PRELIMINARY INFORMATION

MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family

User's Guide

Literature Number: SLAU367B
October 2012–Revised October 2013

Contents

Preface **26**

1 System Resets, Interrupts, and Operating Modes, System Control Module (SYS) **28**

1.1 System Control Module (SYS) Introduction 29
1.2 System Reset and Initialization 29
1.2.1 Device Initial Conditions After System Reset 31
1.3 Interrupts 31
1.3.1 (Non)Maskable Interrupts (NMIs) 32
1.3.2 SNMI Timing 32
1.3.3 Maskable Interrupts 32
1.3.4 Interrupt Processing 33
1.3.5 Interrupt Nesting 34
1.3.6 Interrupt Vectors 34
1.3.7 SYS Interrupt Vector Generators 35
1.4 Operating Modes 37
1.4.1 Low-Power Modes and Clock Requests 39
1.4.2 Entering and Exiting Low-Power Modes LPM0 Through LPM4 40
1.4.3 Low Power Modes LPM3.5 and LPM4.5 (LPMx.5) 40
1.5 Principles for Low-Power Applications 42
1.6 Connection of Unused Pins 43
1.7 Reset Pin (**RST**/NMI) Configuration 43
1.8 Configuring JTAG Pins 43
1.9 Vacant Memory Space 44
1.10 Boot Code 44
1.10.1 IP Encapsulation (IPE) Instantiation by Boot Code 44
1.10.2 IP Encapsulation Signatures 44
1.10.3 IP Encapsulation Init Structure 44
1.10.4 IP Encapsulation Removal 45
1.11 Bootstrap Loader (BSL) 45
1.12 JTAG Mailbox (JMB) System 46
1.12.1 JMB Configuration 46
1.12.2 JMBOUT0 and JMBOUT1 Outgoing Mailbox 46
1.12.3 JMBIN0 and JMBIN1 Incoming Mailbox 46
1.12.4 JMB NMI Usage 47
1.13 JTAG and SBW Lock Mechanism Using the Electronic Fuse 47
1.13.1 JTAG and SBW Lock Without Password 47
1.13.2 JTAG and SBW Lock With Password 48
1.14 Device Descriptor Table 48
1.14.1 Identifying Device Type 49
1.14.2 TLV Descriptors 50
1.14.3 Calibration Values 51
1.15 SFR Registers 54
1.15.1 SFRIE1 Register 55
1.15.2 SFRIFG1 Register 56
1.15.3 SFRRPCR Register 58

1.16 SYS Registers 59
1.16.1 SYSCTL Register 60
1.16.2 SYSJMBC Register 61
1.16.3 SYSJMBI0 Register 62
1.16.4 SYSJMBI1 Register 62
1.16.5 SYSJMBO0 Register 63
1.16.6 SYSJMBO1 Register 63
1.16.7 SYSUNIV Register 64
1.16.8 SYSSNIV Register 64
1.16.9 SYSRSTIV Register 65

2 Power Management Module (PMM) and Supply Voltage Supervisor (SVS) 66
2.1 Power Management Module (PMM) Introduction 67
2.2 PMM Operation 68
2.2.1 V_{CORE} and the Regulator 68
2.2.2 Supply Voltage Supervisor 68
2.2.3 Supply Voltage Supervisor - Power-Up 69
2.2.4 LPM3.5 and LPM4.5 69
2.2.5 Low-Power Reset 69
2.2.6 Brownout Reset (BOR) 70
2.2.7 $\overline{\text{RST}}$/NMI 70
2.2.8 PMM Interrupts 70
2.2.9 Port I/O Control 70
2.3 PMM Registers 71
2.3.1 PMMCTL0 Register (offset = 00h) [reset = 9640h] 72
2.3.2 PMMCTL1 Register (offset = 02h) [reset = 9600h] 73
2.3.3 PMMIFG Register (offset = 0Ah) [reset = 0000h] 74
2.3.4 PM5CTL0 Register (offset = 10h) [reset = 0001h] 75

3 Clock System (CS) Module 76
3.1 Clock System Introduction 77
3.2 Clock System Operation 79
3.2.1 CS Module Features for Low-Power Applications 79
3.2.2 LFXT Oscillator 79
3.2.3 HFXT Oscillator 80
3.2.4 Internal Very-Low-Power Low-Frequency Oscillator (VLO) 81
3.2.5 Module Oscillator (MODOSC) 81
3.2.6 Digitally Controlled Oscillator (DCO) 81
3.2.7 Operation From Low-Power Modes, Requested by Peripheral Modules 81
3.2.8 CS Module Fail-Safe Operation 83
3.2.9 Synchronization of Clock Signals 85
3.3 Module Oscillator (MODOSC) 85
3.3.1 MODOSC Operation 85
3.4 CS Registers 86
3.4.1 CSCTL0 Register 87
3.4.2 CSCTL1 Register 87
3.4.3 CSCTL2 Register 88
3.4.4 CSCTL3 Register 89
3.4.5 CSCTL4 Register 90
3.4.6 CSCTL5 Register 92
3.4.7 CSCTL6 Register 93

4 CPUX 94
4.1 MSP430X CPU (CPUX) Introduction 95
4.2 Interrupts 97
4.3 CPU Registers 98

4.3.1 Program Counter (PC) 98
4.3.2 Stack Pointer (SP) 98
4.3.3 Status Register (SR) 100
4.3.4 Constant Generator Registers (CG1 and CG2) 101
4.3.5 General-Purpose Registers (R4 to R15) 102
4.4 Addressing Modes 104
4.4.1 Register Mode 105
4.4.2 Indexed Mode 106
4.4.3 Symbolic Mode 110
4.4.4 Absolute Mode 115
4.4.5 Indirect Register Mode 117
4.4.6 Indirect Autoincrement Mode 118
4.4.7 Immediate Mode 119
4.5 MSP430 and MSP430X Instructions 121
4.5.1 MSP430 Instructions 121
4.5.2 MSP430X Extended Instructions 126
4.6 Instruction Set Description 137
4.6.1 Extended Instruction Binary Descriptions 138
4.6.2 MSP430 Instructions 140
4.6.3 Extended Instructions 192
4.6.4 Address Instructions 235

5 FRAM Controller (FRCTL) 250

5.1 FRAM Introduction 251
5.2 FRAM Organization 251
5.3 FRCTL Module Operation 251
5.4 Programming FRAM Memory Devices 252
5.4.1 Programming FRAM Memory Via JTAG or Spy-Bi-Wire 252
5.4.2 Programming FRAM Memory Via Bootstrap Loader (BSL) 252
5.4.3 Programming FRAM Memory Via Custom Solution 252
5.5 Wait State Control 252
5.5.1 Wait State and Cache Hit 253
5.6 FRAM ECC 253
5.7 FRAM Write Back 253
5.8 FRAM Power Control 253
5.9 FRAM Cache 254
5.10 FRCTL Registers 255
5.10.1 FRCTL0 Register 256
5.10.2 GCCTL0 Register 257
5.10.3 GCCTL1 Register 258

6 Memory Protection Unit (MPU) 259

6.1 Memory Protection Unit (MPU) Introduction 260
6.2 MPU Segments 261
6.2.1 Main Memory Segments 261
6.2.2 IP Encapsulation Segment 262
6.2.3 Segment Border Setting 263
6.2.4 IP Encapsulation Border Settings 264
6.2.5 Information Memory 265
6.3 MPU Access Management Settings 265
6.4 MPU Violations 266
6.4.1 Interrupt Vector Table and Reset Vector 266
6.4.2 Violation Handling 266
6.5 MPU Lock 266
6.6 MPU Registers 267

6.6.1 MPUCTL0 Register 268
6.6.2 MPUCTL1 Register 269
6.6.3 MPUSEGB2 Register 270
6.6.4 MPUSEGB1 Register 271
6.6.5 MPUSAM Register 272
6.6.6 MPUIPC0 Register 274
6.6.7 MPUIPSEGB2 Register 275
6.6.8 MPUIPSEGB1 Register 276

7 DMA Controller 277

7.1 Direct Memory Access (DMA) Introduction 278
7.2 DMA Operation 280
7.2.1 DMA Addressing Modes 280
7.2.2 DMA Transfer Modes 281
7.2.3 Initiating DMA Transfers 287
7.2.4 Halting Executing Instructions for DMA Transfers 288
7.2.5 Stopping DMA Transfers 288
7.2.6 DMA Channel Priorities 288
7.2.7 DMA Transfer Cycle Time 289
7.2.8 Using DMA With System Interrupts 289
7.2.9 DMA Controller Interrupts 289
7.2.10 Using the eUSCI_B I^2C Module With the DMA Controller 290
7.2.11 Using ADC10 With the DMA Controller 291
7.3 DMA Registers 292
7.3.1 DMACTL0 Register 294
7.3.2 DMACTL1 Register 295
7.3.3 DMACTL2 Register 296
7.3.4 DMACTL3 Register 297
7.3.5 DMACTL4 Register 298
7.3.6 DMAxCTL Register 299
7.3.7 DMAxSA Register 301
7.3.8 DMAxDA Register 302
7.3.9 DMAxSZ Register 303
7.3.10 DMAIV Register 304

8 Digital I/O 305

8.1 Digital I/O Introduction 306
8.2 Digital I/O Operation 307
8.2.1 Input Registers (PxIN) 307
8.2.2 Output Registers (PxOUT) 307
8.2.3 Direction Registers (PxDIR) 307
8.2.4 Pullup or Pulldown Resistor Enable Registers (PxREN) 307
8.2.5 Function Select Registers (PxSEL0, PxSEL1) 308
8.2.6 Port Interrupts 308
8.3 I/O Configuration 310
8.3.1 Configuration After Reset 310
8.3.2 Configuration of Unused Port Pins 311
8.3.3 Configuration for LPMx.5 Low-Power Modes 311
8.4 Digital I/O Registers 313
8.4.1 P1IV Register 326
8.4.2 P2IV Register 326
8.4.3 P3IV Register 327
8.4.4 P4IV Register 327
8.4.5 PxIN Register 328
8.4.6 PxOUT Register 328

8.4.7 PxDIR Register 328
8.4.8 PxREN Register 329
8.4.9 PxSEL0 Register 329
8.4.10 PxSEL1 Register 329
8.4.11 PxSELC Register 330
8.4.12 PxIES Register 330
8.4.13 PxIE Register 330
8.4.14 PxIFG Register 331

9 Capacitive Touch IO 332
9.1 Capacitive Touch IO Introduction 333
9.2 Capacitive Touch IO Operation 334
9.3 CapTouch Registers 335
9.3.1 CAPTIOxCTL Register (offset = 0Eh) [reset = 0000h] 336

10 AES256 Accelerator 337
10.1 AES Accelerator Introduction 338
10.2 AES Accelerator Operation 339
10.2.1 Load the Key (128-Bit, 192-Bit, or 256-Bit Keylength) 340
10.2.2 Load the Data (128-Bit State) 340
10.2.3 Read the Data (128-Bit State) 341
10.2.4 Trigger an Encryption or Decryption 341
10.2.5 Encryption 342
10.2.6 Decryption 342
10.2.7 Decryption Key Generation 343
10.2.8 AES Key Buffer 344
10.2.9 Using the AES Accelerator With Low-Power Modes 345
10.2.10 AES Accelerator Interrupts 345
10.2.11 DMA Operation and Implementing Block Cipher Modes 345
10.3 AES Accelerator Registers 357
10.3.1 AESACTL0 Register 358
10.3.2 AESACTL1 Register 360
10.3.3 AESASTAT Register 361
10.3.4 AESAKEY Register 362
10.3.5 AESADIN Register 363
10.3.6 AESADOUT Register 364
10.3.7 AESAXDIN Register 365
10.3.8 AESAXIN Register 366

11 CRC Module 367
11.1 Cyclic Redundancy Check (CRC) Module Introduction 368
11.2 CRC Standard and Bit Order 368
11.3 CRC Checksum Generation 369
11.3.1 CRC Implementation 369
11.3.2 Assembler Examples 370
11.4 CRC Registers 372
11.4.1 CRCDI Register 373
11.4.2 CRCDIRB Register 373
11.4.3 CRCINIRES Register 374
11.4.4 CRCRESR Register 374

12 Watchdog Timer (WDT_A) 375
12.1 WDT_A Introduction 376
12.2 WDT_A Operation 378
12.2.1 Watchdog Timer Counter (WDTCNT) 378
12.2.2 Watchdog Mode 378

12.2.3 Interval Timer Mode 378
12.2.4 Watchdog Timer Interrupts 378
12.2.5 Fail-Safe Features 379
12.2.6 Operation in Low-Power Modes 379
12.3 WDT_A Registers 380
12.3.1 WDTCTL Register 381
13 Timer_A 382
13.1 Timer_A Introduction 383
13.2 Timer_A Operation 385
13.2.1 16-Bit Timer Counter 385
13.2.2 Starting the Timer 385
13.2.3 Timer Mode Control 386
13.2.4 Capture/Compare Blocks 389
13.2.5 Output Unit 391
13.2.6 Timer_A Interrupts 395
13.3 Timer_A Registers 397
13.3.1 TAxCTL Register 398
13.3.2 TAxR Register 399
13.3.3 TAxCCTLn Register 400
13.3.4 TAxCCRn Register 402
13.3.5 TAxIV Register 402
13.3.6 TAxEX0 Register 403
14 Timer_B 404
14.1 Timer_B Introduction 405
14.1.1 Similarities and Differences From Timer_A 405
14.2 Timer_B Operation 407
14.2.1 16-Bit Timer Counter 407
14.2.2 Starting the Timer 407
14.2.3 Timer Mode Control 408
14.2.4 Capture/Compare Blocks 411
14.2.5 Output Unit 414
14.2.6 Timer_B Interrupts 418
14.3 Timer_B Registers 420
14.3.1 TBxCTL Register 421
14.3.2 TBxR Register 423
14.3.3 TBxCCTLn Register 424
14.3.4 TBxCCRn Register 426
14.3.5 TBxIV Register 427
14.3.6 TBxEX0 Register 428
15 Real-Time Clock B (RTC_B) 429
15.1 Real-Time Clock RTC_B Introduction 430
15.2 RTC_B Operation 432
15.2.1 Real-Time Clock and Prescale Dividers 432
15.2.2 Real-Time Clock Alarm Function 432
15.2.3 Reading or Writing Real-Time Clock Registers 433
15.2.4 Real-Time Clock Interrupts 433
15.2.5 Real-Time Clock Calibration 435
15.2.6 Real-Time Clock Operation in LPMx.5 Low-Power Mode 436
15.3 RTC_B Registers 437
15.3.1 RTCCTL0 Register 439
15.3.2 RTCCTL1 Register 440
15.3.3 RTCCTL2 Register 441
15.3.4 RTCCTL3 Register 441

15.3.5 RTCSEC Register – Hexadecimal Format 442
15.3.6 RTCSEC Register – BCD Format 442
15.3.7 RTCMIN Register – Hexadecimal Format 443
15.3.8 RTCMIN Register – BCD Format 443
15.3.9 RTCHOUR Register – Hexadecimal Format 444
15.3.10 RTCHOUR Register – BCD Format 444
15.3.11 RTCDOW Register 445
15.3.12 RTCDAY Register – Hexadecimal Format 445
15.3.13 RTCDAY Register – BCD Format 445
15.3.14 RTCMON Register – Hexadecimal Format 446
15.3.15 RTCMON Register – BCD Format 446
15.3.16 RTCYEAR Register – Hexadecimal Format 447
15.3.17 RTCYEAR Register – BCD Format 447
15.3.18 RTCAMIN Register – Hexadecimal Format 448
15.3.19 RTCAMIN Register – BCD Format 448
15.3.20 RTCAHOUR Register – Hexadecimal Format 449
15.3.21 RTCAHOUR Register – BCD Format 449
15.3.22 RTCADOW Register 450
15.3.23 RTCADAY Register – Hexadecimal Format 451
15.3.24 RTCADAY Register – BCD Format 451
15.3.25 RTCPS0CTL Register 452
15.3.26 RTCPS1CTL Register 453
15.3.27 RTCPS0 Register 454
15.3.28 RTCPS1 Register 454
15.3.29 RTCIV Register 455
15.3.30 BIN2BCD Register 456
15.3.31 BCD2BIN Register 456

16 32-Bit Hardware Multiplier (MPY32) 457

16.1 32-Bit Hardware Multiplier (MPY32) Introduction 458
16.2 MPY32 Operation 460
16.2.1 Operand Registers 461
16.2.2 Result Registers 462
16.2.3 Software Examples 463
16.2.4 Fractional Numbers 464
16.2.5 Putting It All Together 467
16.2.6 Indirect Addressing of Result Registers 470
16.2.7 Using Interrupts 470
16.2.8 Using DMA 471
16.3 MPY32 Registers 472
16.3.1 MPY32CTL0 Register 474

17 REF_A 475

17.1 REF_A Introduction 476
17.2 Principle of Operation 477
17.2.1 Low-Power Operation 477
17.2.2 Reference System Requests 477
17.3 REF_A Registers 479
17.3.1 REFCTL0 Register (offset = 00h) [reset = 0000h] 480

18 ADC12_B 482

18.1 ADC12_B Introduction 483
18.2 ADC12_B Operation 485
18.2.1 12-Bit ADC Core 485
18.2.2 ADC12_B Inputs and Multiplexer 485

18.2.3 Voltage References 486
18.2.4 Auto Power Down 486
18.2.5 Sample Frequency Mode Selection 486
18.2.6 Sample and Conversion Timing 486
18.2.7 Conversion Memory 488
18.2.8 ADC12_B Conversion Modes 489
18.2.9 Window Comparator 494
18.2.10 Using the Integrated Temperature Sensor 495
18.2.11 ADC12_B Grounding and Noise Considerations 496
18.2.12 ADC12_B Interrupts 497
18.3 ADC12_B Registers 499
18.3.1 ADC12CTL0 Register (offset = 00h) [reset = 0000h] 505
18.3.2 ADC12CTL1 Register (offset = 02h) [reset = 0000h] 507
18.3.3 ADC12CTL2 Register (offset = 04h) [reset = 0020h] 509
18.3.4 ADC12CTL3 Register (offset = 06h) [reset = 0000h] 510
18.3.5 ADC12MEMx Register (x = 0 to 31) 511
18.3.6 ADC12MCTLx Register (x = 0 to 31) 512
18.3.7 ADC12HI Register (offset = 0Ah) [reset = 0FFFh] 514
18.3.8 ADC12LO Register (offset = 08h) [reset = 0000h] 514
18.3.9 ADC12IER0 Register (offset = 12h) [reset = 0000h] 515
18.3.10 ADC12IER1 Register (offset = 14h) [reset = 0000h] 517
18.3.11 ADC12IER2 Register (offset = 16h) [reset = 0000h] 519
18.3.12 ADC12IFGR0 Register (offset = 0Ch) [reset = 0000h] 520
18.3.13 ADC12IFGR1 Register (offset = 0Eh) [reset = 0000h] 522
18.3.14 ADC12IFGR2 Register (offset = 10h) [reset = 0000h] 524
18.3.15 ADC12IV Register (offset = 18h) [reset = 0000h] 525

19 Comparator E (COMP_E) Module 527
19.1 COMP_E Introduction 528
19.2 COMP_E Operation 529
19.2.1 Comparator 529
19.2.2 Analog Input Switches 529
19.2.3 Port Logic 529
19.2.4 Input Short Switch 529
19.2.5 Output Filter 530
19.2.6 Reference Voltage Generator 531
19.2.7 Port Disable Register (CEPD) 532
19.2.8 Comparator_E Interrupts 532
19.2.9 Comparator_E Used to Measure Resistive Elements 532
19.3 COMP_E Registers 535
19.3.1 CECTL0 Register (offset = 00h) [reset = 0000h] 536
19.3.2 CECTL1 Register (offset = 02h) [reset = 0000h] 537
19.3.3 CECTL2 Register (offset = 04h) [reset = 0000h] 539
19.3.4 CECTL3 Register (offset = 06h) [reset = 0000h] 540
19.3.5 CEINT Register (offset = 0Ch) [reset = 0000h] 542
19.3.6 CEIV Register (offset = 0Eh) [reset = 0000h] 543

20 Enhanced Universal Serial Communication Interface (eUSCI) – UART Mode 544
20.1 Enhanced Universal Serial Communication Interface A (eUSCI_A) Overview 545
20.2 eUSCI_A Introduction – UART Mode 545
20.3 eUSCI_A Operation – UART Mode 547
20.3.1 eUSCI_A Initialization and Reset 547
20.3.2 Character Format 547
20.3.3 Asynchronous Communication Format 547
20.3.4 Automatic Baud-Rate Detection 550

20.3.5 IrDA Encoding and Decoding 551
20.3.6 Automatic Error Detection 552
20.3.7 eUSCI_A Receive Enable 553
20.3.8 eUSCI_A Transmit Enable 553
20.3.9 UART Baud-Rate Generation 554
20.3.10 Setting a Baud Rate 556
20.3.11 Transmit Bit Timing - Error calculation 557
20.3.12 Receive Bit Timing – Error Calculation 557
20.3.13 Typical Baud Rates and Errors 558
20.3.14 Using the eUSCI_A Module in UART Mode With Low-Power Modes 560
20.3.15 eUSCI_A Interrupts 560
20.4 eUSCI_A UART Registers 562
20.4.1 UCAxCTLW0 Register 563
20.4.2 UCAxCTLW1 Register 564
20.4.3 UCAxBRW Register 565
20.4.4 UCAxMCTLW Register 565
20.4.5 UCAxSTATW Register 566
20.4.6 UCAxRXBUF Register 567
20.4.7 UCAxTXBUF Register 567
20.4.8 UCAxABCTL Register 568
20.4.9 UCAxIRCTL Register 569
20.4.10 UCAxIE Register 570
20.4.11 UCAxIFG Register 571
20.4.12 UCAxIV Register 572

21 Enhanced Universal Serial Communication Interface (eUSCI) – SPI Mode 573

21.1 Enhanced Universal Serial Communication Interfaces (eUSCI_A, eUSCI_B) Overview 574
21.2 eUSCI Introduction – SPI Mode 574
21.3 eUSCI Operation – SPI Mode 576
21.3.1 eUSCI Initialization and Reset 576
21.3.2 Character Format 577
21.3.3 Master Mode 577
21.3.4 Slave Mode 578
21.3.5 SPI Enable 579
21.3.6 Serial Clock Control 579
21.3.7 Using the SPI Mode With Low-Power Modes 580
21.3.8 SPI Interrupts 580
21.4 eUSCI_A SPI Registers 582
21.4.1 UCAxCTLW0 Register 583
21.4.2 UCAxBRW Register 585
21.4.3 UCAxSTATW Register 586
21.4.4 UCAxRXBUF Register 587
21.4.5 UCAxTXBUF Register 588
21.4.6 UCAxIE Register 589
21.4.7 UCAxIFG Register 590
21.4.8 UCAxIV Register 591
21.5 eUSCI_B SPI Registers 592
21.5.1 UCBxCTLW0 Register 593
21.5.2 UCBxBRW Register 595
21.5.3 UCBxSTATW Register 595
21.5.4 UCBxRXBUF Register 596
21.5.5 UCBxTXBUF Register 596
21.5.6 UCBxIE Register 597
21.5.7 UCBxIFG Register 597

21.5.8 UCBxIV Register 598

22 Enhanced Universal Serial Communication Interface (eUSCI) – I²C Mode 599

22.1 Enhanced Universal Serial Communication Interface B (eUSCI_B) Overview 600
22.2 eUSCI_B Introduction – I²C Mode 600
22.3 eUSCI_B Operation – I²C Mode 601
22.3.1 eUSCI_B Initialization and Reset 602
22.3.2 I²C Serial Data 602
22.3.3 I²C Addressing Modes 603
22.3.4 I²C Quick Setup 604
22.3.5 I²C Module Operating Modes 605
22.3.6 Glitch Filtering 615
22.3.7 I²C Clock Generation and Synchronization 615
22.3.8 Byte Counter 616
22.3.9 Multiple Slave Addresses 617
22.3.10 Using the eUSCI_B Module in I²C Mode With Low-Power Modes 617
22.3.11 eUSCI_B Interrupts in I²C Mode 618
22.4 eUSCI_B I2C Registers 621
22.4.1 UCBxCTLW0 Register 622
22.4.2 UCBxCTLW1 Register 624
22.4.3 UCBxBRW Register 626
22.4.4 UCBxSTATW 626
22.4.5 UCBxTBCNT Register 627
22.4.6 UCBxRXBUF Register 628
22.4.7 UCBxTXBUF 628
22.4.8 UCBxI2COA0 Register 629
22.4.9 UCBxI2COA1 Register 630
22.4.10 UCBxI2COA2 Register 630
22.4.11 UCBxI2COA3 Register 631
22.4.12 UCBxADDRX Register 631
22.4.13 UCBxADDMASK Register 632
22.4.14 UCBxI2CSA Register 632
22.4.15 UCBxIE Register 633
22.4.16 UCBxIFG Register 635
22.4.17 UCBxIV Register 637

23 Embedded Emulation Module (EEM) 638

23.1 Embedded Emulation Module (EEM) Introduction 639
23.2 EEM Building Blocks 641
23.2.1 Triggers 641
23.2.2 Trigger Sequencer 641
23.2.3 State Storage (Internal Trace Buffer) 641
23.2.4 Cycle Counter 641
23.2.5 Clock Control 642
23.3 EEM Configurations 642

Revision History 643

List of Figures

1-1. BOR, POR, and PUC Reset Circuit 30
1-2. Interrupt Priority 31
1-3. Interrupt Processing 33
1-4. Return From Interrupt 34
1-5. Operation Modes 38
1-6. Devices Descriptor Table 49
1-7. SFRIE1 Register 55
1-8. SFRIFG1 Register 56
1-9. SFRRPCR Register 58
1-10. SYSCTL Register 60
1-11. SYSJMBC Register 61
1-12. SYSJMBI0 Register 62
1-13. SYSJMBI1 Register 62
1-14. SYSJMBO0 Register 63
1-15. SYSJMBO1 Register 63
1-16. SYSUNIV Register 64
1-17. SYSSNIV Register 64
1-18. SYSRSTIV Register 65
2-1. PMM Block Diagram 67
2-2. Voltage Failure and Resulting PMM Actions 68
2-3. PMM Action at Device Power-Up 69
2-4. PMMCTL0 Register 72
2-5. PMMCTL1 Register 73
2-6. PMMIFG Register 74
2-7. PM5CTL0 Register 75
3-1. Clock System Block Diagram 78
3-2. Module Request Clock System 82
3-3. Oscillator Fault Logic 84
3-4. Switch MCLK from DCOCLK to LFXTCLK 85
3-5. CSCTL0 Register 87
3-6. CSCTL1 Register 87
3-7. CSCTL2 Register 88
3-8. CSCTL3 Register 89
3-9. CSCTL4 Register 90
3-10. CSCTL5 Register 92
3-11. CSCTL6 Register 93
4-1. MSP430X CPU Block Diagram 96
4-2. PC Storage on the Stack for Interrupts 97
4-3. Program Counter 98
4-4. PC Storage on the Stack for CALLA 98
4-5. Stack Pointer 99
4-6. Stack Usage 99
4-7. PUSHX.A Format on the Stack 99
4-8. PUSH SP, POP SP Sequence 99
4-9. SR Bits 100
4-10. Register-Byte and Byte-Register Operation 102
4-11. Register-Word Operation 102

