is disabled.	
		0	Odd parity
		1	Even parity
UCMSB	Bit 5	MSB first select. Controls the direction of the receive and transmit shift register.	
		0	LSB first
		1	MSB first
UC7BIT	Bit 4	Character length. Selects 7-bit or 8-bit character length.	
		0	8-bit data
		1	7-bit data
UCSPB	Bit 3	Stop bit select. Number of stop bits.	
		0	One stop bit
		1	Two stop bits
UCMODEx	Bits 2-1	USCI mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0.	
		00	UART mode
		01	Idle-line multiprocessor mode
		10	Address-bit multiprocessor mode
		11	UART mode with automatic baud-rate detection
UCSYNC	Bit 0	Synchronous mode enable	
		0	Asynchronous mode
		1	Synchronous mode

USCI_Ax Control Register 1 (UCAxCTL1)

7	6	5	4	3	2	1	0
UCSSELx		**UCRXEIE**	**UCBRKIE**	**UCDORM**	**UCTXADDR**	**UCTXBRK**	**UCSWRST**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

UCSSELx	Bits 7-6	USCI clock source select. These bits select the BRCLK source clock. 00 UCAxCLK (external USCI clock) 01 ACLK 10 SMCLK 11 SMCLK
UCRXEIE	Bit 5	Receive erroneous-character interrupt enable 0 Erroneous characters rejected and UCRXIFG is not set. 1 Erroneous characters received set UCRXIFG.
UCBRKIE	Bit 4	Receive break character interrupt enable 0 Received break characters do not set UCRXIFG. 1 Received break characters set UCRXIFG.
UCDORM	Bit 3	Dormant. Puts USCI into sleep mode. 0 Not dormant. All received characters set UCRXIFG. 1 Dormant. Only characters that are preceded by an idle-line or with address bit set UCRXIFG. In UART mode with automatic baud-rate detection, only the combination of a break and synch field sets UCRXIFG.
UCTXADDR	Bit 2	Transmit address. Next frame to be transmitted is marked as address, depending on the selected multiprocessor mode. 0 Next frame transmitted is data. 1 Next frame transmitted is an address.
UCTXBRK	Bit 1	Transmit break. Transmits a break with the next write to the transmit buffer. In UART mode with automatic baud-rate detection, 055h must be written into UCAxTXBUF to generate the required break/synch fields. Otherwise, 0h must be written into the transmit buffer. 0 Next frame transmitted is not a break. 1 Next frame transmitted is a break or a break/synch.
UCSWRST	Bit 0	Software reset enable 0 Disabled. USCI reset released for operation. 1 Enabled. USCI logic held in reset state.

USCI_Ax Baud Rate Control Register 0 (UCAxBR0)

7	6	5	4	3	2	1	0
UCBRx - low byte							
rw	rw	rw	rw	rw	rw	rw	rw

USCI_Ax Baud Rate Control Register 1 (UCAxBR1)

7	6	5	4	3	2	1	0
UCBRx - high byte							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx	Clock prescaler setting of the baud-rate generator. The 16-bit value of (UCAxBR0 + UCAxBR1 × 256) forms the prescaler value UCBRx.

USCI_Ax Modulation Control Register (UCAxMCTL)

7	6	5	4	3	2	1	0
UCBRFx				UCBRSx			UCOS16
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCBRFx	Bits 7-4	First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. Table 19-3 shows the modulation pattern.
UCBRSx	Bits 3-1	Second modulation stage select. These bits determine the modulation pattern for BITCLK. Table 19-2 shows the modulation pattern.
UCOS16	Bit 0	Oversampling mode enabled 0 Disabled 1 Enabled

USCI_Ax Status Register (UCAxSTAT)

7	6	5	4	3	2	1	0
UCLISTEN	**UCFE**	**UCOE**	**UCPE**	**UCBRK**	**UCRXERR**	**UCADDR/ UCIDLE**	**UCBUSY**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

UCLISTEN	Bit 7	Listen enable. The UCLISTEN bit selects loopback mode. 0 Disabled 1 Enabled. UCAxTXD is internally fed back to the receiver.
UCFE	Bit 6	Framing error flag 0 No error 1 Character received with low stop bit
UCOE	Bit 5	Overrun error flag. This bit is set when a character is transferred into UCAxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly. 0 No error 1 Overrun error occurred.
UCPE	Bit 4	Parity error flag. When UCPEN = 0, UCPE is read as 0. 0 No error 1 Character received with parity error
UCBRK	Bit 3	Break detect flag 0 No break condition 1 Break condition occurred.
UCRXERR	Bit 2	Receive error flag. This bit indicates a character was received with error(s). When UCRXERR = 1, on or more error flags, UCFE, UCPE, or UCOE is also set. UCRXERR is cleared when UCAxRXBUF is read. 0 No receive errors detected 1 Receive error detected
UCADDR	Bit 1	Address received in address-bit multiprocessor mode. UCADDR is cleared when UCAxRXBUF is read. 0 Received character is data. 1 Received character is an address.
UCIDLE		Idle line detected in idle-line multiprocessor mode. UCIDLE is cleared when UCAxRXBUF is read. 0 No idle line detected 1 Idle line detected
UCBUSY	Bit 0	USCI busy. This bit indicates if a transmit or receive operation is in progress. 0 USCI inactive 1 USCI transmitting or receiving

USCI_Ax Receive Buffer Register (UCAxRXBUF)

7	6	5	4	3	2	1	0
			UCRXBUFx				
r	r	r	r	r	r	r	r

UCRXBUFx	Bits 7-0	The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCAxRXBUF resets the receive-error bits, the UCADDR or UCIDLE bit, and UCRXIFG. In 7-bit data mode, UCAxRXBUF is LSB justified and the MSB is always reset.

USCI_Ax Transmit Buffer Register (UCAxTXBUF)

7	6	5	4	3	2	1	0
			UCTXBUFx				
rw	rw	rw	rw	rw	rw	rw	rw

UCTXBUFx	Bits 7-0	The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted on UCAxTXD. Writing to the transmit data buffer clears UCTXIFG. The MSB of UCAxTXBUF is not used for 7-bit data and is reset.

USCI_Ax IrDA Transmit Control Register (UCAxIRTCTL)

7	6	5	4	3	2	1	0
UCIRTXPLx						**UCIRTXCLK**	**UCIREN**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCIRTXPLx Bits 7-2 Transmit pulse length
Pulse length t_{PULSE} = (UCIRTXPLx + 1) / (2 × $f_{IRTXCLK}$)

UCIRTXCLK Bit 1 IrDA transmit pulse clock select
- 0 BRCLK
- 1 BITCLK16 when UCOS16 = 1. Otherwise, BRCLK.

UCIREN Bit 0 IrDA encoder/decoder enable
- 0 IrDA encoder/decoder disabled
- 1 IrDA encoder/decoder enabled

USCI_Ax IrDA Receive Control Register (UCAxIRRCTL)

7	6	5	4	3	2	1	0
UCIRRXFLx						**UCIRRXPL**	**UCIRRXFE**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCIRRXFLx Bits 7-2 Receive filter length. The minimum pulse length for receive is given by:
t_{MIN} = (UCIRRXFLx + 4) / (2 × f_{BRCLK})

UCIRRXPL Bit 1 IrDA receive input UCAxRXD polarity
- 0 IrDA transceiver delivers a high pulse when a light pulse is seen.
- 1 IrDA transceiver delivers a low pulse when a light pulse is seen.

UCIRRXFE Bit 0 IrDA receive filter enabled
- 0 Receive filter disabled
- 1 Receive filter enabled

USCI_Ax Auto Baud Rate Control Register (UCAxABCTL)

7	6	5	4	3	2	1	0
Reserved		**UCDELIMx**		**UCSTOE**	**UCBTOE**	**Reserved**	**UCABDEN**
r-0	r-0	rw-0	rw-0	rw-0	rw-0	r-0	rw-0

Reserved Bits 7-6 Reserved

UCDELIMx Bits 5-4 Break/synch delimiter length
- 00 1 bit time
- 01 2 bit times
- 10 3 bit times
- 11 4 bit times

UCSTOE Bit 3 Synch field time out error
- 0 No error
- 1 Length of synch field exceeded measurable time.

UCBTOE Bit 2 Break time out error
- 0 No error
- 1 Length of break field exceeded 22 bit times.

Reserved Bit 1 Reserved

UCABDEN Bit 0 Automatic baud-rate detect enable
- 0 Baud-rate detection disabled. Length of break and synch field is not measured.
- 1 Baud-rate detection enabled. Length of break and synch field is measured and baud-rate settings are changed accordingly.

USCI_Ax Interrupt Enable Register (UCAxIE)

7	6	5	4	3	2	1	0
Reserved						**UCTXIE**	**UCRXIE**
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Reserved	Bits 7-2	Reserved
UCTXIE	Bit 1	Transmit interrupt enable 0 Interrupt disabled 1 Interrupt enabled
UCRXIE	Bit 0	Receive interrupt enable 0 Interrupt disabled 1 Interrupt enabled

USCI_Ax Interrupt Flag Register (UCAxIFG)

7	6	5	4	3	2	1	0
Reserved						**UCTXIFG**	**UCRXIFG**
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-0

Reserved	Bits 7-2	Reserved
UCTXIFG	Bit 1	Transmit interrupt flag. UCTXIFG is set when UCAxTXBUF empty. 0 No interrupt pending 1 Interrupt pending
UCRXIFG	Bit 0	Receive interrupt flag. UCRXIFG is set when UCAxRXBUF has received a complete character. 0 No interrupt pending 1 Interrupt pending

USCI_Ax Interrupt Vector Register (UCAxIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**UCIVx**		**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

UCIVx Bits 15-0 USCI interrupt vector value

UCAxIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending		
002h	Data received	UCRXIFG	Highest
004h	Transmit buffer empty	UCTXIFG	Lowest

Universal Serial Communication Interface – SPI Mode

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface (SPI) mode.

Topic **Page**

20.1	**Universal Serial Communication Interface (USCI) Overview**	**498**
20.2	**USCI Introduction – SPI Mode**	**499**
20.3	**USCI Operation – SPI Mode**	**501**
20.4	**USCI Registers – SPI Mode**	**506**

20.1 Universal Serial Communication Interface (USCI) Overview

The universal serial communication interface (USCI) modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI_A is different from USCI_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. See the device-specific data sheet to determine which USCI modules, if any, are implemented on which devices.

USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud-rate detection for LIN communications
- SPI mode

USCI_Bx modules support:

- I^2C mode
- SPI mode

20.2 USCI Introduction – SPI Mode

In synchronous mode, the USCI connects the device to an external system via three or four pins: UCxSIMO, UCxSOMI, UCxCLK, and UCxSTE. SPI mode is selected when the UCSYNC bit is set, and SPI mode (3-pin or 4-pin) is selected with the UCMODEx bits.

SPI mode features include:

- 7-bit or 8-bit data length
- LSB-first or MSB-first data transmit and receive
- 3-pin and 4-pin SPI operation
- Master or slave modes
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- Continuous transmit and receive operation
- Selectable clock polarity and phase control
- Programmable clock frequency in master mode
- Independent interrupt capability for receive and transmit
- Slave operation in LPM4

Figure 20-1 shows the USCI when configured for SPI mode.

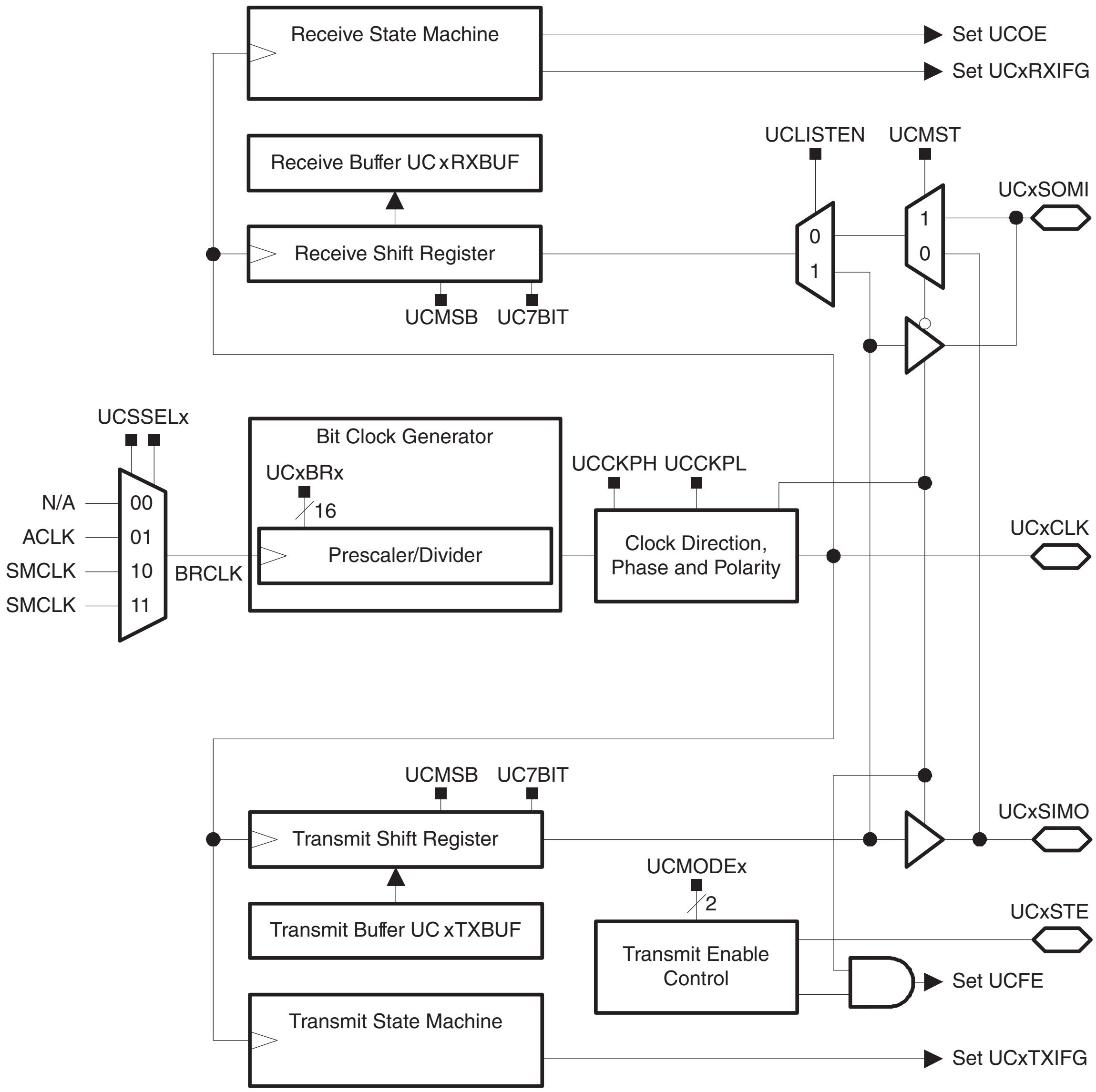


Figure 20-1. USCI Block Diagram – SPI Mode

20.3 USCI Operation – SPI Mode

In SPI mode, serial data is transmitted and received by multiple devices using a shared clock provided by the master. An additional pin, UCxSTE, is provided to enable a device to receive and transmit data and is controlled by the master.

Three or four signals are used for SPI data exchange:

- UCxSIMO slave in, master out Master mode: UCxSIMO is the data output line. Slave mode: UCxSIMO is the data input line.
- UCxSOMI slave out, master in Master mode: UCxSOMI is the data input line. Slave mode: UCxSOMI is the data output line.
- UCxCLK USCI SPI clock Master mode: UCxCLK is an output. Slave mode: UCxCLK is an input.
- UCxSTE slave transmit enable. Used in 4-pin mode to allow multiple masters on a single bus. Not used in 3-pin mode. Table 20-1 describes the UCxSTE operation.

Table 20-1. UCxSTE Operation

UCMODEx	UCxSTE Active State	UCxSTE	Slave	Master
01	High	0	Inactive	Active
		1	Active	Inactive
10	Low	0	Active	Inactive
		1	Inactive	Active

20.3.1 USCI Initialization and Reset

The USCI is reset by a PUC or by the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the USCI in a reset condition. When set, the UCSWRST bit resets the UCRXIE, UCTXIE, UCRXIFG, UCOE, and UCFE bits, and sets the UCTXIFG flag. Clearing UCSWRST releases the USCI for operation.

NOTE: Initializing or reconfiguring the USCI module

The recommended USCI initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST,&UCxCTL1`).
2. Initialize all USCI registers with UCSWRST = 1 (including UCxCTL1).
3. Configure ports.
4. Clear UCSWRST via software (`BIC.B #UCSWRST,&UCxCTL1`).
5. Enable interrupts (optional) via UCRXIE and/or UCTXIE.

20.3.2 Character Format

The USCI module in SPI mode supports 7-bit and 8-bit character lengths selected by the UC7BIT bit. In 7-bit data mode, UCxRXBUF is LSB justified and the MSB is always reset. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first.

NOTE: Default character format

The default SPI character transmission is LSB first. For communication with other SPI interfaces, MSB-first mode may be required.

NOTE: Character format for Figures

Figures throughout this chapter use MSB-first format.

20.3.3 Master Mode

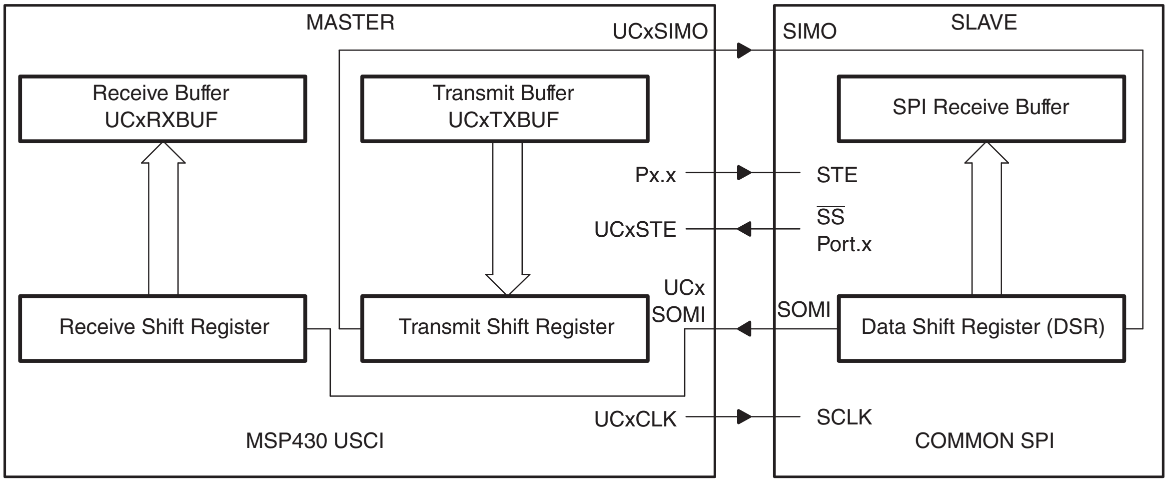


Figure 20-2. USCI Master and External Slave

Figure 20-2 shows the USCI as a master in both 3-pin and 4-pin configurations. The USCI initiates data transfer when data is moved to the transmit data buffer UCxTXBUF. The UCxTXBUF data is moved to the transmit (TX) shift register when the TX shift register is empty, initiating data transfer on UCxSIMO starting with either the MSB or LSB, depending on the UCMSB setting. Data on UCxSOMI is shifted into the receive shift register on the opposite clock edge. When the character is received, the receive data is moved from the receive (RX) shift register to the received data buffer UCxRXBUF and the receive interrupt flag UCRXIFG is set, indicating the RX/TX operation is complete.

A set transmit interrupt flag, UCTXIFG, indicates that data has moved from UCxTXBUF to the TX shift register and UCxTXBUF is ready for new data. It does not indicate RX/TX completion.

To receive data into the USCI in master mode, data must be written to UCxTXBUF, because receive and transmit operations operate concurrently.

20.3.3.1 4-Pin SPI Master Mode

In 4-pin master mode, UCxSTE is used to prevent conflicts with another master and controls the master as described in Table 20-1. When UCxSTE is in the master-inactive state:

- UCxSIMO and UCxCLK are set to inputs and no longer drive the bus.
- The error bit UCFE is set, indicating a communication integrity violation to be handled by the user.
- The internal state machines are reset and the shift operation is aborted.

If data is written into UCxTXBUF while the master is held inactive by UCxSTE, it is transmit as soon as UCxSTE transitions to the master-active state. If an active transfer is aborted by UCxSTE transitioning to the master-inactive state, the data must be rewritten into UCxTXBUF to be transferred when UCxSTE transitions back to the master-active state. The UCxSTE input signal is not used in 3-pin master mode.

20.3.4 Slave Mode

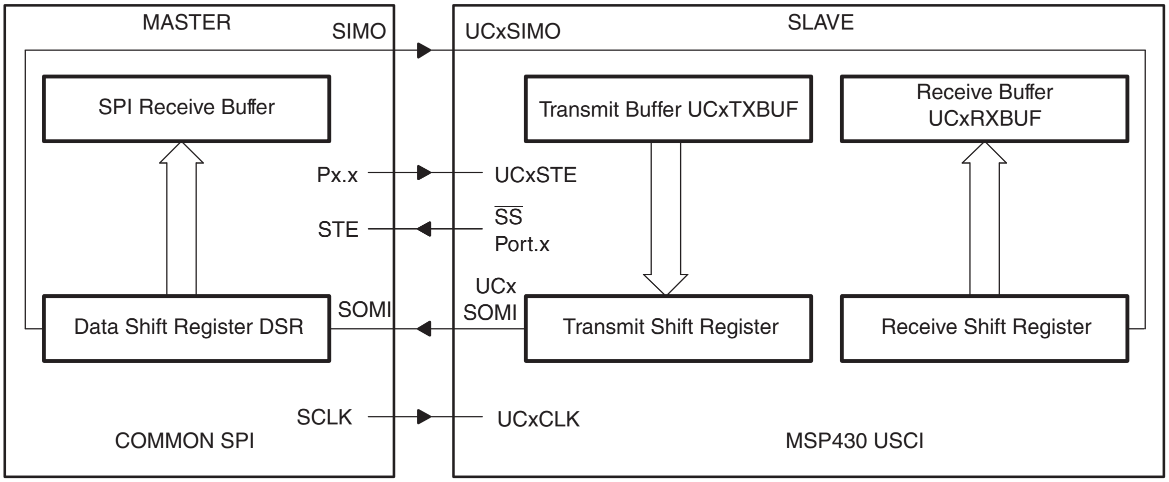


Figure 20-3. USCI Slave and External Master

Figure 20-3 shows the USCI as a slave in both 3-pin and 4-pin configurations. UCxCLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal bit clock generator. Data written to UCxTXBUF and moved to the TX shift register before the start of UCxCLK is transmitted on UCxSOMI. Data on UCxSIMO is shifted into the receive shift register on the opposite edge of UCxCLK and moved to UCxRXBUF when the set number of bits are received. When data is moved from the RX shift register to UCxRXBUF, the UCRXIFG interrupt flag is set, indicating that data has been received. The overrun error bit UCOE is set when the previously received data is not read from UCxRXBUF before new data is moved to UCxRXBUF.

20.3.4.1 4-Pin SPI Slave Mode

In 4-pin slave mode, UCxSTE is used by the slave to enable the transmit and receive operations and is provided by the SPI master. When UCxSTE is in the slave-active state, the slave operates normally. When UCxSTE is in the slave- inactive state:

- Any receive operation in progress on UCxSIMO is halted.
- UCxSOMI is set to the input direction.
- The shift operation is halted until the UCxSTE line transitions into the slave transmit active state.

The UCxSTE input signal is not used in 3-pin slave mode.

20.3.5 SPI Enable

When the USCI module is enabled by clearing the UCSWRST bit, it is ready to receive and transmit. In master mode, the bit clock generator is ready, but is not clocked nor producing any clocks. In slave mode, the bit clock generator is disabled and the clock is provided by the master.

A transmit or receive operation is indicated by UCBUSY = 1.

A PUC or set UCSWRST bit disables the USCI immediately and any active transfer is terminated.

20.3.5.1 Transmit Enable

In master mode, writing to UCxTXBUF activates the bit clock generator, and the data begins to transmit.

In slave mode, transmission begins when a master provides a clock and, in 4-pin mode, when the UCxSTE is in the slave-active state.

20.3.5.2 Receive Enable

The SPI receives data when a transmission is active. Receive and transmit operations operate concurrently.

20.3.6 Serial Clock Control

UCxCLK is provided by the master on the SPI bus. When UCMST = 1, the bit clock is provided by the USCI bit clock generator on the UCxCLK pin. The clock used to generate the bit clock is selected with the UCSSELx bits. When UCMST = 0, the USCI clock is provided on the UCxCLK pin by the master, the bit clock generator is not used, and the UCSSELx bits are don't care. The SPI receiver and transmitter operate in parallel and use the same clock source for data transfer.

The 16-bit value of UCBRx in the bit rate control registers (UCxxBR1 and UCxxBR0) is the division factor of the USCI clock source, BRCLK. The maximum bit clock that can be generated in master mode is BRCLK. Modulation is not used in SPI mode, and UCAxMCTL should be cleared when using SPI mode for USCI_A. The UCAxCLK/UCBxCLK frequency is given by:

$$f_{BitClock} = f_{BRCLK}/UCBRx$$

20.3.6.1 Serial Clock Polarity and Phase

The polarity and phase of UCxCLK are independently configured via the UCCKPL and UCCKPH control bits of the USCI. Timing for each case is shown in Figure 20-4.

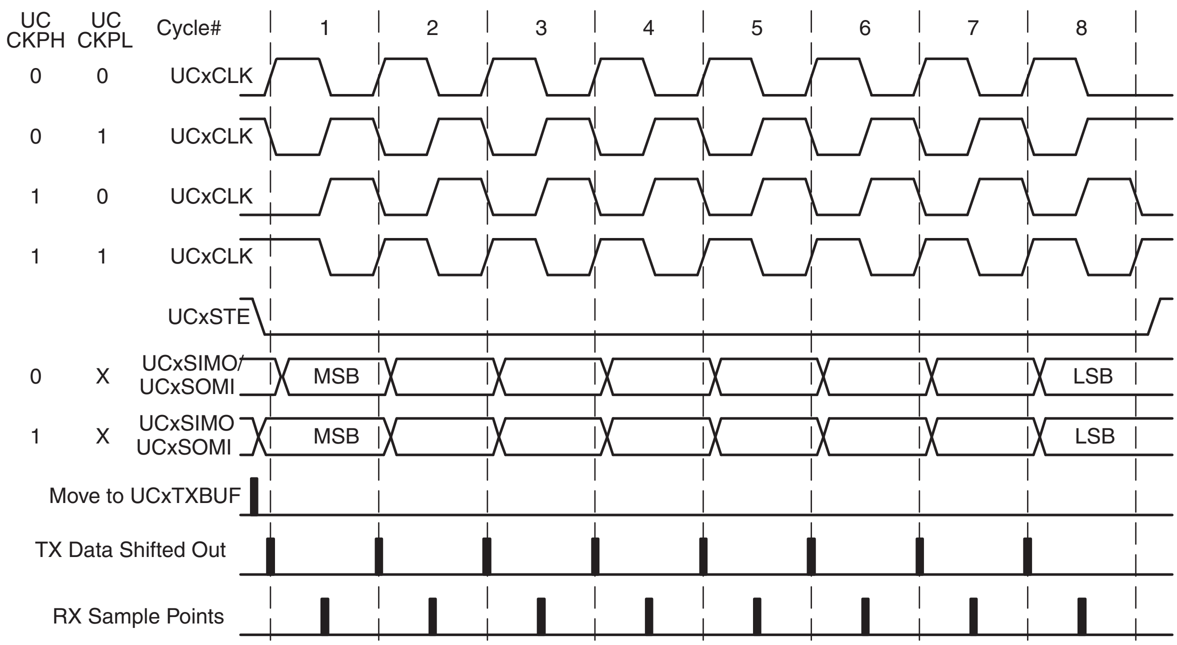


Figure 20-4. USCI SPI Timing With UCMSB = 1

20.3.7 Using the SPI Mode With Low-Power Modes

The USCI module provides automatic clock activation for use with low-power modes. When the USCI clock source is inactive because the device is in a low-power mode, the USCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the USCI module returns to its idle condition. After the USCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

In SPI slave mode, no internal clock source is required because the clock is provided by the external master. It is possible to operate the USCI in SPI slave mode while the device is in LPM4 and all clock sources are disabled. The receive or transmit interrupt can wake up the CPU from any low-power mode.