4-12. Word-Register Operation 103
4-13. Register – Address-Word Operation 103
4-14. Address-Word – Register Operation 104
4-15. Indexed Mode in Lower 64 KB 106
4-16. Indexed Mode in Upper Memory 107
4-17. Overflow and Underflow for Indexed Mode 108
4-18. Example for Indexed Mode 109
4-19. Symbolic Mode Running in Lower 64 KB 111
4-20. Symbolic Mode Running in Upper Memory 112
4-21. Overflow and Underflow for Symbolic Mode 113
4-22. MSP430 Double-Operand Instruction Format 121
4-23. MSP430 Single-Operand Instructions 122
4-24. Format of Conditional Jump Instructions 123
4-25. Extension Word for Register Modes 126
4-26. Extension Word for Non-Register Modes 126
4-27. Example for Extended Register or Register Instruction 127
4-28. Example for Extended Immediate or Indexed Instruction 128
4-29. Extended Format I Instruction Formats 129
4-30. 20-Bit Addresses in Memory 129
4-31. Extended Format II Instruction Format 130
4-32. PUSHM and POPM Instruction Format 131
4-33. RRCM, RRAM, RRUM, and RLAM Instruction Format 131
4-34. BRA Instruction Format 131
4-35. CALLA Instruction Format 131
4-36. Decrement Overlap 157
4-37. Stack After a RET Instruction 176
4-38. Destination Operand—Arithmetic Shift Left 178
4-39. Destination Operand—Carry Left Shift 179
4-40. Rotate Right Arithmetically RRA.B and RRA.W 180
4-41. Rotate Right Through Carry RRC.B and RRC.W 181
4-42. Swap Bytes in Memory 188
4-43. Swap Bytes in a Register 188
4-44. Rotate Left Arithmetically—RLAM[.W] and RLAM.A 215
4-45. Destination Operand-Arithmetic Shift Left 216
4-46. Destination Operand-Carry Left Shift 217
4-47. Rotate Right Arithmetically RRAM[.W] and RRAM.A 218
4-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode 220
4-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode 220
4-50. Rotate Right Through Carry RRCM[.W] and RRCM.A 222
4-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode 224
4-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode 224
4-53. Rotate Right Unsigned RRUM[.W] and RRUM.A 225
4-54. Rotate Right Unsigned RRUX(.B,.A) – Register Mode 226
4-55. Swap Bytes SWPBX.A Register Mode 230
4-56. Swap Bytes SWPBX.A In Memory 230
4-57. Swap Bytes SWPBX[.W] Register Mode 231
4-58. Swap Bytes SWPBX[.W] In Memory 231
4-59. Sign Extend SXTX.A 232
4-60. Sign Extend SXTX[.W] 232

5-1. FRAM Controller Block Diagram 251
5-2. FRAM Power Control Diagram 254
5-3. FRCTL0 Register 256
5-4. GCCTL0 Register 257
5-5. GCCTL1 Register 258
6-1. Memory Protection Unit Overview 260
6-2. Segment Border Register 261
6-3. Segment Border Register - Fixed Bits 261
6-4. Segmentation of Main Memory 262
6-5. IP Encapsulation Access Rights Equivalent Schematic 263
6-6. MPUCTL0 Register 268
6-7. MPUCTL1 Register 269
6-8. MPUSEGB2 Register 270
6-9. MPUSEGB1 Register 271
6-10. MPUSAM Register 272
6-11. MPUIPC0 Register 274
6-12. MPUIPSEGB2 Register 275
6-13. MPUIPSEGB1 Register 276
7-1. DMA Controller Block Diagram 279
7-2. DMA Addressing Modes 280
7-3. DMA Single Transfer State Diagram 282
7-4. DMA Block Transfer State Diagram 284
7-5. DMA Burst-Block Transfer State Diagram 286
7-6. DMACTL0 Register 294
7-7. DMACTL1 Register 295
7-8. DMACTL2 Register 296
7-9. DMACTL3 Register 297
7-10. DMACTL4 Register 298
7-11. DMAxCTL Register 299
7-12. DMAxSA Register 301
7-13. DMAxDA Register 302
7-14. DMAxSZ Register 303
7-15. DMAIV Register 304
8-1. P1IV Register 326
8-2. P2IV Register 326
8-3. P3IV Register 327
8-4. P4IV Register 327
8-5. PxIN Register 328
8-6. PxOUT Register 328
8-7. PxDIR Register 328
8-8. PxREN Register 329
8-9. PxSEL0 Register 329
8-10. PxSEL1 Register 329
8-11. PxSELC Register 330
8-12. PxIES Register 330
8-13. PxIE Register 330
8-14. PxIFG Register 331
9-1. Capacitive Touch IO Principle 333
9-2. Capacitive Touch IO Block Diagram 334

9-3. CAPTIOxCTL Register 336
10-1. AES Accelerator Block Diagram 338
10-2. AES State Array Input and Output 339
10-3. AES Encryption Process for 128-Bit Key 342
10-4. AES Decryption Process Using AESOPx = 01 for 128-bit key 343
10-5. AES Decryption Process using AESOPx = 10 and 11 for 128-bit key 344
10-6. ECB encryption 347
10-7. ECB decryption 348
10-8. CBC Encryption 349
10-9. CBC Decryption 350
10-10. OFB Encryption 352
10-11. OFB Decryption 353
10-12. CFB Encryption 355
10-13. CFB Decryption 356
10-14. AESACTL0 Register 358
10-15. AESACTL1 Register 360
10-16. AESASTAT Register 361
10-17. AESAKEY Register 362
10-18. AESADIN Register 363
10-19. AESADOUT Register 364
10-20. AESAXDIN Register 365
10-21. AESAXIN Register 366
11-1. LFSR Implementation of CRC-CCITT Standard, Bit 0 is the MSB of the Result 368
11-2. Implementation of CRC-CCITT Using the CRCDI and CRCINIRES Registers 370
11-3. CRCDI Register 373
11-4. CRCDIRB Register 373
11-5. CRCINIRES Register 374
11-6. CRCRESR Register 374
12-1. Watchdog Timer Block Diagram 377
12-2. WDTCTL Register 381
13-1. Timer_A Block Diagram 384
13-2. Up Mode 386
13-3. Up Mode Flag Setting 386
13-4. Continuous Mode 387
13-5. Continuous Mode Flag Setting 387
13-6. Continuous Mode Time Intervals 387
13-7. Up/Down Mode 388
13-8. Up/Down Mode Flag Setting 388
13-9. Output Unit in Up/Down Mode 389
13-10. Capture Signal (SCS = 1) 390
13-11. Capture Cycle 390
13-12. Output Example – Timer in Up Mode 392
13-13. Output Example – Timer in Continuous Mode 393
13-14. Output Example – Timer in Up/Down Mode 394
13-15. Capture/Compare Interrupt Flag 395
13-16. TAxCTL Register 398
13-17. TAxR Register 399
13-18. TAxCCTLn Register 400
13-19. TAxCCRn Register 402

13-20. TAxIV Register 402
13-21. TAxEX0 Register 403
14-1. Timer_B Block Diagram 406
14-2. Up Mode 408
14-3. Up Mode Flag Setting 408
14-4. Continuous Mode 409
14-5. Continuous Mode Flag Setting 409
14-6. Continuous Mode Time Intervals 409
14-7. Up/Down Mode 410
14-8. Up/Down Mode Flag Setting 410
14-9. Output Unit in Up/Down Mode 411
14-10. Capture Signal (SCS = 1) 412
14-11. Capture Cycle 412
14-12. Output Example – Timer in Up Mode 415
14-13. Output Example – Timer in Continuous Mode 416
14-14. Output Example – Timer in Up/Down Mode 417
14-15. Capture/Compare TBxCCR0 Interrupt Flag 418
14-16. TBxCTL Register 421
14-17. TBxR Register 423
14-18. TBxCCTLn Register 424
14-19. TBxCCRn Register 426
14-20. TBxIV Register 427
14-21. TBxEX0 Register 428
15-1. RTC_B Block Diagram 431
15-2. RTCCTL0 Register 439
15-3. RTCCTL1 Register 440
15-4. RTCCTL2 Register 441
15-5. RTCCTL3 Register 441
15-6. RTCSEC Register 442
15-7. RTCSEC Register 442
15-8. RTCMIN Register 443
15-9. RTCMIN Register 443
15-10. RTCHOUR Register 444
15-11. RTCHOUR Register 444
15-12. RTCDOW Register 445
15-13. RTCDAY Register 445
15-14. RTCDAY Register 445
15-15. RTCMON Register 446
15-16. RTCMON Register 446
15-17. RTCYEAR Register 447
15-18. RTCYEAR Register 447
15-19. RTCAMIN Register 448
15-20. RTCAMIN Register 448
15-21. RTCAHOUR Register 449
15-22. RTCAHOUR Register 449
15-23. RTCADOW Register 450
15-24. RTCADAY Register 451
15-25. RTCADAY Register 451
15-26. RTCPS0CTL Register 452

15-27. RTCPS1CTL Register 453
15-28. RTCPS0 Register 454
15-29. RTCPS1 Register 454
15-30. RTCIV Register 455
15-31. BIN2BCD Register 456
15-32. BCD2BIN Register 456
16-1. MPY32 Block Diagram 459
16-2. Q15 Format Representation 464
16-3. Q14 Format Representation 464
16-4. Saturation Flow Chart 466
16-5. Multiplication Flow Chart 468
16-6. MPY32CTL0 Register 474
17-1. REF_A Block Diagram 476
17-2. REFCTL0 Register 480
18-1. ADC12_B Block Diagram 484
18-2. Analog Multiplexer 486
18-3. Extended Sample Mode in 12-Bit Mode 487
18-4. Pulse Sample Mode in 12-Bit Mode 488
18-5. Analog Input Equivalent Circuit 488
18-6. Single-Channel Single-Conversion Mode 490
18-7. Sequence-of-Channels Mode 491
18-8. Repeat-Single-Channel Mode 492
18-9. Repeat-Sequence-of-Channels Mode 493
18-10. Typical Temperature Sensor Transfer Function 495
18-11. ADC12_B Grounding and Noise Considerations 496
18-12. ADC12CTL0 Register 505
18-13. ADC12CTL1 Register 507
18-14. ADC12CTL2 Register 509
18-15. ADC12CTL3 Register 510
18-16. ADC12MEMx Register 511
18-17. ADC12MCTLx Register 512
18-18. ADC12HI Register 514
18-19. ADC12LO Register 514
18-20. ADC12IER0 Register 515
18-21. ADC12IER1 Register 517
18-22. ADC12IER2 Register 519
18-23. ADC12IFGR0 Register 520
18-24. ADC12IFGR1 Register 522
18-25. ADC12IFGR2 Register 524
18-26. ADC12IV Register 525
19-1. Comparator_E Block Diagram 528
19-2. Comparator_E Sample-And-Hold 530
19-3. RC-Filter Response at the Output of the Comparator 531
19-4. Reference Generator Block Diagram 531
19-5. Transfer Characteristic and Power Dissipation in a CMOS Inverter and Buffer 532
19-6. Temperature Measurement System 533
19-7. Timing for Temperature Measurement Systems 533
19-8. CECTL0 Register 536
19-9. CECTL1 Register 537

19-10. CECTL2 Register 539
19-11. CECTL3 Register 540
19-12. CEINT Register 542
19-13. CEIV Register 543
20-1. eUSCI_Ax Block Diagram – UART Mode (UCSYNC = 0) 546
20-2. Character Format 547
20-3. Idle-Line Format 548
20-4. Address-Bit Multiprocessor Format 549
20-5. Auto Baud-Rate Detection – Break/Synch Sequence 550
20-6. Auto Baud-Rate Detection – Synch Field 550
20-7. UART vs IrDA Data Format 551
20-8. Glitch Suppression, eUSCI_A Receive Not Started 553
20-9. Glitch Suppression, eUSCI_A Activated 553
20-10. BITCLK Baud-Rate Timing With UCOS16 = 0 554
20-11. Receive Error 558
20-12. UCAxCTLW0 Register 563
20-13. UCAxCTLW1 Register 564
20-14. UCAxBRW Register 565
20-15. UCAxMCTLW Register 565
20-16. UCAxSTATW Register 566
20-17. UCAxRXBUF Register 567
20-18. UCAxTXBUF Register 567
20-19. UCAxABCTL Register 568
20-20. UCAxIRCTL Register 569
20-21. UCAxIE Register 570
20-22. UCAxIFG Register 571
20-23. UCAxIV Register 572
21-1. eUSCI Block Diagram – SPI Mode 575
21-2. eUSCI Master and External Slave (UCSTEM = 0) 577
21-3. eUSCI Slave and External Master 578
21-4. eUSCI SPI Timing With UCMSB = 1 580
21-5. UCAxCTLW0 Register 583
21-6. UCAxBRW Register 585
21-7. UCAxSTATW Register 586
21-8. UCAxRXBUF Register 587
21-9. UCAxTXBUF Register 588
21-10. UCAxIE Register 589
21-11. UCAxIFG Register 590
21-12. UCAxIV Register 591
21-13. UCBxCTLW0 Register 593
21-14. UCBxBRW Register 595
21-15. UCBxSTATW Register 595
21-16. UCBxRXBUF Register 596
21-17. UCBxTXBUF Register 596
21-18. UCBxIE Register 597
21-19. UCBxIFG Register 597
21-20. UCBxIV Register 598
22-1. eUSCI_B Block Diagram – I^2C Mode 601
22-2. I^2C Bus Connection Diagram 602

22-3. I²C Module Data Transfer 603
22-4. Bit Transfer on I²C Bus 603
22-5. I²C Module 7-Bit Addressing Format 603
22-6. I²C Module 10-Bit Addressing Format 604
22-7. I²C Module Addressing Format With Repeated START Condition 604
22-8. I²C Time-Line Legend 606
22-9. I²C Slave Transmitter Mode 607
22-10. I²C Slave Receiver Mode 608
22-11. I²C Slave 10-Bit Addressing Mode 609
22-12. I²C Master Transmitter Mode 611
22-13. I²C Master Receiver Mode 613
22-14. I²C Master 10-Bit Addressing Mode 614
22-15. Arbitration Procedure Between Two Master Transmitters 614
22-16. Synchronization of Two I²C Clock Generators During Arbitration 615
22-17. UCBxCTLW0 Register 622
22-18. UCBxCTLW1 Register 624
22-19. UCBxBRW Register 626
22-20. UCBxSTATW Register 626
22-21. UCBxTBCNT Register 627
22-22. UCBxRXBUF Register 628
22-23. UCBxTXBUF Register 628
22-24. UCBxI2COA0 Register 629
22-25. UCBxI2COA1 Register 630
22-26. UCBxI2COA2 Register 630
22-27. UCBxI2COA3 Register 631
22-28. UCBxADDRX Register 631
22-29. UCBxADDMASK Register 632
22-30. UCBxI2CSA Register 632
22-31. UCBxIE Register 633
22-32. UCBxIFG Register 635
22-33. UCBxIV Register 637
23-1. Large Implementation of EEM 640

List of Tables

1-1. Interrupt Sources, Flags, and Vectors 34
1-2. Operation Modes 39
1-3. Requested vs Actual LPM 39
1-4. Connection of Unused Pins 43
1-5. IPE_Signatures 44
1-6. IPE_Init_Structure 45
1-7. Tag Values 50
1-8. REF Calibration Tags 51
1-9. ADC Calibration Tags 52
1-10. Random Number Tags 53
1-11. BSL Configuration Tags 53
1-12. BSL_COM_IF Values 54
1-13. BSL_CIF_CONFIG Values 54
1-14. SFR Registers 54
1-15. SFRIE1 Register Description 55
1-16. SFRIFG1 Register Description 56
1-17. SFRRPCR Register Description 58
1-18. SYS Registers 59
1-19. SYSCTL Register Description 60
1-20. SYSJMBC Register Description 61
1-21. SYSJMBI0 Register Description 62
1-22. SYSJMBI1 Register Description 62
1-23. SYSJMBO0 Register Description 63
1-24. SYSJMBO1 Register Description 63
1-25. SYSUNIV Register Description 64
1-26. SYSSNIV Register Description 64
1-27. SYSRSTIV Register Description 65
2-1. PMM Registers 71
2-2. PMMCTL0 Register Description 72
2-3. PMMCTL1 Register Description 73
2-4. PMMIFG Register Description 74
2-5. PM5CTL0 Register Description 75
3-1. HFFREQ Settings 80
3-2. System Clocks, Power Modes, and Clock Requests 83
3-3. CS Registers 86
3-4. CSCTL0 Register Description 87
3-5. CSCTL1 Register Description 87
3-6. CSCTL2 Register Description 88
3-7. CSCTL3 Register Description 89
3-8. CSCTL4 Register Description 90
3-9. CSCTL5 Register Description 92
3-10. CSCTL6 Register Description 93
4-1. SR Bit Description 100
4-2. Values of Constant Generators CG1, CG2 101
4-3. Source and Destination Addressing 104
4-4. MSP430 Double-Operand Instructions 122
4-5. MSP430 Single-Operand Instructions 122

4-6. Conditional Jump Instructions 123
4-7. Emulated Instructions 123
4-8. Interrupt, Return, and Reset Cycles and Length 124
4-9. MSP430 Format II Instruction Cycles and Length 124
4-10. MSP430 Format I Instructions Cycles and Length 125
4-11. Description of the Extension Word Bits for Register Mode 126
4-12. Description of Extension Word Bits for Non-Register Modes 127
4-13. Extended Double-Operand Instructions 128
4-14. Extended Single-Operand Instructions 130
4-15. Extended Emulated Instructions 132
4-16. Address Instructions, Operate on 20-Bit Register Data 133
4-17. MSP430X Format II Instruction Cycles and Length 134
4-18. MSP430X Format I Instruction Cycles and Length 135
4-19. Address Instruction Cycles and Length 136
4-20. Instruction Map of MSP430X 137
5-1. FRCTL Registers 255
5-2. FRCTL0 Register Description 256
5-3. GCCTL0 Register Description 257
5-4. GCCTL1 Register Description 258
6-1. IP Encapsulation Access Rights 263
6-2. MPU Border Selection Example 64k (004000h to 013FFFh) 264
6-3. Segment Access Rights 265
6-4. MPU Registers 267
6-5. MPUCTL0 Register Description 268
6-6. MPUCTL1 Register Description 269
6-7. MPUSEGB2 Register Description 270
6-8. MPUSEGB1 Register Description 271
6-9. MPUSAM Register Description 272
6-10. MPUIPC0 Register Description 274
6-11. MPUIPSEGB2 Register Description 275
6-12. MPUIPSEGB1 Register Description 276
7-1. DMA Transfer Modes 281
7-2. DMA Trigger Operation 288
7-3. Maximum Single-Transfer DMA Cycle Time 289
7-4. DMA Registers 292
7-5. DMACTL0 Register Description 294
7-6. DMACTL1 Register Description 295
7-7. DMACTL2 Register Description 296
7-8. DMACTL3 Register Description 297
7-9. DMACTL4 Register Description 298
7-10. DMAxCTL Register Description 299
7-11. DMAxSA Register Description 301
7-12. DMAxDA Register Description 302
7-13. DMAxSZ Register Description 303
7-14. DMAIV Register Description 304
8-1. I/O Configuration 307
8-2. I/O Function Selection 308
8-3. Digital I/O Registers 313
8-4. P1IV Register Description 326

8-5. P2IV Register Description 326
8-6. P3IV Register Description 327
8-7. P4IV Register Description 327
8-8. PxIN Register Description 328
8-9. PxOUT Register Description 328
8-10. P1DIR Register Description 328
8-11. PxREN Register Description 329
8-12. PxSEL0 Register Description 329
8-13. PxSEL1 Register Description 329
8-14. PxSELC Register Description 330
8-15. PxIES Register Description 330
8-16. PxIE Register Description 330
8-17. PxIFG Register Description 331
9-1. CapTouch Registers 335
9-2. CAPTIOxCTL Register Description 336
10-1. AES Operation Modes Overview 339
10-2. 'AES trigger 0-2' Operation When AESCMEN = 1 345
10-3. AES and DMA Configuration for ECB Encryption 347
10-4. AES DMA Configuration for ECB Decryption 348
10-5. AES and DMA Configuration for CBC Encryption 349
10-6. AES and DMA Configuration for CBC Decryption 350
10-7. AES and DMA Configuration for OFB Encryption 352
10-8. AES and DMA Configuration for OFB Decryption 353
10-9. AES and DMA Configuration for CFB Encryption 355
10-10. AES and DMA Configuration for CFB Decryption 356
10-11. AES256 Registers 357
10-12. AESACTL0 Register Description 358
10-13. AESACTL1 Register Description 360
10-14. AESASTAT Register Description 361
10-15. AESAKEY Register Description 362
10-16. AESADIN Register Description 363
10-17. AESADOUT Register Description 364
10-18. AESAXDIN Register Description 365
10-19. AESAXIN Register Description 366
11-1. CRC Registers 372
11-2. CRCDI Register Description 373
11-3. CRCDIRB Register Description 373
11-4. CRCINIRES Register Description 374
11-5. CRCRESR Register Description 374
12-1. WDT_A Registers 380
12-2. WDTCTL Register Description 381
13-1. Timer Modes 386
13-2. Output Modes 391
13-3. Timer_A Registers 397
13-4. TAxCTL Register Description 398
13-5. TAxR Register Description 399
13-6. TAxCCTLn Register Description 400
13-7. TAxCCRn Register Description 402
13-8. TAxIV Register Description 402

13-9. TAxEX0 Register Description 403
14-1. Timer Modes 408
14-2. TBxCLn Load Events 413
14-3. Compare Latch Operating Modes 414
14-4. Output Modes 414
14-5. Timer_B Registers 420
14-6. TBxCTL Register Description 421
14-7. TBxR Register Description 423
14-8. TBxCCTLn Register Description 424
14-9. TBxCCRn Register Description 426
14-10. TBxIV Register Description 427
14-11. TBxEX0 Register Description 428
15-1. RTC_B Registers 437
15-2. RTCCTL0 Register Description 439
15-3. RTCCTL1 Register Description 440
15-4. RTCCTL2 Register Description 441
15-5. RTCCTL3 Register Description 441
15-6. RTCSEC Register Description 442
15-7. RTCSEC Register Description 442
15-8. RTCMIN Register Description 443
15-9. RTCMIN Register Description 443
15-10. RTCHOUR Register Description 444
15-11. RTCHOUR Register Description 444
15-12. RTCDOW Register Description 445
15-13. RTCDAY Register Description 445
15-14. RTCDAY Register Description 445
15-15. RTCMON Register Description 446
15-16. RTCMON Register Description 446
15-17. RTCYEAR Register Description 447
15-18. RTCYEAR Register Description 447
15-19. RTCAMIN Register Description 448
15-20. RTCAMIN Register Description 448
15-21. RTCAHOUR Register Description 449
15-22. RTCAHOUR Register Description 449
15-23. RTCADOW Register Description 450
15-24. RTCADAY Register Description 451
15-25. RTCADAY Register Description 451
15-26. RTCPS0CTL Register Description 452
15-27. RTCPS1CTL Register Description 453
15-28. RTCPS0 Register Description 454
15-29. RTCPS1 Register Description 454
15-30. RTCIV Register Description 455
15-31. BIN2BCD Register Description 456
15-32. BCD2BIN Register Description 456
16-1. Result Availability (MPYFRAC = 0, MPYSAT = 0) 460
16-2. OP1 Registers 461
16-3. OP2 Registers 461
16-4. SUMEXT and MPYC Contents 462
16-5. Result Availability in Fractional Mode (MPYFRAC = 1, MPYSAT = 0) 465

16-6. Result Availability in Saturation Mode (MPYSAT = 1) 466
16-7. MPY32 Registers 472
16-8. Alternative Registers 473
16-9. MPY32CTL0 Register Description 474
17-1. REF_A Registers 479
17-2. REFCTL0 Register Description 480
18-1. ADC12_B Conversion Result Formats 489
18-2. Conversion Mode Summary 489
18-3. ADC12_B Registers 499
18-4. ADC12CTL0 Register Description 505
18-5. ADC12CTL1 Register Description 507
18-6. ADC12CTL2 Register Description 509
18-7. ADC12CTL3 Register Description 510
18-8. ADC12MEMx Register Description 511
18-9. ADC12MCTLx Register Description 512
18-10. ADC12HI Register Description 514
18-11. ADC12LO Register Description 514
18-12. ADC12IER0 Register Description 515
18-13. ADC12IER1 Register Description 517
18-14. ADC12IER2 Register Description 519
18-15. ADC12IFGR0 Register Description 520
18-16. ADC12IFGR1 Register Description 522
18-17. ADC12IFGR2 Register Description 524
18-18. ADC12IV Register Description 525
19-1. COMP_E Registers 535
19-2. CECTL0 Register Description 536
19-3. CECTL1 Register Description 537
19-4. CECTL2 Register Description 539
19-5. CECTL3 Register Description 540
19-6. CEINT Register Description 542
19-7. CEIV Register Description 543
20-1. Receive Error Conditions 552
20-2. Modulation Pattern Examples 554
20-3. BITCLK16 Modulation Pattern 555
20-4. UCBRSx Settings for Fractional Portion of N = f_{BRCLK}/Baudrate 556
20-5. Recommended Settings for Typical Crystals and Baudrates 559
20-6. UART State Change Interrupt Flags 561
20-7. eUSCI_A UART Registers 562
20-8. UCAxCTLW0 Register Description 563
20-9. UCAxCTLW1 Register Description 564
20-10. UCAxBRW Register Description 565
20-11. UCAxMCTLW Register Description 565
20-12. UCAxSTATW Register Description 566
20-13. UCAxRXBUF Register Description 567
20-14. UCAxTXBUF Register Description 567
20-15. UCAxABCTL Register Description 568
20-16. UCAxIRCTL Register Description 569
20-17. UCAxIE Register Description 570
20-18. UCAxIFG Register Description 571

20-19. UCAxIV Register Description 572
21-1. UCxSTE Operation 576
21-2. eUSCI_A SPI Registers 582
21-3. UCAxCTLW0 Register Description 583
21-4. UCAxBRW Register Description 585
21-5. UCAxSTATW Register Description 586
21-6. UCAxRXBUF Register Description 587
21-7. UCAxTXBUF Register Description 588
21-8. UCAxIE Register Description 589
21-9. UCAxIFG Register Description 590
21-10. UCAxIV Register Description 591
21-11. eUSCI_B SPI Registers 592
21-12. UCBxCTLW0 Register Description 593
21-13. UCBxBRW Register Description 595
21-14. UCBxSTATW Register Description 595
21-15. UCBxRXBUF Register Description 596
21-16. UCBxTXBUF Register Description 596
21-17. UCBxIE Register Description 597
21-18. UCBxIFG Register Description 597
21-19. UCBxIV Register Description 598
22-1. Glitch Filter Length Selection Bits 615
22-2. I^2C State Change Interrupt Flags 619
22-3. eUSCI_B Registers 621
22-4. UCBxCTLW0 Register Description 622
22-5. UCBxCTLW1 Register Description 624
22-6. UCBxBRW Register Description 626
22-7. UCBxSTATW Register Description 626
22-8. UCBxTBCNT Register Description 627
22-9. UCBxRXBUF Register Description 628
22-10. UCBxTXBUF Register Description 628
22-11. UCBxI2COA0 Register Description 629
22-12. UCBxI2COA1 Register Description 630
22-13. UCBxI2COA2 Register Description 630
22-14. UCBxI2COA3 Register Description 631
22-15. UCBxADDRX Register Description 631
22-16. UCBxADDMASK Register Description 632
22-17. UCBxI2CSA Register Description 632
22-18. UCBxIE Register Description 633
22-19. UCBxIFG Register Description 635
22-20. UCBxIV Register Description 637
23-1. EEM Configurations 642

Read This First

About This Manual

This manual describes the modules and peripherals of the MSP430FR58xx and MSP430FR59xx 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. Consult the device-specific data sheet for these details.

Related Documentation From Texas Instruments

For related documentation, see the MSP430 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; for example:

```
MOV  #255,R10
XOR  @R5,R6
```

Glossary

ACLK	Auxiliary clock
ADC	Analog-to-digital converter
BOR	Brownout reset
BSL	Bootstrap loader; see www.ti.com/msp430 for application reports
CPU	Central processing unit
DAC	Digital-to-analog converter
DCO	Digitally controlled oscillator
dst	Destination
FLL	Frequency locked loop
GIE Modes	General interrupt enable
INT(N/2)	Integer portion of N/2
I/O	Input/output
ISR	Interrupt service routine
LSB	Least-significant bit

LSD	Least-significant digit
LPM	Low-power mode; 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; also split to UNMI (user NMI) and SNMI (system NMI)
PC	Program counter
PM	Power mode
POR	Power-on reset
PUC	Power-up clear
RAM	Random access memory
SCG	System clock generator
SFR	Special function register
SMCLK	Subsystem master clock
SNMI	System NMI
SP	Stack pointer
SR	Status register
src	Source
TOS	Top of stack
UNMI	User NMI
WDT	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 always reads 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

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

The system control module (SYS) is available on all devices. The basic features of SYS are:

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

Topic **Page**

1.1 System Control Module (SYS) Introduction 29
1.2 System Reset and Initialization 29
1.3 Interrupts 31
1.4 Operating Modes 37
1.5 Principles for Low-Power Applications 42
1.6 Connection of Unused Pins 43
1.7 Reset Pin ($\overline{\text{RST}}$/NMI) Configuration 43
1.8 Configuring JTAG Pins 43
1.9 Vacant Memory Space 44
1.10 Boot Code 44
1.11 Bootstrap Loader (BSL) 45
1.12 JTAG Mailbox (JMB) System 46
1.13 JTAG and SBW Lock Mechanism Using the Electronic Fuse 47
1.14 Device Descriptor Table 48
1.15 SFR Registers 54
1.16 SYS Registers 59

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 generated only by the following events:

- Powering up the device
- Low signal on the $\overline{\text{RST}}$/NMI pin when configured in the reset mode
- Wakeup event from LPMx.5 (that is, LPM3.5 or LPM4.5) mode
- SVS_H low condition, when enabled (see the PMM and SVS chapter for details)
- Software BOR event (see the PMM and SVS chapter for details)

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:

- BOR signal
- Software POR event (see the PMM and SVS chapter for details)

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:

- 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)
- FRAM memory password violation (see the FRAM Controller chapter for details)
- Power Management Module password violation (see the PMM and SVS chapter for details)
- Memory Protection Unit password violation (see the MPU chapter for details)
- Memory segment violation (see the MPU chapter for details)
- Clock System password violation (see the Clock System chapter for details)
- Fetch from peripheral area
- Uncorrectable FRAM bit error detection

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.

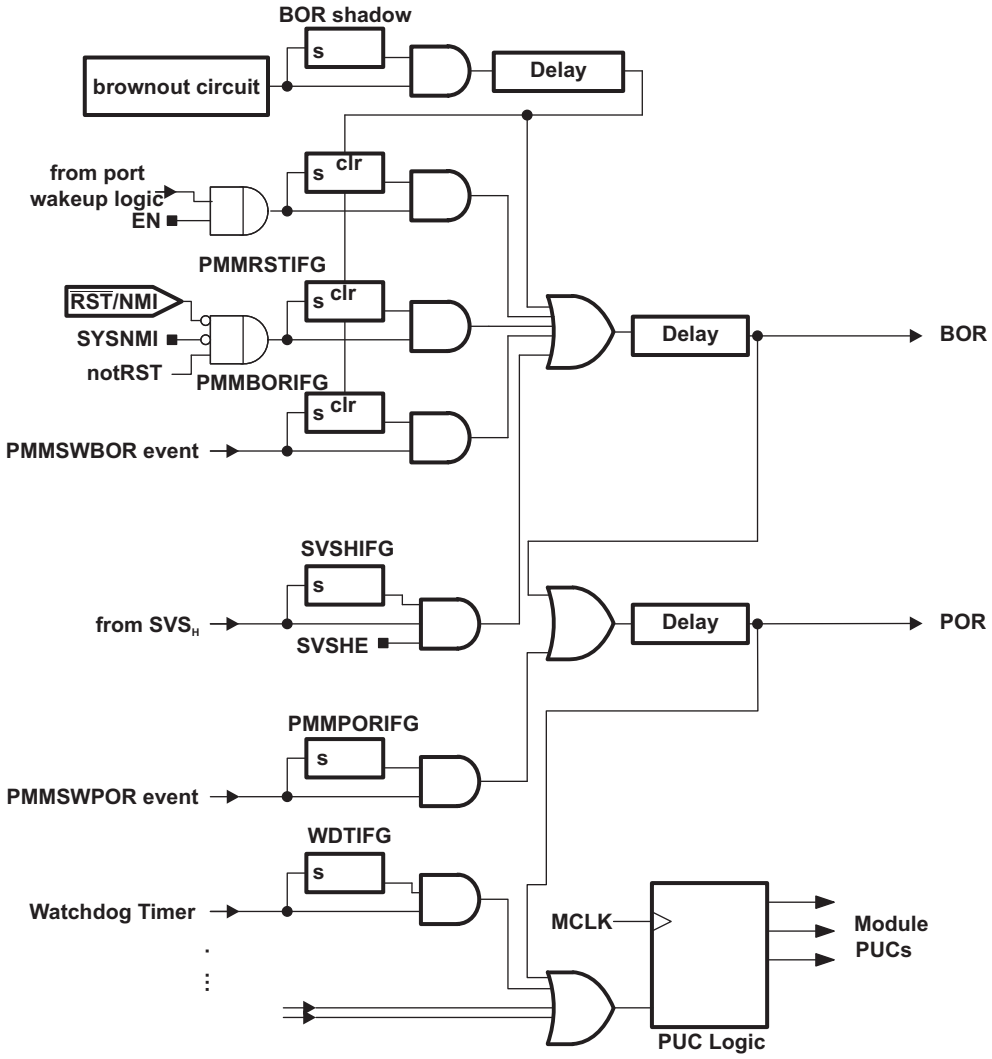


Figure 1-1. BOR, POR, and PUC Reset Circuit

1.2.1 Device Initial Conditions After System Reset

After a BOR, the initial device conditions are:

- The $\overline{\text{RST}}$/NMI pin is configured in the reset mode. See Section 1.7 for details on configuring the $\overline{\text{RST}}$/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.
- 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 1.10 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 when available, otherwise FRAM location.
- Initialize the watchdog to the requirements of the application.
- Configure peripheral modules to the requirements of the application.

NOTE: A device that is unprogrammed or blank is defined as having its reset vector value, residing at memory address FFFEh, equal to FFFFh. Upon system reset of a blank device, the device automatically enters operating mode LPM4. See Section 1.4 for information on operating modes and Section 1.3.6 for details on interrupt vectors.

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

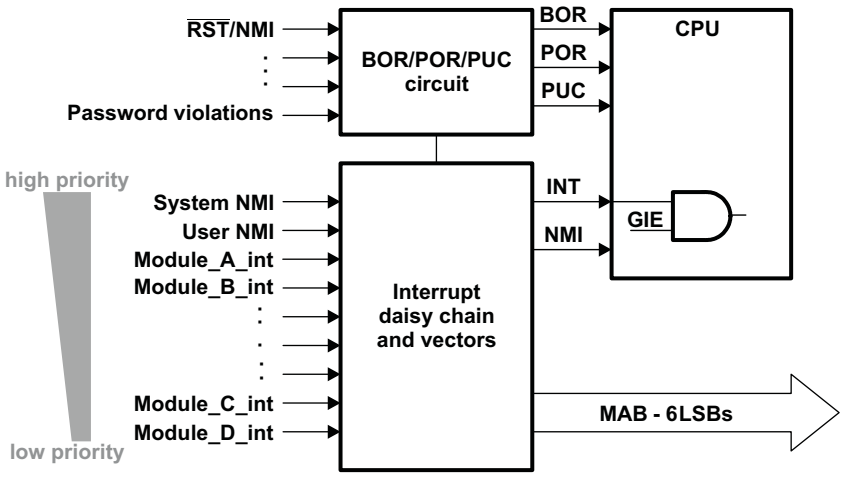


Figure 1-2. Interrupt Priority

NOTE: The types of interrupt sources available and their respective priorities change from device to device. 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. Two levels of NMIs are supported — 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 Section 1.3.6. 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 Section 1.3.

A UNMI interrupt can be generated by following sources:

- An edge on the $\overline{\text{RST}}$/NMI pin when configured in NMI mode
- An oscillator fault occurs

A SNMI interrupt can be generated by following sources:

- FRAM errors (see the FRAM Controller chapter for details)
- 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.

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 1-3. 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. All bits of SR are cleared except SCG0, thereby terminating 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.

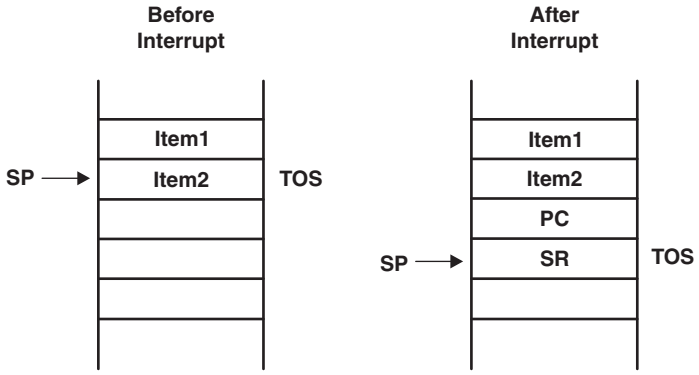


Figure 1-3. Interrupt Processing

NOTE: Enable and Disable Interrupt

Due to the pipelined CPU architecture, the instruction following the enable interrupt instruction (EINT) is always executed, even if an interrupt service request is pending when the interrupts are enabled.

If the enable interrupt instruction (EINT) is immediately followed by a disable interrupt instruction (DINT), a pending interrupt might not be serviced. Further instructions after DINT might execute incorrectly and result in unexpected CPU execution. It is recommended to always insert at least one instruction between EINT and DINT. Note that any alternative instruction use that sets and immediately clears the CPU status register GIE bit must be considered in the same fashion.

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 1-4.

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

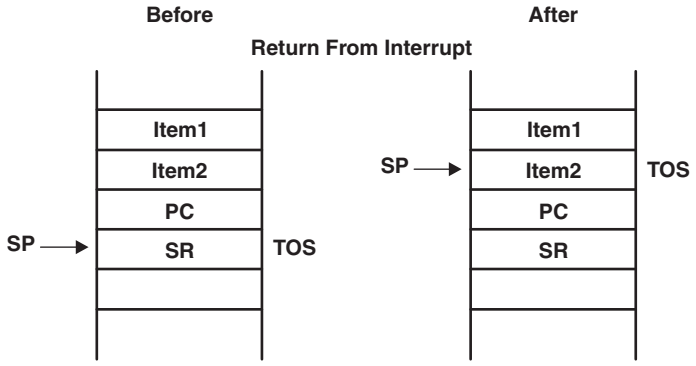


Figure 1-4. Return From Interrupt

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, FRAM password	... WDTIFG FRCTLPW	... Reset	... 0FFFEh	... Highest
System NMI: JTAG Mailbox	JMBINIFG, JMBOUTIFG	(Non)maskable	0FFFCh	
User NMI: NMI oscillator fault	... NMIIFG OFIFG	... (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

On devices that contain RAM, 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 vectors 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, or 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   DBD_ISR        ; Vector 2: DBDIFG
            JMP   ACCTIM_ISR     ; Vector 4: ACCTIMIFG
            JMP   RSVD1_ISR      ; Vector 6: Reserved for future usage.
            JMP   RSVD2_ISR      ; Vector 8: Reserved for future usage.
            JMP   RSVD3_ISR      ; Vector 10: Reserved for future usage.
            JMP   RSVD4_ISR      ; Vector 12: Reserved for future usage.
            JMP   ACCV_ISR       ; Vector 14: ACCVIFG
            JMP   VMA_ISR        ; Vector 16: VMAIFG
            JMP   JMBI_ISR       ; Vector 18: JMBINIFG
            JMP   JMBO_ISR       ; Vector 20: JMBOUTIFG
            JMP   SBD_ISR        ; Vector 22: SBDIFG

DBD_ISR:                         ; Vector 2: DBDIFG
            ...                  ; Task_2 starts here
            RETI                 ; Return
ACCTIM_ISR:                      ; Vector 4
            ...                  ; Task_4 starts here
            RETI                 ; Return
RSVD1_ISR:                       ; Vector 6
            ...                  ; Task_6 starts here
            RETI                 ; Return
RSVD2_ISR:                       ; Vector 8
            ...                  ; Task_8 starts here
            RETI                 ; Return
RSVD3_ISR:                       ; Vector 10
            ...                  ; Task_10 starts here
            RETI                 ; Return
RSVD4_ISR:                       ; Vector 12
            ...                  ; Task_12 starts here
            RETI                 ; Return
ACCV_ISR:                        ; Vector 14
            ...                  ; Task_14 starts here
            RETI                 ; Return
VMA_ISR:                         ; Vector 16
            ...                  ; Task_16 starts here
            RETI                 ; Return
JMBI_ISR:                        ; Vector 18
            ...                  ; Task_18 starts here
JMBO_ISR:                        ; Vector 20
            ...                  ; Task_20 starts here
            RETI                 ; Return
SBD_ISR:                         ; Vector 22
            ...                  ; Task_22 starts here
            RETI                 ; Return
```

1.4 Operating Modes

The MSP430 family is designed for ultralow-power applications and uses different operating modes shown in Figure 1-5.

The operating modes take into account three different needs:

- Ultra-low 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, RAM, and 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. Although the I/O register contents are lost, the I/O pin states are locked upon LPMx.5 entry. See the Digital I/O chapter for further details. 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: The TEST/SBWTCK 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 (for example, MSP-FET430UIF). See the PMM and SVS chapter for further details.

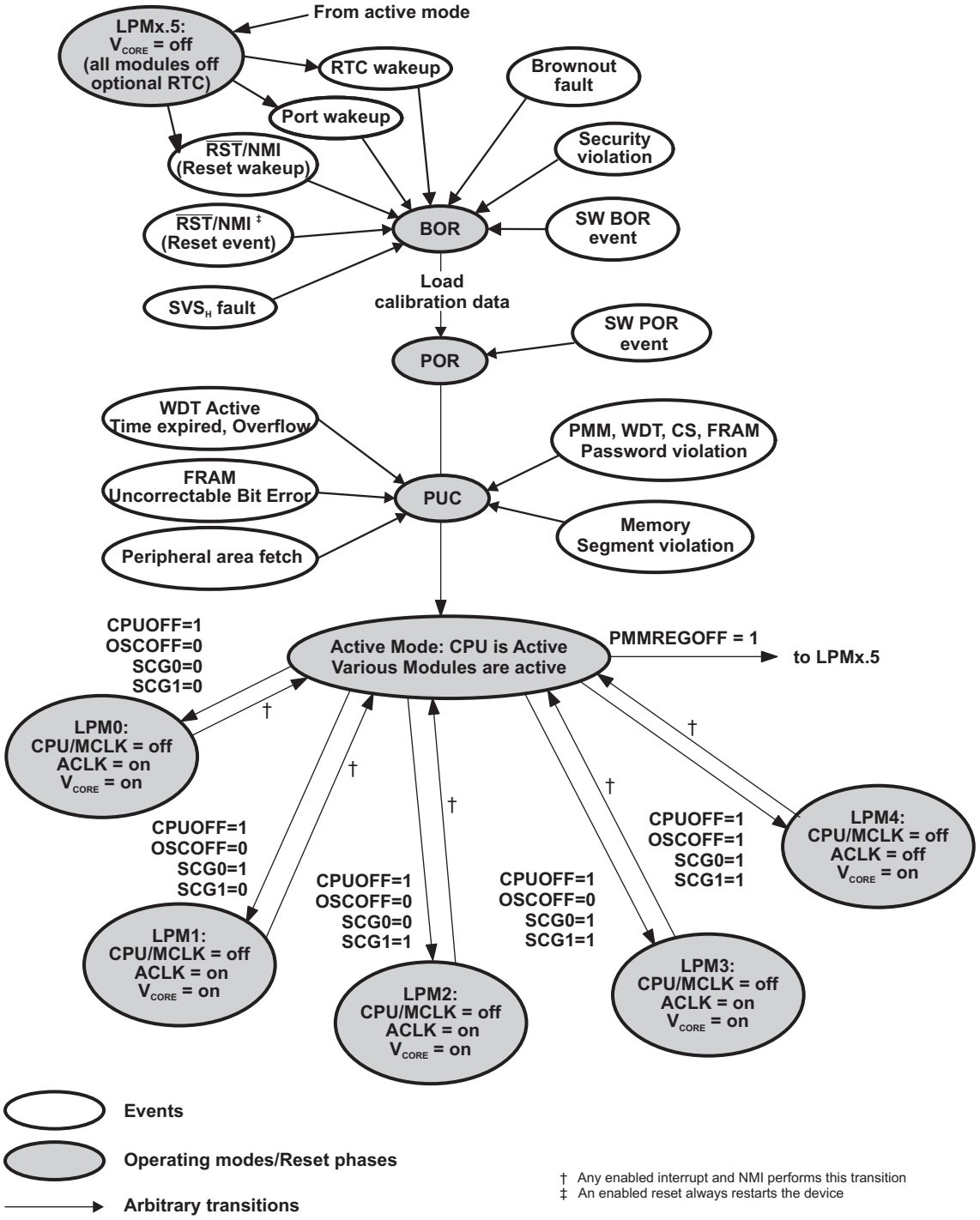


Figure 1-5. Operation Modes

Table 1-2. Operation Modes

SCG1[1]	SCG0	OSCOFF[1]	CPUOFF[1]	Mode	CPU and Clocks Status[2]
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).
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).
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).
1	0	0	1	LPM2	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled.
1	1	0	1	LPM3	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled.
1	1	1	1	LPM4	CPU and all clocks are disabled.
1	1	1	1	LPM3.5	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, RTC operation is possible when configured properly. See the *RTC* module for further details.
1	1	1	1	LPM4.5	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, all clock sources are disabled; that is, no RTC operation is possible.

(1) This bit is automatically reset when exiting low-power modes. See Section 1.4.2 for details.
(2) The low-power modes and, hence, the system clocks can be affected by the clock request system. See the Clock System chapter for details.

1.4.1 Low-Power Modes and Clock Requests

A peripheral module requests its clock sources automatically from the clock system (CS) module if it is required for its proper operation, regardless of the current power mode of operation. Refer to the "Operation From Low-Power Modes, Requested by Peripheral Modules" section in the Clock System chapter.

Because of the clock request mechanism the system might not reach the low-power modes requested by the bits set in the CPU's status register SR as listed in Table 1-3.

Table 1-3. Requested vs Actual LPM

Requested LPM (SR Bits according to Table 1-2)	Actual LPM...		
	if no clock requested	if only ACLK requested	if SMCLK requested
LPM0	LPM0	LPM0	LPM0
LPM1	LPM1	LPM1	LPM1
LPM2	LPM2	LPM2	*LPM0*
LPM3	LPM3	LPM3	*LPM1*
LPM4	LPM4	*LPM3*	*LPM1*

1.4.2 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
```

1.4.3 Low Power Modes LPM3.5 and LPM4.5 (LPMx.5)

The low-power modes LPM3.5 and LPM4.5 (LPMx.5 [1]) give the lowest power consumption on a device. In LPMx.5 the core LDO of the device is switched off. This has the following effects:

- Most of the modules are powered down.
  - In LPM3.5, only modules powered by the RTC LDO continue to operate. At least an RTC module is connected to the RTC LDO. Refer to the device's data sheet for other modules (if any) that are connected to the RTC LDO.
  - In LPM4.5 the RTC LDO and the connected modules are switched off.
- The register content of all modules and the CPU is lost.
- The SRAM content is lost.
- A wake-up from LPMx.5 causes a complete reset of the core.
- The application must initialize the complete device after a wake-up from LPMx.5.

The wake-up time from LPMx.5 is much longer than the wake-up time from any other power mode (refer to the device's data sheet). This is because the core domain must power up and the device internal initialization must be done. In addition, the application must be initialized again. Therefore, use LPMx.5 only when the application is in LPMx.5 for a long time.

[1] The abbreviation "LPMx.5" is used in this document to indicate both LPM3.5 and LPM4.5.

1.4.3.1 Enter LPMx.5

Do the following steps to enter LPMx.5:

1. Store any information that must be available after wakeup from LPMx.5 in FRAM.
2. For LPM4.5 set all ports to general-purpose I/Os (PxSEL0=00h and PxSEL1=00h).
   For LPM3.5 if the LF crystal oscillator is used do not change the settings for the I/Os shared with the LF-crystal-oscillator. These pins must be configured as LFXIN and LFXOUT. Set all other port pins to general-purpose I/Os with PxSEL0 and PxSEL1 bits equal to 0.
3. Set the port pin direction and output bits as necessary for the application.
4. To enable a wakeup from an I/O do the following:
   (a) Select the wakeup edge (PxIES)
   (b) Clear the interrupt flag (PxIFG)
   (c) Set the interrupt enable bit (PxIE)
5. For LPM3.5 the modules that stay active must be enabled. For example, the RTC must be enabled if necessary. Only modules connected to the RTC LDO can stay active.
6. For LPM3.5 if necessary enable any interrupt sources from these modules as wakeup sources. Refer to the corresponding module chapter.
7. Disable the watchdog timer WDT if it is enabled and in watchdog mode. If the WDT is enabled and in watchdog mode, the device does not enter LPMx.5.
8. Clear the GIE bit:

```
BIC #GIE, SR
```

9. Do the following steps to set the PMMREGOFF bit in the PMMCTL0 register:
   (a) Write the correct PMM password to get write access to the PMM control registers.

```
MOV.B #PMMPW_H, &PMMCTL0_H
```

   (b) Set PMMREGOFF bit in the PMMCTL0 register.

```
BIS.B #PMMREGOFF, &PMMCTL0_L
```

   (c) If you want to disabled the SVS during LPMx.5 clear the SVSHE bit in PMMCTL0.

```
BIC.B #SVSHE, &PMMCTL0_L
```

   (d) Write an incorrect PMM password to disable the write access to the PMM control registers.

```
MOV.B #000h, &PMMCTL0_H
```

10. Enter LPMx.5 with the following instruction:

```
BIS #CPUOFF+OSCOFF+SCG0+SCG1, SR
```

The device will enter LPM3.5 if modules connected to the RTC LDO are enabled. It will enter LPM4.5 if none of the modules connected to the RTC LDO are enabled.

1.4.3.2 Exit from LPMx.5

The following conditions will cause an exit from LPMx.5:

- A wakeup event on an I/O if configured and enabled. The interrupt flag of the corresponding port pin is set (PxIFG). The PMMLPM5IFG bit is set.
- A wakeup event from a module connected to the RTC LDO if enabled. The corresponding interrupt flag in the module is set. The PMMLPM5IFG bit is set.
- A wakeup from the RST pin.
- A power-cycle. Either the SVSHIFG or none of the PMMIFGs is set.

Any exit from LPMx.5 causes a BOR. The program execution starts at the address the reset vector points to. PMMLPM5IFG=1 indicates a wakeup from LPMx.5 or the System Reset Vector Word register SYSRSTIV can be used to decode the reset condition (refer to the device's data sheet).

After wakeup from LPMx.5 the state of the I/Os and the modules connected to the RTC LDO are locked and remain unchanged until you clear the LOCKLPM5 bit in the PM5CTL0 register.

1.4.3.3 Wake-Up from LPM3.5

Do the following steps after a wake-up from LPM3.5:

1. Initialize the registers of the modules connected to the RTC LDO exactly the same way as they were configured before the device entered LPM3.5 but do not enable the interrupts.
2. Initialize the port registers exactly the same way as they were configured before the device entered LPM3.5 but do not enable port interrupts.
3. If the LF-crystal-oscillator was used in LPM3.5 the corresponding I/Os must be configured as LFXIN and LFXOUT. The LF-crystal-oscillator must be enabled in the clock system (refer to the clock system CS chapter).
4. Clear the LOCKLPM5 bit in the PM5CTL0 register.
5. Enable port interrupts as necessary.
6. Enable module interrupts.
7. After enabling the port and module interrupts the wake-up interrupt will be serviced as a normal interrupt.

1.4.3.4 Wake-Up from LPM4.5

Do the following steps after a wake-up from LPM4.5:

1. Initialize the port registers exactly the same way as they were configured before the device entered LPM4.5 but do not enable port interrupts.
2. Clear the LOCKLPM5 bit in the PM5CTL0 register.
3. Enable port interrupts as necessary.
4. After enabling the port interrupts the wake-up interrupt will be serviced as a normal interrupt.

If a crystal oscillator is needed after a wake-up from LPM4.5 then configure the corresponding pins and start the oscillator after you cleared the LOCKLPM5 bit.

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 and 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 1-4.

Table 1-4. Connection of Unused Pins(1)

Pin	Potential	Comment
AVCC	DV_{CC}	
AVSS	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(2)) pulldown
PJ.0/TDO PJ.1/TDI PJ.2/TMS PJ.3/TCK	Open	The JTAG pins are shared with general-purpose I/O function (PJ.x). If not being used, these should be switched to port function, output direction. When used as JTAG pins, these pins should remain open.
TEST	Open	This pin always has an internal pulldown enabled.

(1) Any unused pin with a secondary function that is shared with general-purpose I/O should follow the Px.0 to Px.7 unused pin connection guidelines.

(2) 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 is set.

The $\overline{RST}$/NMI pin can have either a pullup or pulldown present or not. SYSRSTUP selects either pullup or pulldown, and SYSRSTRE causes the pullup or pulldown to be enabled or not. If the $\overline{RST}$/NMI pin is unused, it is required to have either the internal pullup selected and enabled or an external resistor connected to the $\overline{RST}$/NMI pin as shown in Table 1-4.

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-purpose I/O. See 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 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 does not change from four-wire JTAG mode to general-purpose I/O.

1.9 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.

1.10 Boot Code

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

1.10.1 IP Encapsulation (IPE) Instantiation by Boot Code

The boot code can preload user-defined setting prior to the start of application code. This ensures that the encapsulation is active before any user controlled accesses to the memory can be performed.

1.10.2 IP Encapsulation Signatures

Two IPE signatures, IPE Signature 1 (memory location 0FF88h) and IPE Signature 2 (memory location 0FF8Ah) reside in FRAM and can be used to control the initialization of the IP Encapsulation. Writing 0xAAAA to IPE Signature 1 triggers the evaluation of the IPE Signature 2 as the IPE structure pointer.

```
// example c-code fragment for definition of IPE structure pointer
IPE_STR_PTR_SRC = (&IPE_Init_Structure) >> 4;
```

Table 1-5. IPE_Signatures

Signature	Address	Symbolic name	Description
IPE Signature 1	0FF88h	IPE_SIG_VALID	IPE signature valid flag
IPE Signature 2	0FF8Ah	IPE_STR_PTR_SRC	source for pointer (nibble address) to MPU IPE structure

1.10.2.1 Trapdoor Mechanism for IP Structure Pointer Transfer

The boot code performs a special sequence to ensure the integrity of the IPE structure pointer. On bootcode execution, a valid IPE Signature 1 triggers the transfer of the IPE Signature 2 (IPE structure pointer source) to a secured system data area (saved IPE structure pointer). This transfer only happens once if no previous secured IPE structure pointer exist. Subsequent of a successful transfer of the IPE structure pointer, the IPE Signatures can be overwritten by any value without compromising the existing IP Encapsulation.

NOTE: Memory locations for IPE Signatures are shared with the JTAG password. This gives the limitation that the first word of the JTAG password cannot be set to the 0AAAAh for a nonprotected device, since this would trigger the trapdoor mechanism unintentionally.

1.10.3 IP Encapsulation Init Structure

By evaluating the saved IPE structure pointer, the bootcode can program the IP Encapsulation related register by transferring the values defined in the IP Encapsulation init structure to the corresponding fields in the MPU control registers. The definition of the structure can be seen in Table 1-6 . The checkcode is calculated as an odd bit interleaved parity of the previous three words. As an example see the following code:

```
// IPE data structures definition, reusable for ALL projects
#define IPE_MPUIPLOCK  0x0080
#define IPE_MPUIPENA   0x0040
#define IPE_MPUIPPUC   0x0020
#define IPE_SEGREG(a)  (a >> 4)
#define IPE_BIP(a,b,c) (a ^ b ^ c ^ 0xFFFF)
```

```
#define IPE_FILLSTRUCT(a,b,c)
{a,IPE_SEGREG(b),IPE_SEGREG(c),IPE_BIP(a,IPE_SEGREG(b),IPE_SEGREG(c))}



typedef struct IPE_Init_Structure {
   unsigned int MPUIPC0 ;
   unsigned int MPUIPB2 ;
   unsigned int MPUIPB1 ;
   unsigned int MPUCHECK ;
} IPE_Init_Structure;          // this struct should be placed inside IPB1/IPB2 boundaries

// This is the project dependant part

#define IPE_START 0x0D000      // This defines the Start of the IP protected area
#define IPE_END   0x0F000      // This defines the End of the IP protected area

// define borders of protected code
#pragma DATA_SECTION(IPE_STRUCT,".pbss"); // pbbs is defined in a adopted linker control file
                                          // pbbs is the section for protected data;

IPE_Init_Structure IPE_STRUCT = IPE_FILLSTRUCT(IPE_MPUIPLOCK + IPE_MPUIPENA, IPE_END,IPE_START);
```

Table 1-6. IPE_Init_Structure

Field Name	Address Offset	Length	Description
MPUIPC0	0h	word	Control setting for IP Encapsulation. Value is written to MPUIPC0
MPUIPB2	2h	word	Upper border of IP Encapsulation segment. Value is written to MPUIPSEGB2.
MPUIPB1	4h	word	Lower border of IP Encapsulation segment. Value is written to MPUIPSEGB1.
MPUCHECK	6h	word	Odd bit interleaved parity

NOTE: Although the user is completely free to select the location for the IPE Init Structure, protection against unwanted modification is given only if the structure is placed inside of the protected area checked by the structure itself. This allows a reconfiguration from within the protected area but prevents malicious modification from outside.