20.3.8 SPI Interrupts

The USCI has only one interrupt vector that is shared for transmission and for reception. USCI_Ax and USC_Bx do not share the same interrupt vector.

20.3.8.1 SPI Transmit Interrupt Operation

The UCTXIFG interrupt flag is set by the transmitter to indicate that UCxTXBUF is ready to accept another character. An interrupt request is generated if UCTXIE and GIE are also set. UCTXIFG is automatically reset if a character is written to UCxTXBUF. UCTXIFG is set after a PUC or when UCSWRST = 1. UCTXIE is reset after a PUC or when UCSWRST = 1.

NOTE: Writing to UCxTXBUF in SPI mode

Data written to UCxTXBUF when UCTXIFG = 0 may result in erroneous data transmission.

20.3.8.2 SPI Receive Interrupt Operation

The UCRXIFG interrupt flag is set each time a character is received and loaded into UCxRXBUF. An interrupt request is generated if UCRXIE and GIE are also set. UCRXIFG and UCRXIE are reset by a system reset PUC signal or when UCSWRST = 1. UCRXIFG is automatically reset when UCxRXBUF is read.

20.3.8.3 UCxIV, Interrupt Vector Generator

The USCI interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCxIV is used to determine which flag requested an interrupt. The highest-priority enabled interrupt generates a number in the UCxIV register that can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCxIV value.

Any access, read or write, of the UCxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

UCxIV Software Example

The following software example shows the recommended use of UCxIV. The UCxIV value is added to the PC to automatically jump to the appropriate routine. The following example is given for USCI_B0.

```
USCI_SPI_ISR
         ADD      &UCB0IV, PC  ; Add offset to jump table
         RETI                  ; Vector 0: No interrupt
         JMP      RXIFG_ISR    ; Vector 2: RXIFG
TXIFG_ISR                      ; Vector 4: TXIFG
         ...                   ; Task starts here
         RETI                  ; Return
RXIFG_ISR                      ; Vector 2
         ...                   ; Task starts here
         RETI                  ; Return
```

20.4 USCI Registers – SPI Mode

The USCI registers applicable in SPI mode are listed in Table 20-2 and Table 20-3. The base addresses can be found in the device-specific data sheet. The address offsets are listed in Table 20-2 and Table 20-3.

Table 20-2. USCI_Ax Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
USCI_Ax Control Word 0	UCAxCTLW0	Read/write	Word	00h	0001h
USCI_Ax Control 1	UCAxCTL1	Read/write	Byte	00h	01h
USCI_Ax Control 0	UCAxCTL0	Read/write	Byte	01h	00h
USCI_Ax Bit Rate Control Word	UCAxBRW	Read/write	Word	06h	0000h
USCI_Ax Bit Rate Control 0	UCAxBR0	Read/write	Byte	06h	00h
USCI_Ax Bit Rate Control 1	UCAxBR1	Read/write	Byte	07h	00h
USCI_Ax Modulation Control	UCAxMCTL	Read/write	Byte	08h	00h
USCI_Ax Status	UCAxSTAT	Read/write	Byte	0Ah	00h
Reserved - reads zero		Read	Byte	0Bh	00h
USCI_Ax Receive Buffer	UCAxRXBUF	Read/write	Byte	0Ch	00h
Reserved - reads zero		Read	Byte	0Dh	00h
USCI_Ax Transmit Buffer	UCAxTXBUF	Read/write	Byte	0Eh	00h
Reserved - reads zero		Read	Byte	0Fh	00h
USCI_Ax Interrupt Control	UCAxICTL	Read/write	Word	1Ch	0200h
USCI_Ax Interrupt Enable	UCAxIE	Read/write	Byte	1Ch	00h
USCI_Ax Interrupt Flag	UCAxIFG	Read/write	Byte	1Dh	02h
USCI_Ax Interrupt Vector	UCAxIV	Read	Word	1Eh	0000h

Table 20-3. USCI_Bx Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
USCI_Bx Control Word 0	UCBxCTLW0	Read/write	Word	00h	0101h
USCI_Bx Control 1	UCBxCTL1	Read/write	Byte	00h	01h
USCI_Bx Control 0	UCBxCTL0	Read/write	Byte	01h	01h
USCI_Bx Bit Rate Control Word	UCBxBRW	Read/write	Word	06h	0000h
USCI_Bx Bit Rate Control 0	UCBxBR0	Read/write	Byte	06h	00h
USCI_Bx Bit Rate Control 1	UCBxBR1	Read/write	Byte	07h	00h
USCI_Bx Status	UCBxSTAT	Read/write	Byte	0Ah	00h
Reserved - reads zero		Read	Byte	0Bh	00h
USCI_Bx Receive Buffer	UCBxRXBUF	Read/write	Byte	0Ch	00h
Reserved - reads zero		Read	Byte	0Dh	00h
USCI_Bx Transmit Buffer	UCBxTXBUF	Read/write	Byte	0Eh	00h
Reserved - reads zero		Read	Byte	0Fh	00h
USCI_Bx Interrupt Control	UCBxICTL	Read/write	Word	1Ch	0200h
USCI_Bx Interrupt Enable	UCBxIE	Read/write	Byte	1Ch	00h
USCI_Bx Interrupt Flag	UCBxIFG	Read/write	Byte	1Dh	02h
USCI_Bx Interrupt Vector	UCBxIV	Read	Word	1Eh	0000h

USCI_Ax Control Register 0 (UCAxCTL0)
USCI_Bx Control Register 0 (UCBxCTL0)

7	6	5	4	3	2	1	0
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC=1
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0[(1)] rw-1[(2)]

UCCKPH	Bit 7	Clock phase select 0 Data is changed on the first UCLK edge and captured on the following edge. 1 Data is captured on the first UCLK edge and changed on the following edge.
UCCKPL	Bit 6	Clock polarity select 0 The inactive state is low. 1 The inactive state is high.
UCMSB	Bit 5	MSB first select. Controls the direction of the receive and transmit shift register. 0 LSB first 1 MSB first
UC7BIT	Bit 4	Character length. Selects 7-bit or 8-bit character length. 0 8-bit data 1 7-bit data
UCMST	Bit 3	Master mode select 0 Slave mode 1 Master mode
UCMODEx	Bits 2-1	USCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00 3-pin SPI 01 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11 I^2C mode
UCSYNC	Bit 0	Synchronous mode enable 0 Asynchronous mode 1 Synchronous mode

(1) UCAxCTL0 (USCI_Ax)
(2) UCBxCTL0 (USCI_Bx)

USCI_Ax Control Register 1 (UCAxCTL1)
USCI_Bx Control Register 1 (UCBxCTL1)

7	6	5	4	3	2	1	0
UCSSELx		Unused					UCSWRST
rw-0	rw-0	rw-0[(1)] r0[(2)]	rw-0	rw-0	rw-0	rw-0	rw-1

UCSSELx	Bits 7-6	USCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode. 00 NA 01 ACLK 10 SMCLK 11 SMCLK
Unused	Bits 5-1	Unused
UCSWRST	Bit 0	Software reset enable 0 Disabled. USCI reset released for operation. 1 Enabled. USCI logic held in reset state.

(1) UCAxCTL1 (USCI_Ax)
(2) UCBxCTL1 (USCI_Bx)

USCI_Ax Bit Rate Control Register 0 (UCAxBR0)
USCI_Bx Bit Rate Control Register 0 (UCBxBR0)

7	6	5	4	3	2	1	0
UCBRx - low byte							
rw	rw	rw	rw	rw	rw	rw	rw

USCI_Ax Bit Rate Control Register 1 (UCAxBR1)
USCI_Bx Bit Rate Control Register 1 (UCBxBR1)

7	6	5	4	3	2	1	0
UCBRx - high byte							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx	Bits 7-0	Bit clock prescaler. The 16-bit value of (UCxxBR0 + UCxxBR1 × 256) forms the prescaler value UCBRx.

USCI_Ax Modulation Control Register (UCAxMCTL)

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

	Bits 7-0	Write as 0

USCI_Ax Status Register (UCAxSTAT)
USCI_Bx Status Register (UCBxSTAT)

7	6	5	4	3	2	1	0
UCLISTEN	**UCFE**	**UCOE**	**Unused**				**UCBUSY**
rw-0	rw-0	rw-0	rw-0 [(1)] r0 [(2)]	rw-0 [(1)] r0 [(2)]	rw-0 [(1)] r0 [(2)]	rw-0 [(1)] r0 [(2)]	r-0

UCLISTEN Bit 7 Listen enable. The UCLISTEN bit selects loopback mode.
- 0 Disabled
- 1 Enabled. The transmitter output is internally fed back to the receiver.

UCFE Bit 6 Framing error flag. This bit indicates a bus conflict in 4-wire master mode. UCFE is not used in 3-wire master or any slave mode.
- 0 No error
- 1 Bus conflict occurred.

UCOE Bit 5 Overrun error flag. This bit is set when a character is transferred into UCxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly.
- 0 No error
- 1 Overrun error occurred.

Unused Bits 4-1 Unused

UCBUSY Bit 0 USCI busy. This bit indicates if a transmit or receive operation is in progress.
- 0 USCI inactive
- 1 USCI transmitting or receiving

(1) UCAxSTAT (USCI_Ax)
(2) UCBxSTAT (USCI_Bx)

USCI_Ax Receive Buffer Register (UCAxRXBUF)
USCI_Bx Receive Buffer Register (UCBxRXBUF)

7	6	5	4	3	2	1	0
UCRXBUFx							
r	r	r	r	r	r	r	r

UCRXBUFx Bits 7-0 The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCxRXBUF resets the receive-error bits and UCRXIFG. In 7-bit data mode, UCxRXBUF is LSB justified and the MSB is always reset.

USCI_Ax Transmit Buffer Register (UCAxTXBUF)
USCI_Bx Transmit Buffer Register (UCBxTXBUF)

7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCTXBUFx Bits 7-0 The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted. Writing to the transmit data buffer clears UCTXIFG. The MSB of UCxTXBUF is not used for 7-bit data and is reset.

USCI_Ax Interrupt Enable Register (UCAxIE)
USCI_Bx Interrupt Enable Register (UCBxIE)

7	6	5	4	3	2	1	0
Reserved						**UCTXIE**	**UCRXIE**
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Reserved	Bits 7-2	Reserved
UCTXIE	Bit 1	Transmit interrupt enable 0 Interrupt disabled 1 Interrupt enabled
UCRXIE	Bit 0	Receive interrupt enable 0 Interrupt disabled 1 Interrupt enabled

USCI_Ax Interrupt Flag Register (UCAxIFG)
USCI_Bx Interrupt Flag Register (UCBxIFG)

7	6	5	4	3	2	1	0
Reserved						**UCTXIFG**	**UCRXIFG**
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-0

Reserved	Bits 7-2	Reserved
UCTXIFG	Bit 1	Transmit interrupt flag. UCTXIFG is set when UCxxTXBUF empty. 0 No interrupt pending 1 Interrupt pending
UCRXIFG	Bit 0	Receive interrupt flag. UCRXIFG is set when UCxxRXBUF has received a complete character. 0 No interrupt pending 1 Interrupt pending

USCI_Ax Interrupt Vector Register (UCAxIV)
USCI_Bx Interrupt Vector Register (UCBxIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**UCIVx**		**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

UCIVx Bits 15-0 USCI interrupt vector value

UCAxIV/ UCBxIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	Data received	UCRXIFG	Highest
004h	Transmit buffer empty	UCTXIFG	Lowest

Chapter 21

SLAU208F–June 2008–Revised March 2010

Universal Serial Communication Interface – I2C Mode

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the I²C mode.

Topic **Page**

21.1 Universal Serial Communication Interface (USCI) Overview 512
21.2 USCI Introduction – I²C Mode 513
21.3 USCI Operation – I²C Mode 514
21.4 USCI Registers– I²C Mode 533

21.1 Universal Serial Communication Interface (USCI) Overview

The USCI modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI_A is different from USCI_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. See the device-specific data sheet to determine which USCI modules, if any, are implemented on each device.

USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud-rate detection for LIN communications
- SPI mode

USCI_Bx modules support:

- I^2C mode
- SPI mode

21.2 USCI Introduction – I²C Mode

In I²C mode, the USCI module provides an interface between the device and I²C-compatible devices connected by the two-wire I²C serial bus. External components attached to the I²C bus serially transmit and/or receive serial data to/from the USCI module through the 2-wire I²C interface.

The I²C mode features include:

- Compliance to the Philips Semiconductor I²C specification v2.1
- 7-bit and 10-bit device addressing modes
- General call
- START/RESTART/STOP
- Multi-master transmitter/receiver mode
- Slave receiver/transmitter mode
- Standard mode up to 100 kbps and fast mode up to 400 kbps support
- Programmable UCxCLK frequency in master mode
- Designed for low power
- Slave receiver START detection for auto wake up from LPMx modes
- Slave operation in LPM4

Figure 21-1 shows the USCI when configured in I²C mode.

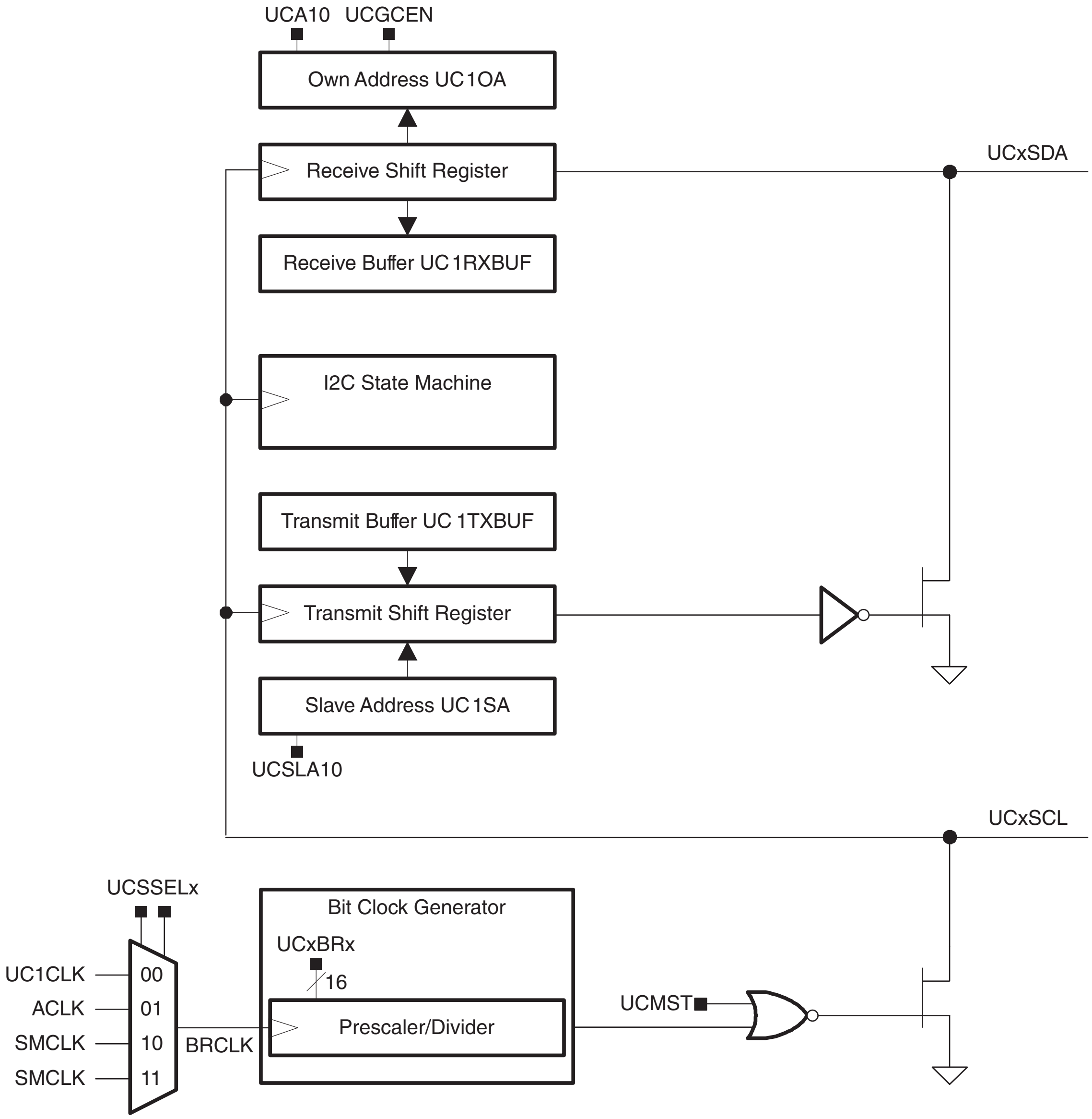


Figure 21-1. USCI Block Diagram – I²C Mode

21.3 USCI Operation – I²C Mode

The I²C mode supports any slave or master I²C-compatible device. Figure 21-2 shows an example of an I²C bus. Each I²C device is recognized by a unique address and can operate as either a transmitter or a receiver. A device connected to the I²C bus can be considered as the master or the slave when performing data transfers. A master initiates a data transfer and generates the clock signal SCL. Any device addressed by a master is considered a slave.

I²C data is communicated using the serial data (SDA) pin and the serial clock (SCL) pin. Both SDA and SCL are bidirectional and must be connected to a positive supply voltage using a pullup resistor.

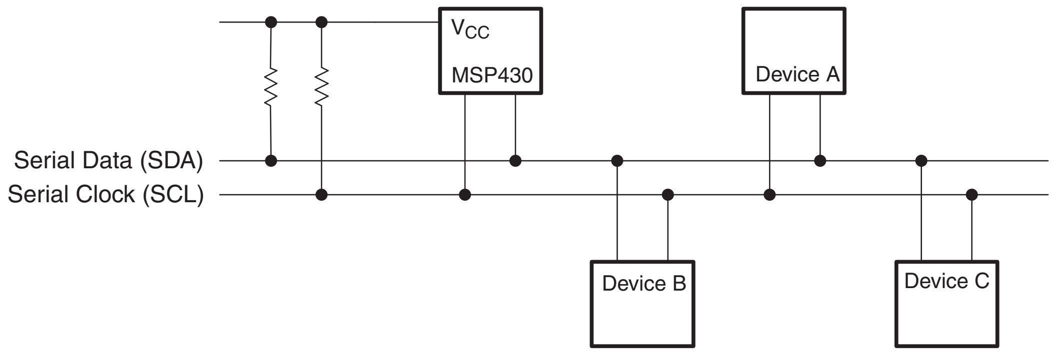


Figure 21-2. I²C Bus Connection Diagram

NOTE: SDA and SCL levels

The SDA and SCL pins must not be pulled up above the device V_{CC} level.

21.3.1 USCI Initialization and Reset

The USCI is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the USCI in a reset condition. To select I²C operation, the UCMODEx bits must be set to 11. After module initialization, it is ready for transmit or receive operation. Clearing UCSWRST releases the USCI for operation.

Configuring and reconfiguring the USCI module should be done when UCSWRST is set to avoid unpredictable behavior. Setting UCSWRST in I²C mode has the following effects:

- I²C communication stops.
- SDA and SCL are high impedance.
- UCBxI2CSTAT, bits 6–0 are cleared.
- Registers UCBxIE and UCBxIFG are cleared.
- All other bits and register remain unchanged.

NOTE: Initializing or re-configuring the USCI module

The recommended USCI initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST,&UCxCTL1`).
2. Initialize all USCI registers with UCSWRST = 1 (including UCxCTL1).
3. Configure ports.
4. Clear UCSWRST via software (`BIC.B #UCSWRST,&UCxCTL1`).
5. Enable interrupts (optional).

21.3.2 I²C Serial Data

One clock pulse is generated by the master device for each data bit transferred. The I²C mode operates with byte data. Data is transferred MSB first as shown in Figure 21-3.

The first byte after a START condition consists of a 7-bit slave address and the R/$\overline{W}$ bit. When R/$\overline{W}$ = 0, the master transmits data to a slave. When R/$\overline{W}$ = 1, the master receives data from a slave. The ACK bit is sent from the receiver after each byte on the ninth SCL clock.

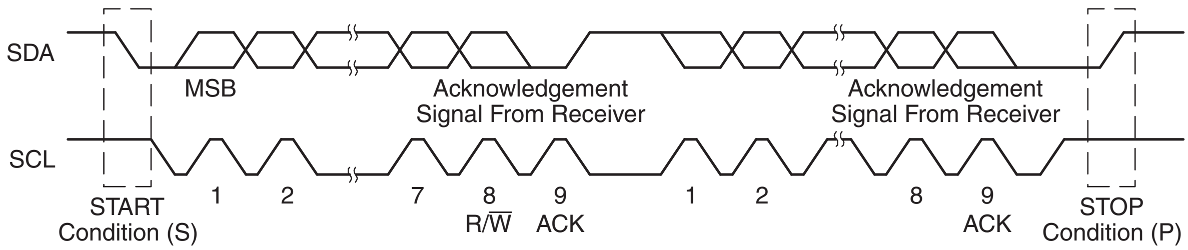


Figure 21-3. I²C Module Data Transfer

START and STOP conditions are generated by the master and are shown in Figure 21-3. A START condition is a high-to-low transition on the SDA line while SCL is high. A STOP condition is a low-to-high transition on the SDA line while SCL is high. The bus busy bit, UCBBUSY, is set after a START and cleared after a STOP.

Data on SDA must be stable during the high period of SCL (see Figure 21-4). The high and low state of SDA can only change when SCL is low, otherwise START or STOP conditions are generated.

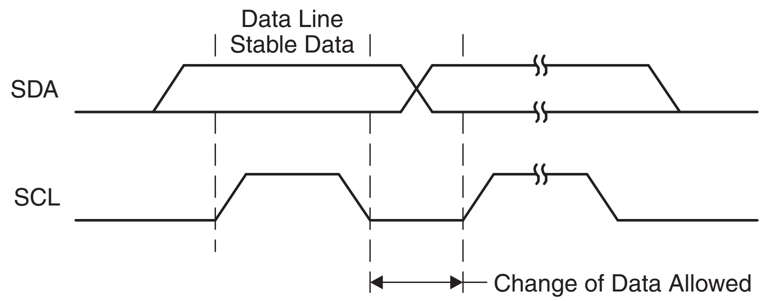


Figure 21-4. Bit Transfer on I²C Bus

21.3.3 I²C Addressing Modes

The I²C mode supports 7-bit and 10-bit addressing modes.

21.3.3.1 7-Bit Addressing

In the 7-bit addressing format (see Figure 21-5), the first byte is the 7-bit slave address and the R/$\overline{W}$ bit. The ACK bit is sent from the receiver after each byte.

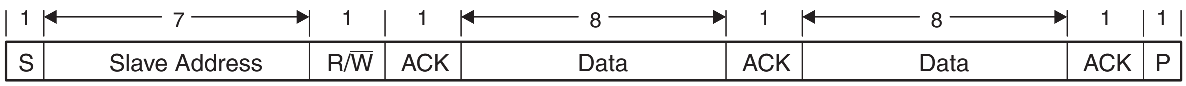


Figure 21-5. I²C Module 7-Bit Addressing Format

21.3.3.2 10-Bit Addressing

In the 10-bit addressing format (see Figure 21-6), the first byte is made up of 11110b plus the two MSBs of the 10-bit slave address and the R/$\overline{W}$ bit. The ACK bit is sent from the receiver after each byte. The next byte is the remaining eight bits of the 10-bit slave address, followed by the ACK bit and the 8-bit data. See I2C Slave 10-bit Addressing Mode and I2C Master 10-bit Addressing Mode for details how to use the 10-bit addressing mode with the USCI module.

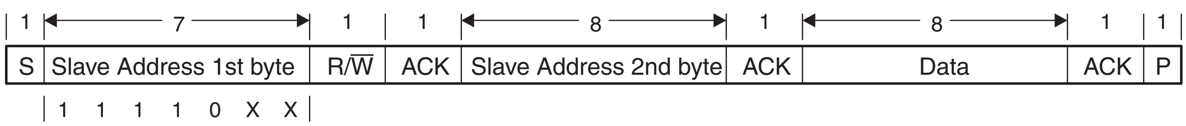


Figure 21-6. I²C Module 10-Bit Addressing Format

21.3.3.3 Repeated Start Conditions

The direction of data flow on SDA can be changed by the master, without first stopping a transfer, by issuing a repeated START condition. This is called a RESTART. After a RESTART is issued, the slave address is again sent out with the new data direction specified by the R/$\overline{W}$ bit. The RESTART condition is shown in Figure 21-7.

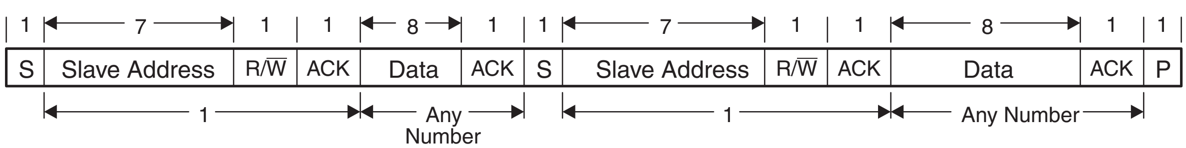


Figure 21-7. I²C Module Addressing Format With Repeated START Condition

21.3.4 I^{2}C Module Operating Modes

In I^{2}C mode, the USCI module can operate in master transmitter, master receiver, slave transmitter, or slave receiver mode. The modes are discussed in the following sections. Time lines are used to illustrate the modes.

Figure 21-8 shows how to interpret the time-line figures. Data transmitted by the master is represented by grey rectangles; data transmitted by the slave is represented by white rectangles. Data transmitted by the USCI module, either as master or slave, is shown by rectangles that are taller than the others.

Actions taken by the USCI module are shown in grey rectangles with an arrow indicating where in the the data stream the action occurs. Actions that must be handled with software are indicated with white rectangles with an arrow pointing to where in the data stream the action must take place.

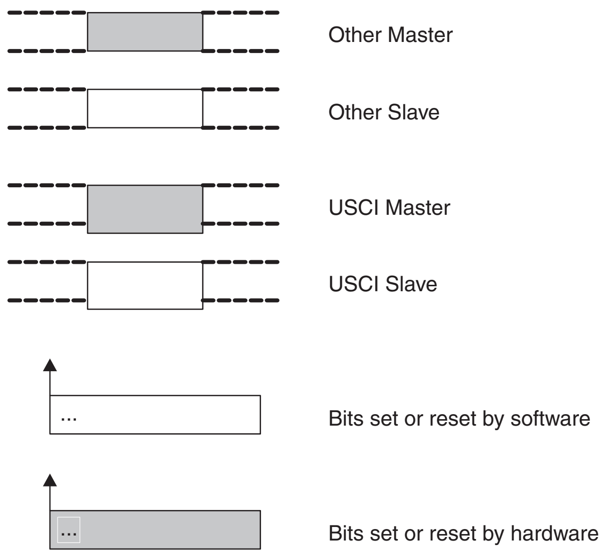


Figure 21-8. I^{2}C Time-Line Legend

21.3.4.1 Slave Mode

The USCI module is configured as an I^{2}C slave by selecting the I^{2}C mode with UCMODEx = 11 and UCSYNC = 1 and clearing the UCMST bit.

Initially, the USCI module must to be configured in receiver mode by clearing the UCTR bit to receive the I^{2}C address. Afterwards, transmit and receive operations are controlled automatically, depending on the R/$\overline{W}$ bit received together with the slave address.

The USCI slave address is programmed with the UCBxI2COA register. When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the slave responds to a general call.

When a START condition is detected on the bus, the USCI module receives the transmitted address and compare it against its own address stored in UCBxI2COA. The UCSTTIFG flag is set when address received matches the USCI slave address.

I²C Slave Transmitter Mode

Slave transmitter mode is entered when the slave address transmitted by the master is identical to its own address with a set R/$\overline{W}$ bit. The slave transmitter shifts the serial data out on SDA with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it does hold SCL low while intervention of the CPU is required after a byte has been transmitted.

If the master requests data from the slave, the USCI module is automatically configured as a transmitter and UCTR and UCTXIFG become set. The SCL line is held low until the first data to be sent is written into the transmit buffer UCBxTXBUF. Then the address is acknowledged, the UCSTTIFG flag is cleared, and the data is transmitted. As soon as the data is transferred into the shift register, the UCTXIFG is set again. After the data is acknowledged by the master, the next data byte written into UCBxTXBUF is transmitted or, if the buffer is empty, the bus is stalled during the acknowledge cycle by holding SCL low until new data is written into UCBxTXBUF. If the master sends a NACK succeeded by a STOP condition, the UCSTPIFG flag is set. If the NACK is succeeded by a repeated START condition, the USCI I²C state machine returns to its address-reception state.

Figure 21-9 shows the slave transmitter operation.

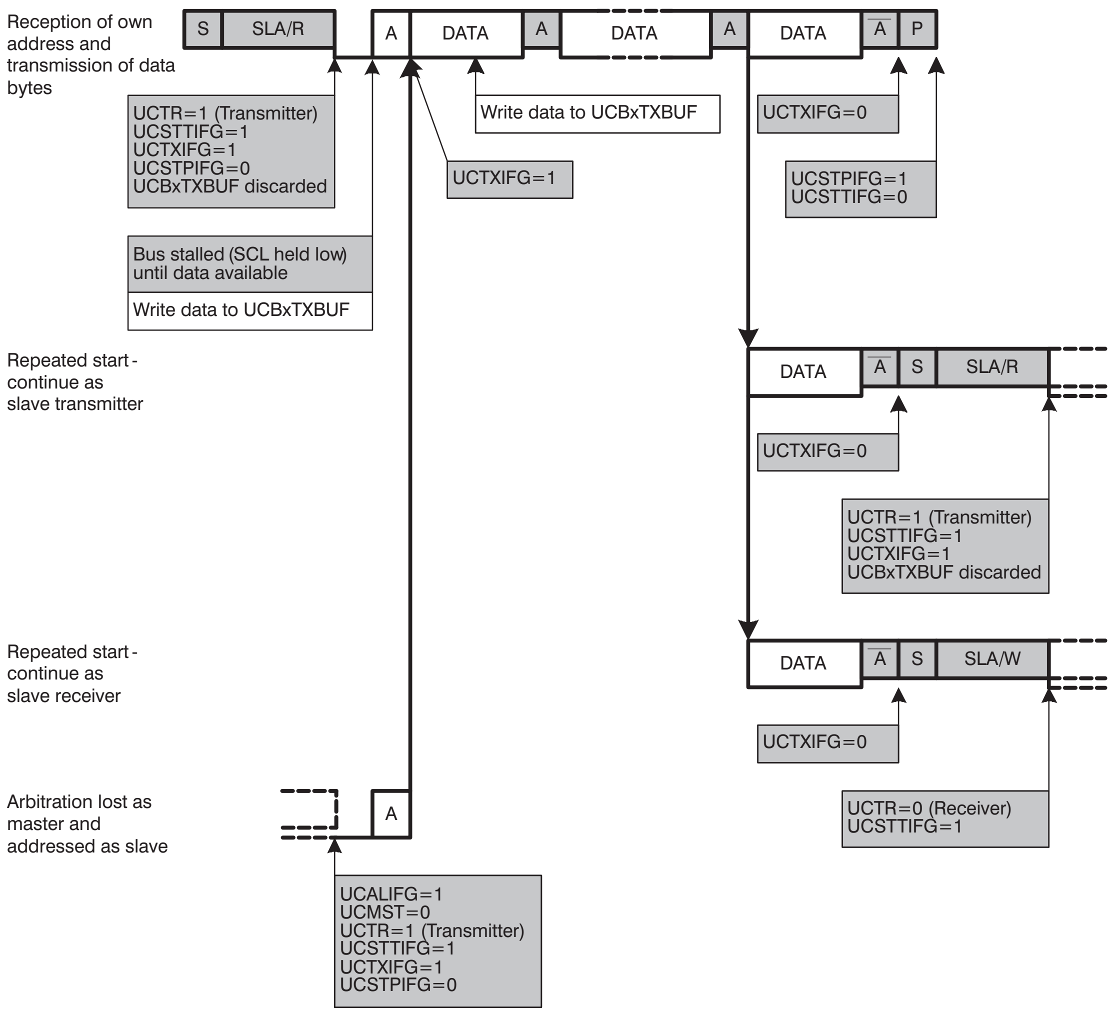


Figure 21-9. I²C Slave Transmitter Mode

I²C Slave Receiver Mode

Slave receiver mode is entered when the slave address transmitted by the master is identical to its own address and a cleared R/$\overline{W}$ bit is received. In slave receiver mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it can hold SCL low if intervention of the CPU is required after a byte has been received.

If the slave should receive data from the master, the USCI module is automatically configured as a receiver and UCTR is cleared. After the first data byte is received, the receive interrupt flag UCRXIFG is set. The USCI module automatically acknowledges the received data and can receive the next data byte.

If the previous data was not read from the receive buffer UCBxRXBUF at the end of a reception, the bus is stalled by holding SCL low. As soon as UCBxRXBUF is read, the new data is transferred into UCBxRXBUF, an acknowledge is sent to the master, and the next data can be received.

Setting the UCTXNACK bit causes a NACK to be transmitted to the master during the next acknowledgment cycle. A NACK is sent even if UCBxRXBUF is not ready to receive the latest data. If the UCTXNACK bit is set while SCL is held low, the bus is released, a NACK is transmitted immediately, and UCBxRXBUF is loaded with the last received data. Because the previous data was not read, that data is lost. To avoid loss of data, the UCBxRXBUF must be read before UCTXNACK is set**.**

When the master generates a STOP condition, the UCSTPIFG flag is set.

If the master generates a repeated START condition, the USCI I²C state machine returns to its address reception state.

Figure 21-10 shows the the I²C slave receiver operation.

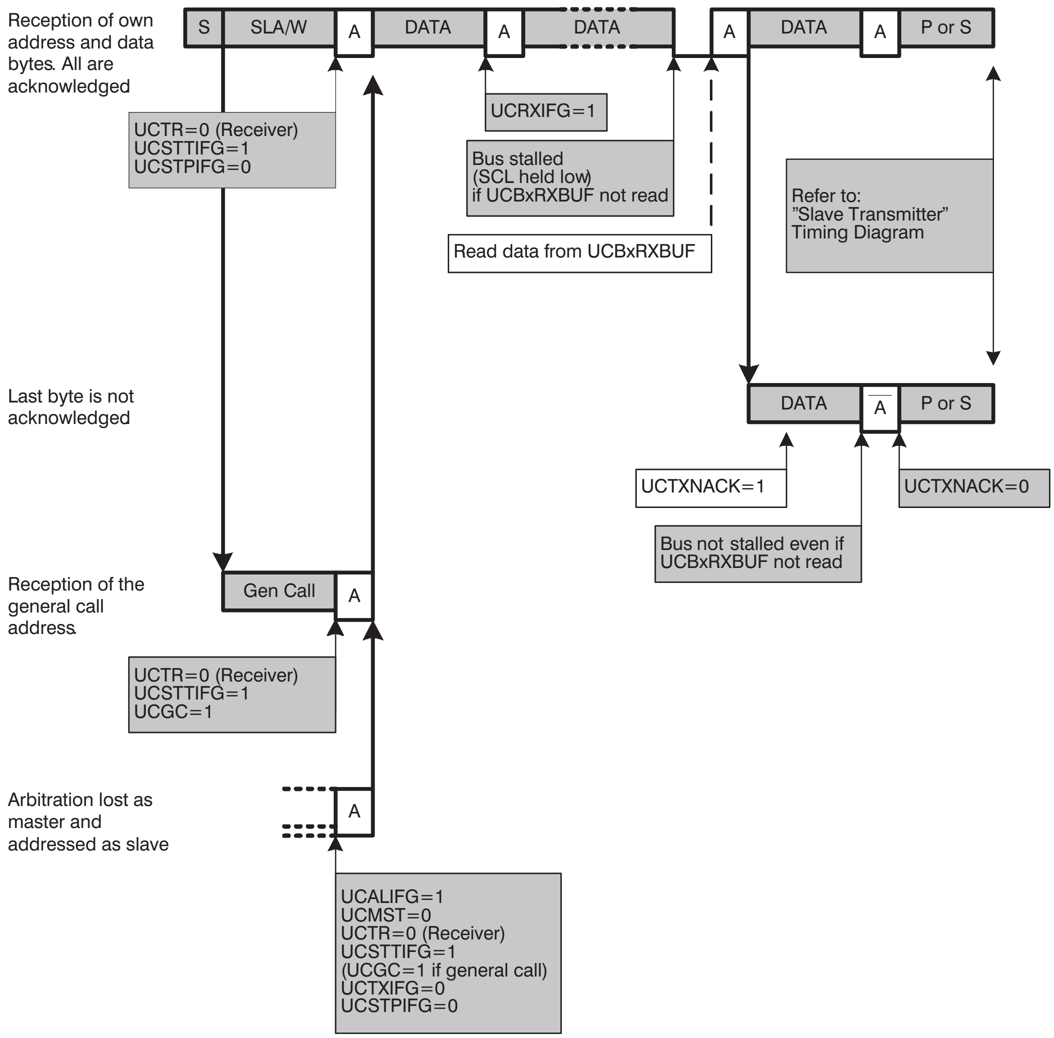


Figure 21-10. I²C Slave Receiver Mode

I²C Slave 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCA10 = 1 and is as shown in Figure 21-11. In 10-bit addressing mode, the slave is in receive mode after the full address is received. The USCI module indicates this by setting the UCSTTIFG flag while the UCTR bit is cleared. To switch the slave into transmitter mode, the master sends a repeated START condition together with the first byte of the address but with the R/$\overline{W}$ bit set. This sets the UCSTTIFG flag if it was previously cleared by software, and the USCI modules switches to transmitter mode with UCTR = 1.

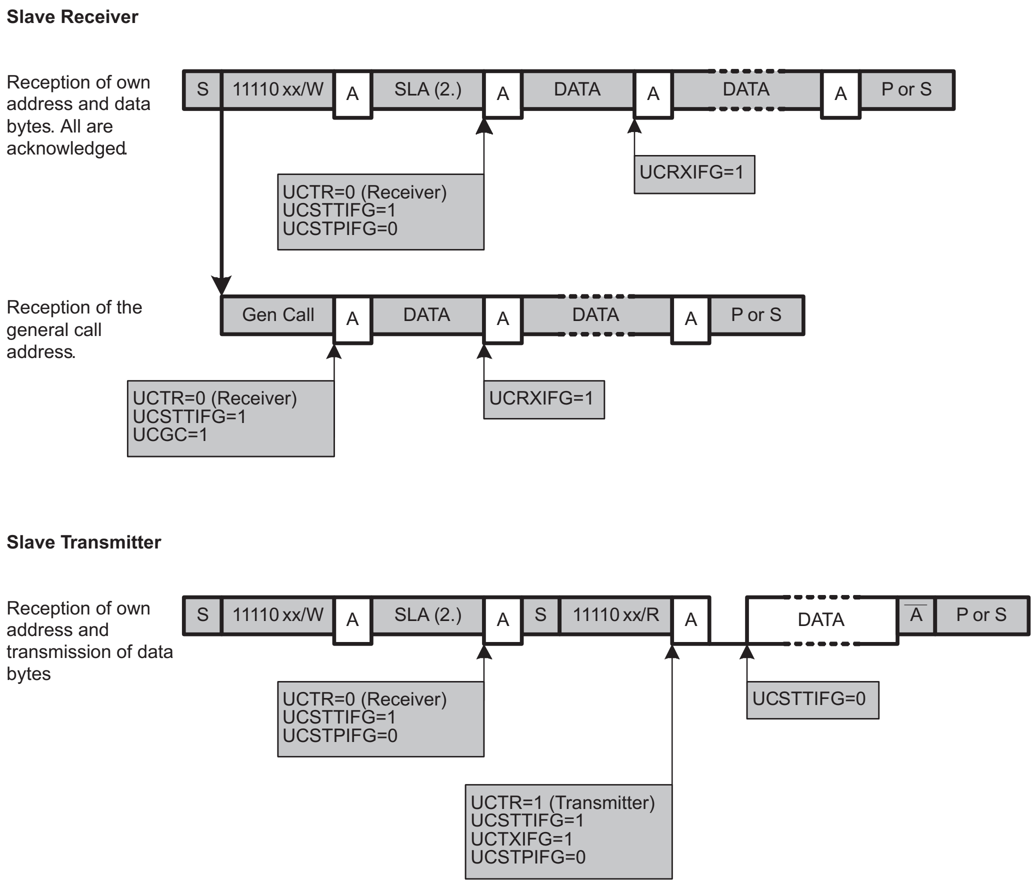


Figure 21-11. I²C Slave 10-Bit Addressing Mode

21.3.4.2 Master Mode

The USCI module is configured as an I²C master by selecting the I²C mode with UCMODEx = 11 and UCSYNC = 1 and setting the UCMST bit. When the master is part of a multi-master system, UCMM must be set and its own address must be programmed into the UCBxI2COA register. When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the USCI module responds to a general call.

I^2C Master Transmitter Mode

After initialization, master transmitter mode is initiated by writing the desired slave address to the UCBxI2CSA register, selecting the size of the slave address with the UCSLA10 bit, setting UCTR for transmitter mode, and setting UCTXSTT to generate a START condition.

The USCI module checks if the bus is available, generates the START condition, and transmits the slave address. The UCTXIFG bit is set when the START condition is generated and the first data to be transmitted can be written into UCBxTXBUF. As soon as the slave acknowledges the address, the UCTXSTT bit is cleared.

NOTE: Handling of TXIFG in a multi-master system

In a multi-master system (UCMM =1), if the bus is unavailable, the USCI module waits and checks for bus release. Bus unavailability can occur even after the UCTXSTT bit has been set. While waiting for the bus to become available, the USCI may update the TXIFG based on SCL clock line activity. Checking the UCTXSTT bit to verify if the START condition has been sent ensures that the TXIFG is being serviced correctly.

The data written into UCBxTXBUF is transmitted if arbitration is not lost during transmission of the slave address. UCTXIFG is set again as soon as the data is transferred from the buffer into the shift register. If there is no data loaded to UCBxTXBUF before the acknowledge cycle, the bus is held during the acknowledge cycle with SCL low until data is written into UCBxTXBUF. Data is transmitted or the bus is held, as long as the UCTXSTP bit or UCTXSTT bit is not set.

Setting UCTXSTP generates a STOP condition after the next acknowledge from the slave. If UCTXSTP is set during the transmission of the slave's address or while the USCI module waits for data to be written into UCBxTXBUF, a STOP condition is generated, even if no data was transmitted to the slave. When transmitting a single byte of data, the UCTXSTP bit must be set while the byte is being transmitted or anytime after transmission begins, without writing new data into UCBxTXBUF. Otherwise, only the address is transmitted. When the data is transferred from the buffer to the shift register, UCTXIFG is set, indicating data transmission has begun, and the UCTXSTP bit may be set.

Setting UCTXSTT generates a repeated START condition. In this case, UCTR may be set or cleared to configure transmitter or receiver, and a different slave address may be written into UCBxI2CSA if desired.

If the slave does not acknowledge the transmitted data, the not-acknowledge interrupt flag UCNACKIFG is set. The master must react with either a STOP condition or a repeated START condition. If data was already written into UCBxTXBUF, it is discarded. If this data should be transmitted after a repeated START, it must be written into UCBxTXBUF again. Any set UCTXSTT is also discarded. To trigger a repeated START, UCTXSTT must be set again.

Figure 21-12 shows the I^2C master transmitter operation.

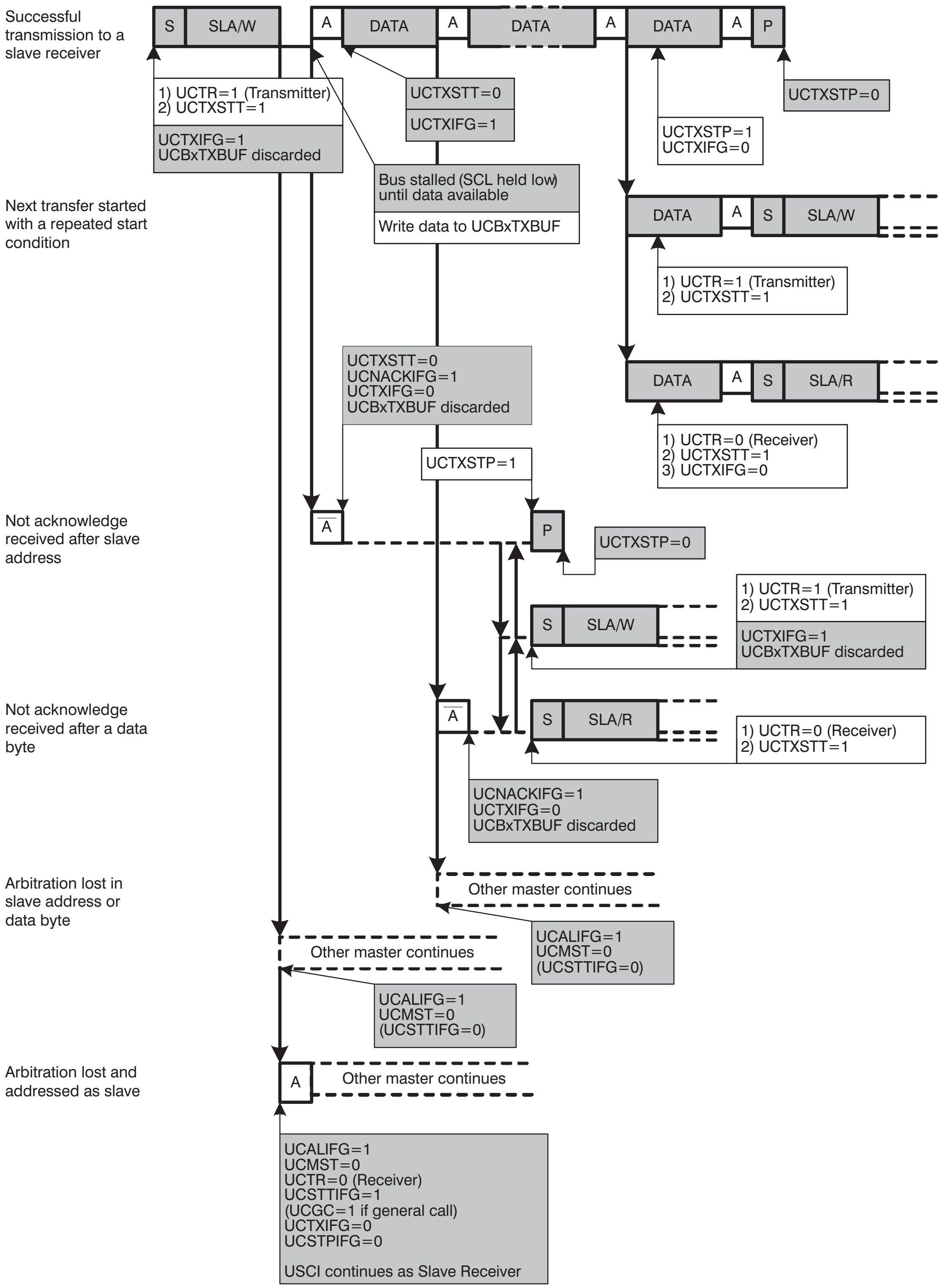


Figure 21-12. I²C Master Transmitter Mode

I²C Master Receiver Mode

After initialization, master receiver mode is initiated by writing the desired slave address to the UCBxI2CSA register, selecting the size of the slave address with the UCSLA10 bit, clearing UCTR for receiver mode, and setting UCTXSTT to generate a START condition.

The USCI module checks if the bus is available, generates the START condition, and transmits the slave address. As soon as the slave acknowledges the address, the UCTXSTT bit is cleared.

After the acknowledge of the address from the slave, the first data byte from the slave is received and acknowledged and the UCRXIFG flag is set. Data is received from the slave, as long as UCTXSTP or UCTXSTT is not set. If UCBxRXBUF is not read, the master holds the bus during reception of the last data bit and until the UCBxRXBUF is read.

If the slave does not acknowledge the transmitted address, the not-acknowledge interrupt flag UCNACKIFG is set. The master must react with either a STOP condition or a repeated START condition.

Setting the UCTXSTP bit generates a STOP condition. After setting UCTXSTP, a NACK followed by a STOP condition is generated after reception of the data from the slave, or immediately if the USCI module is currently waiting for UCBxRXBUF to be read.

If a master wants to receive a single byte only, the UCTXSTP bit must be set while the byte is being received. For this case, the UCTXSTT may be polled to determine when it is cleared:

```
           BIS.B   #UCTXSTT, &UCB0CTL1  ;Transmit START cond.
POLL_STT   BIT.B   #UCTXSTT, &UCB0CTL1  ;Poll UCTXSTT bit
           JC      POLL_STT             ;When cleared,
           BIS.B   #UCTXSTP, &UCB0CTL1  ;transmit STOP cond.
```

Setting UCTXSTT generates a repeated START condition. In this case, UCTR may be set or cleared to configure transmitter or receiver, and a different slave address may be written into UCBxI2CSA if desired.

Figure 21-13 shows the I²C master receiver operation.

NOTE: Consecutive master transactions without repeated START

When performing multiple consecutive I²C master transactions without the repeated START feature, the current transaction must be completed before the next one is initiated. This can be done by ensuring that the transmit STOP condition flag UCTXSTP is cleared before the next I²C transaction is initiated with setting UCTXSTT = 1. Otherwise, the current transaction might be affected.

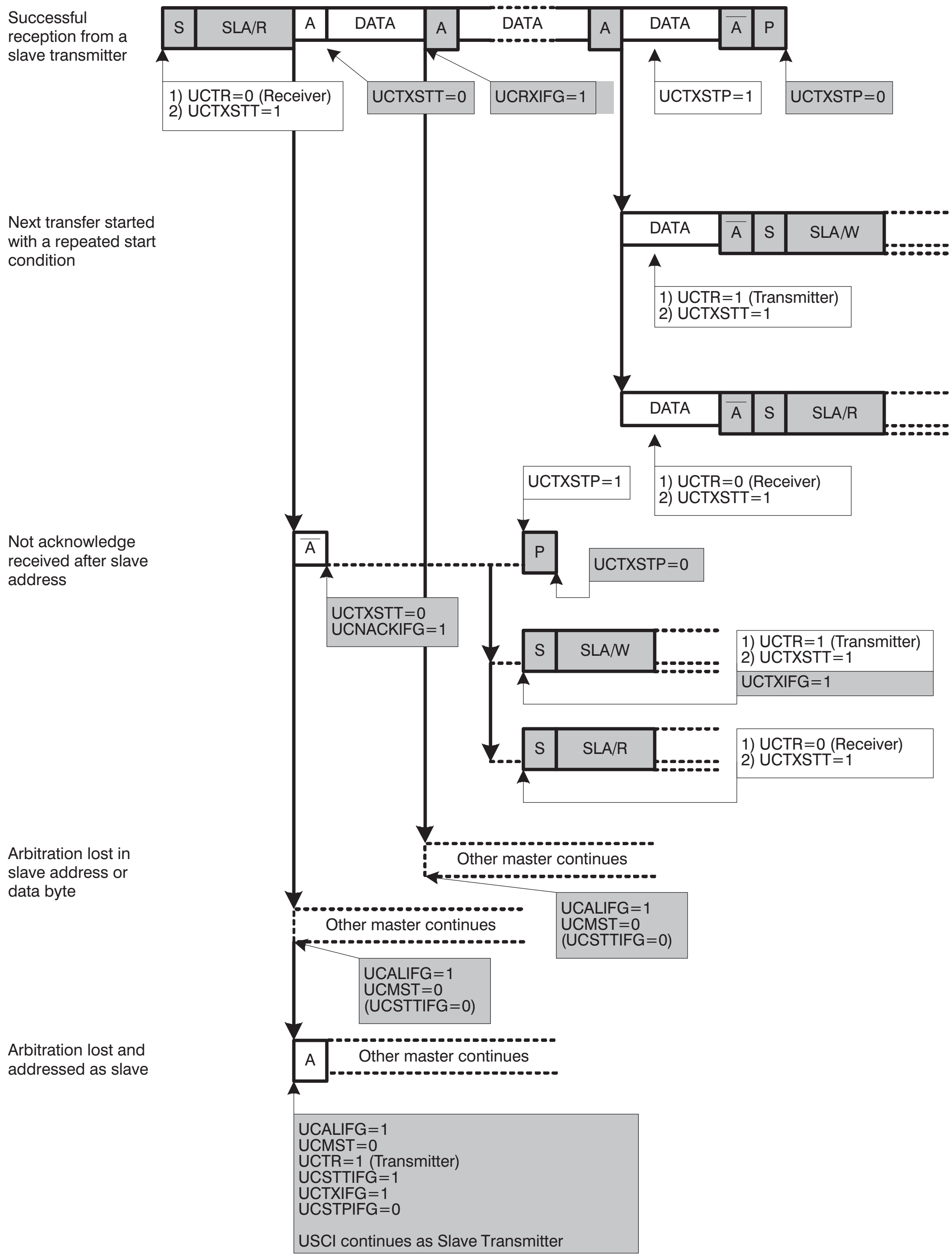


Figure 21-13. I²C Master Receiver Mode

I²C Master 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCSLA10 = 1 and is shown in Figure 21-14.

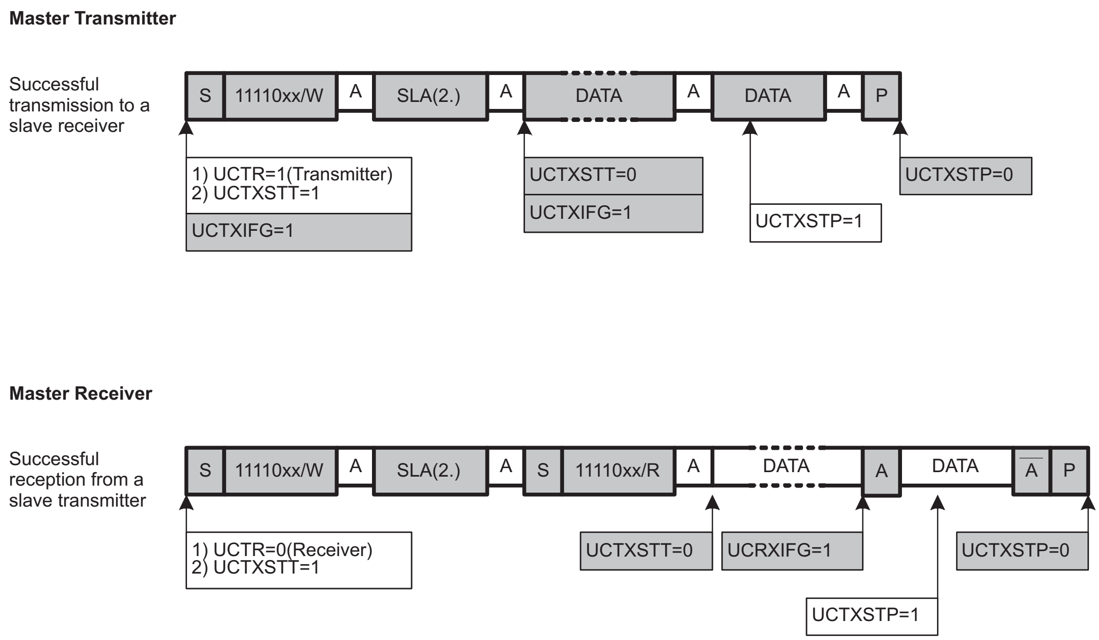


Figure 21-14. I²C Master 10-Bit Addressing Mode

21.3.4.3 Arbitration

If two or more master transmitters simultaneously start a transmission on the bus, an arbitration procedure is invoked. Figure 21-15 shows the arbitration procedure between two devices. The arbitration procedure uses the data presented on SDA by the competing transmitters. The first master transmitter that generates a logic high is overruled by the opposing master generating a logic low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. The master transmitter that lost arbitration switches to the slave receiver mode and sets the arbitration lost flag UCALIFG. If two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

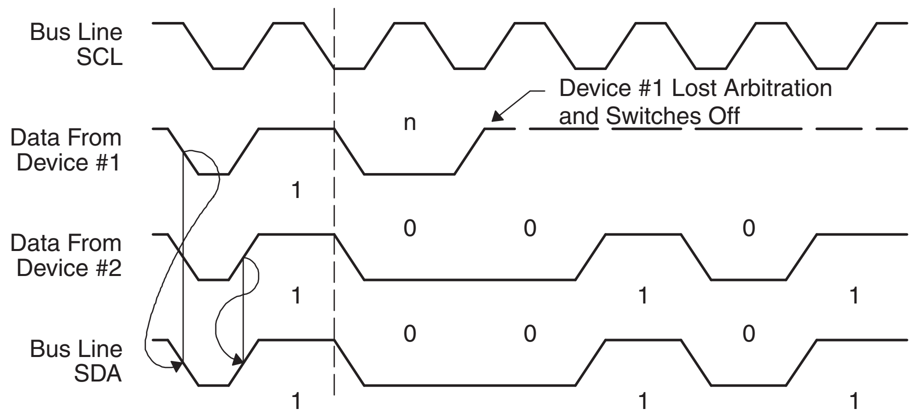


Figure 21-15. Arbitration Procedure Between Two Master Transmitters

If the arbitration procedure is in progress when a repeated START condition or STOP condition is transmitted on SDA, the master transmitters involved in arbitration must send the repeated START condition or STOP condition at the same position in the format frame. Arbitration is not allowed between:

- A repeated START condition and a data bit
- A STOP condition and a data bit
- A repeated START condition and a STOP condition

21.3.5 I²C Clock Generation and Synchronization

The I²C clock SCL is provided by the master on the I²C bus. When the USCI is in master mode, BITCLK is provided by the USCI bit clock generator and the clock source is selected with the UCSSELx bits. In slave mode, the bit clock generator is not used and the UCSSELx bits are don't care.