1.10.4 IP Encapsulation Removal

After successful instantiation of an IP protected memory area, a mass erase only erases the memory area outside of the IP Encapsulation. To perform a erase of all memory location in main memory and to remove the IPE structure pointer, a total erase must be performed. For more details, see the *MSP430™ Programming Via the JTAG Interface User's Guide* (SLAU320).

NOTE: An invalid IP Encapsulation init structure or a saved IPE structure pointer with an invalid target (not pointing to a valid IP Encapsulation init structure) causes a total erase during bootcode execution. This setup error leads to a completely unprogrammed device after the next bootcode execution. This mechanism ensures that no exposure of IP code can happen by a misconfiguration or a memory corruption.

1.11 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, the data memory (RAM), and the peripherals, can be modified by the BSL as required.

A basic BSL program is provided by TI and resides in ROM at memory space 01000h through 017FFh. The BSL supports the commonly used UART protocol with RS232 interfacing, allowing flexible use of both hardware and software. Depending on the device, additional BSL communication interfaces are supported. For details of the available and configured BSL communication interfaces see Section 1.14.3.5.

To use the BSL, a specific BSL entry sequence must be applied to the $\overline{\text{RST}}$/NMI and TEST pins. A correct entry sequence causes SYSBSLIND to be set. An added sequence of commands initiates the desired function. A bootstrap-loading session can be exited by continuing operation at a defined user program address or by applying the standard reset sequence. Access to the device memory via the BSL is protected against misuse by a user-defined password.

Two BSL signatures, BSL Signature 1 (memory location 0FF84h) and BSL Signature 2 (memory location 0FF86h) reside in FRAM and can be used to control the behavior of the BSL. Writing 05555h to BSL Signature 1 or BSL Signature 2 disables the BSL function and any access to the BSL memory space causes a vacant memory access as described in Section 1.9. Most BSL commands require the BSL to be unlocked by a user-defined password. An incorrect password erases the device memory as a security feature. Writing 0AAAAh to BSL Signature 1 or BSL Signature 2 disables this security feature. This causes a password error to be returned by the BSL, but the device memory is not erased. In this case, unlimited password attempts are possible.

For more details, see the *MSP430 Programming Via the Bootstrap Loader (BSL) User's Guide* (SLAU319).

Some JTAG commands are still possible after the device is secured, including the BYPASS command (see IEEE Std 1149-2001) and the JMB_EXCHANGE command, which allows access to the JTAG Mailbox System (see Section 1.12 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 devices of this family and uses only few or no user application resources. The JTAG interface was chosen because it is available on all devices and is a dedicated resource for debugging, programming, and test.

Applications of the JMB are:

- Providing entry password for device lock or 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 JTAG and SBW Lock Mechanism Using the Electronic Fuse

A device can be protected from unauthorized access by restricting accessibility of JTAG commands that can be transferred to the device by the JTAG and SBW interface. This is achieved by programming the electronic fuse. When the device is protected, the JTAG and SBW interface still remains functional, but JTAG commands that give direct access into the device are completely disabled. There are two ways to lock the device. Both of these require the programming of two signatures that reside in FRAM. JTAG Signature 1 (memory location 0FF80h) and JTAG Signature 2 (memory location 0FF82h) control the behavior of the device locking mechanism.

NOTE: When a device has been protected, Texas Instruments cannot access the device for a customer return. Access is only possible if a BSL is provided with its corresponding key or an unlock mechanism is provided by the customer.

1.13.1 JTAG and SBW Lock Without Password

A device can be locked by writing 05555h to both JTAG Signature 1 and JTAG Signature 2. In this case, the JTAG and SBW interfaces grant access to a limited JTAG command set that restricts accessibility into the device. The only way to unlock the device in this case is to use the BSL to overwrite the JTAG signatures with anything other than 05555h or 0AAAAh. 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 Section 1.12 for details).

NOTE: Signatures that have been entered do not take effect until the next BOR event has occurred, at which time the signatures are checked.

1.13.2 JTAG and SBW Lock With Password

A device can also be locked by writing 0AAAAh to JTAG Signature 1 and writing JTAG Signature 2 with any value except 05555h. In this case, JTAG and SBW interfaces grant access to a limited JTAG command set that restricts accessibility into the device as in Section 1.13.1, but an additional mechanism is available that can unlock the device with a user-defined password. In this case, JTAG Signature 2 represents a user-defined length in words of the user defined password. For example, a password length of four words would require writing 0004h to JTAG Signature 2. The starting location of the password is fixed at location 0FF88h. As an example, for a password of length 4, the password memory locations would reside at 0FF88h, 0FF8Ah, 0FF8Ch, and 0FF8Eh.

The password is not checked after each BOR; it is checked only if a specific signature is present in the JTAG incoming mailbox. If the JTAG incoming mailbox contains 0A55Ah and 01E1Eh in JMBIN0 and JMBIN1, respectively, the device is expecting a password to be applied. The entered password is compared to the password that is stored in the device password memory locations. If they match, the device unlocks the JTAG and SBW to the complete JTAG command set until the next BOR event occurs.

NOTE: Memory locations 0FF80h through 0FFFFh may also be used for interrupt vector address locations (see the device-specific data sheet). Therefore, if using the password mechanism for JTAG and SBW lock, which uses address locations 0FF88h and higher, these locations may also have interrupt vector addresses assigned to them. Therefore, the same values assigned for any interrupt vector addresses must also be used as password values.

NOTE: Entering the password via the tool chain is done using 32-bit mode (two words). The least-significant word is entered first for each two-word transfer. For example, if the password location contains: @0xFF8C = 0x45670123CDEF89AB4321, the password must be entered as 0x0123456789ABCDEF4321 via the tool chain.

NOTE: Signatures that have been entered do not take effect until the next BOR event has occurred, at which time the signatures are checked. For example, entering a correct password that grants entry into the device followed by an incorrect password without a BOR sequence may still grant access to the device.

1.14 Device Descriptor Table

Each device provides a data structure in memory that allows an unambiguous identification of the device. The validity of the device descriptor can be verified by cyclic redundancy check (CRC). Figure 1-6 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.

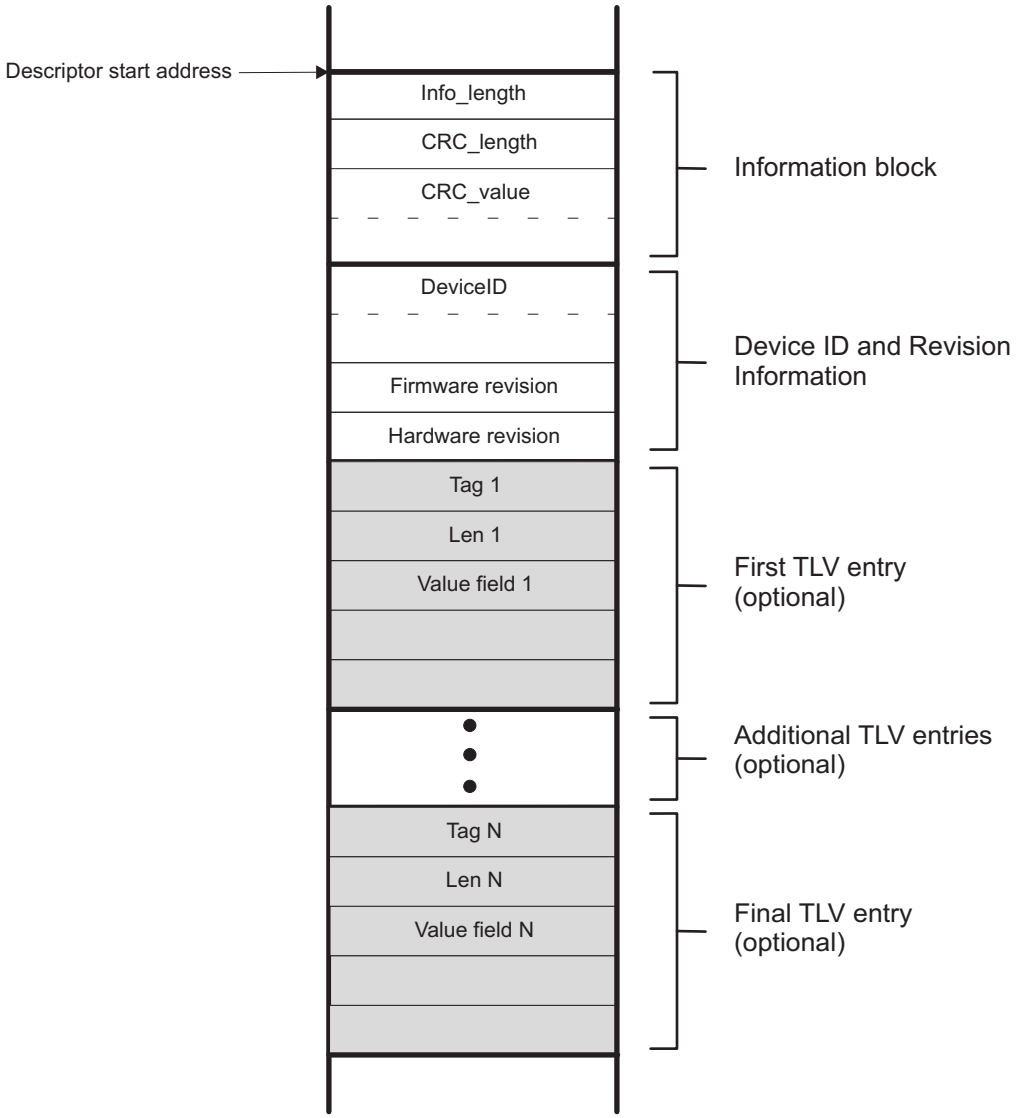


Figure 1-6. Devices Descriptor Table

1.14.1 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 1-6 contains the device ID, die revisions, firmware revisions, and other manufacturer and tool related information.

The length of the descriptors represented by Info_length is computed as shown in Equation 1:

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

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

1.14.2 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 MSP430FR57xx 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 followed by the descriptor block length.

Each TLV descriptor contains a tag field which identifies the descriptor type. Table 1-7 shows the currently supported tags.

Table 1-7. Tag Values

Short Name	Value	Description
LDTAG	01h	Legacy descriptor (1xx, 2xx, 4xx families)
PDTAG	02h	Peripheral discovery descriptor
Reserved	03h	Reserved for future use
Reserved	04h	Reserved for future use
BLANK	05h	Blank descriptor
Reserved	06h	Reserved for future use
Reserved	07h	Reserved for future use
Reserved	08h	Unique Die Record
Reserved	09h-0Fh	Reserved for future use
Reserved	10h	Reserved
ADC12CAL	11h	ADC12 calibration (see Section 1.14.3.2 and Section 1.14.3.3)
REFCAL	12h	REF calibration (see Section 1.14.3.1)
ADC10CAL	13h	ADC10 calibration (see Section 1.14.3.2 and Section 1.14.3.3)
Reserved	14h	Reserved for future use
RANDTAG	15h	Random Number Seed (see Section 1.14.3.4)
Reserved	16h-1Bh	Reserved for future use
BSLTAG	1Ch	BSL Configuration
Reserved	1Dh-FDh	Reserved for future use
TAGEXT	FEh	Tag extender

Each tag field is unique to its respective descriptor and is always followed 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 the following pseudo code:

```
// Identify 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 descriptor 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
}
```

1.14.3 Calibration Values

The TLV structure contains calibration 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.

1.14.3.1 REF Calibration

Table 1-8 shows the REF calibration tags.

Table 1-8. REF Calibration Tags

REF Calibration	TAG	12h
	Length	06h
	Low Byte	CAL_ADC_12VREF_FACTOR
	High Byte	
	Low Byte	CAL_ADC_20VREF_FACTOR
	High Byte	
	Low Byte	CAL_ADC_25VREF_FACTOR
	High Byte	

The calibration data for the REF module consists of three words, one word for each reference voltage available (1.2 V, 2.0 V, and 2.5 V). The reference voltages are measured at room temperature. The measured values are normalized by 1.2 V, 2.0 V, or 2.5 V before being stored into the TLV structure, as shown in Equation 2:

$$\text{CAL\_ADC\_12VREF\_FACTOR} = \frac{V_{REF+}}{1.2V} \times 2^{15}$$

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

$$\text{CAL\_ADC\_25VREF\_FACTOR} = \frac{V_{REF+}}{2.5V} \times 2^{15} \quad (2)$$

In this way, a conversion result is corrected by multiplying it with the CAL_12VREF_FACTOR (or CAL_20VREF_FACTOR, CAL_25VREF_FACTOR) and dividing the result by 2^{15}as shown in Equation 3 for each of the respective reference voltages:

$$\text{ADC(corrected)} = \text{ADC(raw)} \times \text{CAL\_ADC12VREF\_FACTOR} \times \frac{1}{2^{15}}$$

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

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

In the following example, the integrated 1.2-V reference voltage is used during a conversion.

- Conversion result: 0x0100 = 256 decimal
- Reference voltage calibration factor (CAL_12VREF_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_12VREF_FACTOR: 0x200 x 0x7FEE = 0x00F7_7600
- Divide the result by 2^{16}: 0x00F7_7600 / 0x0001_0000 = 0x0000_00F7 = 247 decimal

1.14.3.2 ADC Offset and Gain Calibration

Table 1-9 shows the ADC calibration tags.

Table 1-9. ADC Calibration Tags

ADC Calibration	TAG	ADC10: 13h ADC12: 11h
	Length	10h
	Low Byte	CAL_ADC_GAIN_FACTOR
	High Byte	
	Low Byte	CAL_ADC_OFFSET
	High Byte	
	Low Byte	CAL_ADC_12T30
	High Byte	
	Low Byte	CAL_ADC_12T85
	High Byte	
	Low Byte	CAL_ADC_20T30
	High Byte	
	Low Byte	CAL_ADC_20T85
	High Byte	
	Low Byte	CAL_ADC_25T30
	High Byte	
	Low Byte	CAL_ADC_25T85
	High Byte	

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.

$$ADC(offset\_corrected) = ADC(raw) + CAL\_ADC\_OFFSET \quad (4)$$

The gain of the ADC12 is calculated by Equation 5:

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

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

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

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

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

$$ADC(final) = ADC(gain\_corrected) + CAL\_ADC\_OFFSET \quad (7)$$

1.14.3.3 Temperature Sensor Calibration

The temperature sensor calibration data is part of the ADC tag as shown in Table 1-9.

The temperature sensor is calibrated using the internal voltage references. Each reference voltage (1.2 V, 2.0 V, or 2.5 V) 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 millivolts is:

$$V_{SENSE} = TC_{SENSOR} \times Temp + V_{SENSOR} \quad (8)$$

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:

$$\text{Temp}\,[°C] = (\text{ADC(raw)} - \text{CAL\_ADC\_12T30}) \times \left(\frac{85-30}{\text{CAL\_ADC\_12T85} - \text{CAL\_ADC\_12T30}}\right) + 30$$

$$\text{Temp}\,[°C] = (\text{ADC(raw)} - \text{CAL\_ADC\_20T30}) \times \left(\frac{85-30}{\text{CAL\_ADC\_20T85} - \text{CAL\_ADC\_20T30}}\right) + 30$$

$$\text{Temp}\,[°C] = (\text{ADC(raw)} - \text{CAL\_ADC\_25T30}) \times \left(\frac{85-30}{\text{CAL\_ADC\_25T85} - \text{CAL\_ADC\_25T30}}\right) + 30 \quad (9)$$

1.14.3.4 Random Number Seed

Table 1-10 shows the tags used for the random number seed.

Table 1-10. Random Number Tags

Random Number	TAG	15h
	Length	10h
	16 bytes	128-bit random number seed

The random number stored as a seed for a deterministic random number generator is programmed during test of the device. It is generated on the test system using a cryptographic random number generator.

1.14.3.5 BSL Configuration

Table 1-11 shows the tags used for the BSL configuration. The BSL configuration stores the communication interface selection and corresponding communication interface settings. The Tag is optional for devices only providing the basic UART BSL interface. The TAG length field is variable and determinated by the length of the configuration option field BSL_CIF_CONFIG. The BSL configuration cannot be changed by the user.

Table 1-11. BSL Configuration Tags

BSL Configuration	TAG	1Ch
	Length	Depends on the BSL_COM_IF value (actual: 02h for UART or I2C)
	Low Byte	BSL_COM_IF
	High Byte	BSL_CIF_CONFIG[0]
	Low Byte	BSL_CIF_CONFIG[1] (optional)
	High Byte	BSL_CIF_CONFIG[2] (optional)
	Low Byte	BSL_CIF_CONFIG[3](optional)
	High Byte	BSL_CIF_CONFIG[4](optional)
	⋮	⋮
	⋮	⋮
	High Byte	BSL_CIF_CONFIG[n] (optional)

Table 1-12. BSL_COM_IF Values

BSL_COM_IF	Description	Length
00h	UART interface selected	02h
01h	I2C interface selected	02h
02h to FFh	Reserved for future communication interface	reserved

Table 1-12 shows the defined value for the BSL_COM_IF field. Depending on the selected communication interface, the subsequent bytes in the BSL config tag are interpreted to configure the communication interface. The interpretation is shown in Table 1-13. Unused bytes in BSL_CIF_CONFIG are defined as 00h.

Table 1-13. BSL_CIF_CONFIG Values

BSL_CIF_CONFIG_IF[n]	UART [BSL_COM_IF == 00h]	I2C [BSL_COM_IF == 01h]
0	00h	I2C address (valid values: 0..7Fh)
1 to FFh	N/A	N/A

Table 1-13 shows the defined configuration options for the given BSL communication interface.

1.15 SFR Registers

The SFRs are listed in Table 1-14. The base address for the SFRs is 00100h. Many of the bits inside the SFRs are described in other chapters throughout this user's guide. These bits are marked with a note and a reference. See 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 1-14. SFR Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	SFRIE1	Interrupt Enable	Read/write	Word	0000h	Section 1.15.1
00h	SFRIE1_L (IE1)		Read/write	Byte	00h	
01h	SFRIE1_H (IE2)		Read/write	Byte	00h	
02h	SFRIFG1	Interrupt Flag	Read/write	Word	0082h	Section 1.15.2
02h	SFRIFG1_L (IFG1)		Read/write	Byte	82h	
03h	SFRIFG1_H (IFG2)		Read/write	Byte	00h	
04h	SFRRPCR	Reset Pin Control	Read/write	Word	001Ch	Section 1.15.3
04h	SFRRPCR_L		Read/write	Byte	1Ch	
05h	SFRRPCR_H		Read/write	Byte	00h	

1.15.1 SFRIE1 Register

Interrupt Enable Register

Figure 1-7. SFRIE1 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
JMBOUTIE	JMBINIE	Reserved	NMIIE	VMAIE	Reserved	OFIE(1)	WDTIE(2)
rw-0	rw-0	r-0	rw-0	rw-0	r0	rw-0	rw-0

(1) See the Clock System chapter for details.
(2) See the WDT_A chapter for details.

Table 1-15. SFRIE1 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7	JMBOUTIE	RW	0h	JTAG mailbox output interrupt enable 0b = Interrupts disabled 1b = Interrupts enabled
6	JMBINIE	RW	0h	JTAG mailbox input interrupt enable 0b = Interrupts disabled 1b = Interrupts enabled
5	Reserved	R	0h	Reserved. Always reads as 0.
4	NMIIE	RW	0h	NMI pin interrupt enable 0b = Interrupts disabled 1b = Interrupts enabled
3	VMAIE	RW	0h	Vacant memory access interrupt enable 0b = Interrupts disabled 1b = Interrupts enabled
2	Reserved	R	0h	Reserved. Always reads as 0.
1	OFIE	RW	0h	Oscillator fault interrupt enable 0b = Interrupts disabled 1b = Interrupts enabled
0	WDTIE	RW	0h	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 SFRIE1 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. 0b = Interrupts disabled 1b = Interrupts enabled

1.15.2 SFRIFG1 Register

Interrupt Flag Register

Figure 1-8. SFRIFG1 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
JMBOUTIFG	JMBINIFG	Reserved	NMIIFG	VMAIFG	Reserved	OFIFG(1)	WDTIFG(2)
rw-(1)	rw-(0)	r0	rw-0	rw-0	r0	rw-(1)	rw-0

(1) See the Clock System chapter for details.
(2) See the WDT_A chapter for details.

Table 1-16. SFRIFG1 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7	JMBOUTIFG	RW	1h	JTAG mailbox output interrupt flag 0b = 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. 1b = 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 module and are ready for new messages from the CPU.
6	JMBINIFG	RW	0h	JTAG mailbox input interrupt flag 0b = 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 1b = 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.
5	Reserved	R	0h	Reserved. Always reads as 0.
4	NMIIFG	RW	0h	NMI pin interrupt flag 0b = No interrupt pending 1b = Interrupt pending
3	VMAIFG	RW	0h	Vacant memory access interrupt flag 0b = No interrupt pending 1b = Interrupt pending
2	Reserved	R	0h	Reserved. Always reads as 0.
1	OFIFG	RW	1h	Oscillator fault interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Table 1-16. SFRIFG1 Register Description (continued)

Bit	Field	Type	Reset	Description
0	WDTIFG	RW	0h	Watchdog timer interrupt flag. In watchdog mode, WDTIFG clears itself upon a watchdog timeout event. The SYSRSTIV can be read to determine if the reset was caused by a watchdog timeout event. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or can be reset by software. Because other bits in SFRIFG1 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. 0b = No interrupt pending 1b = Interrupt pending

1.15.3 SFRRPCR Register

Reset Pin Control Register

Figure 1-9. SFRRPCR 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			SYSRSTFE	SYSRSTRE	SYSRSTUP	SYSNMIIES	SYSNMI
r0	r0	r0	rw-1	rw-1	rw-1	rw-0	rw-0

Table 1-17. SFRRPCR Register Description

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Always reads as 0.
4	SYSRSTFE	RW	1h	Reset pin filter enable. The filter is only available in Reset function (SYSNMI = 0). 0b = Filter at the RST/NMI pin is disabled. 1b = Filter at the RST/NMI pin is enabled. Only available in Reset function (SYSNMI = 0)
3	SYSRSTRE	RW	1h	Reset pin resistor enable 0b = Pullup or pulldown resistor at the RST/NMI pin is disabled. 1b = Pullup or pulldown resistor at the RST/NMI pin is enabled.
2	SYSRSTUP	RW	1h	Reset resistor pin pullup or pulldown 0b = Pulldown is selected. 1b = Pullup is selected.
1	SYSNMIIES	RW	0h	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. 0b = NMI on rising edge 1b = NMI on falling edge
0	SYSNMI	RW	0h	NMI select. This bit selects the function for the RST/NMI pin. 0b = Reset function 1b = NMI function

1.16 SYS Registers

The SYS configuration registers are listed in Table 1-18 and the base address is 00180h. A detailed description of each register and its bits is also provided. Each register starts at a word boundary. Either 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 1-18. SYS Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	SYSCTL	System Control	Read/write	Word	0000h	Section 1.16.1
00h	SYSCTL_L		Read/write	Byte	00h	
01h	SYSCTL_H		Read/write	Byte	00h	
06h	SYSJMBC	JTAG Mailbox Control	Read/write	Word	000Ch	Section 1.16.2
06h	SYSJMBC_L		Read/write	Byte	0Ch	
07h	SYSJMBC_H		Read/write	Byte	00h	
08h	SYSJMBI0	JTAG Mailbox Input 0	Read/write	Word	0000h	Section 1.16.3
08h	SYSJMBI0_L		Read/write	Byte	00h	
09h	SYSJMBI0_H		Read/write	Byte	00h	
0Ah	SYSJMBI1	JTAG Mailbox Input 1	Read/write	Word	0000h	Section 1.16.4
0Ah	SYSJMBI1_L		Read/write	Byte	00h	
0Bh	SYSJMBI1_H		Read/write	Byte	00h	
0Ch	SYSJMBO0	JTAG Mailbox Output 0	Read/write	Word	0000h	
0Ch	SYSJMBO0_L		Read/write	Byte	00h	
0Dh	SYSJMBO0_H		Read/write	Byte	00h	
0Eh	SYSJMBO1	JTAG Mailbox Output 1	Read/write	Word	0000h	Section 1.16.6
0Eh	SYSJMBO1_L		Read/write	Byte	00h	
0Fh	SYSJMBO1_H		Read/write	Byte	00h	
1Ah	SYSUNIV	User NMI Vector Generator	Read	Word	0000h	Section 1.16.7
1Ch	SYSSNIV	System NMI Vector Generator	Read	Word	0000h	Section 1.16.8
1Eh	SYSRSTIV	Reset Vector Generator	Read	Word	0002h	Section 1.16.9

1.16.1 SYSCTL Register

SYS Control Register

Figure 1-10. SYSCTL 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		SYSJTAGPIN	SYSBSLIND	Reserved	SYSPMMPE	Reserved	SYSRIVECT
r0	r0	rw-[0]	r-0	r0	rw-[0]	r0	rw-[0]

Table 1-19. SYSCTL Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5	SYSJTAGPIN	RW	0h	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. 0b = Shared JTAG pins (JTAG mode selectable using SBW sequence) 1b = Dedicated JTAG pins (explicit 4-wire JTAG mode selection)
4	SYSBSLIND	R	0h	BSL entry indication. This bit indicates a BSL entry sequence detected on the Spy-Bi-Wire pins. 0b = No BSL entry sequence detected 1b = BSL entry sequence detected
3	Reserved	R	0h	Reserved. Always reads as 0.
2	SYSPMMPE	RW	0h	PMM access protect. This controls the accessibility of the PMM control registers. Once set to 1, it only can be cleared by a BOR. 0b = Access from anywhere in memory 1b = Access only from the BSL segments
1	Reserved	R	0h	Reserved. Always reads as 0.
0	SYSRIVECT	RW	0h	RAM-based interrupt vectors 0b = Interrupt vectors generated with end address TOP of lower 64K FRAM FFFFh 1b = Interrupt vectors generated with end address TOP of RAM, when RAM available.

1.16.2 SYSJMBC Register

JTAG Mailbox Control Register

Figure 1-11. SYSJMBC 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
JMBCLR1OFF	JMBCLR0OFF	Reserved	JMBM0DE	JMBOUT1FG	JMBOUT0FG	JMBIN1FG	JMBIN0FG
rw-(0)	rw-(0)	r0	rw-0	r-(1)	r-(1)	rw-(0)	rw-(0)

Table 1-20. SYSJMBC Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7	JMBCLR1OFF	RW	0h	Incoming JTAG Mailbox 1 flag auto-clear disable 0b = JMBIN1FG cleared on read of JMB1IN register 1b = JMBIN1FG cleared by software
6	JMBCLR0OFF	RW	0h	Incoming JTAG Mailbox 0 flag auto-clear disable 0b = JMBIN0FG cleared on read of JMB0IN register 1b = JMBIN0FG cleared by software
5	Reserved	R	0h	Reserved. Always reads as 0.
4	JMBMODE	RW	0h	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. 0b = 16-bit transfers using JMBO0 and JMBI0 only 1b = 32-bit transfers using JMBO0 with JMBO1 and JMBI0 with JMBI1
3	JMBOUT1FG	R	1h	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. 0b = JMBO1 is not ready to receive new data. 1b = JMBO1 is ready to receive new data.
2	JMBOUT0FG	R	1h	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. 0b = JMBO0 is not ready to receive new data. 1b = JMBO0 is ready to receive new data.
1	JMBIN1FG	RW	0h	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. 0b = JMBI1 has no new data. 1b = JMBI1 has new data available.
0	JMBIN0FG	RW	0h	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. 0b = JMBI1 has no new data. 1b = JMBI1 has new data available.