The 16-bit value of UCBRx in registers UCBxBR1 and UCBxBR0 is the division factor of the USCI clock source, BRCLK. The maximum bit clock that can be used in single master mode is $f_{BRCLK}/4$. In multi-master mode, the maximum bit clock is $f_{BRCLK}/8$. The BITCLK frequency is given by:

$f_{BitClock} = f_{BRCLK}/UCBRx$

The minimum high and low periods of the generated SCL are:

$t_{LOW,MIN} = t_{HIGH,MIN} = (UCBRx/2)/f_{BRCLK}$ when UCBRx is even

$t_{LOW,MIN} = t_{HIGH,MIN} = (UCBRx – 1/2)/f_{BRCLK}$ when UCBRx is odd

The USCI clock source frequency and the prescaler setting UCBRx must to be chosen such that the minimum low and high period times of the I²C specification are met.

During the arbitration procedure the clocks from the different masters must be synchronized. A device that first generates a low period on SCL overrules the other devices, forcing them to start their own low periods. SCL is then held low by the device with the longest low period. The other devices must wait for SCL to be released before starting their high periods. Figure 21-16 shows the clock synchronization. This allows a slow slave to slow down a fast master.

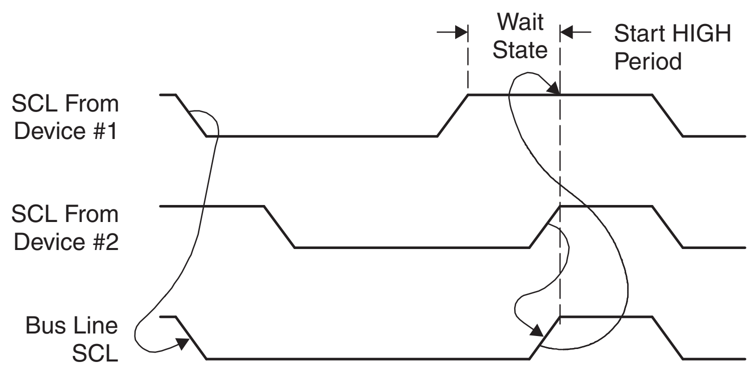


Figure 21-16. Synchronization of Two I²C Clock Generators During Arbitration

21.3.5.1 Clock Stretching

The USCI module supports clock stretching and also makes use of this feature as described in the Operation Mode sections.

The UCSCLLOW bit can be used to observe if another device pulls SCL low while the USCI module already released SCL due to the following conditions:

- USCI is acting as master and a connected slave drives SCL low.
- USCI is acting as master and another master drives SCL low during arbitration.

The UCSCLLOW bit is also active if the USCI holds SCL low because it is waiting as transmitter for data being written into UCBxTXBUF or as receiver for the data being read from UCBxRXBUF.

The UCSCLLOW bit might get set for a short time with each rising SCL edge because the logic observes the external SCL and compares it to the internally generated SCL.

21.3.6 Using the USCI Module in I^2C Mode With Low-Power Modes

The USCI module provides automatic clock activation for use with low-power modes. When the USCI clock source is inactive because the device is in a low-power mode, the USCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the USCI module returns to its idle condition. After the USCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

In I^2C slave mode, no internal clock source is required because the clock is provided by the external master. It is possible to operate the USCI in I^2C slave mode while the device is in LPM4 and all internal clock sources are disabled. The receive or transmit interrupts can wake up the CPU from any low-power mode.

21.3.7 USCI Interrupts in I^2C Mode

The USCI has only one interrupt vector that is shared for transmission, reception, and the state change. USCI_Ax and USC_Bx do not share the same interrupt vector.

Each interrupt flag has its own interrupt enable bit. When an interrupt is enabled and the GIE bit is set, the interrupt flag generates an interrupt request. DMA transfers are controlled by the UCTXIFG and UCRXIFG flags on devices with a DMA controller.

21.3.7.1 I^2C Transmit Interrupt Operation

The UCTXIFG interrupt flag is set by the transmitter to indicate that UCBxTXBUF is ready to accept another character. An interrupt request is generated if UCTXIE and GIE are also set. UCTXIFG is automatically reset if a character is written to UCBxTXBUF or if a NACK is received. UCTXIFG is set when UCSWRST = 1 and the I^2C mode is selected. UCTXIE is reset after a PUC or when UCSWRST = 1.

21.3.7.2 I^2C Receive Interrupt Operation

The UCRXIFG interrupt flag is set when a character is received and loaded into UCBxRXBUF. An interrupt request is generated if UCRXIE and GIE are also set. UCRXIFG and UCRXIE are reset after a PUC signal or when UCSWRST = 1. UCRXIFG is automatically reset when UCxRXBUF is read.

21.3.7.3 I²C State Change Interrupt Operation

Table 21-1 describes the I²C state change interrupt flags.

Table 21-1. I²C State Change Interrupt Flags

Interrupt Flag	Interrupt Condition
UCALIFG	Arbitration-lost. Arbitration can be lost when two or more transmitters start a transmission simultaneously, or when the USCI operates as master but is addressed as a slave by another master in the system. The UCALIFG flag is set when arbitration is lost. When UCALIFG is set, the UCMST bit is cleared and the I²C controller becomes a slave.
UCNACKIFG	Not-acknowledge interrupt. This flag is set when an acknowledge is expected but is not received. UCNACKIFG is automatically cleared when a START condition is received.
UCSTTIFG	START condition detected interrupt. This flag is set when the I²C module detects a START condition together with its own address while in slave mode. UCSTTIFG is used in slave mode only and is automatically cleared when a STOP condition is received.
UCSTPIFG	STOP condition detected interrupt. This flag is set when the I²C module detects a STOP condition while in slave mode. UCSTPIFG is used in slave mode only and is automatically cleared when a START condition is received.

21.3.7.4 UCBxIV, Interrupt Vector Generator

The USCI interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCBxIV is used to determine which flag requested an interrupt. The highest-priority enabled interrupt generates a number in the UCBxIV register that can be evaluated or added to the PC to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCBxIV value.

Any access, read or write, of the UCBxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

UCBxIV Software Example

The following software example shows the recommended use of UCBxIV. The UCBxIV value is added to the PC to automatically jump to the appropriate routine. The example is given for USCI_B0.

```
USCI_I2C_ISR
         ADD      &UCB0IV, PC  ; Add offset to jump table
         RETI                  ; Vector 0: No interrupt
         JMP      ALIFG_ISR    ; Vector 2: ALIFG
         JMP      NACKIFG_ISR  ; Vector 4: NACKIFG
         JMP      STTIFG_ISR   ; Vector 6: STTIFG
         JMP      STPIFG_ISR   ; Vector 8: STPIFG
         JMP      RXIFG_ISR    ; Vector 10: RXIFG
TXIFG_ISR                      ; Vector 12
         ...                   ; Task starts here
         RETI                  ; Return
ALIFG_ISR                      ; Vector 2
         ...                   ; Task starts here
         RETI                  ; Return
NACKIFG_ISR                    ; Vector 4
         ...                   ; Task starts here
         RETI                  ; Return
STTIFG_ISR                     ; Vector 6
         ...                   ; Task starts here
         RETI                  ; Return
STPIFG_ISR                     ; Vector 8
         ...                   ; Task starts here
         RETI                  ; Return
RXIFG_ISR                      ; Vector 10
         ...                   ; Task starts here
         RETI                  ; Return
```

21.4 USCI Registers– I2C Mode

The USCI registers applicable in I2C mode are listed in Table 21-2. The base address can be found in the device-specific data sheet. The address offsets are listed in Table 21-2.

Table 21-2. USCI_Bx Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
USCI_Bx Control Word 0	UCBxCTLW0	Read/write	Word	00h	0101h
USCI_Bx Control 1	UCBxCTL1	Read/write	Byte	00h	01h
USCI_Bx Control 0	UCBxCTL0	Read/write	Byte	01h	01h
USCI_Bx Bit Rate Control Word	UCBxBRW	Read/write	Word	06h	0000h
USCI_Bx Bit Rate Control 0	UCBxBR0	Read/write	Byte	06h	00h
USCI_Bx Bit Rate Control 1	UCBxBR1	Read/write	Byte	07h	00h
USCI_Bx Status	UCBxSTAT	Read/write	Byte	0Ah	00h
Reserved - reads zero		Read	Byte	0Bh	00h
USCI_Bx Receive Buffer	UCBxRXBUF	Read/write	Byte	0Ch	00h
Reserved - reads zero		Read	Byte	0Dh	00h
USCI_Bx Transmit Buffer	UCBxTXBUF	Read/write	Byte	0Eh	00h
Reserved - reads zero		Read	Byte	0Fh	00h
USCI_Bx I2C Own Address	UCBxI2COA	Read/write	Word	10h	0000h
USCI_Bx I2C Slave Address	UCBxI2CSA	Read/write	Word	12h	0000h
USCI_Bx Interrupt Control	UCBxICTL	Read/write	Word	1Ch	0200h
USCI_Bx Interrupt Enable	UCBxIE	Read/write	Byte	1Ch	00h
USCI_Bx Interrupt Flag	UCBxIFG	Read/write	Byte	1Dh	02h
USCI_Bx Interrupt Vector	UCBxIV	Read	Word	1Eh	0000h

USCI_Bx Control Register 0 (UCBxCTL0)

7	6	5	4	3	2	1	0
UCA10	UCSLA10	UCMM	Unused	UCMST	UCMODEx=11		UCSYNC=1
R/W-0							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-1

UCA10 Bit 7 Own addressing mode select
- 0 Own address is a 7-bit address.
- 1 Own address is a 10-bit address.

UCSLA10 Bit 6 Slave addressing mode select
- 0 Address slave with 7-bit address
- 1 Address slave with 10-bit address

UCMM Bit 5 Multi-master environment select
- 0 Single master environment. There is no other master in the system. The address compare unit is disabled.
- 1 Multi-master environment

Unused Bit 4 Unused

UCMST Bit 3 Master mode select. When a master loses arbitration in a multi-master environment (UCMM = 1), the UCMST bit is automatically cleared and the module acts as slave.
- 0 Slave mode
- 1 Master mode

UCMODEx Bits 2-1 USCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1.
- 00 3-pin SPI
- 01 4-pin SPI (master/slave enabled if STE = 1)
- 10 4-pin SPI (master/slave enabled if STE = 0)
- 11 I²C mode

UCSYNC Bit 0 Synchronous mode enable
- 0 Asynchronous mode
- 1 Synchronous mode

USCI_Bx Control Register 1 (UCBxCTL1)

7	6	5	4	3	2	1	0
UCSSELx		**Unused**	**UCTR**	**UCTXNACK**	**UCTXSTP**	**UCTXSTT**	**UCSWRST**
rw-0	rw-0	r0	rw-0	rw-0	rw-0	rw-0	rw-1

UCSSELx	Bits 7-6	USCI clock source select. These bits select the BRCLK source clock. 00 UCLKI 01 ACLK 10 SMCLK 11 SMCLK
Unused	Bit 5	Unused
UCTR	Bit 4	Transmitter/receiver 0 Receiver 1 Transmitter
UCTXNACK	Bit 3	Transmit a NACK. UCTXNACK is automatically cleared after a NACK is transmitted. 0 Acknowledge normally 1 Generate NACK
UCTXSTP	Bit 2	Transmit STOP condition in master mode. Ignored in slave mode. In master receiver mode, the STOP condition is preceded by a NACK. UCTXSTP is automatically cleared after STOP is generated. 0 No STOP generated 1 Generate STOP
UCTXSTT	Bit 1	Transmit START condition in master mode. Ignored in slave mode. In master receiver mode, a repeated START condition is preceded by a NACK. UCTXSTT is automatically cleared after START condition and address information is transmitted. Ignored in slave mode. 0 Do not generate START condition 1 Generate START condition
UCSWRST	Bit 0	Software reset enable 0 Disabled. USCI reset released for operation. 1 Enabled. USCI logic held in reset state.

USCI_Bx Baud Rate Control Register 0 (UCBxBR0)

7	6	5	4	3	2	1	0
UCBRx - low byte							
rw	rw	rw	rw	rw	rw	rw	rw

USCI_Bx Baud Rate Control Register 1 (UCBxBR1)

7	6	5	4	3	2	1	0
UCBRx - high byte							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx	Bits 7-0	Bit clock prescaler. The 16-bit value of (UCxxBR0 + UCxxBR1 × 256) forms the prescaler value UCBRx.

USCI_Bx Status Register (UCBxSTAT)

7	6	5	4	3	2	1	0
Unused	**UCSCLLOW**	**UCGC**	**UCBBUSY**	**Unused**			
rw-0	r-0	rw-0	r-0	r0	r0	r0	r0

Unused	Bit 7	Unused
UCSCLLOW	Bit 6	SCL low 0 SCL is not held low. 1 SCL is held low.
UCGC	Bit 5	General call address received. UCGC is automatically cleared when a START condition is received. 0 No general call address received 1 General call address received
UCBBUSY	Bit 4	Bus busy 0 Bus inactive 1 Bus busy
Unused	Bits 3-0	Unused

USCI_Bx Receive Buffer Register (UCBxRXBUF)

7	6	5	4	3	2	1	0
UCRXBUFx							
r	r	r	r	r	r	r	r

UCRXBUFx	Bits 7-0	The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCBxRXBUF resets UCRXIFG.

USCI_Bx Transmit Buffer Register (UCBxTXBUF)

7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCTXBUFx	Bits 7-0	The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted. Writing to the transmit data buffer clears UCTXIFG.

USCIBx I²C Own Address Register (UCBxI2COA)

15	14	13	12	11	10	9	8
UCGCEN	**0**	**0**	**0**	**0**	**0**	**I2COAx**	
rw-0	r0	r0	r0	r0	r0	rw-0	rw-0

7	6	5	4	3	2	1	0
I2COAx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCGCEN	Bit 15	General call response enable 0 Do not respond to a general call 1 Respond to a general call
I2COAx	Bits 9-0	I²C own address. The I2COAx bits contain the local address of the USCI_Bx I²C controller. The address is right justified. In 7-bit addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit addressing mode, bit 9 is the MSB.

USCI_Bx I²C Slave Address Register (UCBxI2CSA)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**I2CSAx**	
r0	r0	r0	r0	r0	r0	rw-0	rw-0

7	6	5	4	3	2	1	0
I2CSAx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

I2CSAx	Bits 9-0	I²C slave address. The I2CSAx bits contain the slave address of the external device to be addressed by the USCI_Bx module. It is only used in master mode. The address is right justified. In 7-bit slave addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit slave addressing mode, bit 9 is the MSB.

USCI_Bx I^2C Interrupt Enable Register (UCBxIE)

7	6	5	4	3	2	1	0
Reserved		UCNACKIE	UCALIE	UCSTPIE	UCSTTIE	UCTXIE	UCRXIE
r-0	r-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Reserved	Bits 7-6	Reserved
UCNACKIE	Bit 5	Not-acknowledge interrupt enable 0 Interrupt disabled 1 Interrupt enabled
UCALIE	Bit 4	Arbitration lost interrupt enable 0 Interrupt disabled 1 Interrupt enabled
UCSTPIE	Bit 3	STOP condition interrupt enable 0 Interrupt disabled 1 Interrupt enabled
UCSTTIE	Bit 2	START condition interrupt enable 0 Interrupt disabled 1 Interrupt enabled
UCTXIE	Bit 1	Transmit interrupt enable 0 Interrupt disabled 1 Interrupt enabled
UCRXIE	Bit 0	Receive interrupt enable 0 Interrupt disabled 1 Interrupt enabled

USCI_Bx I^2C Interrupt Flag Register (UCBxIFG)

7	6	5	4	3	2	1	0
Reserved		UCNACKIFG	UCALIFG	UCSTPIFG	UCSTTIFG	UCTXIFG	UCRXIFG
r-0	r-0	rw-0	rw-0	rw-0	rw-0	rw-1	rw-0

Reserved	Bits 7-6	Reserved
UCNACKIFG	Bit 5	Not-acknowledge received interrupt flag. UCNACKIFG is automatically cleared when a START condition is received. 0 No interrupt pending 1 Interrupt pending
UCALIFG	Bit 4	Arbitration lost interrupt flag 0 No interrupt pending 1 Interrupt pending
UCSTPIFG	Bit 3	STOP condition interrupt flag. UCSTPIFG is automatically cleared when a START condition is received. 0 No interrupt pending 1 Interrupt pending
UCSTTIFG	Bit 2	START condition interrupt flag. UCSTTIFG is automatically cleared if a STOP condition is received. 0 No interrupt pending 1 Interrupt pending
UCTXIFG	Bit 1	USCI transmit interrupt flag. UCTXIFG is set when UCBxTXBUF is empty. 0 No interrupt pending 1 Interrupt pending
UCRXIFG	Bit 0	USCI receive interrupt flag. UCRXIFG is set when UCBxRXBUF has received a complete character. 0 No interrupt pending 1 Interrupt pending

USCI_Bx Interrupt Vector Register (UCBxIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**UCIVx**			**0**
r0	r0	r0	r0	r-0	r-0	r-0	r0

UCIVx Bits 15-0 USCI interrupt vector value

UCBxIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	Arbitration lost	UCALIFG	Highest
004h	Not acknowledgement	UCNACKIFG	
006h	Start condition received	UCSTTIFG	
008h	Stop condition received	UCSTPIFG	
00Ah	Data received	UCRXIFG	
00Ch	Transmit buffer empty	UCTXIFG	Lowest

USB Module

This chapter describes the USB module that is available in some devices.

Topic **Page**

22.1 USB Introduction 542
22.2 USB Operation 544
22.3 USB Transfers 555
22.4 Registers 561

22.1 USB Introduction

The features of the USB module include:

- Fully compliant with the USB 2.0 Full-speed specification
  - Full-speed device (12 Mbps) with integrated USB transceiver (PHY)
  - Up to eight input and eight output endpoints
  - Supports control, interrupt, and bulk transfers
  - Supports USB suspend, resume, and remote wakeup
- A power supply system independent from the PMM system
  - Integrated 3.3-V LDO regulator with sufficient output to power entire MSP430 and system circuitry from 5-V VBUS
  - Integrated 1.8-V LDO regulator for PHY and PLL
  - Easily used in either bus-powered or self-powered operation
  - Current-limiting capability on 3.3-V LDO output
  - Autonomous power-up of MSP430 upon arrival of USB power possible (low/no battery condition)
- Internal 48-MHz USB clock
  - Integrated programmable PLL
  - Highly-flexible input clock frequencies for use with lowest-cost crystals
- 1904 bytes of dedicated USB buffer space for endpoints, with fully configurable size to a granularity of eight bytes
- Timestamp generator with 62.5-ns resolution
- When USB is disabled
  - Buffer space is mapped into general RAM, providing additional 2 KB to the system
  - USB interface pins become high-current general purpose I/O pins

---

NOTE: Use of the word *device*

The word *device* is used throughout the chapter. This word can mean one of two things, depending on the context. In a USB context, it means what the USB specification refers to as a device, function, or peripheral; that is, a piece of equipment that can be attached to a USB host or hub. In a semiconductor context, it refers to an integrated circuit such as the MSP430.

To avoid confusion, the term *USB device* in this document refers to the USB-context meaning of the word. The word *device* by itself refers to silicon devices such as the MSP430.

Figure 22-1 shows a block diagram of the USB module.

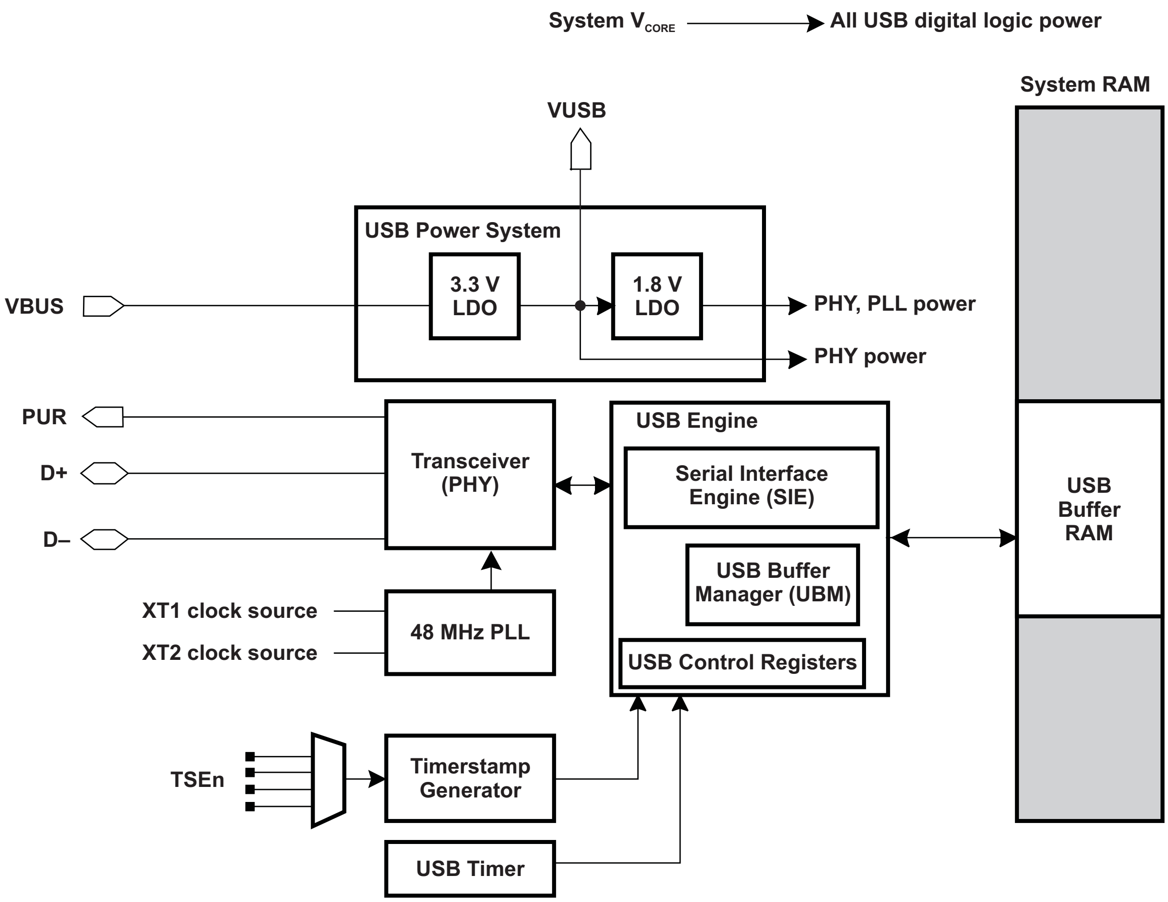


Figure 22-1. USB Block Diagram

22.2 USB Operation

The USB module is a comprehensive, full-speed USB device compliant with the USB 2.0 specification.

The USB engine coordinates all USB-related traffic. It consists of the USB SIE (serial interface engine) and USB Buffer Manager (UBM). All traffic received on the USB receive path is de-serialized and placed into receive buffers in the USB buffer RAM. Data in the buffer RAM marked ‘ready to be sent' are serialized into packets and sent to the USB host.

The USB engine requires an accurate 48-MHz clock to sample the incoming data stream. This is generated by a PLL that is fed from one of the system oscillators (XT1/XT2). A crystal greater than 1.5 MHz is required. However, the PLL is very flexible and can adapt to a wide range of frequencies, allowing design to the most cost-effective crystal frequency.

NOTE: The reference clock to the PLL depends on the device configuration. On devices that contain the optional XT2, the reference clock to the PLL is XT2CLK, regardless if XT1 is available. If the device has only XT1, then the reference will be XT1CLK. Please refer to the device specific datasheet for clock sources available.

The USB buffer memory is where data is exchanged between the USB interface and the application software. It is also where the usage of endpoints 1 to 7 are defined. This buffer memory is implemented such that it can be easily accessed like RAM by the CPU and/or DMA while USB module is not in suspend condition.

22.2.1 USB Transceiver (PHY)

The physical layer interface (USB transceiver) is a differential line driver directly powered from VUSB (3.3 V). The line driver is connected to the DP/DM pins, which form the signaling mechanism of the USB interface.

When the PUSEL bit is set, DP/DM are configured to function as USB drivers controlled by the USB core logic. When the bit is cleared, these two pins become "Port U", which is a pair of high-current general purpose I/O pins. In this case, the pins are controlled by the UPCR register. Port U is powered from the VUSB rail, separate from the main device DVCC. If these pins are to be used, whether for USB or general purpose use, it is necessary that VUSB be properly powered – either from the internal regulators or an external source.

22.2.1.1 D+ Pullup Via PUR Pin

When a full-speed USB device is attached to a USB host, it must pull up the D+ line (DP pin) in order for the host to recognize its presence. The MSP430 USB module implements this with a software-controlled pin that activates a pullup resistor. The bit that controls this function is PUR_EN. If software control is not desired, the pullup can be connected directly to VUSB.

22.2.1.2 Shorts on Damaged Cables and Clamping

USB devices must tolerate connection to a cable that is damaged, such that it has developed shorts on either ground or VBUS. The device should not become damaged by this event, either electrically or physically. To this end, the MSP430 USB power system features a current limitation mechanism that limits the available transceiver current in the event of a short to ground. The transceiver interface itself therefore does not need a current limiting function.

Note that if VUSB is to be powered from a source other than the integrated regulator, the absence of current-limiting in the transceiver means that the external power source must itself be tolerant of this same shorting event, through its own means of current limiting.

22.2.1.3 Port U Control

When PUSEL is cleared, the Port U pins (PU0/PU1, corresponding with DP/DM, respectively) function as general-purpose, high-current I/O pins. PUDIR controls the enable of both outputs residing on the Port U pins. The Port U pins are either both driving out, or both acting as inputs. When configured as inputs, the PUIN0/1 pins can be read to determine the input values. When Port U outputs are enabled, the PUIN0/1 will mirror what is present on the outputs.

When PUDIR is set, both Port U pins function as outputs, controlled by PUOUT0/PUOUT1. When driven high, they use the VUSB rail, and they are capable of a drive current higher than other I/O pins on the device. See the device-specific datasheet for parameters.

By default, PUDIR is cleared. PU0/PU1 therefore become high-impedance when the USB module is disabled.

22.2.2 USB Power System

The USB power system incorporates dual LDO regulators (3.3 V and 1.8 V) that allow the entire MSP430 device to be powered from 5-V VBUS when it is made available from the USB host. Alternatively, the power system can supply power only to the USB module, or it can be unused altogether, as in a fully self-powered device. The block diagram is shown in Figure 22-2.

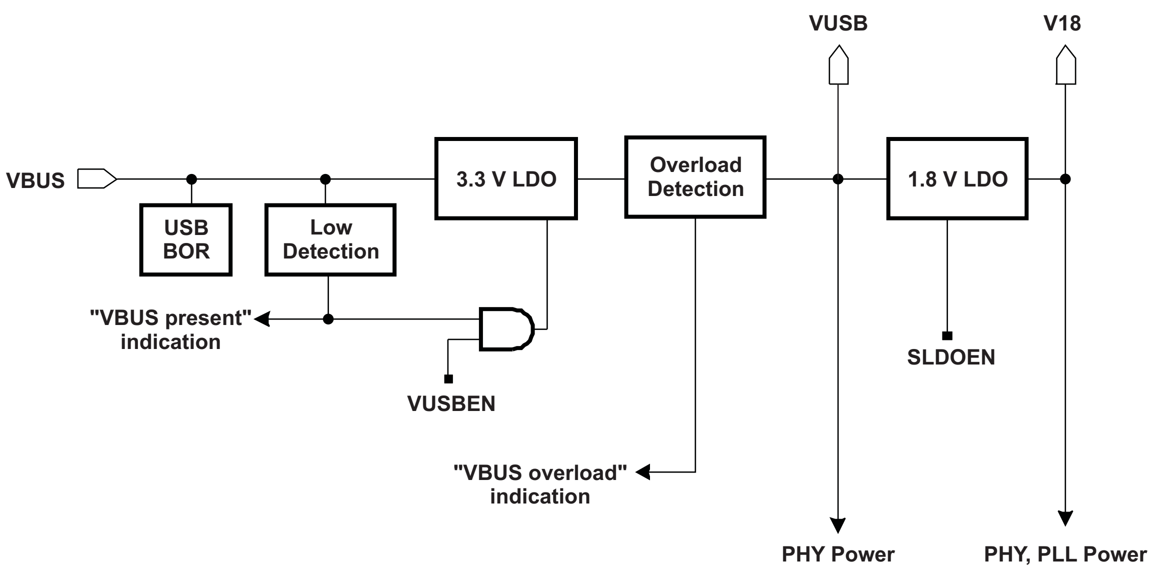


Figure 22-2. USB Power System

The 3.3-V LDO receives 5 V from VBUS and provides power to the transceiver, as well as the VUSB pin. Using this setup prevents the relatively high load of the transceiver and PLL from loading a local system power supply, if used. Thus it is very useful in battery-powered devices.

The 1.8-V LDO receives power from the VUSB pin – which is to be sourced either from the internal 3.3-V LDO or externally – and provides power to the USB PLL and transceiver. The 1.8-V LDO in the USB module is not related to the LDO that resides in the MSP430 Power Management Module (PMM).

The inputs and outputs of the LDOs are shown in Figure 22-2. VBUS, VUSB, and V18 need to be connected to external capacitors. The V18 pin is not intended to source other components in the system, rather it exists solely for the attachment of a load capacitor.

22.2.2.1 Enabling/Disabling

The 3.3-V LDO is enabled/disabled by setting/clearing VUSBEN. Even if enabled, if the voltage on VBUS is detected to be low or nonexistent, the LDO is suspended. When VBUS rises above the USB power brownout level, the LDO reference and low voltage detection become enabled. When VBUS rises further above the launch voltage V_{LAUNCH}, the LDO module becomes enabled (see Figure 22-3).

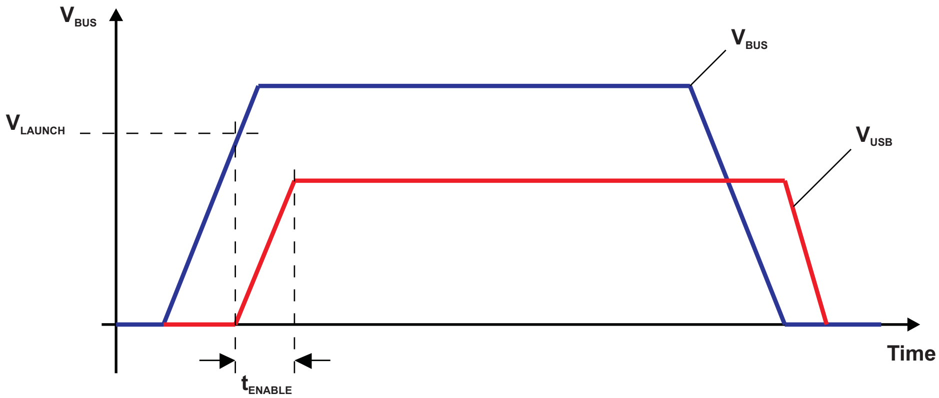


Figure 22-3. USB Power Up/Down Profile

The 1.8-V LDO can be enabled/disabled by setting SLDOEN accordingly. By default, SLDOEN is controlled automatically according to whether power is available on VBUS. This auto-enable feature is controlled by SLDOAON. If providing VUSB from an external source, rather than through the integrated 3.3-V LDO, keep in mind that if 5 V is not present on VBUS, the 1.8-V LDO is not automatically enabled. In this situation, either VBUS much be attached to USB bus power, or the SLDOAON bit must be cleared and SLDOEN set.

It is required that power from the USB cable's VBUS be directed through a Schottky diode prior to entering the VBUS terminal. This prevents current from draining into the cable's VBUS from the LDO input, allowing the MSP430 to tolerate a suspended/unpowered USB cable that remains electrically connected.

22.2.2.2 Powering the Rest of the MSP430 From USB Bus Power via VUSB

The output of the 3.3-V LDO can be used to power the entire MSP430 device, sourcing the DVCC rail. If this is desired, the VUSB and DVCC should be connected externally. Power from the 3.3-V LDO is sourced into DVCC (see Figure 22-4).

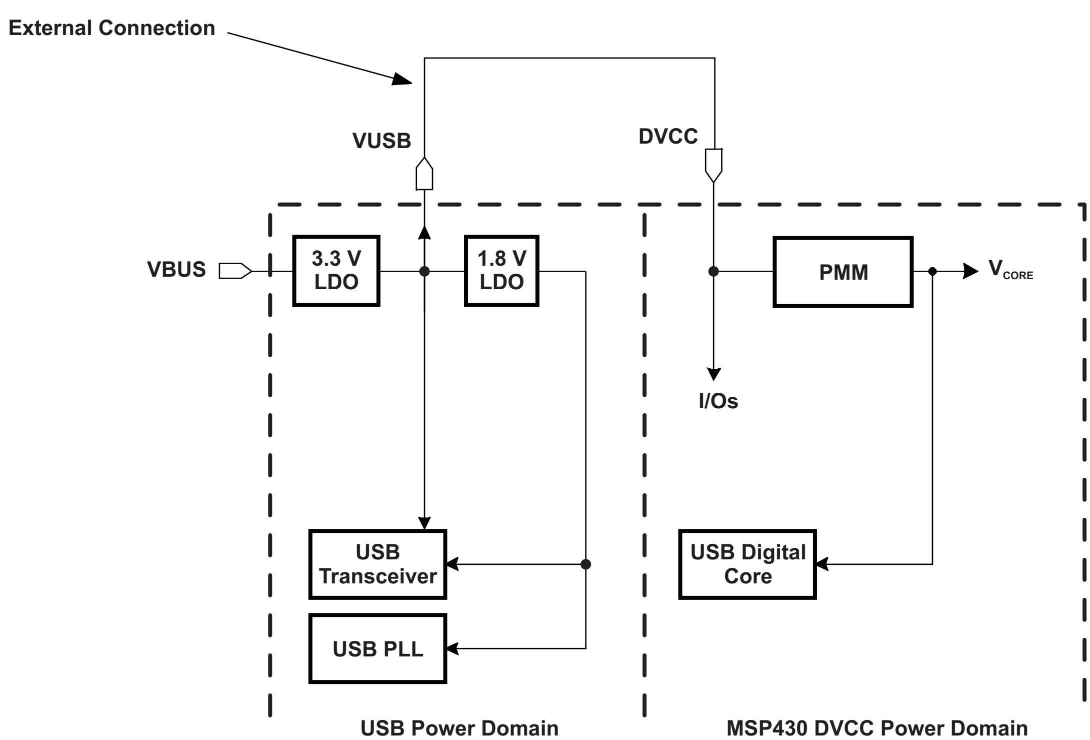


Figure 22-4. Powering Entire MSP430 From VBUS

With this connection made, the MSP430 allows for autonomous power up of the device when VBUS rises above V_{LAUNCH}. If no voltage is present on V_{CORE} – meaning the device is unpowered (or, in LPM5 mode) – then both the 3.3-V and 1.8-V LDOs automatically turn on when VBUS rises above V_{LAUNCH}.

Note that if DVCC is being driven from VUSB in this manner, and if power is available from VUSB, attempting to place the device into LPM5 results in the device immediately re-powering. This is because it re-creates the conditions of the autonomous feature described above (no V_{CORE} but power available on VBUS). The resulting drop of V_{CORE} would cause the system to immediately power up again.

When DVCC is being powered from VUSB, it is up to the user to ensure that the total current being drawn from VBUS stays below I_{DET}.

22.2.2.3 Powering Other Components in the System from VUSB

There is sufficient current capacity available from the 3.3-V LDO to power not only the entire MSP430 but also other components in the system, via the VUSB pin.

If the device is to always be connected to USB, then perhaps no other power system is needed. If it only occasionally connects to USB and is battery-powered otherwise, then sourcing system power via the 3.3-V LDO takes power burden away from the battery. Alternatively, if the battery is rechargeable, the recharging can be driven from VUSB.

22.2.2.4 Current Limitation / Overload Protection

The 3.3-V LDO features current limitation to protect the transceiver during shorted-cable conditions. A short/overload condition – that is, when the output of the LDO becomes current-limited to I_{DET} – is reported to software via the VUOVLIFG flag.

If this event occurs, it means USB operation may become unreliable, due to insufficient power supply. As a result, software may wish to cease USB operation. If the OVLAOFF bit is set, USB operation is automatically terminated by clearing VUSBEN.

During overload conditions, VUSB and V18 drop below their nominal output voltage. In power scenarios where DVCC is exclusively supplied from VUSB, repetitive system restarts may be triggered as long the short/overload condition exists. For this reason, firmware should avoid re-enabling USB after detection of an overload on the previous power session, until the cause of failure can be identified.

The USB power system brownout circuit is supplied from VBUS or DVCC, whichever carries the higher voltage.

Ultimately, it is the user's responsibility to ensure that the current drawn from VBUS does not exceed I_{DET}.

22.2.3 USB Phase-Locked Loop (PLL)

The PLL provides the low-jitter high-accuracy clock needed for USB operation (see Figure 22-5).

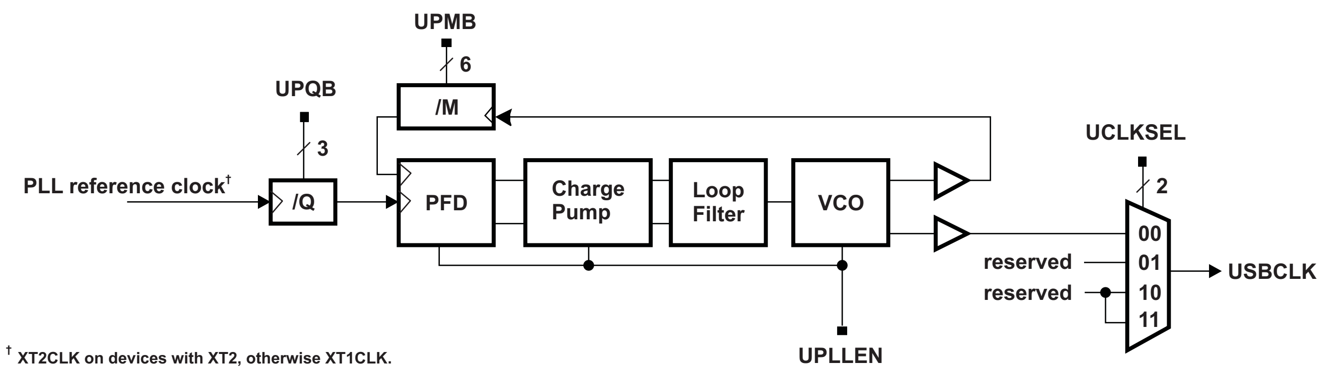


Figure 22-5. USB-PLL Analog Block Diagram

The reference clock to the PLL depends on the device configuration. On devices that contain the optional XT2, the reference clock to the PLL is XT2CLK, regardless if XT1 is available. If the device has only XT1, then the reference will be XT1CLK. A four-bit prescale counter controlled by the UPQB bits allows division of the reference to generate the PLL update clock. The UPMB bits control the divider in the feedback path and define the multiplication rate of the PLL (see Equation 11).

$$f_{OUT} = CLK_{SEL} \times \frac{DIVM}{DIVQ} \quad \text{with} \quad \frac{CLK_{SEL}}{DIVQ} = f_{UPD} \geq 1.5\text{ MHz} \qquad (11)$$

Where

CLK_{SEL} is the PLL reference frequency

DIVQ is derived from Table 22-1

DIVM represents the value of UPMB field

If USB operation is used in a bus-powered configuration, disabling the PLL is necessary in order to pass the USB requirement of not consuming more than 500 μA. The UPLLEN bit enables/disables the PLL. The PFDEN bit must be set in order to enable the phase/frequency discriminator. Out-of-lock, loss-of-signal, and out-of-range are indicated and flagged in the interrupt flags OOLIFG, LOSIFG, OORIFG, respectively.

NOTE: UCLKSEL bits should always be cleared, which is the default operation. All other combinations are reserved for future usages.

Table 22-1. USB-PLL Pre-Scale Divider

UPQB	DIVQ
000	1
001	2
010	3
011	4
100	6
101	8
110	12
111	16

Table 22-2. Register Settings to Generate 48 MHz Using Common Crystals

CLKSEL (MHz)	UPQB	UPMB	DIVQ	DIVM	CLKLOOP (MHz)	UPLLCLK (MHz)	ACCURACY (ppm)
1.5	000	011111	1	32	1.5	48	0
1.6	000	011101	1	30	1.6	48	0
1.7778	000	011010	1	27	1.7778	48	0
1.8432	000	011001	1	26	1.8432	47.92	-1570
1.8461	000	011001	1	26	1.8461	48	0
1.92	000	011000	1	25	1.92	48	0
2	000	010111	1	24	2	48	0
2.4	000	010011	1	20	2.4	48	0
2.6667	000	010001	1	18	2.6667	48	0
3	000	001111	1	16	3	48	0
3.2	001	011110	2	30	1.6	48	0
3.5556	001	011010	2	27	1.7778	48	0
3.579545	001	011010	2	27	1.79	48.32	6666
3.84	001	011001	2	25	1.92	48	0
4 [1]	001	010111	2	24	2	48	0

(1) This frequency can be automatically detected by the factory-supplied BSL, for use in production programming of the MSP430 via USB. Refer to the *MSP430 Memory Programming User's Guide* for details.

Table 22-2. Register Settings to Generate 48 MHz Using Common Crystals (continued)

CLKSEL (MHz)	UPQB	UPMB	DIVQ	DIVM	CLKLOOP (MHz)	UPLLCLK (MHz)	ACCURACY (ppm)
4.1739	001	010110	2	23	2.086	48	0
4.1943	001	010110	2	23	2.097	48.23	4884
4.332	001	010101	2	22	2.166	47.652	-7250
4.3636	001	010101	2	22	2.1818	48	0
4.5	010	011111	3	32	1.5	48	0
4.8	001	010011	2	20	2.4	48	0
5.33 X (16/3)	001	010001	2	18	2.6667	48	0
5.76	010	011000	3	25	1.92	48	0
6	010	010111	3	24	2	48	0
6.4	011	011101	4	30	1.6	48	0
7.2	010	010011	3	20	2.4	48	0
7.68	011	011000	4	25	1.92	48	0
8[1]	010	010001	3	18	2.6667	48	0
9	010	001111	3	16	3	48	0
9.6	011	010011	4	20	2.4	48	0
10.66 X (32/3)	011	010001	4	18	2.6667	48	0
12[2]	011	001111	4	16	3	48	0
12.8	101	011101	8	30	1.6	48	0
14.4	100	010011	6	20	2.4	48	0
16	100	010001	6	18	2.6667	48	0
16.9344	100	010000	6	17	2.8224	47.98	-400
16.94118	100	010000	6	17	2.8235	48	0
18	100	001111	6	16	3	48	0
19.2	101	010011	8	20	2.4	48	0
24[2]	101	001111	8	16	3	48	0
25.6	111	011101	16	30	1.6	48	0
32	111	010111	16	24	2.6667	48	0

(2) This frequency can be automatically detected by the factory-supplied BSL, for use in production programming of the MSP430 via USB. Refer to the *MSP430 Memory Programming User's Guide* for details.

22.2.3.1 Modifying the Divider Values

Updating the values of UPQB (DIVQ) and UPMB (DIVM) to select the desired PLL frequency must occur simultaneously to avoid spurious frequency artifacts. The values of UPQB and UPMB can be calculated and written to their buffer registers; the final update of UPQB and UPMB occurs when the upper byte of UPLLDIVB (UPQB) is written.

22.2.3.2 PLL Error Indicators

The PLL can detect three kinds of errors. Out-of-lock (OOL) is indicated if a frequency correction is performed in the same direction (i.e., up/down) for four consecutive update periods. Loss-of-signal (LOS) is indicated if a frequency correction is performed in the same direction (i.e., up/down) for 16 consecutive update periods. Out-of-range (OOR) is indicated if PLL was unable to lock for more than 32 update periods.

OOL, LOS, and OOR trigger their respective interrupt flags (USBOOLIFG, USBLOSIFG, USBOORIFG) if errors occur, and interrupts are generated if enabled by their enable bits (USBOOLIE, USBLOSIE, USBOORIE).

22.2.3.3 PLL Startup Sequence

To achieve the fastest startup of the PLL, the following sequence is recommended.

1. Enable VUSB and V18.
2. Wait 2 ms for external capacitors to charge, so that proper VUSB is in place. (During this time, the USB registers and buffers can be initialized.)
3. Activate the PLL, using the required divider values.
4. Wait 2 ms and check PLL. If it stays locked, it is ready to be used.

22.2.4 USB Controller Engine

The USB controller engine transfers data packets arriving from the USB host into the USB buffers, and also transmits valid data from the buffers to the USB host. The controller engine has dedicated, fixed buffer space for input endpoint 0 and output endpoint 0, which are the default USB endpoints for control transfers.

The 14 remaining endpoints (seven input and seven output) may have one or more USB buffers assigned to them. All the buffers are located in the USB buffer memory. This memory is implemented as "multiport" memory, in that it can be accessed both by the USB buffer manager and also by the CPU and DMA.

Each endpoint has a dedicated set of descriptor registers that describe the use of that endpoint (see Figure 22-6). Configuration of each endpoint is performed by setting its descriptor registers. These data structures are located in the USB buffer memory and contain address pointers to the next memory buffer for receive/transmit.

Assigning one or two data buffers to an endpoint, of up to 64 bytes, requires no further software involvement after configuration. If more than three buffers per endpoint are desired, however, software must change the address pointers on the fly during a receive/transmit process.

Synchronization of empty and full buffers is done using validation flags. All events are indicated by flags and fire a vector interrupt when enabled. Transfer event indication can be enabled separately.

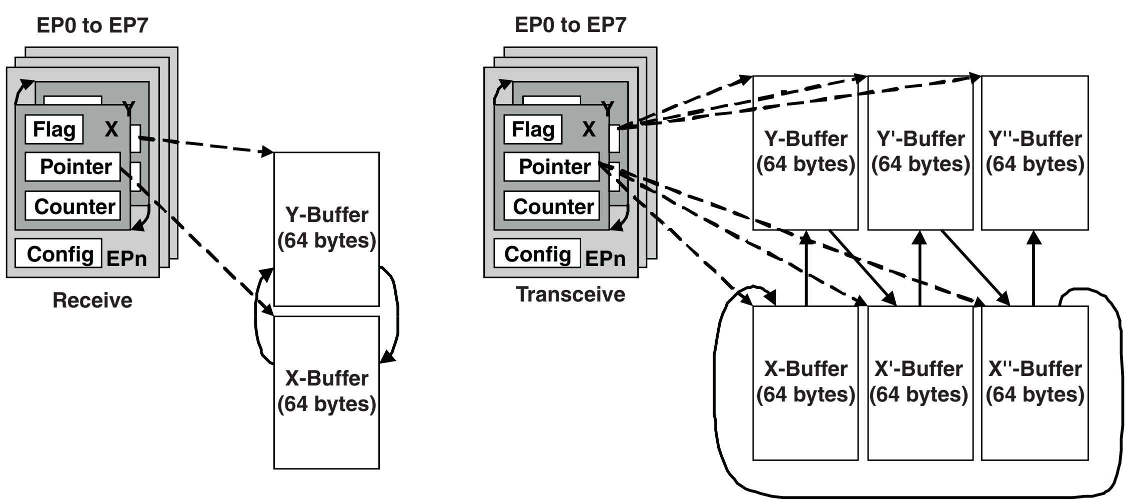


Figure 22-6. Data Buffers and Descriptors

22.2.4.1 USB Serial Interface Engine (SIE)

The SIE logic manages the USB packet protocol requirements for the packets being received and transmitted on the bus. For packets being received, the SIE decodes the packet identifier field (packet ID) to determine the type of packet being received and to ensure the packet ID is valid. For token and data packets being received, the SIE calculates the packet cycle redundancy check (CRC) and compares the value to the CRC contained in the packet to verify that the packet was not corrupted during transmission.

For token and data packets being transmitted, the SIE generates the CRC that is transmitted with the packet. For packets being transmitted, the SIE also generates the synchronization field (SYNC), which is an eight-bit field at the beginning of each packet. In addition, the SIE generates the correct packet ID for all packets being transmitted.

Another major function of the SIE is the overall serial-to-parallel conversion of the data packets being received/transmitted.

22.2.4.2 USB Buffer Manager (UBM)

The USB buffer manager provides the control logic that interfaces the SIE to the USB endpoint buffers.

One of the major functions of the UBM is to decode the USB device address to determine if the USB host is addressing this particular USB device. In addition, the endpoint address field and direction signal are decoded to determine which particular USB endpoint is being addressed. Based on the direction of the USB transaction and the endpoint number, the UBM either writes or reads the data packet to/from the appropriate USB endpoint data buffer.

The TOGGLE bit for each output endpoint configuration register is used by the UBM to track successful output data transactions. If a valid data packet is received and the data packet ID matches the expected packet ID, the TOGGLE bit is toggled. Similarly, the TOGGLE bit for each input endpoint configuration is used by the UBM to track successful input data transactions. If a valid data packet is transmitted, the TOGGLE bit is toggled. If the TOGGLE bit is cleared, a DATA0 packet ID is transmitted in the data packet to the host. If the TOGGLE bit is set, a DATA1 packet ID is transmitted in the data packet to the host. Please refer to Section 22.3 regarding details of USB transfers.

22.2.4.3 USB Buffer Memory

The USB buffer memory contains the data buffers for all endpoints and for SETUP packets. In that the buffers for endpoints 1 to 7 are flexible, there are USB buffer configuration registers that define them, and these too are in the USB buffer memory. (Endpoint 0 is defined with a set of registers in the USB control register space.) Storing these in open memory allows for efficient, flexible use, which is advantageous because use of these endpoints is very application-specific.

This memory is implemented as "multiport" memory, in that it can be accessed both by the USB buffer manager and also by the CPU and DMA. The SIE allows CPU/DMA access, but reserves priority. As a result, CPU/DMA access is delayed using wait states if a conflict arises with an SIE access.

When the USB module is disabled (USBEN = 0), the buffer memory behaves like regular RAM. When changing the state of the USBEN bit (enabling or disabling the USB module), the USB buffer memory should not be accessed within four clocks before and eight clocks after changing this bit, as doing so reconfigures the access method to the USB memory.

Accessing of the USB buffer memory by CPU or DMA is only possible if the USB PLL is active. When a host requests suspend condition the application software (e.g. USB stack) of client has to switch off the PLL within 10ms. Note that the MSP430 USB suspend interrupt occurs around 5ms after the host request.

Each endpoint is defined by a block of six configuration "registers" (based in RAM, they are not true registers in the strict sense of the word). These registers specify the endpoint type, buffer address, buffer size and data packet byte count. They define an endpoint buffer space that is 1904 bytes in size. An additional 24 bytes are allotted to three remaining blocks – the EP0_IN buffer, the EP0_OUT buffer, and the SETUP packet buffer (see Table 22-3).

Table 22-3. USB Buffer Memory Map

Memory	Short Form	Access Type	Address Offset
Start of buffer space	STABUFF	Read/Write	0000h
1904 bytes of configurable buffer space	⋮	Read/Write	⋮
End of buffer space	TOPBUFF	Read/Write	076Fh
Output endpoint_0 buffer	USBOEP0BUF	Read/Write	0770h
		Read/Write	⋮
		Read/Write	0777h
Input endpoint_0 buffer	USBIEP0BUF	Read/Write	0778h
		Read/Write	⋮
		Read/Write	077Fh

Table 22-3. USB Buffer Memory Map (continued)

Memory	Short Form	Access Type	Address Offset
Setup Packet Block	USBSUBLK	Read/Write	0780h
		Read/Write	⋮
		Read/Write	0787h

Software can configure each buffer according to the total number of endpoints needed. Single or double buffering of each endpoint is possible.

Unlike the descriptor registers for endpoints 1 to 7, which are defined as memory entries in USB RAM, endpoint 0 is described by a set of four registers (two for output and two for input) in the USB control register set. Endpoint 0 has no base-address register, since these addresses are hardwired. The bit positions have been preserved to provide consistency with endpoint_n (n = 1 to 7).

22.2.4.4 USB Fine Timestamp

The USB module is capable of saving a timestamp associated with particular USB events (see Figure 22-7). This can be useful in compensating for delays in software response. The timestamp values are based on the USB module's internal timer, driven by USBCLK.

Up to four events can be selected to generate the timestamp, selected with the TSESEL bits. When they occur, the value of the USB timer is transferred to the timestamp register USBTSREG, and thus the exact moment of the event is recorded. The trigger options include one of three DMA channels, or a software-driven event. The USB timer cannot be directly accessed by reading.

Furthermore, the value of the USB timer can be used to generate periodic interrupts. Since the USBCLK can have a frequency different from the other system clocks, this gives another option for periodic system interrupts. The UTSEL bits select the divider from the USB clock. UTIE must be set for an interrupt vector to get triggered.

The timestamp register is set to zero on a frame-number-receive event and pseudo-start-of-frame.

TSGEN enables/disables the time stamp generator.