1.16.3 SYSJMBI0 Register

JTAG Mailbox Input 0 Register

Figure 1-12. SYSJMBI0 Register

15	14	13	12	11	10	9	8
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
MSGLO							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-21. SYSJMBI0 Register Description

Bit	Field	Type	Reset	Description
15-8	MSGHI	RW	0h	JTAG mailbox incoming message high byte
7-0	MSGLO	RW	0h	JTAG mailbox incoming message low byte

1.16.4 SYSJMBI1 Register

JTAG Mailbox Input 1 Register

Figure 1-13. SYSJMBI1 Register

15	14	13	12	11	10	9	8
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
MSGLO							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-22. SYSJMBI1 Register Description

Bit	Field	Type	Reset	Description
15-8	MSGHI	RW	0h	JTAG mailbox incoming message high byte
7-0	MSGLO	RW	0h	JTAG mailbox incoming message low byte

1.16.5 SYSJMBO0 Register

JTAG Mailbox Output 0 Register

Figure 1-14. SYSJMBO0 Register

15	14	13	12	11	10	9	8
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
MSGLO							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-23. SYSJMBO0 Register Description

Bit	Field	Type	Reset	Description
15-8	MSGHI	RW	0h	JTAG mailbox outgoing message high byte
7-0	MSGLO	RW	0h	JTAG mailbox outgoing message low byte

1.16.6 SYSJMBO1 Register

JTAG Mailbox Output 1 Register

Figure 1-15. SYSJMBO1 Register

15	14	13	12	11	10	9	8
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
MSGLO							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-24. SYSJMBO1 Register Description

Bit	Field	Type	Reset	Description
15-8	MSGHI	RW	0h	JTAG mailbox outgoing message high byte
7-0	MSGLO	RW	0h	JTAG mailbox outgoing message low byte

1.16.7 SYSUNIV Register

User NMI Vector Register

Figure 1-16. SYSUNIV Register

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

Table 1-25. SYSUNIV Register Description

Bit	Field	Type	Reset	Description
15-0	SYSUNIV	R	0h	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. See the device-specific data sheet for a list of values.

1.16.8 SYSSNIV Register

System NMI Vector Register

Figure 1-17. SYSSNIV Register

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

Table 1-26. SYSSNIV Register Description

Bit	Field	Type	Reset	Description
15-0	SYSSNIV	R	0h	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. See the device-specific data sheet for a list of values.

1.16.9 SYSRSTIV Register

Reset Interrupt Vector Register

Figure 1-18. SYSRSTIV Register

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

(1) Reset value depends on reset source.

Table 1-27. SYSRSTIV Register Description

Bit	Field	Type	Reset	Description
15-0	SYSRSTIV	R	02h-03Eh(1)	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. See the device-specific data sheet for a list of values.

(1) Reset value depends on reset source.

Power Management Module (PMM) and Supply Voltage Supervisor (SVS)

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

Topic **Page**

2.1 Power Management Module (PMM) Introduction 67
2.2 PMM Operation 68
2.3 PMM Registers 71

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})
- Supply voltage supervisor (SVS) for DV_{CC}
- Brownout reset (BOR)
- Software accessible power-fail indicators
- I/O protection during power-fail condition

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 of the voltage applied to the device (DV_{CC}).

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, and the digital modules, while DV_{CC} supplies the I/Os and analog modules. The V_{CORE} output is maintained using a dedicated voltage reference. 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.

Figure 2-1 shows the block diagram of the PMM.

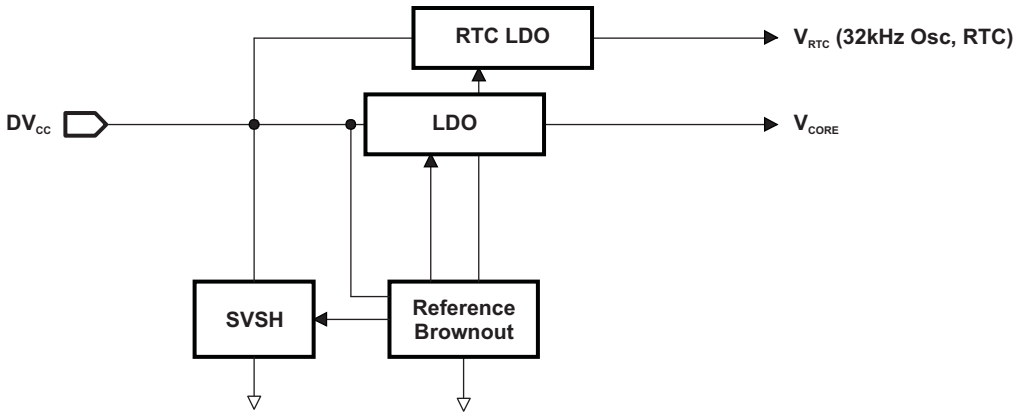


Figure 2-1. PMM Block Diagram

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 (LDO) has been integrated into the PMM. The regulator derives the necessary core voltage (V_{CORE}) from DV_{CC}.

The regulator supports different load settings to optimize power. The hardware controls the load settings automatically, according to the following criteria:

- Selected and active power modes
- Selected and active clocks
- Clock frequencies according to Clock System (CS) settings
- JTAG is active

In addition to the main LDO, an ultralow-power regulator (RTC LDO) provides a regulated voltage to the real-time clock module (including the 32-kHz crystal oscillator) and other ultralow-power modules that remain active during LPM3.5 when the main LDO is off.

2.2.2 Supply Voltage Supervisor

The high-side supervisor (SVSH) oversees DV_{CC}. It is activate in all power modes by default. To disable the SVSH in LPM3, LPM4, LPM3.5, and LPM4.5, set SVSHE = 0.

2.2.2.1 SVS Thresholds

As Figure 2-2 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.

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

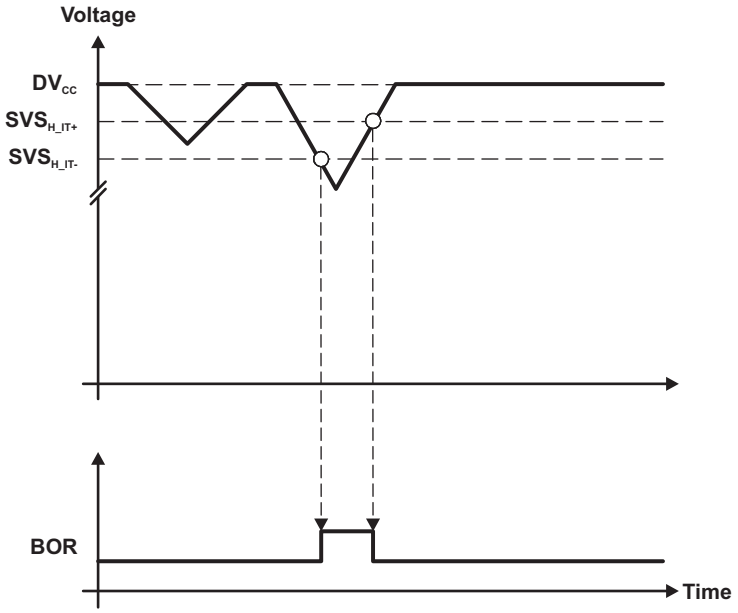


Figure 2-2. Voltage Failure and Resulting PMM Actions

2.2.3 Supply Voltage Supervisor - Power-Up

When the device is powering up, the SVSH function is enabled by default. Initially, DV_{CC} is low, and therefore the PMM holds the device in BOR reset. When the SVSH level is met, the reset is released. Figure 2-3 shows this process.

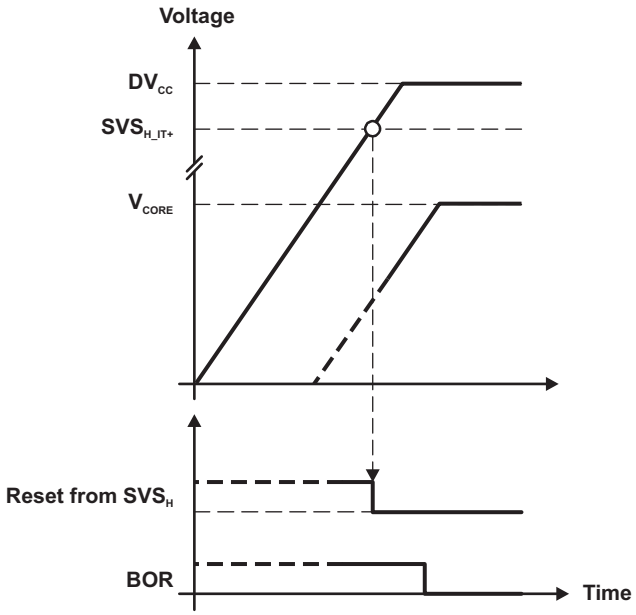


Figure 2-3. PMM Action at Device Power-Up

2.2.4 LPM3.5 and LPM4.5

LPM3.5 and LPM4.5 are additional low-power modes in which the core voltage regulator of the PMM is completely disabled, providing additional power savings. Because there is no power supplied to V_{CORE} during LPMx.5, the CPU and all digital modules including RAM are unpowered. This essentially disables the entire device and thus the contents of the registers and RAM are lost. Any essential values should be stored to FRAM prior to entering LPMx.5.

To enable LPMx.5 the PMMREGOFF bit in the PMMCTL0 register must be set.

The LOCKLPM5 bit in the PM5CTL0 register locks the I/O configuration and other LPMx.5 relevant configurations after a wakeup from LPMx.5 until all the registers are configured again.

LPM3.5 and LPM4.5 can be configured with active SVS (SVSHE = 1) or with SVS disabled (SVSHE = 0). Disabling the SVS results in lower power consumption, whereas enabling it provides the ability to detect supply drops and getting a "wake-up" due to the supply drop below the SVS threshold. Note, the "wake-up" due to a supply failure would not be flagged as a LPMx.5 wake-up but as a SVS reset event. In LPM4.5 enabling the SVS results additionally in an about 4 times faster start-up time than with disabled SVS.

Refer to Section 1.4.3 for complete descriptions and uses of LPMx.5.

NOTE: In watchdog mode, the WDT_A prevents LPMx.5. Refer to Section 12.2.5.

2.2.5 Low-Power Reset

In battery-operated applications, it might be desirable to limit the current drawn by the device to a minimum when the supply drops below the SVS power-down level. By default, when the supply voltage drops below the SVS power-down level, the complete device is reset and prepared to return into active mode quickly when the supply voltage becomes available again. This process results in a current consumption of approximately 50 to 100 µA (typical).

To avoid this relatively high current consumption, a system non-maskable interrupt (NMI) can be generated instead of a reset. Set the PMMLPRST bit in the PMMCTL0 register to enable the SNMI and disable the reset. In this case, the application can configure the I/Os into a defined state, disable any wake-up event (for example, from I/Os by clearing all PxIE bits) and any LPMx.5 supporting module (like RTC), clear SVSHE, and then enter LPM4.5. In this mode, the SVS must be disabled with SVSHE = 0, thus a complete power cycle (that is, the supply needs to drop below the brownout power-down level) is required to reset and restart the device.

Pulling the reset pin during the LPM4.5 low-power reset state causes the device to enter its default reset state (with higher current consumption) and the device starts up when the supply rises above the SVS power-up level.

If the device is already in LPMx.5 (with SVS enabled) before the supply voltage drops below the SVS threshold, then the device automatically enters the low-power reset state; that is, the device transitions into a LPM4.5 state with SVS, RTC domain, and all wake-up events disabled. In LPMx.5, the I/Os are already in a defined state; therefore, no NMI handling required to define the I/O states.

2.2.6 Brownout Reset (BOR)

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 BOR 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 causes a software-driven BOR. PMMBORIFG is set accordingly. 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 is set accordingly. A POR also initiates a PUC. PMMPORIFG can be cleared by software or by reading SYSRSTIV. Both PMMSWBOR and PMMSWPOR are self clearing. See the SYS module for complete descriptions of BOR, POR, and PUC resets.

2.2.7 $\overline{\text{RST}}$/NMI

The external $\overline{\text{RST}}$/NMI terminal is pulled low on a BOR reset condition. The $\overline{\text{RST}}$/NMI can be used as reset source for the rest of the application.

2.2.8 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.

2.2.9 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 or 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.

2.3 PMM Registers

The PMM registers are listed in Table 2-1. 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 2-1.

The password defined in the PMMCTL0 register controls access to all PMM registers except PM5CTL0. PM5CTL0 can be accessed without a password. After the correct password is written, the write access is enabled (this includes byte access to the PMMCTL0 lower byte). 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 2-1. PMM Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	PMMCTL0	PMM control register 0	Read/write	Word	9640h	Section 2.3.1
00h	PMMCTL0_L		Read/write	Byte	40h	
01h	PMMCTL0_H		Read/write	Byte	96h	
02h	PMMCTL1	PMM control register 1	Read/write(1)	Word	9600h	Section 2.3.2
02h	PMMCTL1_L		Read(1)	Byte	00h	
03h	PMMCTL1_H		Read(1)	Byte	96h	
0Ah	PMMIFG	PMM interrupt flag register	Read/write	Word	0000h	Section 2.3.3
0Ah	PMMIFG_L		Read/write	Byte	00h	
0Bh	PMMIFG_H		Read/write	Byte	00h	
10h	PM5CTL0	Power mode 5 control register 0	Read/write	Word	0001h	Section 2.3.4
10h	PM5CTL0_L		Read/write	Byte	01h	
11h	PM5CTL0_H		Read/write	Byte	00h	

(1) PMMCTL1 can be written as word only.

2.3.1 PMMCTL0 Register (offset = 00h) [reset = 9640h]

Power Management Module Control Register 0

Figure 2-4. PMMCTL0 Register

15	14	13	12	11	10	9	8
PMMPW							
rw-1	rw-0	rw-0	rw-1	rw-0	rw-1	rw-1	rw-0
7	6	5	4	3	2	1	0
PMMLPRST	SVSHE	Reserved	PMMREGOFF	PMMSWPOR	PMMSWBOR	Reserved	
rw-[0]	rw-[1]	r0	rw-[0]	rw-(0)	rw-[0]	r0	r0

Table 2-2. PMMCTL0 Register Description

Bit	Field	Type	Reset	Description
15-8	PMMPW	RW	96h	PMM password. Always reads as 096h. Must be written with 0A5h to unlock the PMM registers.
7	PMMLPRST	RW	0h	Low-Power Reset Enable. 0b = Low-Power Reset disabled. SVSH resets device. 1b = Low-Power Reset enabled. SVSH does not reset device but triggers a system NMI.
6	SVSHE	RW	1h	High-side SVS enable. 0b = High-side SVS (SVSH) is disabled in LPM2, LPM3, LPM4, LPM3.5, and LPM4.5. SVSH is always enabled in active mode, LPM0, and LPM1. 1b = SVSH is always enabled.
5	Reserved	R	0h	Reserved. Always reads as 0.
4	PMMREGOFF	RW	0h	Regulator off 0b = Regulator remains on when going into LPM3 or LPM4 1b = Regulator is turned off when going to LPM3 or LPM4. System enters LPM3.5 or LPM4.5, respectively.
3	PMMSWPOR	RW	0h	Software POR. Setting this bit to 1 triggers a POR. This bit is self clearing. 0b = Normal operation 1b = Set to 1 to trigger a POR
2	PMMSWBOR	RW	0h	Software brownout reset. Setting this bit to 1 triggers a BOR. This bit is self clearing. 0b = Normal operation 1b = Set to 1 to trigger a BOR
1-0	Reserved	R	0h	Reserved. Always reads as 0.

2.3.2 PMMCTL1 Register (offset = 02h) [reset = 9600h]

Power Management Module Control Register 1

Figure 2-5. PMMCTL1 Register

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

Table 2-3. PMMCTL1 Register Description

Bit	Field	Type	Reset	Description
15-0	Reserved	R	9600h	Reserved. Always reads as 9600h.

2.3.3 PMMIFG Register (offset = 0Ah) [reset = 0000h]

Power Management Module Interrupt Flag Register

Figure 2-6. PMMIFG Register

15	14	13	12	11	10	9	8
PMMLPM5IFG	Reserved	SVSHIFG	Reserved		PMMPORIFG	PMMRSTIFG	PMMBORIFG
rw-{0}	r0	rw-{0}	r0	r0	rw-[0]	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

Table 2-4. PMMIFG Register Description

Bit	Field	Type	Reset	Description
15	PMMLPM5IFG	RW	0h	LPMx.5 flag. This bit has a specific reset conditions. This bit is only 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 DVCC domain triggered by the high-side SVS (if enabled) or the brownout clears the bit. 0b = Reset not due to wake-up from LPMx.5 1b = Reset due to wake-up from LPMx.5
14	Reserved	R	0h	Reserved. Always reads as 0.
13	SVSHIFG	RW	0h	High-side SVS interrupt flag. This bit has a specific reset conditions. PMMLPRST = 0 (Low-Power Reset disabled): The SVSHIFG interrupt flag is only set if the SVSH is the reset source; that is, if DVCC dropped below the high-side SVS levels but remained above the brownout levels. The bit is cleared by software or by reading the reset vector word SYSRSTIV. PMMLPRST = 1 (Low-Power Reset enabled): The SVSHIFG interrupt flag is set if DVCC dropped below the SVSH power-down level and a system NMI is generated. The bit is cleared by software or by reading the system NMI vector word SYSSNIV. 0b = Reset not due to SVSH 1b = Reset due to SVSH
12-11	Reserved	R	0h	Reserved. Always reads as 0.
10	PMMPORIFG	RW	0h	PMM software POR interrupt flag. This bit has a specific reset conditions. This interrupt flag is only set if a software POR (PMMSWPOR) is triggered. The bit is cleared by software or by reading the reset vector word. 0b = Reset not due to PMMSWPOR 1b = Reset due to PMMSWPOR
9	PMMRSTIFG	RW	0h	PMM reset pin interrupt flag. This bit has a specific reset conditions. This interrupt flag is only 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. 0b = Reset not due to reset pin 1b = Reset due to reset pin
8	PMMBORIFG	RW	0h	PMM software brownout reset interrupt flag. This bit has a specific reset conditions. This interrupt flag is only set if a software BOR (PMMSWBOR) is triggered. The bit is cleared by software or by reading the reset vector word. 0b = Reset not due to PMMSWBOR 1b = Reset due to PMMSWBOR
7-0	Reserved	R	0h	Reserved. Always reads as 0.

2.3.4 PM5CTL0 Register (offset = 10h) [reset = 0001h]

Power Mode 5 Control Register 0

Figure 2-7. PM5CTL0 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							LOCKLPM5
r0	r0	r0	r0	r0	r0	r0	rw-{1}

Table 2-5. PM5CTL0 Register Description

Bit	Field	Type	Reset	Description
15-1	Reserved	R	0h	Reserved. Always reads as 0.
0	LOCKLPM5	RW	1h	Locks I/O pin and other LPMx.5 relevant (for example, RTC) configurations upon exit from LPMx.5. After a power cycle I/O pins are locked in high-impedance state with input Schmitt triggers disabled until LOCKLPM5 is cleared by user. After a wake-up from LPMx.5 I/O pins and other LPMx.5 relevant (for example, RTC) configurations are locked in their states configured prior to LPMx.5 entry until LOCKLPM5 is cleared by user. 0b = I/O pin and LPMx.5 configurations unlocked. 1b = I/O pin and LPMx.5 configuration remains locked.

Clock System (CS) Module

This chapter describes the operation of the clock system, which is implemented in all devices.

Topic **Page**

3.1 Clock System Introduction 77
3.2 Clock System Operation 79
3.3 Module Oscillator (MODOSC) 85
3.4 CS Registers 86

3.1 Clock System Introduction

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

The clock system module includes the following clock sources:

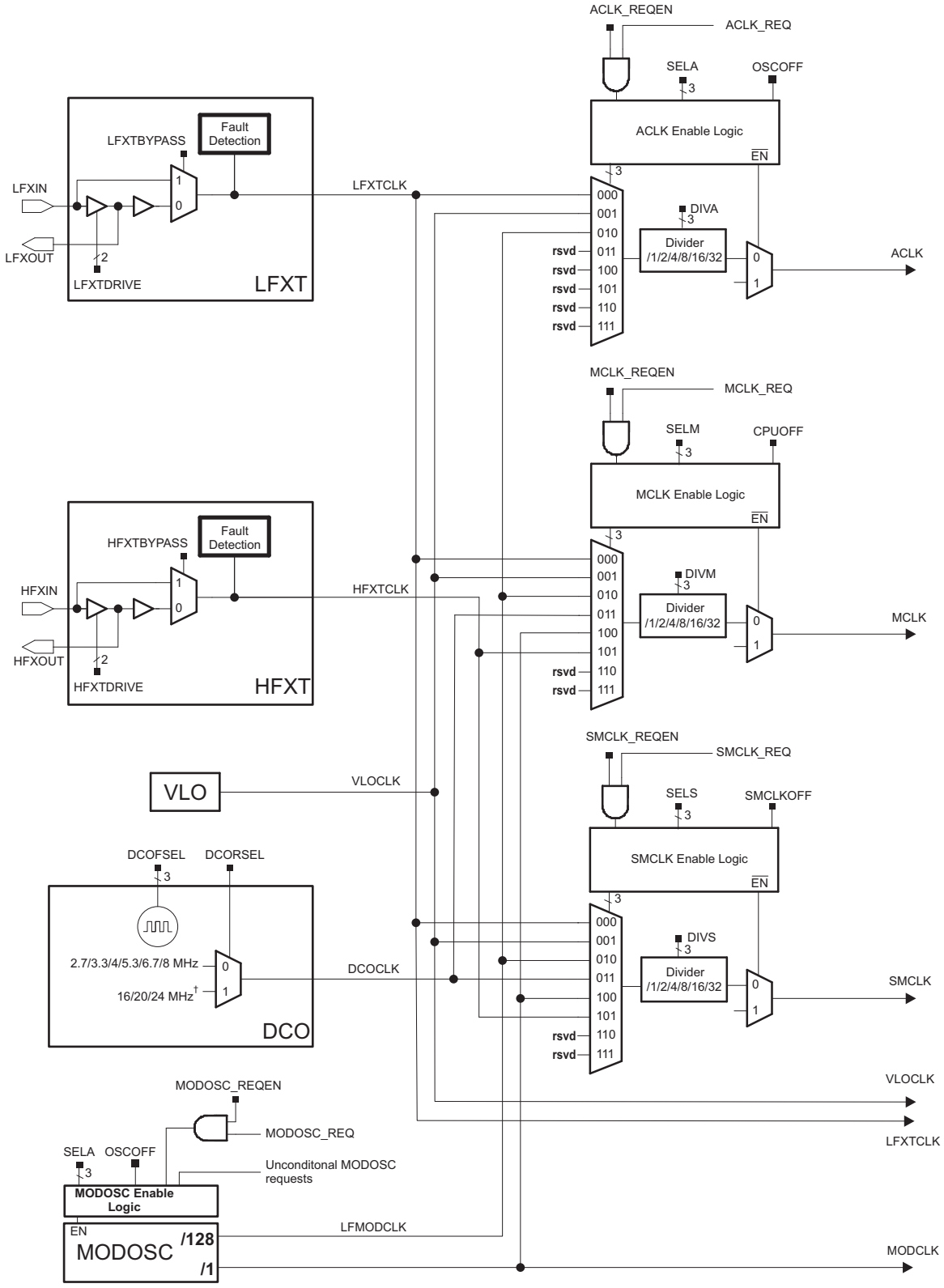
- LFXTCLK: Low-frequency oscillator that can be used either with low-frequency 32768-Hz watch crystals, standard crystals, resonators, or external clock sources in the 50 kHz or below range. When in bypass mode, LFXTCLK can be driven with an external square wave signal.
- VLOCLK: Internal very-low-power low-frequency oscillator with 10-kHz typical frequency
- DCOCLK: Internal digitally controlled oscillator (DCO) with selectable frequencies
- MODCLK: Internal low-power oscillator with 5-MHz typical frequency. LFMODCLK is MODCLK divided by 128.
- HFXTCLK: High-frequency oscillator that can be used with standard crystals or resonators in the 4-MHz to 24-MHz range. When in bypass mode, HFXTCLK can be driven with an external square wave signal.

Four system clock signals are available from the clock module:

- ACLK: Auxiliary clock. The ACLK is software selectable as LFXTCLK, VLOCLK, or LFMODCLK. ACLK can be divided by 1, 2, 4, 8, 16, or 32. ACLK is software selectable by individual peripheral modules.
- MCLK: Master clock. MCLK is software selectable as LFXTCLK, VLOCLK, LFMODCLK, DCOCLK, MODCLK, or HFXTCLK. MCLK can be divided by 1, 2, 4, 8, 16, or 32. MCLK is used by the CPU and system.
- SMCLK: Sub-system master clock. SMCLK is software selectable as LFXTCLK, VLOCLK, LFMODCLK, DCOCLK, MODCLK, or HFXTCLK. SMCLK is software selectable by individual peripheral modules.
- MODCLK: Module clock. MODCLK may also be used by various peripheral modules and is sourced by MODOSC.
- VLOCLK: VLO clock. VLOCLK may also be used directly by various peripheral modules and is sourced by VLO.

NOTE: Not all devices contain both LFXT and HFXT clock sources. See the device-specific data sheet for availability.

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



† Not available on all devices

Figure 3-1. Clock System Block Diagram

3.2 Clock System Operation

After PUC, the CS module default configuration is:

- LFXT is selected as the oscillator source for LFXTCLK. LFXTCLK is selected for ACLK (SELAx = 0) and ACLK is undivided (DIVAx = 0).
- DCOCLK is selected for MCLK and SMCLK (SELMx = SELSx = 3) and each are divided by 8 (DIVMx = DIVSx = 3).
- LFXIN and LFXOUT pins are set to general-purpose I/Os and LFXT remains disabled until the I/O ports are configured for LFXT operation.
- HFXIN and HFXOUT pins are set to general-purpose I/Os and HFXT is disabled.

As previously stated, LFXT is selected by default, but LFXT is disabled. The crystal pins (LFXIN, LFXOUT) are shared with general-purpose I/Os. To enable LFXT, the PSEL bits associated with the crystal pins must be set. When a 32768-Hz crystal is used for LFXTCLK, the fault control logic immediately causes ACLK to be sourced by LFMODCLK, MCLK and SMCLK to be sourced by MODCLK, because LFXT is not stable immediately (see Section 3.2.8).

Status register control bits (SCG0, SCG1, OSCOFF, and CPUOFF) configure the MSP430 operating modes and enable or disable portions of the clock system module (see the *System Resets, Interrupts, and Operating Modes* chapter). Registers CSCTL0 through CSCTL6, configure the CS module.

The CS module can be configured or reconfigured by software at any time during program execution. The CS control registers are password protected to prevent inadvertent access.

3.2.1 CS 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 CS module addresses these conflicting requirements by allowing the user to select from the three available clock signals: ACLK, MCLK, and SMCLK. A flexible clock distribution and divider system is provided to fine tune the individual clock requirements.

3.2.2 LFXT Oscillator

The LFXT oscillator supports ultralow-current consumption using a 32768-Hz watch crystal. A watch crystal connects to LFXIN and LFXOUT and requires external capacitors on both terminals. These capacitors should be sized according to the crystal or resonator specifications. Different crystal or resonator ranges are supported by LFXT by choosing the proper LFXTDRIVE settings.

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

If the PSEL bit associated with LFXIN is cleared, both LFXIN and LFXOUT ports are configured as general-purpose I/Os, and LFXT is disabled.

LFXT is enabled under any of the following conditions:

- LFXT is a source for ACLK (SELAx = 0) and in active mode (AM) through LPM3 (OSCOFF = 0)
- LFXT is a source for MCLK (SELMx = 0) and in active mode (AM) (CPUOFF = 0)
- LFXT is a source for SMCLK (SELSx = 0) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- LFXTOFF = 0. LFXT enabled in active mode (AM) through LPM4.

- LFXT is selected as the source for RTC, RTC is enabled (RTCHOLD = 0), and LPMx.5 is entered.

NOTE: If LFXT is disabled when entering into a low-power mode, it is not fully enabled and stable upon exit from the low-power mode, because its enable time is much longer than the wake-up time. If the application requires or desires to keep LFXT enabled during a low-power mode, the LFXTOFF bit can be cleared prior to entering the low-power mode. This causes LFXT to remain enabled.

3.2.3 HFXT Oscillator

The HFXT high-frequency oscillator can be used with standard crystals or resonators in the 4 MHz to 24 MHz range. The HFXTDRIVE bits select the drive capability of HFXT. HFXTDRIVE bits can be used to provide optimal settings for a given crystal characteristic. HFXT sources HFXTCLK. The HFFREQ bits must be set for the appropriate frequency range of operation as show in Table 3-1 in crystal or bypass modes of operation.