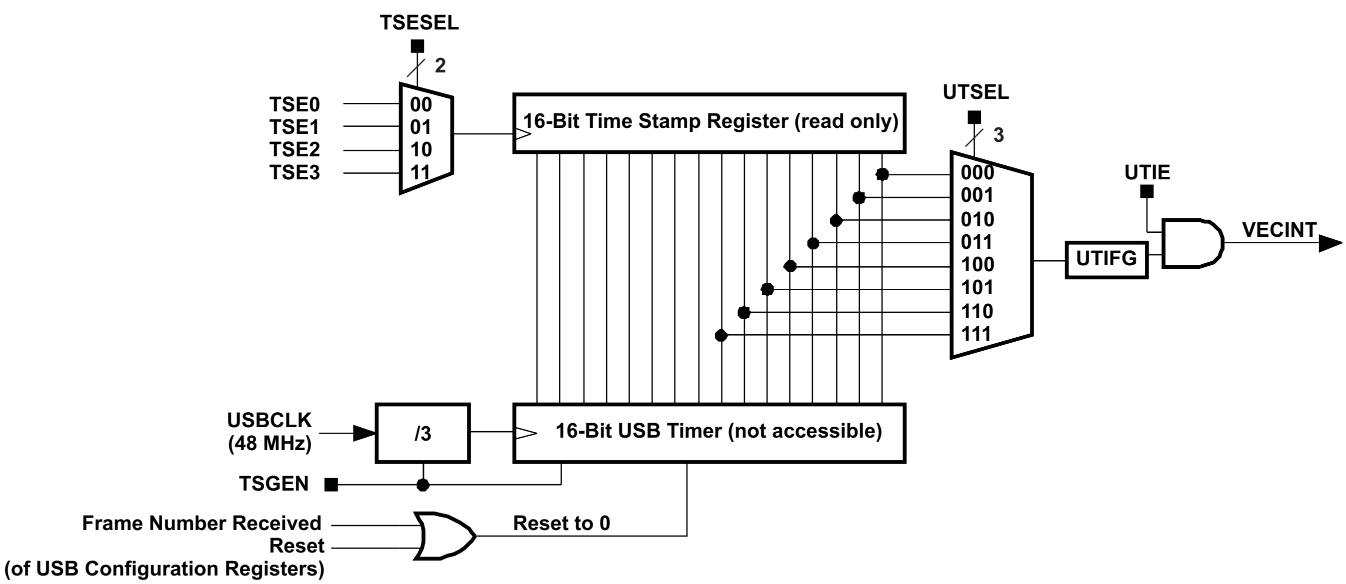


Figure 22-7. USB Timer and Time Stamp Generation

22.2.4.5 Suspend/Resume Logic

The USB suspend/resume logic detects suspend and resume conditions on the USB bus. These events are flagged in SUSRIFG and RESRIFG, respectively, and they fire dedicated interrupts, if the interrupts are enabled (SUSRIE and RESRIE).

The remote wakeup mechanism, in which a USB device can cause the USB host to awaken and resume the device, is triggered by setting the RWUP bit of the USBCTL register.

See Section 22.2.6 for more information.

22.2.4.6 Reset Logic

A PUC resets the USB module logic. When FRSTE = 1, the logic is also reset when a USB reset event occurs on the bus, triggered from the USB host. (A USB reset also sets the RSTRIFG flag.) USB buffer memory is not reset by a USB reset.

22.2.5 USB Vector Interrupts

The USB module uses a single interrupt vector generator register to handle multiple USB interrupts. All USB-related interrupt sources trigger the USBVECINT vector, which then contains a 6-bit vector value that identifies the interrupt source. Each of the interrupt sources results in a different offset value read. The interrupt vector returns zero when no interrupt is pending.

Reading the interrupt vector register clears the corresponding interrupt flag and updates its value. The interrupt with highest priority returns the value 0002h; the interrupt with lowest priority returns the value 003Eh when reading the interrupt vector register. Writing to this register clears all interrupt flags.

For each input and output endpoints resides an USB transaction interrupt indication enable. Software may set this bit to define if interrupts are to be flagged in general. To generate an interrupt the corresponding interrupt enable and flag must be set.

Table 22-4. USB Interrupt Vector Generation

USBVECINT Value	Interrupt Source	Interrupt Flag Bit	Interrupt Enable Bit	Indication Enable Bit
0000h	no interrupt	–	–	–
0002h	USB-PWR drop ind.	USBPWRCTL.VUOVLIFG	USBPWRCTL.VUOVLIE	–
0004h	USB-PLL lock error	USBPLLIR.USBPLLOOLIFG	USBPLLIR.USBPLLOOLIE	–
0006h	USB-PLL signal error	USBPLLIR.USBPLLOSIFG	USBPLLIR.USBPLLLOSIE	–
0008h	USB-PLL range error	USBPLLIR.USBPLLOORIFG	USBPLLIR.USBPLLOORIE	–
000Ah	USB-PWR VBUS-on	USBPWRCTL.VBONIFG	USBPWRCTL.VBONIE	–
000Ch	USB-PWR VBUS-off	USBPWRCTL.VBOFFIFG	USBPWRCTL.VBOFFIE	–
000Eh	reserved	–	–	–
0010h	USB timestamp event	USBMAINTL.UTIFG	USBMAINTL.UTIE	–
0012h	Input Endpoint-0	USBIEPIFG.EP0	USBIEPIE.EP0	USBIEPCNFG_0.USBIIE
0014h	Output Endpoint-0	USBOEPIFG.EP0	USBOEPIE.EP0	USBOEPCNFG_0.USBIIE
0016h	RSTR interrupt	USBIFG.RSTRIFG	USBIE.RSTRIE	–
0018h	SUSR interrupt	USBIFG.SUSRIFG	USBIE.SUSRIE	–
001Ah	RESR interrupt	USBIFG.RESRIFG	USBIE.RESRIE	–
001Ch	reserved	–	–	–
001Eh	reserved	–	–	–
0024h	Input Endpoint-1	USBIEPIFG.EP1	USBIEPIE.EP1	USBIEPCNF_1.USBIIE
0026h	Input Endpoint-2	USBIEPIFG.EP2	USBIEPIE.EP2	USBIEPCNF_2.USBIIE
0028h	Input Endpoint-3	USBIEPIFG.EP3	USBIEPIE.EP3	USBIEPCNF_3.USBIIE
002Ah	Input Endpoint-4	USBIEPIFG.EP4	USBIEPIE.EP4	USBIEPCNF_4.USBIIE
002Ch	Input Endpoint-5	USBIEPIFG.EP5	USBIEPIE.EP5	USBIEPCNF_5.USBIIE
002Eh	Input Endpoint-6	USBIEPIFG.EP6	USBIEPIE.EP6	USBIEPCNF_6.USBIIE
0030h	Input Endpoint-7	USBIEPIFG.EP7	USBIEPIE.EP7	USBIEPCNF_7.USBIIE
0032h	Output Endpoint-1	USBOEPIFG.EP1	USBOEPIE.EP1	USBOEPCNF_1.USBIIE

Table 22-4. USB Interrupt Vector Generation (continued)

USBVECINT Value	Interrupt Source	Interrupt Flag Bit	Interrupt Enable Bit	Indication Enable Bit
0034h	Output Endpoint-2	USBOEPIFG.EP2	USBOEPIE.EP2	USBOEPCNF_2.USBIIE
0036h	Output Endpoint-3	USBOEPIFG.EP3	USBOEPIE.EP3	USBOEPCNF_3.USBIIE
0038h	Output Endpoint-4	USBOEPIFG.EP4	USBOEPIE.EP4	USBOEPCNF_4.USBIIE
003Ah	Output Endpoint-5	USBOEPIFG.EP5	USBOEPIE.EP5	USBOEPCNF_5.USBIIE
003Ch	Output Endpoint-6	USBOEPIFG.EP6	USBOEPIE.EP6	USBOEPCNF_6.USBIIE
003Eh	Output Endpoint-7	USBOEPIFG.EP7	USBOEPIE.EP7	USBOEPCNF_7.USBIIE

22.2.6 Power Consumption

USB functionality consumes more power than is typically drawn in the MSP430. Since most MSP430 applications are power sensitive, the MSP430 USB module has been designed to protect the battery by ensuring that significant power load only occurs when attached to the bus, allowing power to be drawn from VBUS.

The two components of the USB module that draw the most current are the transceiver and the PLL. The transceiver can consume large amounts of power while transmitting, but in its quiescent state – that is, when not transmitting data – the transceiver actually consumes very little power. This is the amount specified as I_{IDLE}. This amount is so little that the transceiver can be kept active during suspend mode without presenting a problem for bus-powered applications. Fortunately the transceiver always has access to VBUS power when drawing the level of current required for transmitting.

The PLL consumes a larger amount of current. However, it need only be active while connected to the host, and the host can supply the power. When the PLL is disabled (for example, during USB suspend), USBCLK automatically is sourced from the VLO.

22.2.7 Suspend and Resume

All USB devices must support the ability to be suspended into a no-activity state, and later resumed. When suspended, a device is not allowed to consume more than 500uA from the USB's VBUS power rail, if the device is drawing any power from that source. A suspended device must also monitor for a resume event on the bus.

The host initiates a suspend condition by creating a constant idle state on the bus for more than 3.0 ms. It is the responsibility of the software to ensure the device enters its low power suspend state within 10 ms of the suspend condition. The USB specification requires that a suspended bus-powered USB device not draw in excess of 500 μA from the bus.

22.2.7.1 Entering Suspend

When the host suspends the USB device, a suspend interrupt is generated (SUSRIFG). From this point, the software has 10 ms to ensure that no more than 500uA is being drawn from the host via VBUS.

For most applications, the integrated 3.3 V LDO is being used. In this case, the following actions should be taken:

- Disable the PLL by clearing UPLLEN (UPLLEN = 0)
- Limit all current sourced from VBUS that causes the total current sourced from VBUS equal to 500 μA minus the suspend current, $I_{SUSPEND}$(refer to the device specific datasheet).

Disabling the PLL eliminates the largest on-chip draw of power from VBUS. During suspend, the USBCLK is automatically sourced by the VLO (VLOCLK), allowing the USB module to detect resume when it occurs. It is a good idea to also then ensure that the RESRIE bit is also set, so that an interrupt will be generated when the host resumes the device. If desired, the high frequency crystal can also be disabled to save additional system power, however it does not contribute to the power from VBUS since it draws power from the DVCC supply.

22.2.7.2 Entering Resume Mode

When the USB device is in a suspended condition, any non-idle signaling, including reset signaling, on the host side will be detected by the suspend/resume logic and device operation will be resumed. RESRIFG will be set, causing an USB interrupt. The interrupt service routine can be used to resume USB operation.

22.3 USB Transfers

The USB module supports control, bulk, and interrupt data transfer types. In accordance with the USB specification, endpoint 0 is reserved for the control endpoint and is bidirectional. In addition to the control endpoint, the USB module is capable of supporting up to 7 input endpoints and 7 output endpoints. These additional endpoints can be configured either as bulk or interrupt endpoints. The software handles all control, bulk, and interrupt endpoint transactions.

22.3.1 Control Transfers

Control transfers are used for configuration, command, and status communication between the host and the USB device. Control transfers to the USB device use input endpoint 0 and output endpoint 0. The three types of control transfers are control write, control write with no data stage, and control read. Note that the control endpoint must be initialized before connecting the USB device to the USB.

22.3.1.1 Control Write Transfer

The host uses a control write transfer to write data to the USB device. A control write transfer consists of a setup stage transaction, at least one output data stage transaction, and an input status stage transaction.

The stage transactions for a control write transfer are:

- Setup stage transaction:
  1. Input endpoint 0 and output endpoint 0 are initialized by programming the appropriate USB endpoint configuration blocks. This entails enabling the endpoint interrupt (USBIIE = 1) and enabling the endpoint (UBME = 1). The NAK bit for both input endpoint 0 and output endpoint 0 must be cleared.
  2. The host sends a setup token packet followed by the setup data packet addressed to output endpoint 0. If the data is received without an error, then the UBM will write the data to the setup data packet buffer, set the setup stage transaction bit (SETUPIFG = 1) in the USB Interrupt Flag register (USBIFG), return an ACK handshake to the host, and assert the setup stage transaction interrupt. Note that as long as SETUPIFG = 1, the UBM will return a NAK handshake for any data stage or status stage transactions regardless of the endpoint 0 NAK or STALL bit values.
  3. The software services the interrupt, reads the setup data packet from the buffer, and then decodes the command. If the command is not supported or invalid, the software should set the STALL bit in the output endpoint 0 configuration register (USBOEPCNFG_0) and the input endpoint 0 configuration register (USBIEPCNFG_0) . This will cause the device to return a STALL handshake for any data or status stage transaction. For control write transfers, the packet ID used by the host for the first data packet output will be a DATA1 packet ID and the TOGGLE bit must match.
- Data stage transaction:
  1. The host sends an OUT token packet followed by a data packet addressed to output endpoint 0. If the data is received without an error, the UBM will write the data to the output endpoint buffer (USBOEP0BUF), update the data count value, toggle the TOGGLE bit, set the NAK bit, return an ACK handshake to the host , and assert the output endpoint interrupt 0 (OEPIFG0).
  2. The software services the interrupt and reads the data packet from the output endpoint buffer. To read the data packet, the software first needs to obtain the data count value inside the USBOEPBCNT_0 register. After reading the data packet, the software should clear the NAK bit to allow the reception of the next data packet from the host.
  3. If the NAK bit is set when the data packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the data packet is received, the UBM simply returns a STALL handshake to the host. If a CRC or bit stuff error occurs when the data packet is received, then no handshake is returned to the host.
- Status stage transaction:
  1. For input endpoint 0, the software updates the data count value to zero, sets the TOGGLE bit, then

clears the NAK bit to enable the data packet to be sent to the host. Note that for a status stage transaction, a null data packet with a DATA1 packet ID is sent to the host.

2. The host sends an IN token packet addressed to input endpoint 0. After receiving the IN token, the UBM transmits a null data packet to the host. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM will then toggle the TOGGLE bit and sets the NAK bit.
3. If the NAK bit is set when the IN token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the IN token packet is received, the UBM simply returns a STALL handshake to the host. If no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.

22.3.1.2 Control Write Transfer with No Data Stage Transfer

The host uses a control write transfer to write data to the USB device. A control write with no data stage transfer consists of a setup stage transaction and an input status stage transaction. For this type of transfer, the data to be written to the USB device is contained in the two byte value field of the setup stage transaction data packet.

The stage transactions for a control write transfer with no data stage transfer are:

- Setup stage transaction:
  1. Input endpoint 0 and output endpoint 0 are initialized by programming the appropriate USB endpoint configuration blocks. This entails programming the buffer size and buffer base address, selecting the buffer mode, enabling the endpoint interrupt (USBIIE = 1), initializing the TOGGLE bit, enabling the endpoint (UBME = 1). The NAK bit for both input endpoint 0 and output endpoint 0 must be cleared.
  2. The host sends a setup token packet followed by the setup data packet addressed to output endpoint 0. If the data is received without an error then the UBM will write the data to the setup data packet buffer, set the setup stage transaction (SETUP) bit in the USB status register, return an ACK handshake to the host, and assert the setup stage transaction interrupt. Note that as long as the setup transaction (SETUP) bit is set, the UBM will return a NAK handshake for any data stage or status stage transaction regardless of the endpoint 0 NAK or STALL bit values.
  3. The software services the interrupt and reads the setup data packet from the buffer then decodes the command. If the command is not supported or invalid, the software should set the STALL bits in the output endpoint 0 and the input endpoint 0 configuration registers before clearing the setup stage transaction (SETUP) bit. This will cause the device to return a STALL handshake for data or status stage transactions. After reading the data packet and decoding the command, the software should clear the interrupt, which will automatically clear the setup stage transaction status bit.
- Status stage transaction:
  1. For input endpoint 0, the software updates the data count value to zero, sets the TOGGLE bit, then clears the NAK bit to enable the data packet to be sent to the host. Note that for a status stage transaction a null data packet with a DATA1 packet ID is sent to the host.
  2. The host sends an IN token packet addressed to input endpoint 0. After receiving the IN token, the UBM transmits a null data packet to the host. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM will then toggle the TOGGLE bit, set the NAK bit, and assert the endpoint interrupt.
  3. If the NAK bit is set when the IN token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the IN token packet is received, the UBM simply returns a STALL handshake to the host. If no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.

22.3.1.3 Control Read Transfer

The host uses a control read transfer to read data from the USB device. A control read transfer consists of a setup stage transaction, at least one input data stage transaction and an output status stage transaction.

The stage transactions for a control read transfer are:

- Setup stage transaction:
  1. Input endpoint 0 and output endpoint 0 are initialized by programming the appropriate USB

endpoint configuration blocks. This entails enabling the endpoint interrupt (USBIIE = 1) and enabling the endpoint (UBME = 1). The NAK bit for both input endpoint 0 and output endpoint 0 must be cleared.

2. The host sends a setup token packet followed by the setup data packet addressed to output endpoint 0. If the data is received without an error, then the UBM will write the data to the setup buffer, set the setup stage transaction (SETUP) bit in the USB status register, return an ACK handshake to the host and assert the setup stage transaction interrupt. Note that as long as the setup transaction (SETUP) bit is set, the UBM will return a NAK handshake for any data stage or status stage transactions regardless of the endpoint 0 NAK or STALL bit values.
3. The software services the interrupt and reads the setup data packet from the buffer then decodes the command. If the command is not supported or invalid, the software should set the STALL bits in the output endpoint 0 and the input endpoint 0 configuration registers before clearing the setup stage transaction (SETUP) bit. This will cause the device to return a STALL handshake for a data stage or status stage transactions. After reading the data packet and decoding the command, the software should clear the interrupt, which will automatically clear the setup stage transaction status bit. The software should also set the TOGGLE bit in the input endpoint 0 configuration register. For control read transfers, the packet ID used by the host for the first input data packet will be a DATA1 packet ID.

- Data stage transaction:
  1. The data packet to be sent to the host is written to the input endpoint 0 buffer by the software. The software also updates the data count value then clears the input endpoint 0 NAK bit to enable the data packet to be sent to the host.
  2. The host sends an IN token packet addressed to input endpoint 0. After receiving the IN token, the UBM transmits the data packet to the host. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM will set the NAK bit and assert the endpoint interrupt.
  3. The software services the interrupt and prepares to send the next data packet to the host.
  4. If the NAK bit is set when the IN token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the IN token packet is received, the UBM simply returns a STALL handshake to the host. If no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.
  5. The software continues to send data packets until all data has been sent to the host.
- Status stage transaction:
  1. For output endpoint 0, the software sets the TOGGLE bit, then clears the NAK bit to enable the data packet to be sent to the host. Note that for a status stage transaction a null data packet with a DATA1 packet ID is sent to the host.
  2. The host sends an OUT token packet addressed to output endpoint 0. If the data packet is received without an error then the UBM will update the data count value, toggle the TOGGLE bit, set the NAK bit, return an ACK handshake to the host, and assert the endpoint interrupt.
  3. The software services the interrupt. If the status stage transaction completed successfully, then the software should clear the interrupt and clear the NAK bit.
  4. If the NAK bit is set when the input data packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the in data packet is received, the UBM simply returns a STALL handshake to the host. If a CRC or bit stuff error occurs when the data packet is received, then no handshake is returned to the host.

22.3.1.4 Control Read Transfer

The host uses a control read transfer to read data from the USB device. A control read transfer consists of a setup stage transaction, at least one input data stage transaction and an output status stage transaction.

The stage transactions for a control read transfer are:

- Setup stage transaction:
  1. Input endpoint 0 and output endpoint 0 are initialized by programming the appropriate USB endpoint configuration blocks. This entails enabling the endpoint interrupt (USBIIE = 1) and enabling the endpoint (UBME = 1). The NAK bit for both input endpoint 0 and output endpoint 0 must be cleared.

2. The host sends a setup token packet followed by the setup data packet addressed to output endpoint 0. If the data is received without an error then the UBM will write the data to the setup buffer, set the setup stage transaction (SETUP) bit to a 1 in the USB status register, return an ACK handshake to the host and assert the setup stage transaction interrupt. Note that as long as the setup transaction (SETUP) bit is set, the UBM will return a NAK handshake for any data stage or status stage transactions regardless of the endpoint 0 NAK or STALL bit values.
3. The software services the interrupt and reads the setup data packet from the buffer then decodes the command. If the command is not supported or invalid, the software should set the STALL bits in the output endpoint 0 the input endpoint 0 configuration registers before clearing the setup stage transaction (SETUP) bit. This will cause the device to return a STALL handshake for a data stage or status stage transactions. After reading the data packet and decoding the command, the software should clear the interrupt, which will automatically clear the setup stage transaction status bit. The software should also set the TOGGLE bit in the input endpoint 0 configuration register. For control read transfers, the packet ID used by the host for the first input data packet will be a DATA1 packet ID.

- Data stage transaction:
  1. The data packet to be sent to the host is written to the input endpoint 0 buffer by the software. The software also updates the data count value then clears the input endpoint 0 NAK bit to enable the data packet to be sent to the host.
  2. The host sends an IN token packet addressed to input endpoint 0. After receiving the IN token, the UBM transmits the data packet to the host device. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM will set the NAK bit and assert the endpoint interrupt.
  3. The software services the interrupt and prepares to send the next data packet to the host.
  4. If the NAK bit is set when the IN token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the IN token packet is received, the UBM simply returns a STALL handshake to the host. If a no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.
  5. The software continues to send data packets until all data has been sent to the host.
- Status stage transaction:
  1. For output endpoint 0, the software sets the TOGGLE bit, then clears the NAK bit to enable the data packet to be sent to the host. Note that for a status stage transaction a null data packet with a DATA1 packet ID is sent to the host.
  2. The host sends an OUT token packet addressed to output endpoint 0. If the data packet is received without an error then the UBM will update the data count value, toggle the TOGGLE bit, set the NAK bit, return an ACK handshake to the host, and assert the endpoint interrupt.
  3. The software services the interrupt. If the status stage transaction completed successfully, then the software should clear the interrupt and clear the NAK bit.
  4. If the NAK bit is set when the input data packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the in data packet is received, the UBM simply returns a STALL handshake to the host. If a CRC or bit stuff error occurs when the data packet is received, then no handshake is returned to the host.

22.3.2 Interrupt Transfers

The USB module supports interrupt data transfers both to and from the host. Devices that need to send or receive a small amount of data with a specified service period are best served by the interrupt transfer type. Input endpoints 1 through 7 and output endpoints 1 through 7 can be configured as interrupt endpoints.

22.3.2.1 Interrupt OUT Transfer

The steps for an interrupt OUT transfer are:

1. The software initializes one of the output endpoints as an output interrupt endpoint by programming the appropriate endpoint configuration block. This entails programming the buffer size and buffer base address, selecting the buffer mode, enabling the endpoint interrupt, initializing the toggle bit, enabling the endpoint, and clearing the NAK bit.

2. The host sends an OUT token packet followed by a data packet addressed to the output endpoint. If the data is received without an error then the UBM will write the data to the endpoint buffer, update the data count value, toggle the toggle bit, set the NAK bit, return an ACK handshake to the host, and assert the endpoint interrupt.
3. The software services the interrupt and reads the data packet from the buffer. To read the data packet, the software first needs to obtain the data count value. After reading the data packet, the software should clear the interrupt and clear the NAK bit to allow the reception of the next data packet from the host.
4. If the NAK bit is set when the data packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the data packet is received, the UBM simply returns a STALL handshake to the host. If a CRC or bit stuff error occurs when the data packet is received, then no handshake is returned to the host device.

In double buffer mode, the UBM selects between the X and Y buffer based on the value of the toggle bit. If the toggle bit is a 0, the UBM will write the data packet to the X buffer. If the toggle bit is a 1, the UBM will write the data packet to the Y buffer. When a data packet is received, the software could determine which buffer contains the data packet by reading the toggle bit. However, when using double buffer mode, the possibility exists for data packets to be received and written to both the X and Y buffer before the software responds to the endpoint interrupt. In this case, simply using the toggle bit to determine which buffer contains the data packet would not work. Hence, in double buffer mode, the software should read the X buffer NAK bit, the Y buffer NAK bit, and the toggle bits to determine the status of the buffers.

22.3.2.2 Interrupt IN Transfer

The steps for an interrupt IN transfer are:

1. The software initializes one of the input endpoints as an input interrupt endpoint by programming the appropriate endpoint configuration block. This entails programming the buffer size and buffer base address, selecting the buffer mode, enabling the endpoint interrupt, initializing the toggle bit, enabling the endpoint, and setting the NAK bit.
2. The data packet to be sent to the host is written to the buffer by the software. The software also updates the data count value then clears the NAK bit to enable the data packet to be sent to the host.
3. The host sends an IN token packet addressed to the input endpoint. After receiving the IN token, the UBM transmits the data packet to the host. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM will then toggle the toggle bit, set the NAK bit and assert the endpoint interrupt.
4. The software services the interrupt and prepares to send the next data packet to the host.
5. If the NAK bit is set when the in token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the IN token packet is received, the UBM simply returns a STALL handshake to the host. If no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.

In double buffer mode, the UBM selects between the X and Y buffer based on the value of the toggle bit. If the toggle bit is a 0, the UBM will read the data packet from the X buffer. If the toggle bit is a 1, the UBM will read the data packet from the Y buffer.

22.3.3 Bulk Transfers

The USB module supports bulk data transfers both to and from the host. Devices that need to send or receive a large amount of data without a suitable bandwidth are best served by the bulk transfer type. In endpoints 1 through 7 and out endpoints 1 through 7 can all be configured as bulk endpoints.

22.3.3.1 Bulk OUT Transfer

The steps for a bulk OUT transfer are:

1. The software initializes one of the output endpoints as an output bulk endpoint by programming the appropriate endpoint configuration block. This entails programming the buffer size and buffer base address, selecting the buffer mode, enabling the endpoint interrupt, initializing the toggle bit, enabling the endpoint, and clearing the NAK bit.

2. The host sends an out token packet followed by a data packet addressed to the output endpoint. If the data is received without an error then the UBM will write the data to the endpoint buffer, update the data count value, toggle the toggle bit, set the NAK bit, return an ACK handshake to the host, and assert the endpoint interrupt.
3. The software services the interrupt and reads the data packet from the buffer. To read the data packet, the software first needs to obtain the data count value. After reading the data packet, the software should clear the interrupt and clear the NAK bit to allow the reception of the next data packet from the host.
4. If the NAK bit is set when the data packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the data packet is received, the UBM simply returns a STALL handshake to the host. If a CRC or bit stuff error occurs when the data packet is received, then no handshake is returned to the host.

In double buffer mode, the UBM selects between the X and Y buffer based on the value of the toggle bit. If the toggle bit is a 0, the UBM will write the data packet to the X buffer. If the toggle bit is a 1, the UBM will write the data packet to the Y buffer. When a data packet is received, the software could determine which buffer contains the data packet by reading the toggle bit. However, when using double buffer mode, the possibility exists for data packets to be received and written to both the X and Y buffer before the software responds to the endpoint interrupt. In this case, simply using the toggle bit to determine which buffer contains the data packet would not work. Hence, in double buffer mode, the software should read the X buffer NAK bit, the Y buffer NAK bit, and the toggle bits to determine the status of the buffers.

22.3.3.2 Bulk IN Transfer

The steps for a bulk IN transfer are:

1. The software initializes one of the input endpoints as an input bulk endpoint by programming the appropriate endpoint configuration block. This entails programming the buffer size and buffer base address, selecting the buffer mode, enabling the endpoint interrupt, initializing the toggle bit, enabling the endpoint, and setting the NAK bit.
2. The data packet to be sent to the host is written to the buffer by the software. The software also updates the data count value then clears the NAK bit to enable the data packet to be sent to the host.
3. The host sends an IN token packet addressed to the input endpoint. After receiving the IN token, the UBM transmits the data packet to the host. If the data packet is received without errors by the host, then an ACK handshake is returned. The UBM will then toggle the toggle bit, set the NAK bit and assert the endpoint interrupt.
4. The software services the interrupt and prepares to send the next data packet to the host.
5. If the NAK bit is set when the in token packet is received, the UBM simply returns a NAK handshake to the host. If the STALL bit is set when the In token packet is received, the UBM simply returns a STALL handshake to the host. If no handshake packet is received from the host, then the UBM prepares to retransmit the same data packet again.

In double buffer mode , the UBM selects between the X and Y buffer based on the value of the toggle bit. If the toggle bit is a 0, the UBM will read the data packet from the X buffer. If the toggle bit is a 1, the UBM will read the data packet from the Y buffer.

22.4 Registers

The USB register space is subdivided into configuration registers, control registers, and USB buffer memory.

The configuration and control registers are physical registers located in peripheral memory, while the buffer memory is implemented in RAM. See the device-specific datasheet for base addresses of these register groupings.

The USB control registers may only be written while the USB module is enabled.

When the USB module is disabled, it no longer uses the RAM buffer memory. This memory then behaves as a 2 KB RAM block, and can be used by the CPU or DMA without any limitation.

22.4.1 USB Configuration Registers

The configuration registers control the hardware functions needed to make a USB connection, including the PHY, PLL, and LDOs.

Access to the configuration registers is allowed or disallowed using the USBKEYPID register. Writing the proper value – 9628h – unlocks the configuration registers and enables access. Writing any other value disables access while leaving the values of the registers intact. Locking should be done intentionally after the configuration is finished.

The configuration registers are listed in Table 22-5. All addresses are expressed as offsets; the base address can be found in the device-specific datasheet.

All registers are byte and word accessible.

Table 22-5. USB Configuration Registers

Register	Short Form	Register Type	Address Offset	Initial State
USB controller key and ID register	USBKEYPID	Read/Write	00h	0000h
USB controller configuration register	USBCNF	Read/Write	02h	0000h
USB-PHY control register	USBPHYCTL	Read/Write	04h	0000h
USB-PWR control register	USBPWRCTL	Read/Write	08h	1850h
USB-PLL control register	USBPLLCTL	Read/Write	10h	0000h
USB-PLL divider buffer register	USBPLLDIVB	Read/Write	12h	0000h
USB-PLL interrupt register	USBPLLIR	Read/Write	14h	0000h

USBKEYPID, USB Key Register

15	14	13	12	11	10	9	8
USBKEY Read as A5h, Must be written as 96h							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
USBKEY							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

USBKEY	Bits 15-0	Key register. Must be written with a value of 9628h in order to be recognized as a valid key. This "unlocks" the configuration registers. If written with any other value, the registers become "locked". Reads back as A528h if the registers are unlocked.

USBCNF, USB Module Configuration Register

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved			FNTEN	BLKRDY	PUR_IN	PUR_EN	USB_EN
r0	r0	r0	rw-0	rw-0	r	rw-0	rw-0

Can be modified only when USBKEYPID is unlocked

Reserved Bits 15-5 Reserved. Read back as 0.

FNTEN Bit 4 Frame number receive trigger enable for DMA transfers
- 0 Frame number receive trigger is blocked.
- 1 Frame number receive trigger is gated through to DMA.

BLKRDY Bit 3 Block transfer ready signaling for DMA transfers
- 0 DMA triggering is disabled.
- 1 DMA is triggered whenever the USB bus interface can accept new write transfers.

PUR_IN Bit 2 PUR input value. This bit reflects the input value present on PUR. This bit may be used as an indication to start a USB based boot loading program (USB-BSL). The PUR input logic is powered by VUSB. PUR_IN returns zero when VUSB is zero

PUR_EN Bit 1 PUR pin enable
- 0 PUR pin is in high-impedance state
- 1 PUR pin is driven high

USB_EN Bit 0 USB module enable
- 0 USB module is disabled
- 1 USB module is enabled

USBPHYCTL, USB-PHY Control Register

15	14	13	12	11	10	9	8
Reserved						Reserved	Reserved
r0	r0	r0	r0	r0	r0	rw-0	rw-0

7	6	5	4	3	2	1	0
PUSEL	Reserved	PUDIR	Reserved	PUIN1	PUIN0	PUOUT1	PUOUT0
rw-0	r	rw-0	rw-0	r	r	rw-0	rw-0

Can be modified only when USBKEYPID is unlocked

Reserved Bits 15-10 Reserved. Reads back as 0.

Reserved Bits 9-8 Reserved. **Must always be written with 0.**

PUSEL Bit 7 USB port function select. This bit selects the function of the PU0/DP and PU1/DM pins.
- 0 PU0 and PU1 function selected (general purpose I/O)
- 1 DP and DM function selected (USB terminals)

Reserved Bit 6 Reserved.

PUDIR Bit 5 USB port direction. This bit controls the direction of both PU0 and PU1. It is only valid when PUSEL = 0.
- 0 PU0 and PU1 output drivers are disabled.
- 1 PU0 and PU1 output drivers are enabled.

Reserved Bit 4 Reserved. **Must always be written with 0.**

PUIN1 Bit 3 PU1 input data, This bit reflects the logic value on the PU1 terminal.

PUIN0 Bit 2 PU0 input data, This bit reflects the logic value on the PU0 terminal.

PUOUT1 Bit 1 PU1 output data. This bits defines the value of the PU1 pin when configured as port function and PUDIR = 1.

PUOUT0 Bit 0 PU0 output data. This bits defines the value of the PU0 pin when configured as port function and PUDIR = 1.

USBPWRCTL, USB-Power Control Register

15	14	13	12	11	10	9	8
Reserved			SLDOEN	VUSBEN	VBOFFIE	VBONIE	VUOVLIE
r0	r0	r0	rw-1	rw-1	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved	SLDOAON	OVLAOFF	USBDETEN	USBBGVBV	VBOFFIFG	VBONIFG	VUOVLIFG
r0	rw-1	rw-0	rw-1	r	rw-0	rw-0	rw-0

Can be modified only when USBKEYPID is unlocked

Reserved	Bits 15-13	Reserved. Reads back as 0.
SLDOEN	Bit 12	1.8 V (secondary) LDO enable. When set, the LDO is enabled.
VUSBEN	Bit 11	3.3-V LDO enable. When set, the LDO is enabled.
VBOFFIE	Bit 10	VBUS "going OFF" interrupt enable 0 Interrupt disabled 1 Interrupt enabled
VBONIE	Bit 9	VBUS "coming ON" interrupt enable 0 Interrupt disabled 1 Interrupt enabled
VUOVLIE	Bit 8	VUSB overload indication interrupt enable 0 Interrupt disabled 1 Interrupt enabled
Reserved	Bit 7	Reserved. Reads back as 0.
SLDOAON	Bit 6	1.8-V LDO auto-on enable 0 LDO needs to be turned on manually via SLDOEN 1 A "VBUS coming on" transition sets SLDOEN
OVLAOFF	Bit 5	LDO overload auto-off enable 0 During an overload on the 3.3-V LDO, the LDO automatically enters current-limiting mode and stays there until the condition stops. 1 An overload indication clears the VUSBEN bit.
USBDETEN	Bit 4	Enable bit for VBUS-on/off events. 0 USB module will not detect USB-PWR VBUS-on/off events 1 USB module will detect USB-PWR VBUS-on/off events
USBBGVBV	Bit 3	VBUS valid 0 VBUS is not valid yet 1 VBUS is valid and within bounds
VBOFFIFG	Bit 2	VBUS "going OFF" interrupt flag. This bit indicates that VBUS fell below the launch voltage. It is automatically cleared when the corresponding vector of the USB interrupt vector register is read, or if a value is written to the interrupt vector register. 0 No interrupt pending 1 Interrupt pending
VBONIFG	Bit 1	VBUS "coming ON" interrupt flag. This bit indicates that VBUS rose above the launch voltage. This bit is automatically cleared when the corresponding vector of the USB interrupt vector register is read, or if a value is written to the interrupt vector register. 0 No interrupt pending 1 Interrupt pending
VUOVLIFG	Bit 0	VUSB overload interrupt flag. This bit indicates that the 3.3-V LDO entered an overload situation. 0 No interrupt pending 1 Interrupt pending

USBPLLCTL, USB-PLL Control Register

15	14	13	12	11	10	9	8
Reserved			Reserved	Reserved		UPFDEN	UPLLEN
r0	r0	r0	rw-0	r0	r0	rw-0	rw-0

7	6	5	4	3	2	1	0
UCLKSEL		Reserved					
rw-0	rw-0	r0	r0	r0	r0	r0	r0

Can be modified only when USBKEYPID is unlocked

Reserved	Bits 15-13	Reserved. Reads back as 0.
Reserved	Bit 12	Reserved. Should always be written with 0.
Reserved	Bits 11-10	Reserved. Reads back as 0.
UPFDEN	Bit 9	Phase frequency discriminator (PFD) enable 0 PFD is disabled 1 PFD is enabled
UPLLEN	Bit 8	PLL enable 0 PLL is disabled 1 PLL is enabled
UCLKSEL	Bits 7-6	USB module clock select. **Must always be written with 00.**

UCLKSEL value	Selected Clock for USB Module
00	PLLCLK (default)
01	Reserved
10	Reserved
11	Reserved

Reserved	Bits 5-0	Reserved. Reads back as 0.

USBPLLDIVB, USB-PLL Clock Divider Buffer Register

15	14	13	12	11	10	9	8
Reserved					UPQB		
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved		UPMB					
r0	r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Can be modified only when USBKEYPID is unlocked

Reserved Bits 15-11 Reserved. Reads back as 0.

UPQB Bits 10-8 PLL pre-scale divider buffer register. These bits select the pre-scale division value. The value of this register is transferred to UPQB as soon it is written.

UPQB value	Pre-Scaling Ratio
000	$f_{UPD} = f_{REF}$
001	$f_{UPD} = f_{REF} / 2$
010	$f_{UPD} = f_{REF} / 3$
011	$f_{UPD} = f_{REF} / 4$
100	$f_{UPD} = f_{REF} / 6$
101	$f_{UPD} = f_{REF} / 8$
110	$f_{UPD} = f_{REF} / 12$
111	$f_{UPD} = f_{REF} / 16$

Reserved Bits 7-6 Reserved. Reads back as 0.

UPMB Bits 5-0 USB PLL feedback divider buffer register. These bits select the value of the feedback divider. The value of this register is transferred to UPMB automatically when UPQB is written.

UPMB value	Multiplying Factor
000000	Feedback division rate: 1
000001	Feedback division rate: 2
⋮	⋮
111111	Feedback division rate: 64

USBPLLIR, USB-PLL Interrupt Register

15	14	13	12	11	10	9	8
Reserved					**USBOORIE**	**USBLOSIE**	**USBOOLIE**
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved					**USBOORIFG**	**USBLOSIFG**	**USBOOLIFG**
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0

Can be modified only when USBKEYPID is unlocked

Reserved	Bits 15-11	Reserved. Reads back as 0.
USBOORIE	Bit 10	PLL out-of-range interrupt enable 0 Interrupt disabled 1 Interrupt enabled
USBLOSIE	Bit 9	PLL loss-of-signal interrupt enable 0 Interrupt disabled 1 Interrupt enabled
USBOOLIE	Bit 8	PLL out-of-lock interrupt enable 0 Interrupt disabled 1 Interrupt enabled
Reserved	Bits 7-3	Reserved. Reads back as 0.
USBOORIFG	Bit 2	PLL out-of-range interrupt flag 0 No interrupt pending 1 Interrupt pending
USBLOSIFG	Bit 1	PLL loss-of-signal interrupt flag 0 No interrupt pending 1 Interrupt pending
USBOOLIFG	Bit 0	PLL out-of-lock interrupt flag 0 No interrupt pending 1 Interrupt pending

22.4.2 USB Control Registers

The control registers affect core USB operations that are fundamental for any USB connection. This includes control endpoint 0, interrupts, bus address and frame, and timestamps. Control of endpoints other than zero are found in the operation registers. Unlike the operation registers, the control registers are actual physical registers, whereas the operation registers exist in RAM, which can be re-allocated to general-purpose use.

The control registers are listed in Table 22-6. All addresses are expressed as offsets; the base address can be found in the device-specific datasheet.

All registers are byte and word accessible.

Table 22-6. USB Control Registers

Register		Short Form	Register Type	Address Offset	Initial State
Endpoint 0 configuration	Input endpoint_0: Configuration	USBIEPCNF_0	Read/Write	00h	00h
	Input endpoint_0: Byte Count	USBIEPCNT_0	Read/Write	01h	80h
	Output endpoint_0: Configuration	USBOEPCNF_0	Read/Write	02h	00h
	Output endpoint_0: Byte count	USBOEPCNT_0	Read/Write	03h	00h
Interrupts	Input endpoint interrupt enables	USBIEPIE	Read/Write	0Eh	00h
	Output endpoint interrupt enables	USBOEPIE	Read/Write	0Fh	00h
	Input endpoint interrupt flags	USBIEPIFG	Read/Write	10h	00h
	Output endpoint interrupt flags	USBOEPIFG	Read/Write	11h	00h
	Vector interrupt register	USBVECINT	Read/Write	12h	0000h
Timestamps	Timestamp maintenance register	USBMAINT	Read/Write	16h	0000h
	Timestamp register	USBTSREG	Read/Write	18h	0000h
Basic USB control	USB frame number	USBFN	Read only	1Ah	0000h
	USB control register	USBCTL	Read/Write	1Ch	00h
	USB interrupt enable register	USBIE	Read/Write	1Dh	00h
	USB interrupt flag register	USBIFG	Read/Write	1Eh	00h
	Function address register	USBFUNADR	Read/Write	1Fh	00h

USBIEPCNF_0 USB Input Endpoint-0 Configuration Register

7	6	5	4	3	2	1	0
UBME	Reserved	TOGGLE	Reserved	STALL	USBIIE	Reserved	
rw-0	r0	r-0	r0	rw-0	rw-0	r0	r0

Can be modified only when USBEN = 1

UBME	Bit 7	UBM in endpoint-0 enable 0 UBM cannot use this endpoint 1 UBM can use this endpoint
Reserved	Bit 6	Reserved. Reads back as 0.
TOGGLE	Bit 5	Toggle bit. Reads back 0, since the configuration endpoint does not need to toggle.
Reserved	Bit 4	Reserved
STALL	Bit 3	USB stall condition. When set, hardware automatically returns a stall handshake to the USB host for any transaction transmitted from endpoint-0. The stall bit is cleared automatically by the next setup transaction. 0 Indicates no stall 1 Indicates stall
USBIIE	Bit 2	USB transaction interrupt indication enable. Software may set this bit to define if interrupts are to be flagged in general. To generate an interrupt the corresponding interrupt flag must be set (IEPIE). 0 Corresponding interrupt flag is not set 1 Corresponding interrupt flag is set
Reserved	Bits 1-0	Reserved. Reads back as 0.

USBIEPBCNT_0 USB Input Endpoint-0 Byte Count Register

7	6	5	4	3	2	1	0
NAK	**Reserved**			**CNT**			
rw-0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0

Can be modified only when USBEN = 1

NAK	Bit 7	No acknowledge status bit. This bit is set by the UBM at the end of a successful USB IN transaction from endpoint-0, to indicate that the EP-0 IN buffer is empty. When this bit is set, all subsequent transactions from endpoint-0 result in a NAK handshake response to the USB host. To re-enable this endpoint to transmit another data packet to the host, this bit must be cleared by software. 0 Buffer contains a valid data packet for host device 1 Buffer is empty (Host-In request receives a NAK)
Reserved	Bits 6-4	Reserved. Reads back as 0.
CNT	Bits 3-0	Byte count. The In_EP-0 buffer data count value should be set by software when a new data packet is written to the buffer. This four-bit value contains the number of bytes in the data packet. 0000b to 1000b are valid numbers for 0 to 8 bytes to be sent 1001b to 1111b are reserved values (if used, defaults to 8)

USBOEPCNFG_0 USB Output Endpoint-0 Configuration Register

7	6	5	4	3	2	1	0
UBME	**Reserved**	**TOGGLE**	**Reserved**	**STALL**	**USBIIE**	**Reserved**	
rw-0	r0	r-0	r0	rw-0	rw-0	r0	r0

Can be modified only when USBEN = 1

UBME	Bit 7	UBM out Endpoint-0 enable 0 UBM cannot use this endpoint 1 UBM can use this endpoint
Reserved	Bit 6	Reserved. Reads back as 0.
TOGGLE	Bit 5	Toggle bit. Reads back 0, since the configuration endpoint does not need to toggle.
Reserved	Bit 4	Reserved. Reads back as 0.
STALL	Bit 3	USB stall condition. When set, hardware automatically returns a stall handshake to the USB host for any transaction transmitted into endpoint-0. The stall bit is cleared automatically by the next setup transaction. 0 Indicates no stall 1 Indicates stall
USBIIE	Bit 2	USB transaction interrupt indication enable. Software may set this bit to define if interrupts are to be flagged in general. To generate an interrupt the corresponding interrupt flag must be set (OEPIE). 0 Corresponding interrupt flag will not be set 1 Corresponding interrupt flag will be set
Reserved	Bits 1-0	Reserved. Reads back as 0.

USBOEPBCNT_0 USB Output Endpoint-0 Byte Count Register

7	6	5	4	3	2	1	0
NAK	**Reserved**			**CNT**			
rw-0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0

Can be modified only when USBEN = 1

NAK Bit 7 No acknowledge status bit. This bit is set by the UBM at the end of a successful USB out transaction into endpoint-0, in order to indicate that the EP-0 buffer contains a valid data packet and that the buffer data count value is valid. When this bit is set, all subsequent transactions to endpoint-0 will result in a NAK handshake response to the USB host. To re-enable this endpoint to receive another data packet from the host, this bit must be cleared by software.

- 0 No valid data in the buffer. The buffer is ready to receive a host OUT transaction
- 1 The buffer contains a valid packet from the host that has not been picked up. (Any subsequent Host-Out requests receive a NAK.)

Reserved Bits 6-4 Reserved. Reads back as 0.

CNT Bits 3-0 Byte count. This data count value is set by the UBM when a new data packet is received by the buffer for the out endpoint-0. The four-bit value contains the number of bytes received in the data buffer.

0000b to 1000b are valid numbers for 0 to 8 received bytes

1001b to 1111b are reserved values

USBIEPIE, USB Input Endpoint Interrupt Enable Register

7	6	5	4	3	2	1	0
IEPIE7	**IEPIE6**	**IEPIE5**	**IEPIE4**	**IEPIE3**	**IEPIE2**	**IEPIE1**	**IEPIE0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Can be modified only when USBEN = 1

IEPIEn Bits 7-0 Input endpoint interrupt enable. These bits enable/disable whether an event can trigger an interrupt; they do not influence whether the event gets flagged. This is enabled/disabled with the interrupt indication enable bit in the Endpoint descriptors.

- 0 Event does not generate an interrupt
- 1 Event does generate an interrupt

USBOEPIE, USB Output Endpoint Interrupt Enable Register

7	6	5	4	3	2	1	0
OEPIE7	**OEPIE6**	**OEPIE5**	**OEPIE4**	**OEPIE3**	**OEPIE2**	**OEPIE1**	**OEPIE0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Can be modified only when USBEN = 1

IEPIEn Bits 7-0 Output endpoint interrupt enable. These bits enable/disable whether an event can trigger an interrupt; they do not influence whether the event gets flagged. This is enabled/disabled with the interrupt indication enable bit in the Endpoint descriptors.

- 0 Event does not generate an interrupt
- 1 Event does generate an interrupt

USBIEPIFG, USB Input Endpoint Interrupt Flag Register

7	6	5	4	3	2	1	0
IEPIFG7	**IEPIFG6**	**IEPIFG5**	**IEPIFG4**	**IEPIFG3**	**IEPIFG2**	**IEPIFG1**	**IEPIFG0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Can be modified only when USBEN = 1

OEPIFGn Bits 7-0 Input Endpoint Interrupt Flag. These bits are set by the UBM when a successful completion of a transaction occurs for this endpoint. When set, a USB interrupt will be generated. The interrupt flag will be cleared when the MCU reads the value from the USBVECINT register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing zero to that bit location.

USBOEPIFG, USB Output Endpoint Interrupt Flag Register

7	6	5	4	3	2	1	0
OEPIFG7	**OEPIFG6**	**OEPIFG5**	**OEPIFG4**	**OEPIFG3**	**OEPIFG2**	**OEPIFG1**	**OEPIFG0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Can be modified only when USBEN = 1

OEPIFGn	Bits 7-0	Output Endpoint Interrupt Flag. The output endpoint interrupt flag bits for a particular USB output endpoint are set to a "1" by the UBM when a successful completion of a transaction occurs to that out endpoint. When a bit is set, a USB interrupt will be generated. The interrupt flag will be cleared when the MCU reads the value from the USBVECINT register corresponding with this interrupt, or when it writes any value to the interrupt vector register. An interrupt flag can also be cleared by writing a zero to that bit location.

USBVECINT, USB Interrupt Vector Register

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**USBIV**					**0**
r0	r0	r-0	r-0	r-0	r-0	r-0	r0

USBIV	Bits 15-0	USB interrupt vector value. This register is to be accessed as a whole word only. When an interrupt is pending, reading this register results in a value that can be added to the program counter to handle the corresponding event. Writing to this register will clear all pending USB interrupt flags independent of the status of USBEN.

USBIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	—	—
02h	See Section 22.2.5		Highest
3Eh			Lowest

USBMAINT, Timestamp Maintenance Register

15	14	13	12	11	10	9	8
UTSEL			Reserved	TSE3	TSESEL		TSGEN
rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved						UTIE	UTIFG
r0	r0	r0	r0	r0	r0	rw-0	rw-0

Can be modified only when USBEN = 1

UTSEL Bits 15-13 USB timer selection

UTSEL	USB Timer Period	Approximate Frequency
000	4096 µs	~250 Hz (244 Hz)
001	2048 µs	~ 500 Hz (488 Hz)
010	1024 µs	~ 1 kHz (977 Hz)
011	512 µs	~ 2 kHz (1953 Hz)
100	256 µs	~ 4 kHz (3906 Hz)
101	128 µs	~ 8 kHz (7812 Hz)
110	64 µs	~ 16 kHz (15625 Hz)
111	32 µs	~ 31 kHz (31250 Hz)

Reserved Bit 12 Reserved. Read back as 0

TSE3 Bit 11 Timestamp Event #3 bit. This bit allows the triggering of a software-driven timestamp event (when TSESEL=11).

- 0 no TSE3 event signaled
- 1 TSE3 event signaled

TSESEL Bits 10-9 Timestamp Event Selection. TSE[2:0] are connected to the event multiplexer of the three DMA channels of the DMA controller if not otherwise noted in datasheet

TSESEL	Source of Timestamp Event
00	TSE0 (DMA0) signal is qualified timestamp event
01	TSE1 (DMA1) signal is qualified timestamp event
10	TSE2 (DMA2) signal is qualified timestamp event
11	Software-driven timestamp event

TSGEN Bit 8 Timestamp Generator Enable

- 0 Timestamp mechanism disabled
- 1 Timestamp mechanism enabled

Reserved Bits 7-2 Reserved. Read back as 0

UTIE Bit 1 USB timer interrupt enable bit

- 0 USB timer interrupt disabled
- 1 USB timer interrupt enabled

UTIFG Bit 0 USB timer interrupt flag

- 0 No interrupt pending
- 1 Interrupt pending

USBTSREG, USB Timestamp Register

15	14	13	12	11	10	9	8
TVAL							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

7	6	5	4	3	2	1	0
TVAL							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

Can be modified only when USBEN = 1

TVAL	Bits 15-0	Timestamp high register. The timestamp value is updated by hardware from the USB timer. A qualified timestamp trigger signal causes the current timer value to be latched into this register.

USBFN, USB Frame Number Register

15	14	13	12	11	10	9	8
Reserved					USBFN		
r0	r0	r0	r0	r0	r-0	r-0	r-0

7	6	5	4	3	2	1	0
USBFN							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

Reserved	Bits 15-11	Reserved. Read back as 0
USBFN	Bits 10-0	USB Frame Number register. The frame number bit values are updated by hardware; each USB frame with the frame number field value received in the USB start-of-frame packet. The frame number can be used as a timestamp. If the local (MSP430's) frame timer is not locked to the USB host's frame timer, then the frame number is automatically incremented from the previous value when a pseudo start-of-frame occurs.

USBCTL, USB Control Register

7	6	5	4	3	2	1	0
Reserved	FEN	RWUP	FRSTE	Reserved			DIR
r0	rw-0	rw-0	rw-0	r0	r0	r0	rw-0

Can be modified only when USBEN = 1

Reserved	Bit 7	Reserved. Read back as 0.
FEN	Bit 6	Function Enable Bit. This bit needs to be set to enable the USB device to respond to USB transactions. If this bit is not set, the UBM will ignore all USB transactions. It is cleared by a USB reset. (This bit is primarily intended for debugging.) 0 Function is disabled 1 Function is enabled
RWUP	Bit 5	Device Remote Wakeup request. The remote wake-up bit is set by software to request the suspend/resume logic to generate resume signaling upstream on the USB. This bit is used to exit a USB low-power suspend state when a remote wake-up event occurs. The bit is self-clearing. 0 Writing 0 has no effect 1 A Remote-Wakeup pulse will be generated
FRSTE	Bit 4	Function Reset Connection Enable. This bit selects whether a bus reset on the USB causes an internal reset of the USB module. 0 Bus reset does not cause a reset of the module 1 Bus reset does cause a reset of the module
Reserved	Bits 3-1	Reserved. Read back as 0.
DIR	Bit 0	Data response to setup packet interrupt status bit. Software must decode the request and set/clear this bit to reflect the data transfer direction. 0 USB data-OUT transaction (from host to device) 1 USB data-IN transaction (from device to host)

USBIE, USB Interrupt Enable Register

7	6	5	4	3	2	1	0
RSTRIE	SUSRIE	RESRIE	Reserved		SETUPIE	Reserved	STPOWIE
rw-0	rw-0	rw-0	r0	r0	rw-0	r0	rw-0

Can be modified only when USBEN = 1

RSTRIE	Bit 7	USB reset interrupt enable. Causes an interrupt to be generated if the RSTRIFG bit is set. 0 Function Reset interrupt disabled 1 Function Reset interrupt enabled
SUSRIE	Bit 6	Suspend interrupt enable. Causes an interrupt to be generated if the SUSRIFG bit is set. 0 Suspend interrupt disabled 1 Suspend interrupt enabled
RESRIE	Bit 5	Resume interrupt enable. Causes an interrupt to be generated if the RESRIFG bit is set. 0 Resume interrupt disabled 1 Resume interrupt enabled
Reserved	Bits 4-3	Reserved. Read back as 0.
SETUPIE	Bit 2	Setup interrupt enable. Causes an interrupt to be generated if the SETUPIFG bit is set. 0 Setup interrupt disabled 1 Setup interrupt enabled
Reserved	Bit 1	Reserved. Read back as 0.
STPOWIE	Bit 0	Setup Overwrite interrupt enable. Causes an interrupt to be generated if the STPOWIFG bit is set. 0 Setup Overwrite interrupt disabled 1 Setup Overwrite interrupt enabled

USBIFG, USB Interrupt Flag Register

7	6	5	4	3	2	1	0
RSTRIFG	SUSRIFG	RESRIFG	Reserved		SETUPIFG	Reserved	STPOWIFG
rw-0	rw-0	rw-0	r0	r0	rw-0	r0	rw-0

Can be modified only when USBEN = 1

RSTRIFG	Bit 7	USB reset request bit. This bit is set to one by hardware in response to the host initiating a USB port reset. A USB reset causes a reset of the USB module logic, but this bit will not be affected.
SUSRIFG	Bit 6	Suspend request bit. This bit is set by hardware in response to the host/hub causing a global or selective suspend condition.
RESRIFG	Bit 5	Resume request bit. This bit is set by hardware in response to the host/hub causing a resume event.
Reserved	Bits 4-3	Reserved. Read back as 0.
SETUPIFG	Bit 2	Setup transaction received bit. This bit is set by hardware when a SETUP transaction is received. As long as this bit is set, transactions on IN and OUT on endpoint-0 receive a NAK, regardless of their corresponding NAK bit value.
Reserved	Bit 1	Reserved. Read back as 0.
STPOWIFG	Bit 0	Setup overwrite bit. This bit is set by hardware when a setup packet is received while there is already a packet in the setup buffer.

USBFUNADR, USB Function Address Register

7	6	5	4	3	2	1	0
Reserved	FA6	FA5	FA4	FA3	FA2	FA1	FA0
r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Can be modified only when USBEN = 1

Reserved	Bit 7	Reserved. Read back as 0.
FA[6:0]	Bits 6-0	Function address (USB address 0 to 127). These bits define the current device address assigned to this USB device. Software must write a value from 0 to 127 when a Set-Address command is received from the host.

22.4.3 USB Buffer Registers and Memory

The data buffers for all endpoints, as well as the registers that define endpoints 1-7, are stored in the USB RAM buffer memory. Doing so allows for efficient, flexible use of this memory. The memory area is known as the USB buffer memory), and the registers that define its use are the buffer descriptor registers.