Table 3-1. HFFREQ Settings

HFXT Frequency Range	HFFREQ[1:0]
0 to 4 MHz	00
> 4 MHz to 8 MHz	01
> 8 MHz to 16 MHz	10
> 16 MHz to 24 MHz	11

NOTE: The HFXT HFFREQ bit settings are also used to control the Power Management Module and must match the intended frequency of operation for proper functioning of the device as listed in Table 3-1. In addition, these bits should be configured properly prior to use of HFXT in either crystal or bypass modes of operation.

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

If the PSEL bit associated with HFXIN is cleared, both HFXIN and HFXOUT ports are configured as general-purpose I/Os, and HFXT is disabled.

HFXT is enabled under any of the following conditions:

- HFXT is a source for MCLK (SELMx = 5) and in active mode (AM) (CPUOFF = 0)
- HFXT is a source for SMCLK (SELSx = 5) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- HFXTOFF = 0. HFXT enabled in active mode (AM) through LPM4.

NOTE: If HFXT is disabled when entering into a low-power mode, it is not fully enabled and stable upon exit from the low-power mode, because its enable time is much longer than the wake-up time. If the application requires or desires to keep HFXT enabled during a low-power mode, the HFXTOFF bit can be cleared prior to entering the low-power mode. This causes HFXT to remain enabled.

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

The internal VLO provides a typical frequency of 10 kHz (see the 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. To conserve power, VLO is powered down when not needed and enabled only when required.

VLO is enabled under any of the following conditions:

- VLO is a source for ACLK (SELAx = 1) and in active mode (AM) through LPM3 (OSCOFF = 0)
- VLO is a source for MCLK (SELMx = 1) and in active mode (AM) (CPUOFF = 0)
- VLO is a source for SMCLK (SELSx = 1) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- VLOOFF = 0. VLO enabled in active mode (AM) through LPM4.
- VLO is selected as the source for RTC, RTC is enabled (RTCHOLD = 0), and LPMx.5 is entered.

3.2.5 Module Oscillator (MODOSC)

The CS module also supports an internal oscillator, MODOSC, that can be used by ACLK, MCLK, or SMCLK, as well as by other modules in the system. It is also used as a fail-safe clock source as described in Section 3.2.8. The MODOSC sources MODCLK and LFMODCLK.

To conserve power, MODOSC is powered down when not needed and is 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; for example, ADC or fail-safe.

MODOSC is enabled under any of the following conditions:

- LFMODCLK is a source for ACLK (SELAx = 2) and in active mode (AM) through LPM3 (OSCOFF = 0)
- LFMODCLK or MODCLK is a source for MCLK (SELMx = 2, 4) and in active mode (AM) (CPUOFF = 0)
- LFMODCLK or MODCLK is a source for SMCLK (SELSx = 2, 4) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- During a fault detection (when fault detection enabled) and LFXT or HFXT is active.
- Unconditional requests from modules that require MODCLK; for example, ADC conversion clock when selected as the source.
- LFMODCLK is used as the fail-safe source for the WDT in watchdog mode. Should the selected source for ACLK or SMCLK not be available, the WDT automatically switches to LFMODCLK as its clock source.

The ADC12 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 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.

3.2.6 Digitally Controlled Oscillator (DCO)

The DCO is an integrated digitally controlled oscillator. The DCO has three frequency settings determined by the DCOFSEL bits. Each frequency is trimmed at the factory. The DCO can be used as a source for MCLK or SMCLK. See the device-specific data sheet for DCO characteristics.

The DCO frequency can be changed at any time, but care should be taken to ensure no other system clock frequency constraints are exceeded with the new frequency selection. Any change in the DCOFSEL or DCORSEL bits causes the DCOCLK to be held for four clock cycles before releasing the new value into the system. This allows for the DCO to settle properly.

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

A peripheral module requests its clock sources automatically from the CS module if required for its proper operation, regardless of the current power mode of operation, as shown in Figure 3-2.

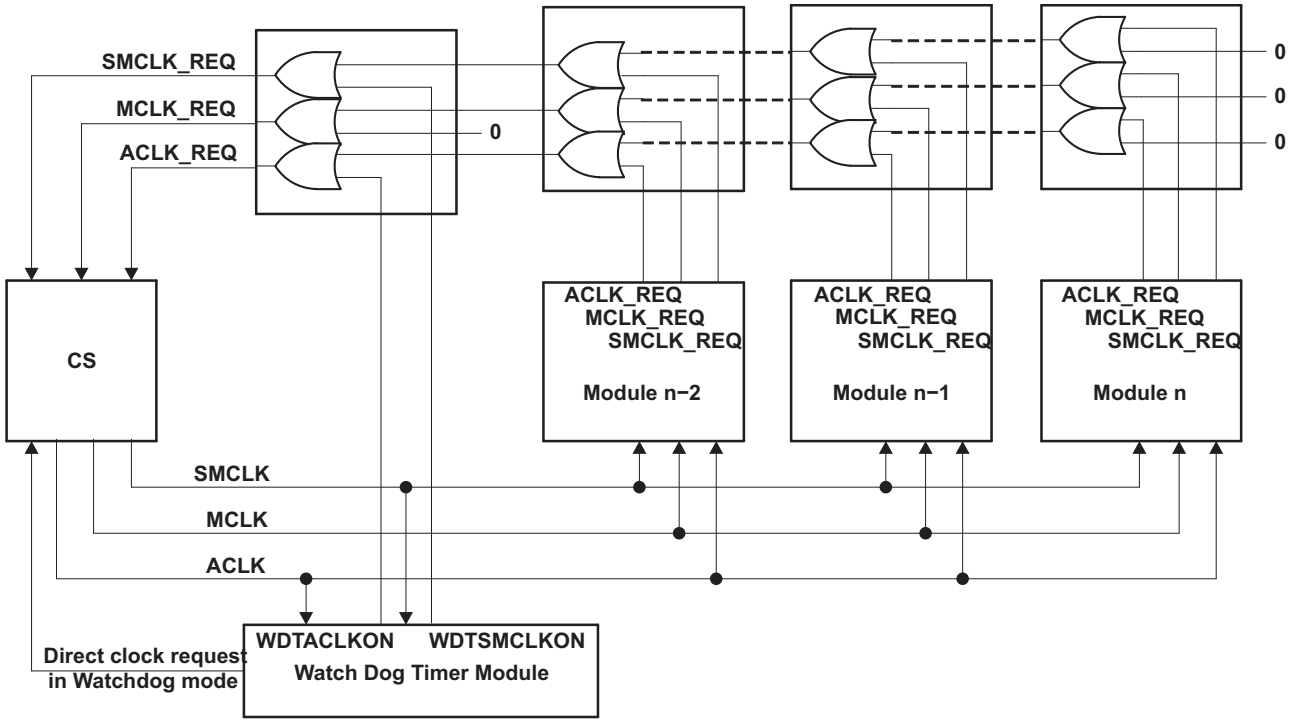


Figure 3-2. Module Request Clock System

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 CS system. The CS, in turn, enables ACLK regardless of the power mode 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 LFXT and LFXT was not enabled, an oscillator fault condition occurs, and the software must handle the event. The watchdog, due to its security requirement, actively selects the LFMODCLK 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 causes more current consumption than the specified values in the data sheet. By default, the clock request feature is enabled. The feature can be disabled for each system clock by clearing ACLKREQEN, MCLKREQEN, or SMCLKREQEN for the respective clocks. This does not disable fail-safe clock requests; for example, those of the watchdog timer or the clock system itself.

The function of the ACLKREQEN, MCLKREQEN, and SMCLKREQEN bits are dependent upon which power mode is selected; that is, they do not have an effect across all power modes. For example, ACLKREQEN is used to enable or disable ACLK requests. It is effective only in LPM4, because ACLK is always active in all other modes (AM, LPM0, LPM1, LPM2, LPM3). SMCLKREQEN is used to enable or disable SMCLK requests. When SMCLKOFF = 0 and in AM, LPM0, or LPM1, it is a don't care, because SMCLK is always on in these cases. For SMCLKOFF = 0 and in LPM2, LPM3, and LPM4, SMCLKREQEN can be used to enable or disable SMCLK requests, because SMCLK is normally off in these modes. When SMCLKOFF = 1, SMCLKREQEN can be used to enable or disable SMCLK requests, because SMCLK is normally off in all power modes under this condition. This is summarized in Table 3-2.

Table 3-2. System Clocks, Power Modes, and Clock Requests

Mode	System Clocks							
	MCLK		ACLK		SMCLK			
					SMCLKOFF = 0		SMCLKOFF = 1	
	MCLKREQEN = 0 and Clock Requested	MCLKREQEN = 1 and Clock Requested	ACLKREQEN = 0 and Clock Requested	ACLKREQEN = 1 and Clock Requested	SMCLKREQEN = 0 and Clock Requested	SMCLKREQEN = 1 and Clock Requested	SMCLKREQEN = 0 and Clock Requested	SMCLKREQEN = 1 and Clock Requested
AM	Active	Active	Active	Active	Active	Active	Disabled	Active
LPM0	Disabled	Active	Active	Active	Active	Active	Disabled	Active
LPM1	Disabled	Active	Active	Active	Active	Active	Disabled	Active
LPM2	Disabled	Active	Active	Active	Disabled	Active	Disabled	Active
LPM3	Disabled	Active	Active	Active	Disabled	Active	Disabled	Active
LPM4	Disabled	Active	Disabled	Active	Disabled	Active	Disabled	Active
LPM3.5	Disabled	Disabled	Disabled[(1)]	Disabled	Disabled	Disabled	Disabled	Disabled
LPM4.5	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled

(1) LFXTCLK is available directly as the clock source to the RTC module.

3.2.8 CS Module Fail-Safe Operation

The CS module incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault for LFXT and HFXT as shown in Figure 3-3. The available fault conditions are:

- Low-frequency oscillator fault (LFXTOFFG) for LFXT
- High-frequency oscillator fault (HFXTOFFG) for HFXT
- External clock signal faults for all bypass modes; that is, LFXTBYPASS = 1 or HFXTBYPASS = 1

The crystal oscillator fault bits LFXTOFFG and HFXTOFFG 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, even 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.

The OFIFG oscillator-fault interrupt flag is set and latched at POR or when any oscillator fault (LFXTOFFG or HFXTOFFG) is detected. When OFIFG is set and OFIE is set, the OFIFG requests a user 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 and 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 LFXT is sourcing any system clock (ACLK, MCLK, or SMCLK) and a fault is detected, the system clock is automatically switched to LFMODCLK for its clock source. The LFXT fault logic works in all power modes, including LPM3.5. Similarly, if HFXT is sourcing MCLK or SMCLK, and a fault is detected, the system clock is automatically switched to MODCLK for its clock source. By default, the HFXT fault logic works in all power modes except LPM3.5 or LPM4.5, because high-frequency operation in these modes is not supported. The fail-safe logic does not change the respective SELA, SELM, and SELS bit settings. The fail-safe mechanism behaves the same in normal and bypass modes.

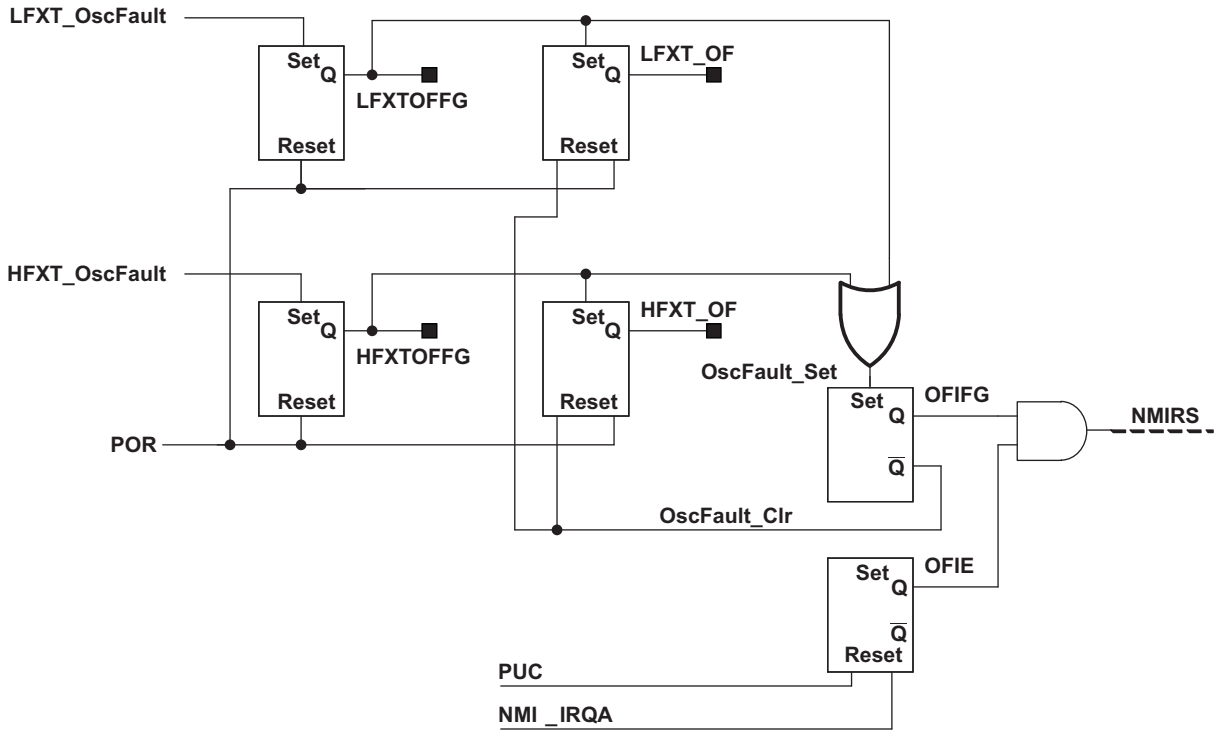


Figure 3-3. Oscillator Fault Logic

NOTE: Fault conditions

LFXT_OscFault: When the fault detection logic is enabled (ENLFXTD = 1), this signal is set after the LFXT oscillator has stopped operation and is cleared after operation resumes. The fault condition causes LFXTOFFG to be set and remain set. If the user clears LFXTOFFG and the fault condition still exists, LFXTOFFG remains set.

HFXT_OscFault: When the fault detection logic is enabled (ENHFXTD = 1), this signal is set after the HFXT oscillator has stopped operation and is cleared after operation resumes. The fault condition causes HFXTOFFG to be set and remain set. If the user clears HFXTOFFG and the fault condition still exists, HFXTOFFG remains set.

NOTE: Fault logic

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: The LFXT startup includes a counter that ensures that 1024 valid clock cycles have passed before LFXT_OscFault signal is cleared. A valid cycle is any cycle that meets the frequency requirement ($f_{Fault,LF}$) as outlined in the device specific data sheet. Any crystal fault restarts the counter. It is recommended that the counter always be enabled, however the counter can be disabled by clearing ENSTFCNT1. Similarly, HFXT startup includes a counter. It can be disabled by clearing ENSTFCNT2. The disabling of the counters is valid for bypass and normal modes of operation.

3.2.9 Synchronization of Clock Signals

When switching ACLK, MCLK or SMCLK from one clock source to the another, the switch is synchronized to avoid critical race conditions as shown in Figure 3-4:

- 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.

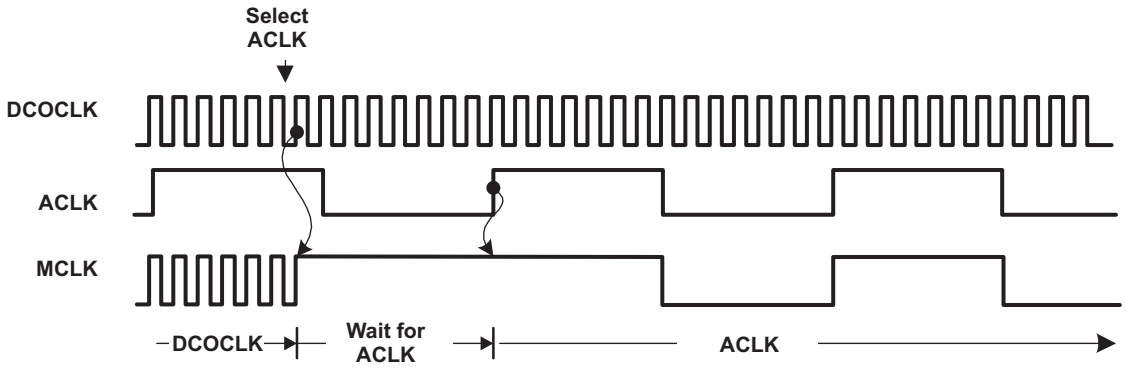


Figure 3-4. Switch MCLK from DCOCLK to LFXTCLK

3.3 Module Oscillator (MODOSC)

The CS module also supports an internal oscillator, MODOSC, that is used by the power management module and, optionally, by other modules in the system. It is also used as a fail-safe clock source as described in Section 3.2.8. 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; for example, ADC, LFXT fail-safe, HFXT fail-safe, or WDT fail-safe.

The ADC12 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 module issues an unconditional request for the ADC12OSC clock source, which enables the MODOSC source if it is not already enabled by other modules' previous requests.

3.4 CS Registers

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

The password defined in CSCTL0 controls access to the CS registers. After the correct password is written in word mode, write access to the CS registers is enabled. Write access is disabled by writing an incorrect password in byte mode to the CSCTL0 upper byte.

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 3-3. CS Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	CSCTL0	Clock System Control 0	Read/write	Word	9600h	Section 3.4.1
00h	CSCTL0_L		Read/write	Byte	00h	
01h	CSCTL0_H		Read/write	Byte	96h	
02h	CSCTL1	Clock System Control 1	Read/write	Word	000Ch	Section 3.4.2
02h	CSCTL1_L		Read/write	Byte	0Ch	
03h	CSCTL1_H		Read/write	Byte	00h	
04h	CSCTL2	Clock System Control 2	Read/write	Word	0033h	Section 3.4.3
04h	CSCTL2_L		Read/write	Byte	33h	
05h	CSCTL2_H		Read/write	Byte	00h	
06h	CSCTL3	Clock System Control 3	Read/write	Word	0033h	Section 3.4.4
06h	CSCTL3_L		Read/write	Byte	33h	
07h	CSCTL3_H		Read/write	Byte	00h	
08h	CSCTL4	Clock System Control 4	Read/write	Word	CDC9h	Section 3.4.5
08h	CSCTL4_L		Read/write	Byte	C9h	
09h	CSCTL4_H		Read/write	Byte	CDh	
0Ah	CSCTL5	Clock System Control 5	Read/write	Word	00C0h	Section 3.4.6
0Ah	CSCTL5_L		Read/write	Byte	C0h	
0Bh	CSCTL5_H		Read/write	Byte	00h	
0Ch	CSCTL6	Clock System Control 6	Read/write	Word	0007h	Section 3.4.7
0Ch	CSCTL6_L		Read/write	Byte	07h	
0Dh	CSCTL6_H		Read/write	Byte	00h	

3.4.1 CSCTL0 Register

Clock System Control 0 Register

Figure 3-5. CSCTL0 Register

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

Table 3-4. CSCTL0 Register Description

Bit	Field	Type	Reset	Description
15-8	CSKEY	RW	96h	CSKEY password. Must always be written with A5h; a PUC is generated if any other value is written. Always reads as 96h. After the correct password is written, all CS registers are available for writing.
7-0	Reserved	R	0h	Reserved. Always reads as 0.

3.4.2 CSCTL1 Register

Clock System Control 1 Register

Figure 3-6. CSCTL1 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	DCORSEL	Reserved		DCOFSEL			Reserved
r0	rw-[0]	r0	r0	rw-[1]	rw-[1]	rw-[0]	r0

Table 3-5. CSCTL1 Register Description

Bit	Field	Type	Reset	Description
15-7	Reserved	R	0h	Reserved. Always reads as 0.
6	DCORSEL	RW	0h	DCO range select. For high speed devices, this bit can be written by the user. For low speed devices, it is always reset. See description of DCOFSEL bit for details.
5-4	Reserved	R	0h	Reserved. Always reads as 0.
3-1	DCOFSEL	RW	6h	DCO frequency select. Selects frequency settings for the DCO. Values shown below are approximate. Please refer to the device specific datasheet. 000b = If DCORSEL = 0: 1 MHz; If DCORSEL = 1: 1 MHz 001b = If DCORSEL = 0: 2.67 MHz; If DCORSEL = 1: 5.33 MHz 010b = If DCORSEL = 0: 3.33 MHz; If DCORSEL = 1: 6.67 MHz 011b = If DCORSEL = 0: 4 MHz; If DCORSEL = 1: 8 MHz 100b = If DCORSEL = 0: 5.33 MHz; If DCORSEL = 1: 16 MHz 101b = If DCORSEL = 0: 6.67 MHz; If DCORSEL = 1: 21 MHz 110b = If DCORSEL = 0: 8 MHz; If DCORSEL = 1: 24 MHz 111b = If DCORSEL = 0: Reserved. Defaults to 8. It is not recommended to use this setting; If DCORSEL = 1: Reserved. Defaults to 24. It is not recommended to use this setting
0	Reserved	R	0h	Reserved. Always reads as 0.

3.4.3 CSCTL2 Register

Clock System Control 2 Register

Figure 3-7. CSCTL2 Register

15	14	13	12	11	10	9	8
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-0	rw-1	rw-1	r0	rw-0	rw-1	rw-1

Table 3-6. CSCTL2 Register Description

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10-8	SELA	RW	0h	Selects the ACLK source 000b = LFXTCLK when LFXT available, otherwise VLOCLK. 001b = VLOCLK 010b = LFMODCLK 011b = Reserved. Defaults to LFMODCLK. Not recommended for use to ensure future compatibility. 100b = Reserved. Defaults to LFMODCLK. Not recommended for use to ensure future compatibility. 101b = Reserved. Defaults to LFMODCLK. Not recommended for use to ensure future compatibility. 110b = Reserved. Defaults to LFMODCLK. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to LFMODCLK. Not recommended for use to ensure future compatibility.
7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	SELS	RW	3h	Selects the SMCLK source 000b = LFXTCLK when LFXT available, otherwise VLOCLK. 001b = VLOCLK 010b = LFMODCLK 011b = DCOCLK 100b = MODCLK 101b = HFXTCLK when HFXT available, otherwise DCOCLK. 110b = Reserved. Defaults to HFXTCLK. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to HFXTCLK. Not recommended for use to ensure future compatibility.
3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	SELM	RW	3h	Selects the MCLK source 000b = LFXTCLK when LFXT available, otherwise VLOCLK 001b = VLOCLK 010b = LFMODCLK 011b = DCOCLK 100b = MODCLK 101b = HFXTCLK when HFXT available, otherwise DCOCLK 110b = Reserved. Defaults to HFXTCLK. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to HFXTCLK. Not recommended for use to ensure future compatibility.

3.4.4 CSCTL3 Register

Clock System Control 3 Register

Figure 3-8. CSCTL3 Register

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

Table 3-7. CSCTL3 Register Description

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10-8	DIVA	RW	0h	ACLK source divider. Divides the frequency of the ACLK clock source. 000b = /1 001b = /2 010b = /4 011b = /8 100b = /16 101b = /32 110b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility.
7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	DIVS	RW	3h	SCLK source divider. Divides the frequency of the SCLK clock source. 000b = /1 001b = /2 010b = /4 011b = /8 100b = /16 101b = /32 110b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility.
3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	DIVM	RW	3h	MCLK source divider. Divides the frequency of the MCLK clock source. 000b = /1 001b = /2 010b = /4 011b = /8 100b = /16 101b = /32 110b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility.

3.4.5 CSCTL4 Register

Clock System Control 4 Register

Figure 3-9. CSCTL4 Register

15	14	13	12	11	10	9	8
HFXTDRIVE		Reserved	HFXTBYPASS	HFFREQ		Reserved	HFXTOFF
rw-1	rw-1	r0	rw-0	rw-1	rw-1	r0	rw-1
7	6	5	4	3	2	1	0
LFXTDRIVE		Reserved	LFXTBYPASS	VLOOFF	Reserved	SMCLKOFF	LFXTOFF
rw-1	rw-1	rw-0	rw-0	rw-1	r0	rw-0	rw-1

Table 3-8. CSCTL4 Register Description

Bit	Field	Type	Reset	Description
15-14	HFXTDRIVE	RW	3h	The HFXT oscillator current can be adjusted to its drive needs. This in combination with the HFFREQ bits can be used for optimizing crystal power based on crystal characteristics. 00b = Lowest current consumption 01b = Increased drive strength HFXT oscillator 10b = Increased drive strength HFXT oscillator 11b = Maximum drive strength HFXT oscillator
13	Reserved	R	0h	Reserved. Always reads as 0.
12	HFXTBYPASS	RW	0h	HFXT bypass select 0b = HFXT sourced from external crystal 1b = HFXT sourced from external clock signal
11-10	HFFREQ	RW	3h	The HFXT frequency selection. These bits must be set to the appropriate frequency for crystal or bypass modes of operation. 00b = 0 to 4 MHz 01b = Greater than 4 MHz to 8 MHz 10b = Greater than 8 MHz to 16 MHz 11b = Greater than 16 MHz to 24 MHz
9	Reserved	R	0h	Reserved. Always reads as 0.
8	HFXTOFF	RW	1h	Turns off the HFXT oscillator 0b = HFXT is on if HFXT is selected via the port selection and HFXT is not in bypass mode of operation 1b = HFXT is off if it is not used as a source for ACLK, MCLK, or SMCLK
7-6	LFXTDRIVE	RW	3h	The LFXT oscillator current can be adjusted to its drive needs. 00b = Lowest drive strength and current consumption LFXT oscillator 01b = Increased drive strength LFXT oscillator 10b = Increased drive strength LFXT oscillator 11b = Maximum drive strength and maximum current consumption LFXT oscillator
5	Reserved	RW	0h	Reserved. Must be written as zero.
4	LFXTBYPASS	RW	0h	LFXT bypass select 0b = LFXT sourced from external crystal 1b = LFXT sourced from external clock signal
3	VLOOFF	RW	1h	VLO off. This bit turns off the VLO. 0b = VLO is on 1b = VLO is off if it is not used as a source for ACLK, MCLK, or SMCLK or if not used as a source for the RTC in LPM3.5
2	Reserved	R	0h	Reserved. Always reads as 0.
1	SMCLKOFF	RW	0h	SMCLK off. This bit turns off the SMCLK. 0b = SMCLK on 1b = SMCLK off

Table 3-8. CSCTL4 Register Description (continued)

Bit	Field	Type	Reset	Description
0	LFXTOFF	RW	1h	LFXT off. This bit turns off the LFXT. 0b = LFXT is on if LFXT is selected via the port selection and LFXT is not in bypass mode of operation 1b = LFXT is off if it is not used as a source for ACLK, MCLK, or SMCLK

3.4.6 CSCTL5 Register

Clock System Control 5 Register

Figure 3-10. CSCTL5 Register

15	14	13	12	11	10	9	8
Reserved							
r0	r0	0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
ENSTFCNT2	ENSTFCNT1	Reserved				HFXTOFFG	LFXTOFFG
rw-(1)	rw-(1)	r0	r0	r0	r0	rw-(0)	rw-(0)

Table 3-9. CSCTL5 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7	ENSTFCNT2	RW	1h	Enable start counter for HFXT when available. 0b = Startup fault counter disabled. Counter is cleared. 1b = Startup fault counter enabled
6	ENSTFCNT1	RW	1h	Enable start counter for LFXT. 0b = Startup fault counter disabled. Counter is cleared. 1b = Startup fault counter enabled
5-2	Reserved	R	0h	Reserved. Always reads as 0.
1	HFXTOFFG	RW	0h	HFXT oscillator fault flag. If this bit is set, the OFIFG flag is also set. HFXTOFFG is set if a HFXT fault condition exists. HFXTOFFG can be cleared via software. If the HFXT fault condition still remains, HFXTOFFG is set. 0b = No fault condition occurred after the last reset 1b = HFXT fault; an HFXT fault occurred after the last reset
0	LFXTOFFG	RW	1h	LFXT oscillator fault flag. If this bit is set, the OFIFG flag is also set. LFXTOFFG is set if a LFXT fault condition exists. LFXTOFFG can be cleared via software. If the LFXT fault condition still remains, LFXTOFFG is set. 0b = No fault condition occurred after the last reset 1b = LFXT fault; an LFXT fault occurred after the last reset

3.4.7 CSCTL6 Register

Clock System Control 6 Register

Figure 3-11. CSCTL6 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				MODCLKREQEN	SMCLKREQEN	MCLKREQEN	ACLKREQEN
r0	r0	r0	r0	rw-(0)	rw-(1)	rw-(1)	rw-(1)

Table 3-10. CSCTL6 Register Description

Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved. Always reads as 0.
3	MODCLKREQEN	RW	0h	MODCLK clock request enable. Setting this enables conditional module requests for MODCLK. 0b = MODCLK conditional requests are disabled 1b = MODCLK conditional requests are enabled
2	SMCLKREQEN	RW	1h	SMCLK clock request enable. Setting this enables conditional module requests for SMCLK. 0b = SMCLK conditional requests are disabled 1b = SMCLK conditional requests are enabled
1	MCLKREQEN	RW	1h	MCLK clock request enable. Setting this enables conditional module requests for MCLK. 0b = MCLK conditional requests are disabled 1b = MCLK conditional requests are enabled
0	ACLKREQEN	RW	1h	ACLK clock request enable. Setting this enables conditional module requests for ACLK. 0b = ACLK conditional requests are disabled 1b = ACLK conditional requests are enabled

CPUX

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

NOTE: The MSP430X CPUX implemented on this device family, formally called CPUXV2, has in some cases, slightly different cycle counts from the MSP430X CPUX implemented on the 2xx and 4xx families.

Topic **Page**

4.1 MSP430X CPU (CPUX) Introduction 95
4.2 Interrupts 97
4.3 CPU Registers 98
4.4 Addressing Modes 104
4.5 MSP430 and MSP430X Instructions 121
4.6 Instruction Set Description 137

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 1MB address range without paging. The MSP430X CPU is completely backward 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.

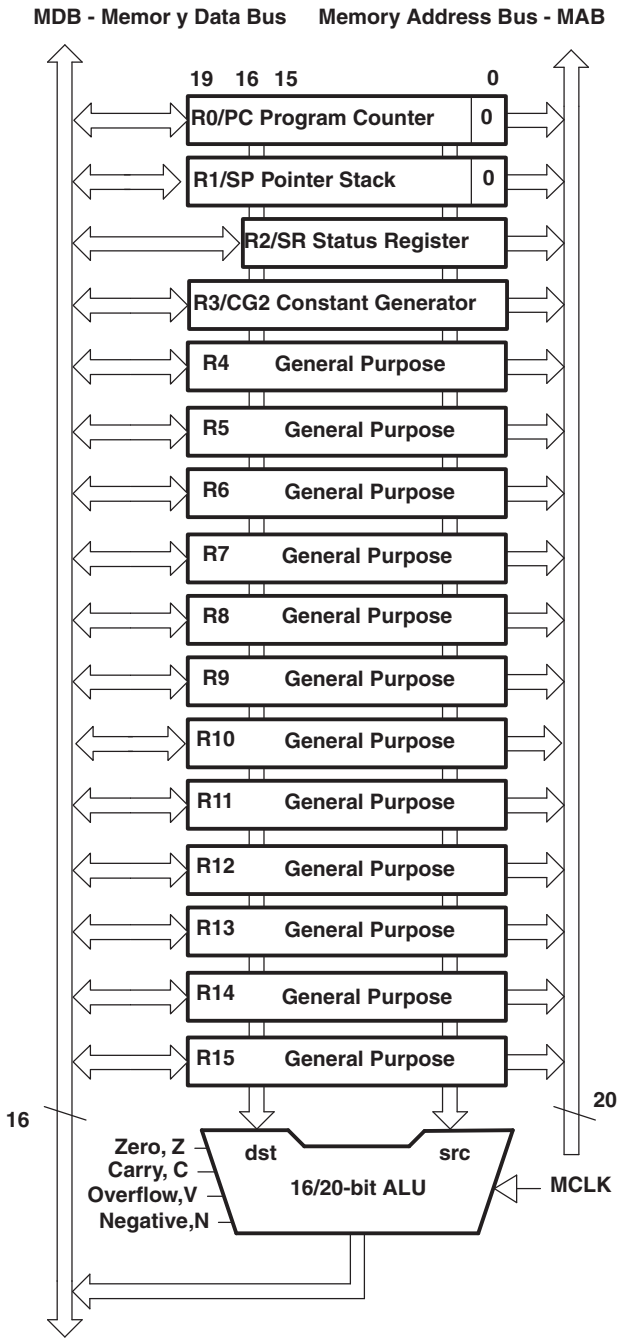


Figure 4-1. MSP430X CPU Block Diagram

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.

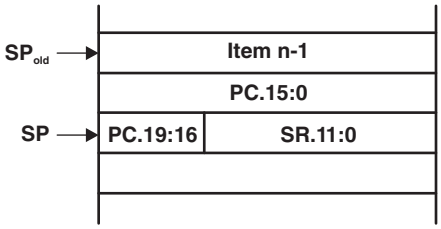


Figure 4-2. PC Storage on the Stack for Interrupts

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 Program Counter (PC, also called 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.

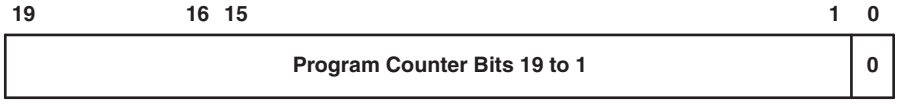


Figure 4-3. Program Counter

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.

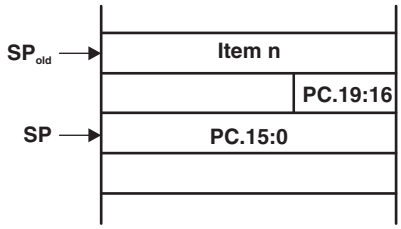


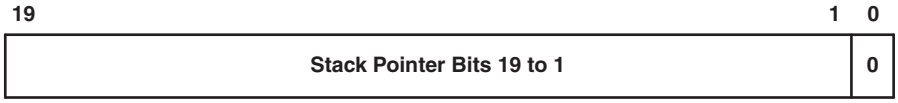
Figure 4-4. PC Storage on the Stack for CALLA

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 Stack Pointer (SP, also called 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

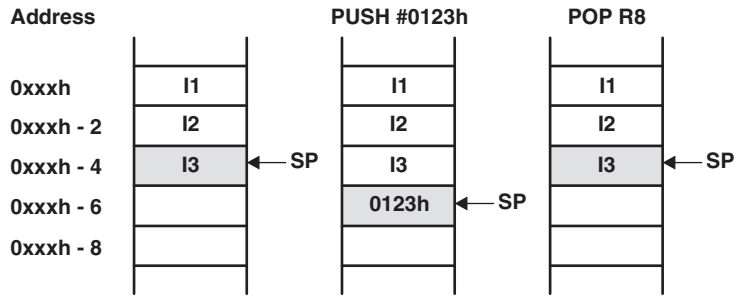


Figure 4-6. Stack Usage

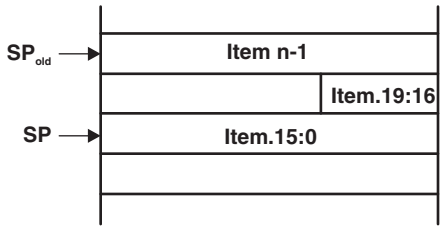


Figure 4-7. PUSHX.A Format on the Stack

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 Status Register (SR, also called 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.