The buffer memory blocks are listed in Table 22-7. The registers are listed in Table 22-8. All addresses are expressed as offsets; the base address can be found in the device-specific datasheet.

All memory is byte and word accessible.

Table 22-7. USB Buffer Memory

Memory	Short Form	Access Type	Address Offset
Start of buffer space	USBSTABUFF	Read/Write	0000h
1904 bytes of configurable buffer space	⋮	Read/Write	⋮
End of buffer space	USBTOPBUFF	Read/Write	076Fh
Output endpoint_0 buffer	USBOEP0BUF	Read/Write	0770h
		Read/Write	⋮
		Read/Write	0777h
Input endpoint_0 buffer	USBIEP0BUF	Read/Write	0778h
		Read/Write	⋮
		Read/Write	077Fh
Setup Packet Block	USBSUBLK	Read/Write	0780h
		Read/Write	⋮
		Read/Write	0787h

Table 22-8. USB Buffer Descriptor Registers

Register		Short Form	Access Type	Address Offset
Output Endpoint_1	Configuration Register	USBOEPCNF_1	Read/Write	0788h
	X-buffer base address Register	USBOEPBBAX_1	Read/Write	0789h
	X-byte count Register	USBOEPBCTX_1	Read/Write	078Ah
	Y-buffer base address Register	USBOEPBBAY_1	Read/Write	078Dh
	Y-byte count Register	USBOEPBCTY_1	Read/Write	078Eh
	X/Y-buffer size Register	USBOEPSIZXY_1	Read/Write	078Fh
Output Endpoint_2	Configuration Register	USBOEPCNF_2	Read/Write	0790h
	X-buffer base address Register	USBOEPBBAX_2	Read/Write	0791h
	X-byte count Register	USBOEPBCTX_2	Read/Write	0792h
	Y-buffer base address Register	USBOEPBBAY_2	Read/Write	0795h
	Y-byte count Register	USBOEPBCTY_2	Read/Write	0796h
	X/Y-buffer size Register	USBOEPSIZXY_2	Read/Write	0797h
Output Endpoint_3	Configuration Register	USBOEPCNF_3	Read/Write	0798h
	X-buffer base address Register	USBOEPBBAX_3	Read/Write	0799h
	X-byte count Register	USBOEPBCTX_3	Read/Write	079Ah
	Y-buffer base address Register	USBOEPBBAY_3	Read/Write	079Dh
	Y-byte count Register	USBOEPBCTY_3	Read/Write	079Eh
	X/Y-buffer size Register	USBOEPSIZXY_3	Read/Write	079Fh
Output Endpoint_4	Configuration Register	USBOEPCNF_4	Read/Write	07A0h
	X-buffer base address Register	USBOEPBBAX_4	Read/Write	07A1h
	X-byte count Register	USBOEPBCTX_4	Read/Write	07A2h
	Y-buffer base address Register	USBOEPBBAY_4	Read/Write	07A5h
	Y-byte count Register	USBOEPBCTY_4	Read/Write	07A6h
	X/Y-buffer size Register	USBOEPSIZXY_4	Read/Write	07A7h

Table 22-8. USB Buffer Descriptor Registers (continued)

Register		Short Form	Access Type	Address Offset
Output Endpoint_5	Configuration Register	USBOEPCNF_5	Read/Write	07A8h
	X-buffer base address Register	USBOEPBBAX_5	Read/Write	07A9h
	X-byte count Register	USBOEPBCTX_5	Read/Write	07AAh
	Y-buffer base address Register	USBOEPBBAY_5	Read/Write	07ADh
	Y-byte count Register	USBOEPBCTY_5	Read/Write	07AEh
	X/Y-buffer size Register	USBOEPSIZXY_5	Read/Write	07AFh
Output Endpoint_6	Configuration Register	USBOEPCNF_6	Read/Write	07B0h
	X-buffer base address Register	USBOEPBBAX_6	Read/Write	07B1h
	X-byte count Register	USBOEPBCTX_6	Read/Write	07B2h
	Y-buffer base address Register	USBOEPBBAY_6	Read/Write	07B5h
	Y-byte count Register	USBOEPBCTY_6	Read/Write	07B6h
	X/Y-buffer size Register	USBOEPSIZXY_6	Read/Write	07B7h
Output Endpoint_7	Configuration Register	USBOEPCNF_7	Read/Write	07B8h
	X-buffer base address Register	USBOEPBBAX_7	Read/Write	07B9h
	X-byte count Register	USBOEPBCTX_7	Read/Write	07BAh
	Y-buffer base address Register	USBOEPBBAY_7	Read/Write	07BDh
	Y-byte count Register	USBOEPBCTY_7	Read/Write	07BEh
	X/Y-buffer size Register	USBOEPSIZXY_7	Read/Write	07BFh
Input Endpoint_1	Configuration Register	USBIEPCNF_1	Read/Write	07C8h
	X-buffer base address Register	USBIEPBBAX_1	Read/Write	07C9h
	X-byte count Register	USBIEPBCTX_1	Read/Write	07CAh
	Y-buffer base address Register	USBIEPBBAY_1	Read/Write	07CDh
	Y-byte count Register	USBIEPBCTY_1	Read/Write	07CEh
	X/Y-buffer size Register	USBIEPSIZXY_1	Read/Write	07CFh
Input Endpoint_2	Configuration Register	USBIEPCNF_2	Read/Write	07D0h
	X-buffer base address Register	USBIEPBBAX_2	Read/Write	07D1h
	X-byte count Register	USBIEPBCTX_2	Read/Write	07D2h
	Y-buffer base address Register	USBIEPBBAY_2	Read/Write	07D5h
	Y-byte count Register	USBIEPBCTY_2	Read/Write	07D6h
	X/Y-buffer size Register	USBIEPSIZXY_2	Read/Write	07D7h
Input Endpoint_3	Configuration Register	USBIEPCNF_3	Read/Write	07D8h
	X-buffer base address Register	USBIEPBBAX_3	Read/Write	07D9h
	X-byte count Register	USBIEPBCTX_3	Read/Write	07DAh
	Y-buffer base address Register	USBIEPBBAY_3	Read/Write	07DDh
	Y-byte count Register	USBIEPBCTY_3	Read/Write	07DEh
	X/Y-buffer size Register	USBIEPSIZXY_3	Read/Write	07DFh
Input Endpoint_4	Configuration Register	USBIEPCNF_4	Read/Write	07E0h
	X-buffer base address Register	USBIEPBBAX_4	Read/Write	07E1h
	X-byte count Register	USBIEPBCTX_4	Read/Write	07E2h
	Y-buffer base address Register	USBIEPBBAY_4	Read/Write	07E5h
	Y-byte count Register	USBIEPBCTY_4	Read/Write	07E6h
	X/Y-buffer size Register	USBIEPSIZXY_4	Read/Write	07E7h

Table 22-8. USB Buffer Descriptor Registers (continued)

Register		Short Form	Access Type	Address Offset
Input Endpoint_5	Configuration Register	USBIEPCNF_5	Read/Write	07E8h
	X-buffer base address Register	USBIEPBBAX_5	Read/Write	07E9h
	X-byte count Register	USBIEPBCTX_5	Read/Write	07EAh
	Y-buffer base address Register	USBIEPBBAY_5	Read/Write	07EDh
	Y-byte count Register	USBIEPBCTY_5	Read/Write	07EEh
	X/Y-buffer size Register	USBIEPSIZXY_5	Read/Write	07EFh
Input Endpoint_6	Configuration Register	USBIEPCNF_6	Read/Write	07F0h
	X-buffer base address Register	USBIEPBBAX_6	Read/Write	07F1h
	X-byte count Register	USBIEPBCTX_6	Read/Write	07F2h
	Y-buffer base address Register	USBIEPBBAY_6	Read/Write	07F5h
	Y-byte count Register	USBIEPBCTY_6	Read/Write	07F6h
	X/Y-buffer size Register	USBIEPSIZXY_6	Read/Write	07F7h
Input Endpoint_7	Configuration Register	USBIEPCNF_7	Read/Write	07F8h
	X-buffer base address Register	USBIEPBBAX_7	Read/Write	07F9h
	X-byte count Register	USBIEPBCTX_7	Read/Write	07FAh
	Y-buffer base address Register	USBIEPBBAY_7	Read/Write	07FDh
	Y-byte count Register	USBIEPBCTY_7	Read/Write	07FEh
	X/Y-buffer size Register	USBIEPSIZXY_7	Read/Write	07FFh

USBOEPCNF_n, Output Endpoint-n Configuration Register

7	6	5	4	3	2	1	0
UBME	Reserved	TOGGLE	DBUF	STALL	USBIIE	Reserved	
rw	r0	rw	rw	rw	rw	r0	r0

(shaded) Can be modified only when USBEN = 1

UBME Bit 7 UBM out endpoint-n enable. This bit is to be set/cleared by software.
- 0 UBM cannot use this endpoint
- 1 UBM can use this endpoint

Reserved Bit 6 Reserved. Read back as 0.

TOGGLE Bit 5 Toggle bit. The toggle bit is controlled by the UBM and is toggled at the end of a successful out data stage transaction, if a valid data packet is received and the data packet's packet ID matches the expected packet ID.

DBUF Bit 4 Double buffer enable. This bit can be set to enable the use of both the X and Y data packet buffers for USB transactions, for a particular out endpoint. Clearing it results in the use of single buffer mode. In this mode, only the X buffer is used.
- 0 Primary buffer only (X-buffer only)
- 1 Toggle bit selects buffer

STALL Bit 3 USB stall condition. This bit can be set to cause endpoint transactions to be stalled. When set, the hardware will automatically return a stall handshake to the host for any transaction received on endpoint-0. The stall bit is cleared automatically by the next setup transaction.
- 0 Indicates no stall
- 1 Indicates stall

USBIIE Bit 2 USB transaction interrupt indication enable. Can be set/cleared to define if interrupts are to be flagged in general. To generate an interrupt, the corresponding interrupt flag must be set (OEPIE).
- 0 Corresponding interrupt flag will not be set
- 1 Corresponding interrupt flag will be set

Reserved Bits 1-0 Reserved. Read back as 0.

USBOEPBBAX_n, Output Endpoint-n X-buffer Base Address Register

7	6	5	4	3	2	1	0
ADR							
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

ADR	Bits 7-0	X-buffer base address. These are the upper seven bits of the X-buffer's base address. The three LSBs are assumed to be zero, for a total of 11 bits. This value needs to be set by software. The UBM uses this value as the start address of a given transaction. It does not change this value at the end of a transaction.

USBOEPBCTX_n, Output Endpoint-n X-byte Count Register

7	6	5	4	3	2	1	0
NAK	CNT						
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

NAK	Bit 7	No-acknowledge status bit. The NAK status bit is set by the UBM at the end of a successful USB out transaction to that endpoint, in order to indicate that the USB endpoint-"n" buffer contains a valid data packet, and that the buffer data count value is valid. When this bit is set, all subsequent transactions to that endpoint will result in a NAK handshake response to the USB host. To re-enable this endpoint to receive another data packet from the host, this bit must be cleared. 0 No valid data in buffer. The buffer is ready to receive OUT packets from the host. 1 The buffer contains a valid packet from the host, and it has not been picked up (subsequent host-out requests receive a NAK)
CNT	Bits 6-0	X-buffer data count. The Out_EP-n data count value is set by the UBM when a new data packet is written to the X-buffer for that out endpoint. It is set to the number of bytes received in the data buffer.

USBOEPBBAY_n, Output Endpoint-n Y-buffer Base Address Register

7	6	5	4	3	2	1	0
ADR							
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

ADR	Bits 7-0	Y-buffer base address. These are the upper seven bits of the Y-buffer's base address. The three LSBs are assumed to be zero, for a total of 11 bits. This value needs to be set by software. The UBM uses this value as the start address of a given transaction. It does not change this value at the end of a transaction.

USBOEPBCTY_n, Output Endpoint-n X-byte Count Register

7	6	5	4	3	2	1	0
NAK	CNT						
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

NAK	Bit 7	No-acknowledge status bit. The NAK status bit is set by the UBM at the end of a successful USB out transaction to that endpoint, in order to indicate that the USB endpoint-"n" buffer contains a valid data packet, and that the buffer data count value is valid. When this bit is set, all subsequent transactions to that endpoint will result in a NAK handshake response to the USB host. To re-enable this endpoint to receive another data packet from the host, this bit must be cleared. 0 No valid data in buffer. The buffer is ready to receive OUT packets from the host. 1 The buffer contains a valid packet from the host, and it has not been picked up (subsequent host-out requests receive a NAK)
CNT	Bits 6-0	Y-buffer data count. The Out_EP-n data count value is set by the UBM when a new data packet is written to the X-buffer for that out endpoint. It is set to the number of bytes received in the data buffer.

USBOEPSIZXY_n, Output Endpoint-n X/Y-buffer Size Register

7	6	5	4	3	2	1	0
Reserved	SIZx						
r0	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

Reserved	Bit 7	Reserved. Read back as 0.
SIZx	Bits 6-0	Buffer size count. This value needs to be set by software to configure the size of the X and Y data packet buffers. Both buffers are set to the same size, based on this value. 000:0000b to 100:0000b are valid numbers for 0 to 64 bytes. Any value ≥ 100:0001b results in unpredictable results.

USBIEPCNF_n, Input Endpoint-n Configuration Register

7	6	5	4	3	2	1	0
UBME	Reserved	TOGGLE	DBUF	STALL	USBIIE	Reserved	
rw	r0	rw	rw	rw	rw	r0	r0

Can be modified only when USBEN = 1

UBME	Bit 7	UBM in endpoint-n enable. This value needs to be set/cleared by software. 0 UBM cannot use this endpoint 1 UBM can use this endpoint
Reserved	Bit 6	Reserved. Read back as 0.
TOGGLE	Bit 5	Toggle bit. The toggle bit is controlled by the UBM and is toggled at the end of a successful in data stage transaction, if a valid data packet is transmitted. If this bit is cleared, a DATA0 packet ID is transmitted in the data packet to the host. If this bit is set, a DATA1 packet ID is transmitted in the data packet.
DBUF	Bit 4	Double buffer enable. This bit can be set to enable the use of both the X and Y data packet buffers for USB transactions, for a particular out endpoint. Clearing it results in the use of single buffer mode. In this mode, only the X buffer is used. 0 Primary buffer only (X-buffer only) 1 Toggle bit selects buffer
STALL	Bit 3	USB stall condition. This bit can be set to cause endpoint transactions to be stalled. When set, the hardware will automatically return a stall handshake to the host for any transaction received on endpoint-0. The stall bit is cleared automatically by the next setup transaction. 0 Indicates no stall 1 Indicates stall
USBIIE	Bit 2	USB transaction interrupt indication enable. Can be set/cleared to define if interrupts are to be flagged in general. To generate an interrupt the corresponding interrupt flag must be set (OEPIE). 0 Corresponding interrupt flag will not be set 1 Corresponding interrupt flag will be set
Reserved	Bits 1-0	Reserved. Read back as 0.

USBIEPBBAX_n, Input Endpoint-n X-buffer Base Address Register

7	6	5	4	3	2	1	0
ADR							
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

ADR	Bits 7-0	X-buffer base address. These are the upper seven bits of the X-buffer's base address. The three LSBs are assumed to be zero, for a total of 11 bits. This value needs to be set by software. The UBM uses this value as the start address of a given transaction. It does not change this value at the end of a transaction.

USBIEPBCTX_n, Input Endpoint-n X-byte Count Register

7	6	5	4	3	2	1	0
NAK	**CNT**						
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

NAK	Bit 7	No-acknowledge status bit. The NAK status bit is set by the UBM at the end of a successful USB in transaction from that endpoint, in order to indicate that the EP-n in buffer is empty. For interrupt or bulk endpoints, when this bit is set, all subsequent transactions from that endpoint result in a NAK handshake response to the USB host. To re-enable this endpoint to transmit another data packet to the host, this bit must be cleared. 0 Buffer contains a valid data packet for the host 1 Buffer is empty (any host-In requests receive a NAK)
CNT	Bits 6-0	X-buffer data count. The In_EP-n X-buffer data count value must be set by software when a new data packet is written to the buffer. It should be the number of bytes in the data packet for interrupt, or bulk endpoint transfers. 000:0000b to 100:0000b are valid numbers for 0 to 64 bytes. Any value ≥ 100:0001b results in unpredictable results.

USBIEPBBAY_n, Input Endpoint-n Y-buffer Base Address Register

7	6	5	4	3	2	1	0
ADR							
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

ADR	Bits 7-0	Y-buffer base address. These are the upper seven bits of the Y-buffer's base address. The three LSBs are assumed to be zero, for a total of 11 bits. This value needs to be set by software. The UBM uses this value as the start address of a given transaction. It does not change this value at the end of a transaction.

USBIEPBCTY_n, Input Endpoint-n Y-byte Count Register

7	6	5	4	3	2	1	0
NAK	**CNT**						
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when USBEN = 1

NAK	Bit 7	No-acknowledge status bit. The NAK status bit is set by the UBM at the end of a successful USB in transaction from that endpoint, in order to indicate that the EP-n in buffer is empty. For interrupt or bulk endpoints, when this bit is set, all subsequent transactions from that endpoint result in a NAK handshake response to the host. To re-enable this endpoint to transmit another data packet to the host, this bit must be cleared. This bit is set by USB SW-init. 0 Buffer contains a valid data packet for host device 1 Buffer is empty (any host-in requests receive a NAK)
CNT	Bits 6-0	Y-Buffer data count. The In EP-n Y-buffer data count value needs to be set by software when a new data packet is written to the buffer. It should be the number of bytes in the data packet for interrupt, or bulk endpoint transfers. 000:0000b to 100:0000b are valid numbers for 0 to 64 bytes. Any value ≥ 100:0001b results in unpredictable results.

USBIEPSIZXY_n, Input Endpoint-n X/Y-buffer Size Register

7	6	5	4	3	2	1	0
Reserved	**SIZ**						
r0	rw	rw	rw	rw	rw	rw	rw

(shaded) Can be modified only when USBEN = 1

Reserved	Bit 7	Reserved. Read back as 0.
SIZ	Bits 6-0	Buffer size count. This value needs to be set by software to configure the size of the X and Y data packet buffers. Both buffers are set to the same size, based on this value. 000:0000b to 100:0000b are valid numbers for 0 to 64 bytes. Any value ≥ 100:0001b results in unpredictable results.

Embedded Emulation Module (EEM)

This chapter describes the embedded emulation module (EEM) that is implemented in all flash devices.

Topic **Page**

23.1 Embedded Emulation Module (EEM) Introduction **582**
23.2 EEM Building Blocks **584**
23.3 EEM Configurations **585**

23.1 Embedded Emulation Module (EEM) Introduction

Every MSP430 flash-based microcontroller implements an EEM. It is accessed and controlled through either 4-wire JTAG mode or Spy-Bi-Wire mode. Each implementation is device dependent and is described in Section 23.3, the EEM Configurations section, and the device-specific data sheet.

In general, the following features are available:

- Nonintrusive code execution with real-time breakpoint control
- Single-step, step-into, and step-over functionality
- Full support of all low-power modes
- Support for all system frequencies, for all clock sources
- Up to eight (device-dependent) hardware triggers/breakpoints on memory address bus (MAB) or memory data bus (MDB)
- Up to two (device-dependent) hardware triggers/breakpoints on CPU register write accesses
- MAB, MDB, and CPU register access triggers can be combined to form up to ten (device dependent) complex triggers/breakpoints
- Up to two (device dependent) cycle counters
- Trigger sequencing (device dependent)
- Storage of internal bus and control signals using an integrated trace buffer (device dependent)
- Clock control for timers, communication peripherals, and other modules on a global device level or on a per-module basis during an emulation stop

Figure 23-1 shows a simplified block diagram of the largest currently-available 5xx EEM implementation.

For more details on how the features of the EEM can be used together with the IAR Embedded Workbench ™ debugger, see the application report *Advanced Debugging Using the Enhanced Emulation Module* (SLAA263) at www.msp430.com. For usage with Code Composer Essentials (CCE), see the application report *Advanced Debugging Using the Enhanced Emulation Module* (SLAA393) at www.msp430.com. Most other debuggers supporting the MSP430 have the same or a similar feature set. For details, see the user's guide of the applicable debugger.

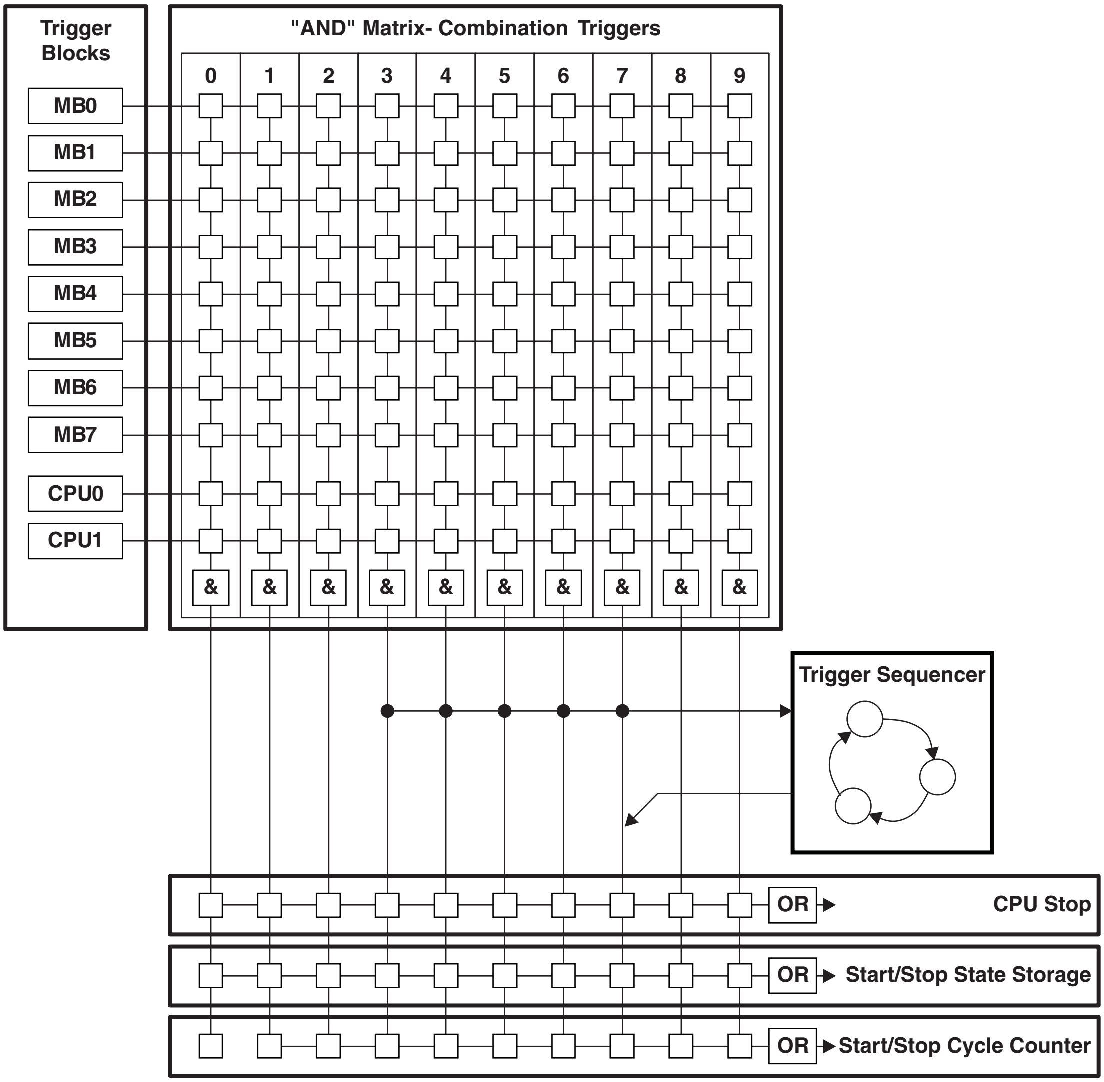


Figure 23-1. Large Implementation of EEM

23.2 EEM Building Blocks

23.2.1 Triggers

The event control in the EEM of the MSP430 system consists of triggers, which are internal signals indicating that a certain event has happened. These triggers may be used as simple breakpoints, but it is also possible to combine two or more triggers to allow detection of complex events and cause various reactions other than stopping the CPU.

In general, the triggers can be used to control the following functional blocks of the EEM:

- Breakpoints (CPU stop)
- State storage
- Sequencer
- Cycle counter

There are two different types of triggers – the memory trigger and the CPU register write trigger.

Each memory trigger block can be independently selected to compare either the MAB or the MDB with a given value. Depending on the implemented EEM, the comparison can be =, ≠, ≥, or ≤. The comparison can also be limited to certain bits with the use of a mask. The mask is either bit-wise or byte-wise, depending upon the device. In addition to selecting the bus and the comparison, the condition under which the trigger is active can be selected. The conditions include read access, write access, DMA access, and instruction fetch.

Each CPU register write trigger block can be independently selected to compare what is written into a selected register with a given value. The observed register can be selected for each trigger independently. The comparison can be =, ≠, ≥, or ≤. The comparison can also be limited to certain bits with the use of a bit mask.

Both types of triggers can be combined to form more complex triggers. For example, a complex trigger can signal when a particular value is written into a user-specified address.

23.2.2 Trigger Sequencer

The trigger sequencer allows the definition of a certain sequence of trigger signals before an event is accepted for a break or state storage event. Within the trigger sequencer, it is possible to use the following features:

- Four states (State 0 to State 3)
- Two transitions per state to any other state
- Reset trigger that resets the sequencer to State 0.

The trigger sequencer always starts at State 0 and must execute to State 3 to generate an action. If State 1 or State 2 are not required, they can be bypassed.

23.2.3 State Storage (Internal Trace Buffer)

The state storage function uses a built-in buffer to store MAB, MDB, and CPU control signal information (i.e., read, write, or instruction fetch) in a nonintrusive manner. The built-in buffer can hold up to eight entries. The flexible configuration allows the user to record the information of interest very efficiently.

23.2.4 Cycle Counter

The cycle counter provides one or two 40-bit counters to measure the cycles used by the CPU to execute certain tasks. On some devices, the cycle counter operation can be controlled using triggers. This allows, for example, conditional profiling, such as profiling a specific section of code.

23.2.5 Clock Control

The EEM provides device-dependent flexible clock control. This is useful in applications where a running clock is needed for peripherals after the CPU is stopped (e.g., to allow a UART module to complete its transfer of a character or to allow a timer to continue generating a PWM signal).

The clock control is flexible and supports both modules that need a running clock and modules that must be stopped when the CPU is stopped due to a breakpoint.

23.3 EEM Configurations

Table 23-1 gives an overview of the EEM configurations in the MSP430 5xx family. The implemented configuration is device dependent, and device-specific details can be found in the application report *Advanced Debugging Using the Enhanced Emulation Module (EEM) With CCE Version 3* (SLAA393), *MSP-FET430 Flash Emulation Tool (FET) (for Use With IAR v3+) User's Guide* (SLAU138), and *MSP-FET430 Flash Emulation Tool (FET) (for Use With CCE v3.1) User's Guide* (SLAU157).

Table 23-1. 5xx EEM Configurations

Feature	XS	S	M	L
Memory bus triggers	2 (=, ≠ only)	3	5	8
Memory bus trigger mask for	1) Low byte 2) High byte 3) Four upper addr bits	1) Low byte 2) High byte 3) Four upper addr bits	1) Low byte 2) High byte 3) Four upper addr bits	All 16 or 20 bits
CPU register write triggers	0	1	1	2
Combination triggers	2	4	6	10
Sequencer	No	No	Yes	Yes
State storage	No	No	No	Yes
Cycle counter	1	1	1	2 (including triggered start/stop)

In general, the following features can be found on any device:

- At least two MAB/MDB triggers supporting:
  - Distinction between CPU, DMA, read, and write accesses
  - =, ≠, ≥, or ≤ comparison (in XS, only =, ≠)
- At least two trigger combination registers
- Hardware breakpoints using the CPU stop reaction
- At least one 40-bit cycle counter
- Enhanced clock control with individual control of module clocks