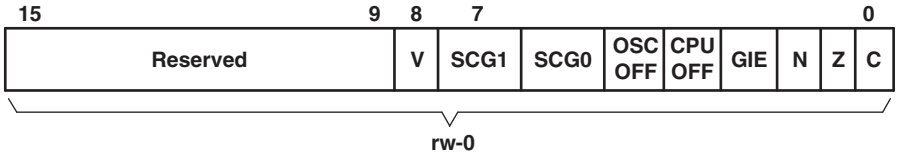


Figure 4-9. SR Bits

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 may be used to enable or disable functions in the clock system depending on the device family; for example, DCO bias enable or disable.	
SCG0	System clock generator 0. This bit may be used to enable or disable functions in the clock system depending on the device family; for example, FLL enable or disable.	
OSCOFF	Oscillator off. This bit, when set, turns off the LFXT1 crystal oscillator when LFXT1CLK is not used for MCLK or SMCLK. In FRAM devices CPUOFF must be 1 to disable the cyrstal oscillator.	
CPUOFF	CPU off. This bit, when set, turns off the CPU and requests a low-power mode according to the settings of bits OSCOFF, SCG0, and SCG1.	
SCG1	The bits CPUOFF, OSCOFF, SCG0 and SCG1 request the system to enter a low-power mode	
SCG0		
OSCOFF		
CPUOFF		
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 by 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 #1,dst
```

4.3.5 General-Purpose Registers (R4 to 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.

Figure 4-10 through Figure 4-14 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.

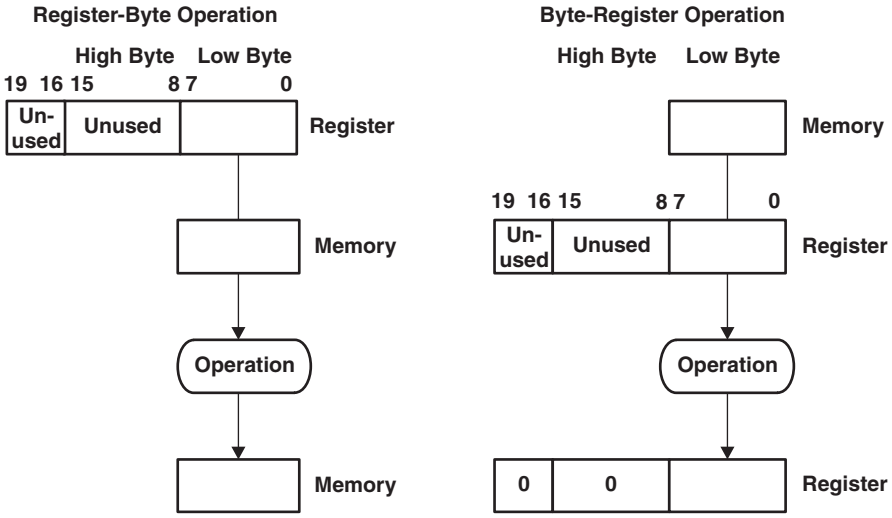


Figure 4-10. Register-Byte and Byte-Register Operation

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.

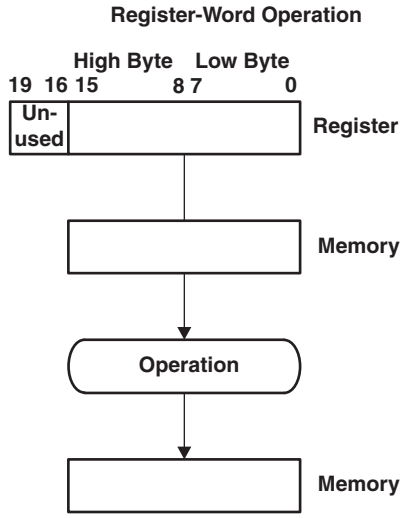


Figure 4-11. Register-Word Operation

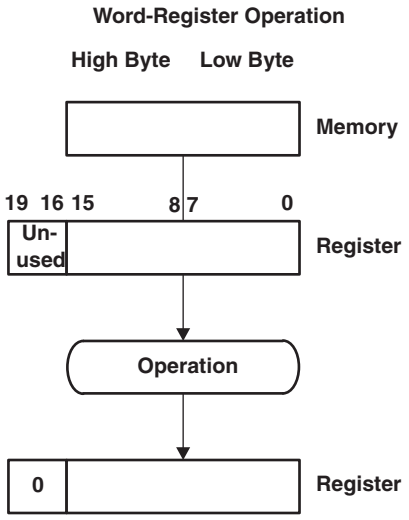


Figure 4-12. Word-Register Operation

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.

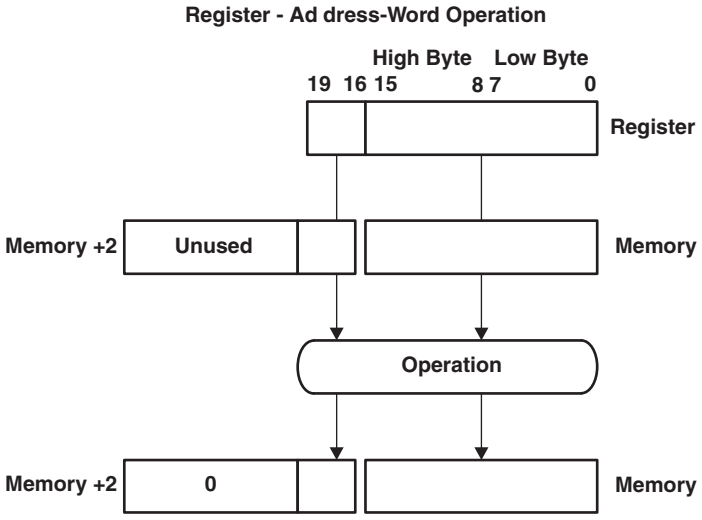


Figure 4-13. Register – Address-Word Operation

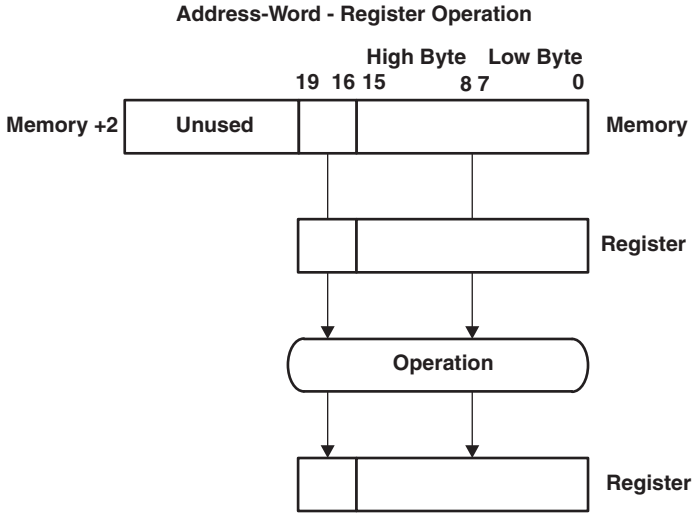


Figure 4-14. Address-Word – Register Operation

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 1MB memory range.

Table 4-3. Source and 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.

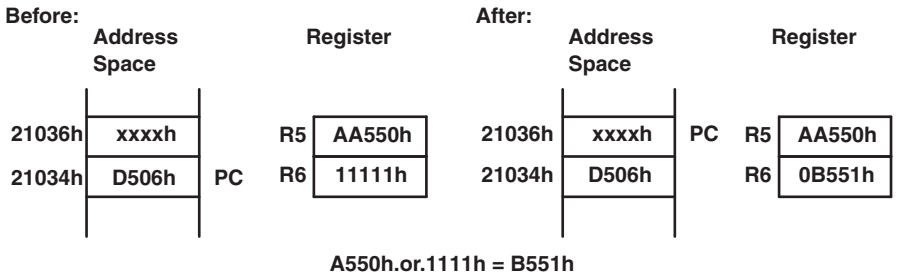
Address-word operation: Address-word operation reads the 20 bits of the source register Rsrc and writes the result to the 20 bits of the destination register Rdst. The register Rsrc is not modified

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.



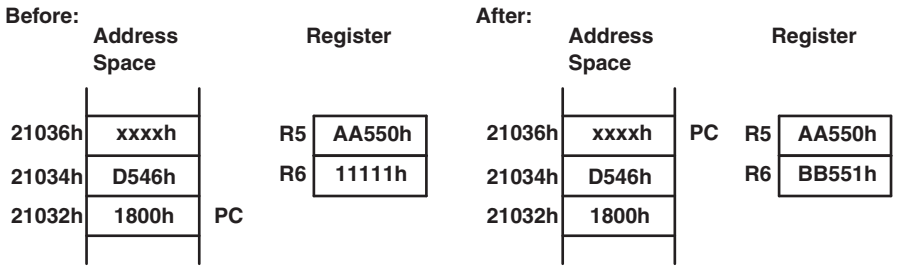
A550h.or.1111h = B551h

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.

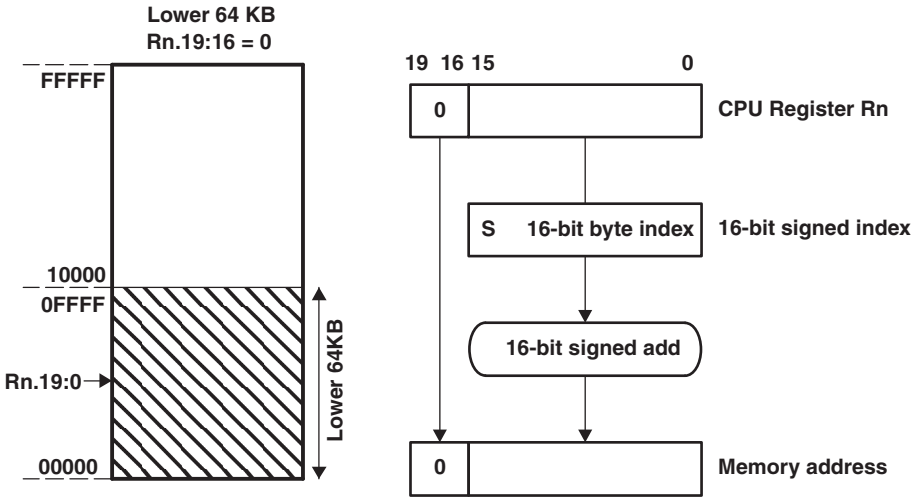
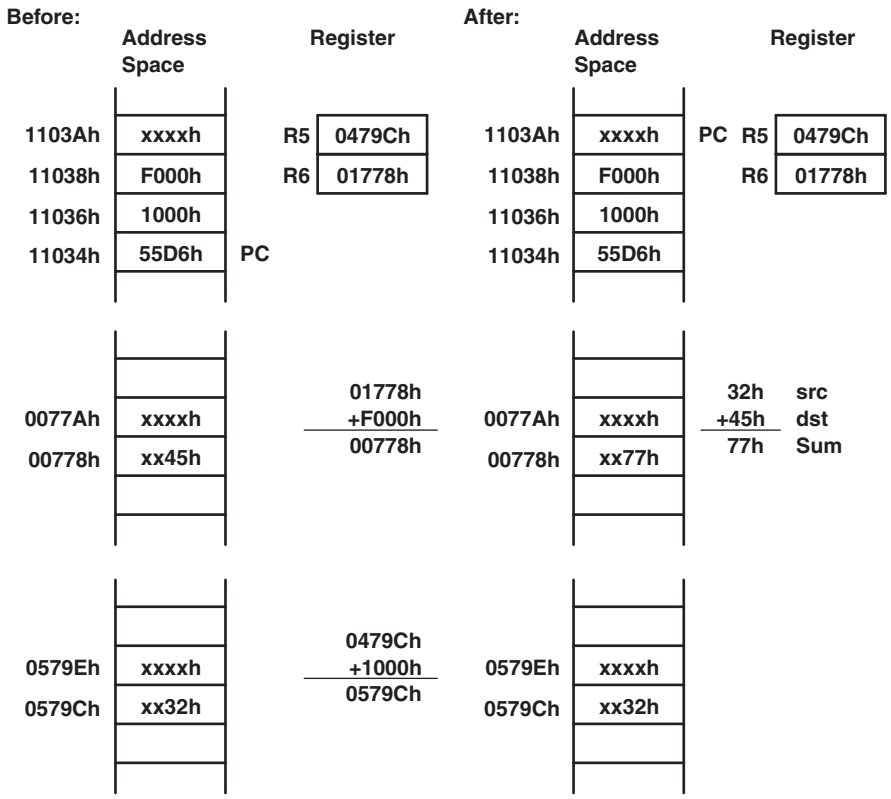


Figure 4-15. Indexed Mode in Lower 64 KB

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).

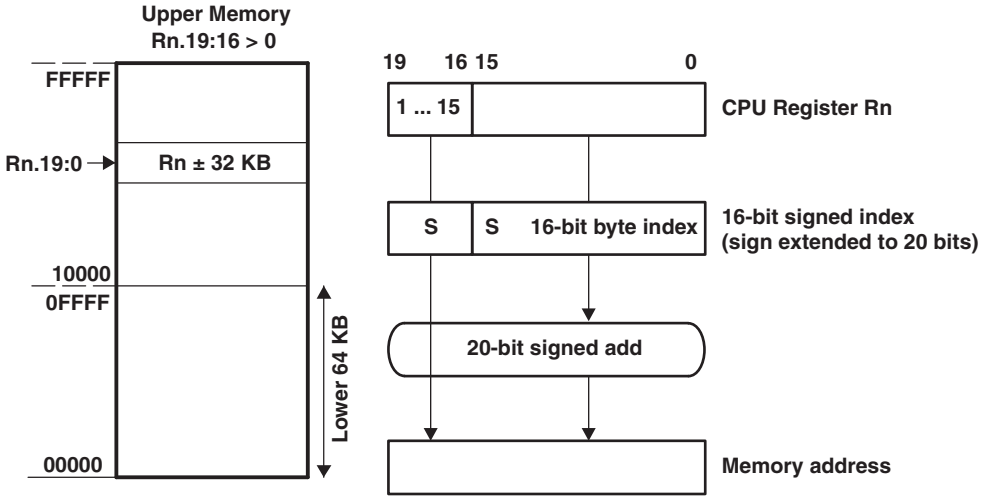


Figure 4-16. Indexed Mode in Upper Memory

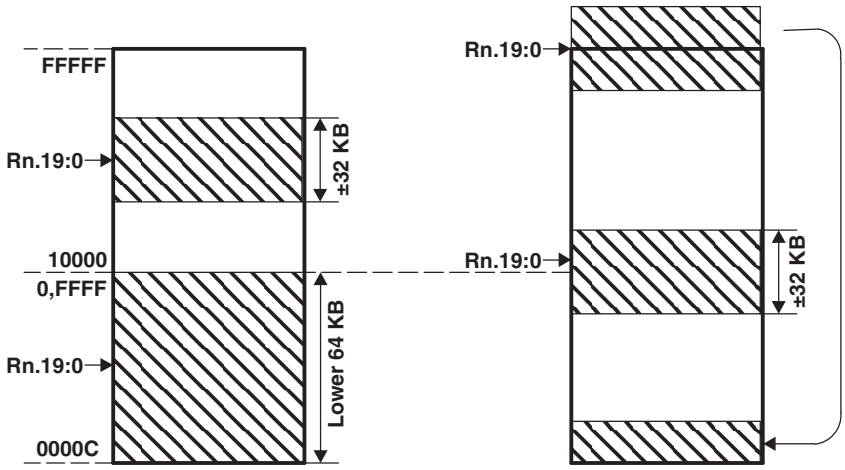


Figure 4-17. Overflow and Underflow for Indexed Mode

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.

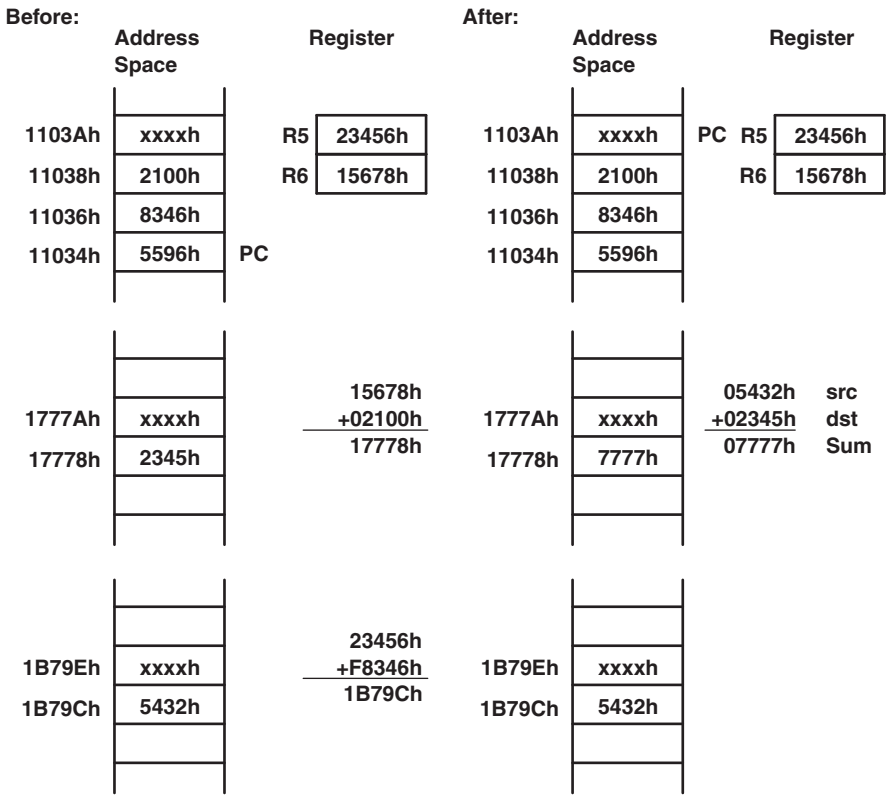


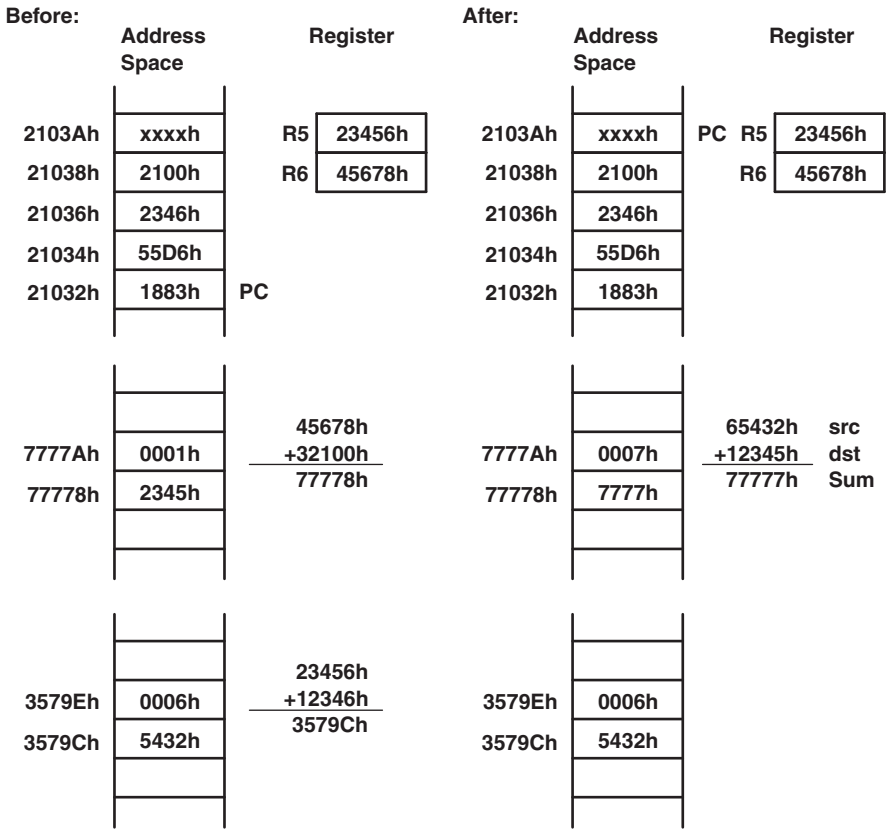
Figure 4-18. Example for Indexed Mode

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.

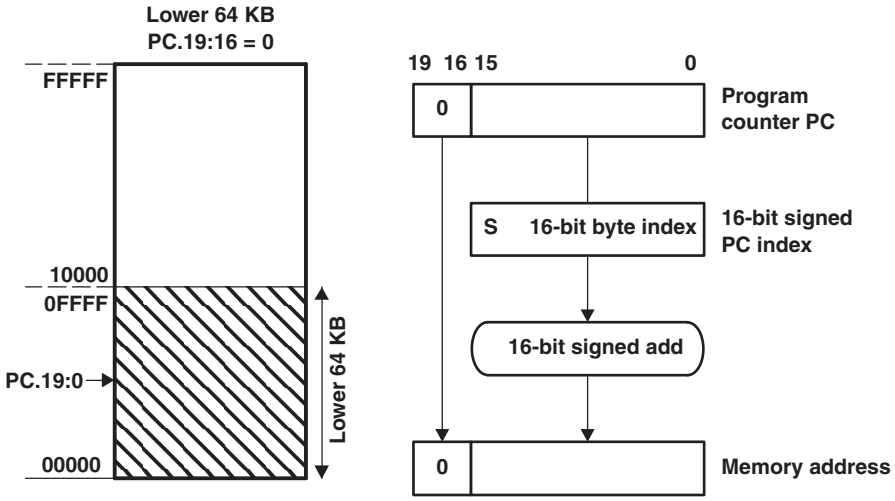


Figure 4-19. Symbolic Mode Running in Lower 64 KB

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.

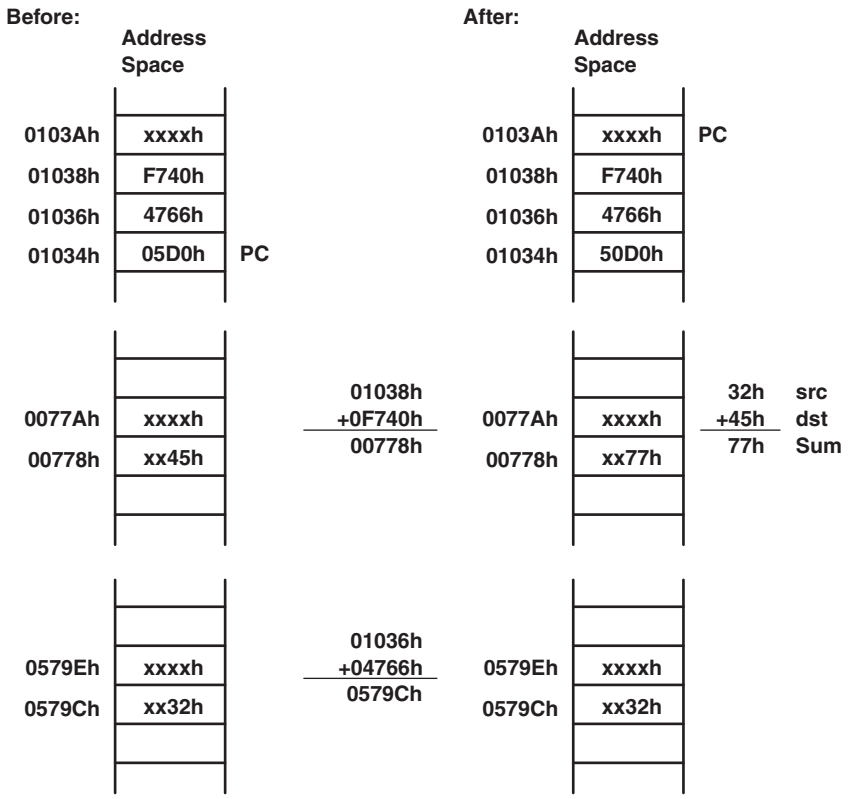
Length: Two or three words

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.

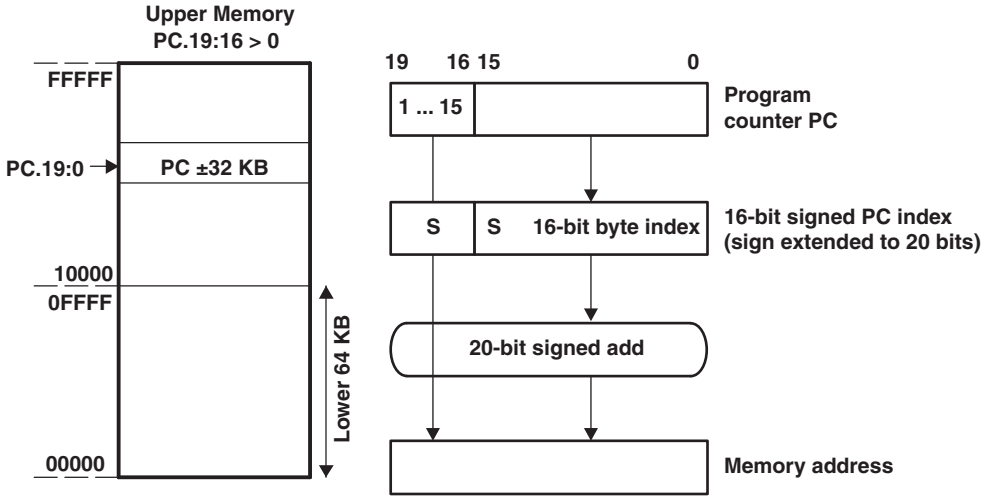


Figure 4-20. Symbolic Mode Running in Upper Memory

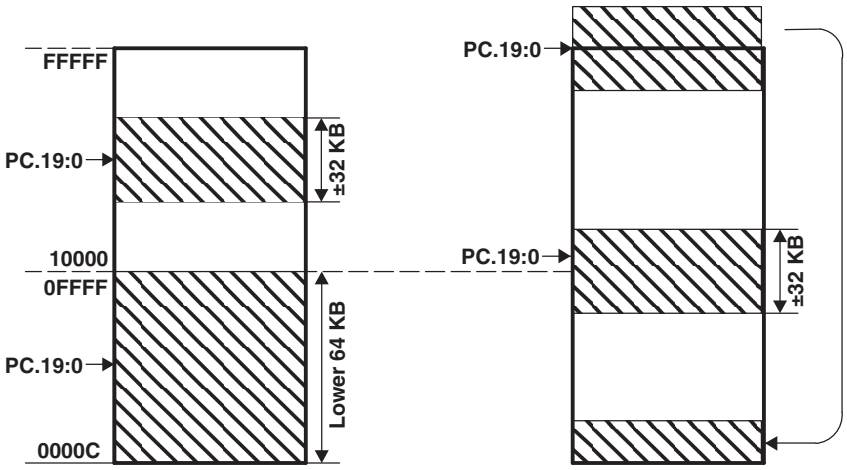
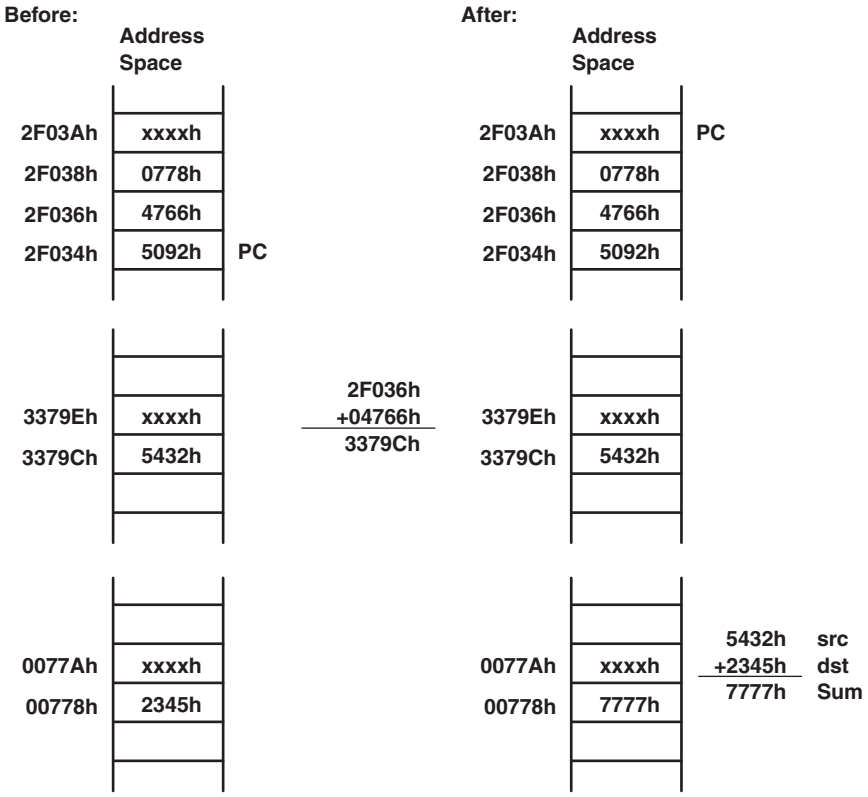


Figure 4-21. Overflow and Underflow for Symbolic Mode

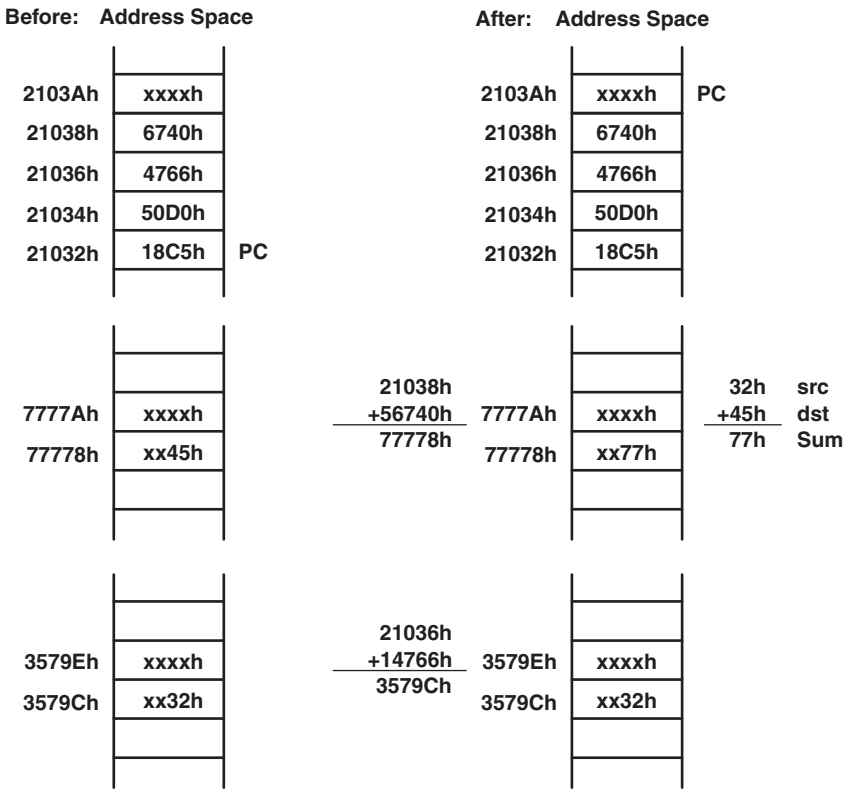
Length:	Two or three words
Operation:	The sign-extended 16-bit index in the next word after the instruction is added to the 20 bits of the PC. This delivers a 20-bit address, which points to an address in the range 0 to FFFFFh. The operand is the content of the addressed memory location.
Comment:	Valid 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

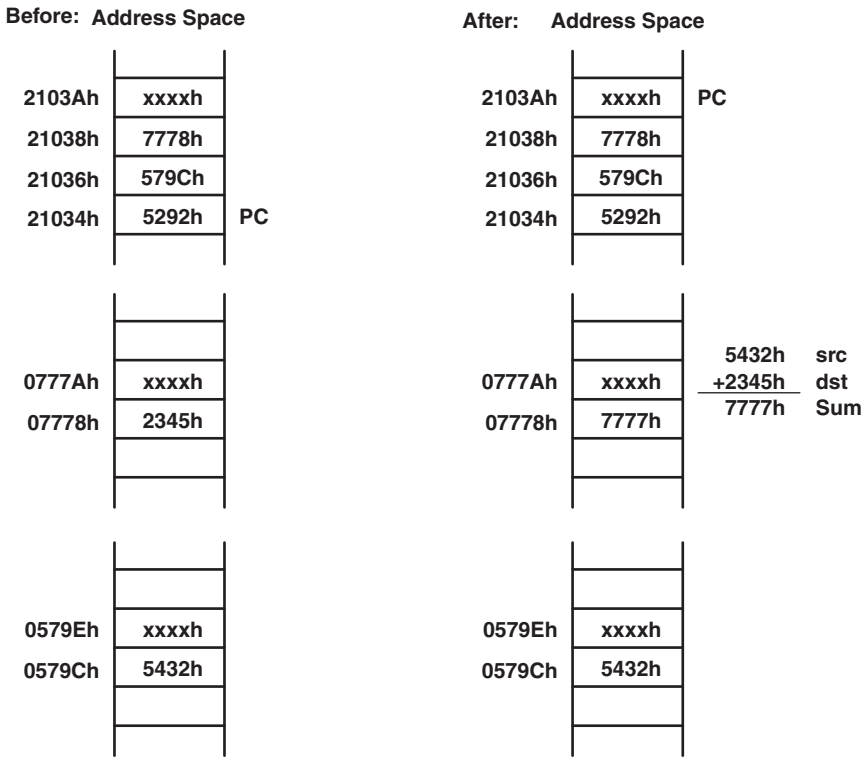
The Absolute mode uses the contents of the word following the instruction as the address of the operand. The Absolute mode has two addressing possibilities:

- 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.

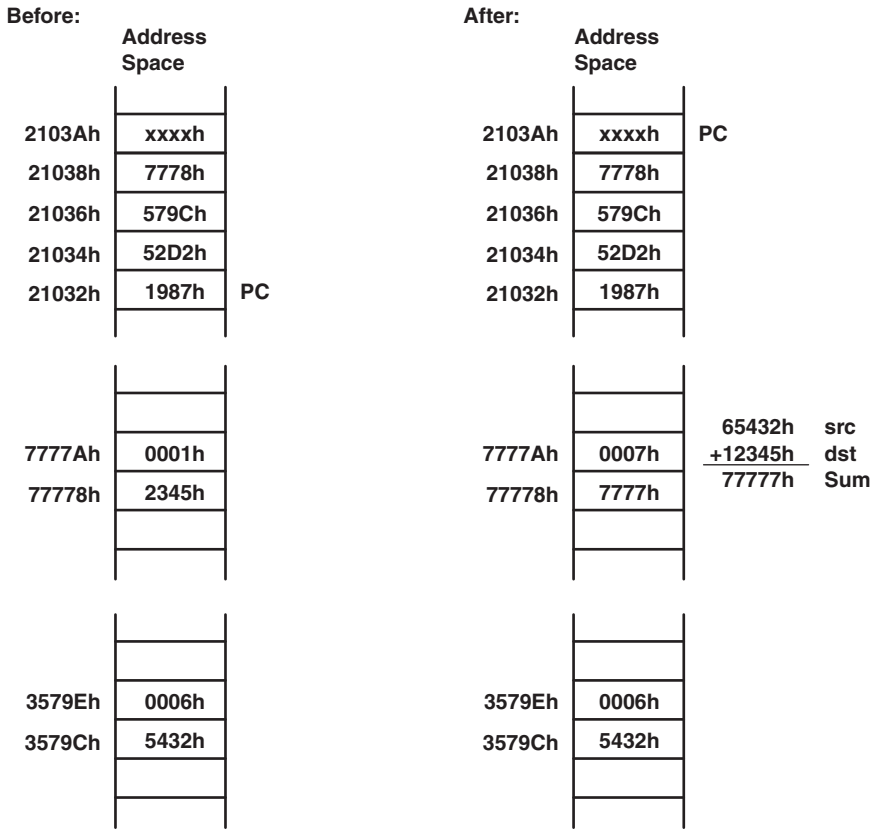
Length:	Two or three words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the index from 0 and inserts it.
Example:	`ADD.W &EDE,&TONI ;` This instruction adds the 16-bit data contained in the absolute source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

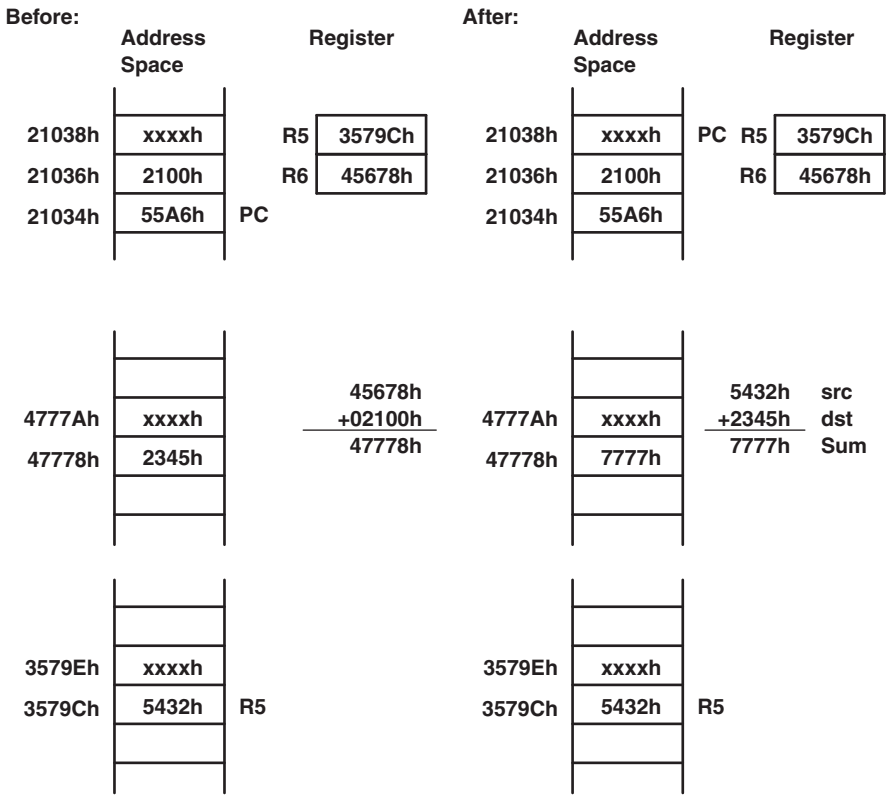
Length:	Three or four words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the index from 0 and inserts it.
Example:	`ADDX.A &EDE,&TONI ;`
	This instruction adds the 20-bit data contained in the absolute source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

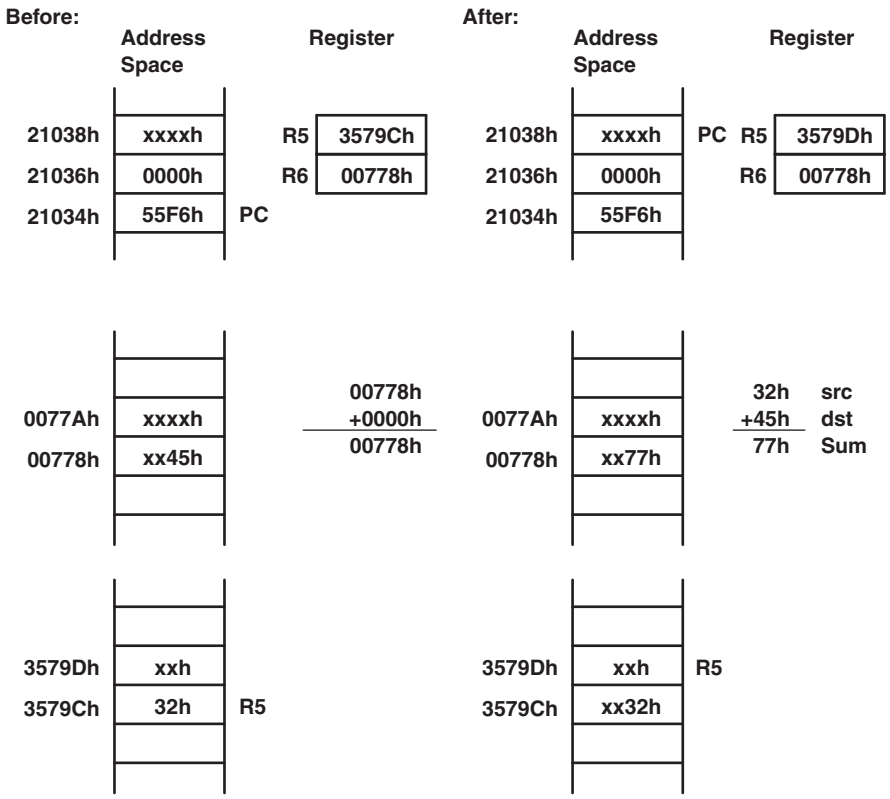
Length:	One, two, or three words
Operation:	The operand is the content the addressed memory location. The source register Rsrc is not modified.
Comment:	Valid only for the source operand. The substitute for the destination operand is 0(Rdst).
Example:	`ADDX.W @R5,2100h(R6)` This instruction adds the 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

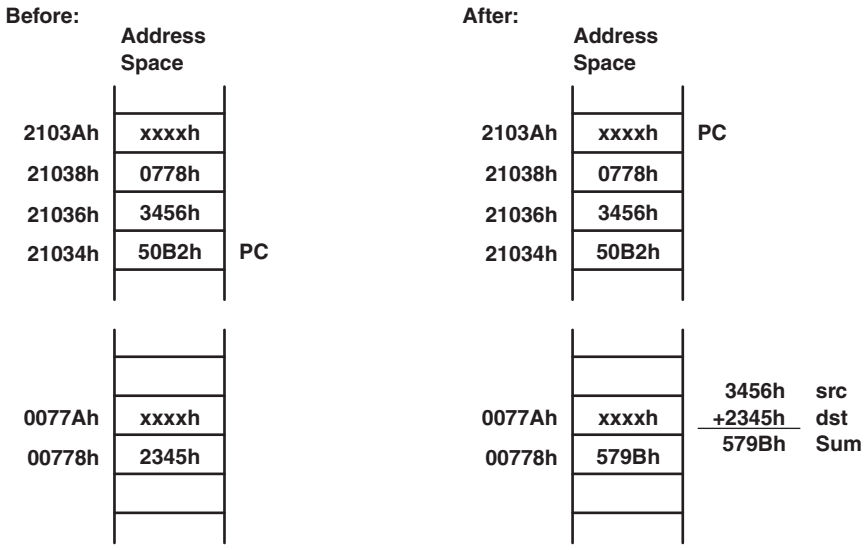
The Immediate mode allows accessing constants as operands by including the constant in the memory location following the instruction. The PC is used with the Indirect Autoincrement mode. The PC points to the immediate value contained in the next word. After the fetching of the immediate operand, the 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.

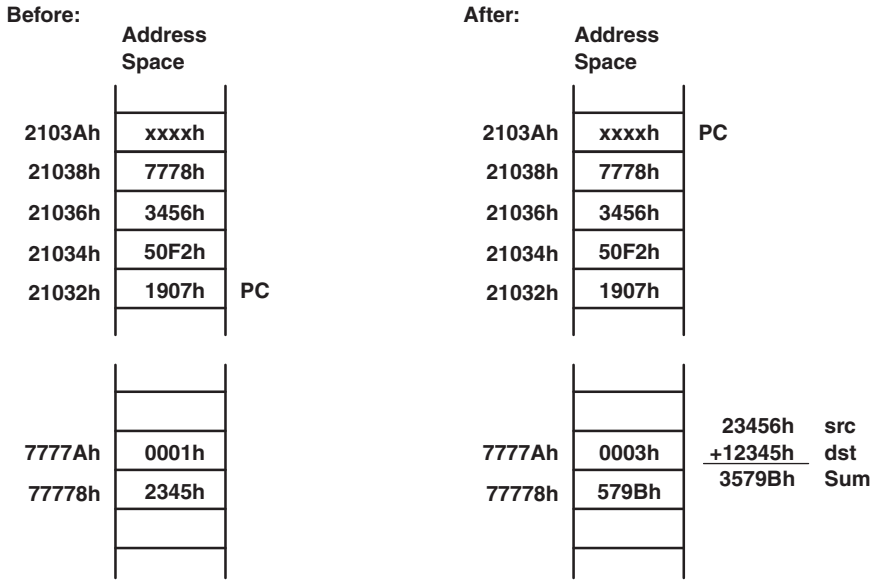
Length:	Two or three words. One word less if a constant of the constant generator can be used for the immediate operand.
Operation:	The 16-bit immediate source operand is used together with the 16-bit destination operand.
Comment:	Valid only for the source operand
Example:	`ADD #3456h,&TONI` This instruction 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 1MB 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:
  - Place 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.
  - Place 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.

Section 4.5.1 lists and describes the MSP430 instructions, and Section 4.5.2 lists and describes the 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.

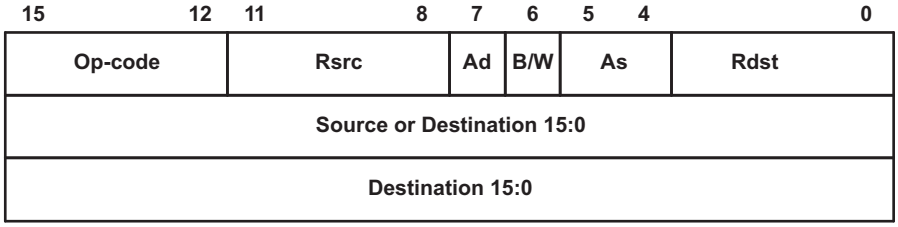


Figure 4-22. MSP430 Double-Operand Instruction Format

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	*	*	$\overline{Z}$
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	*	*	*	$\overline{Z}$
AND(.B)	src,dst	src .and. dst → dst	0	*	*	$\overline{Z}$

(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.

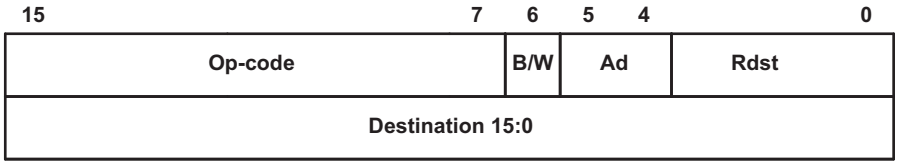


Figure 4-23. MSP430 Single-Operand Instructions

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	0	*	*	*
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	*	*	$\overline{Z}$

(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.

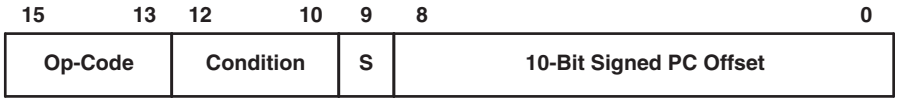


Figure 4-24. Format of Conditional Jump Instructions

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.

4.5.1.5.1 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 ($\overline{\text{RST}}$/NMI)	4	–

4.5.1.5.2 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			Length of Instruction	Example
	RRA, RRC SWPB, SXT	PUSH	CALL		
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

4.5.1.5.3 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.

4.5.1.5.4 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 or 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.

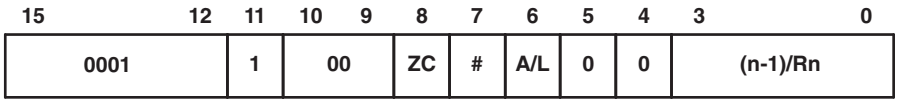


Figure 4-25. Extension Word for Register Modes

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 instruction 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.

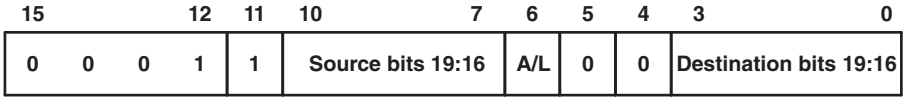


Figure 4-26. Extension Word for Non-Register Modes

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

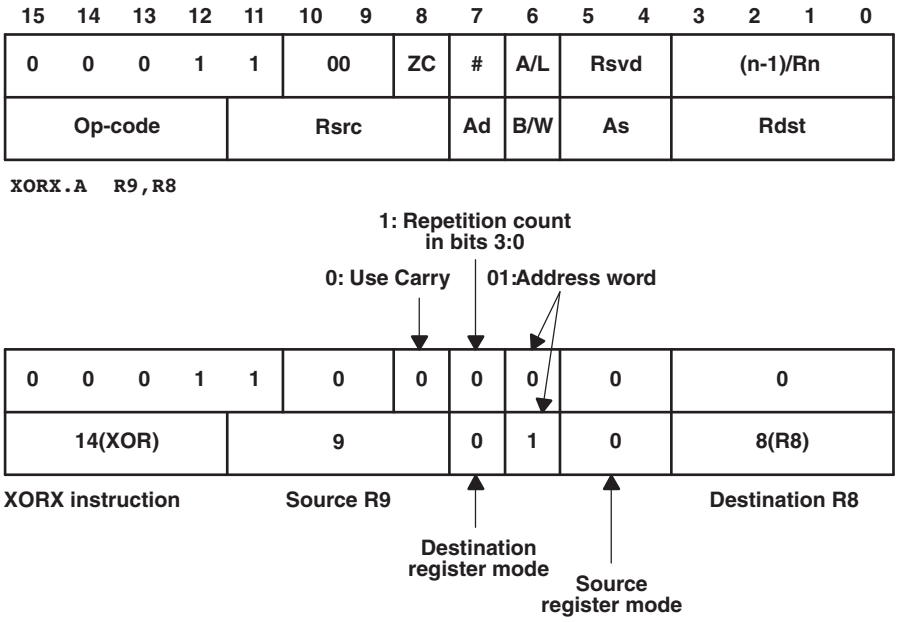


Figure 4-27. Example for Extended Register or Register Instruction

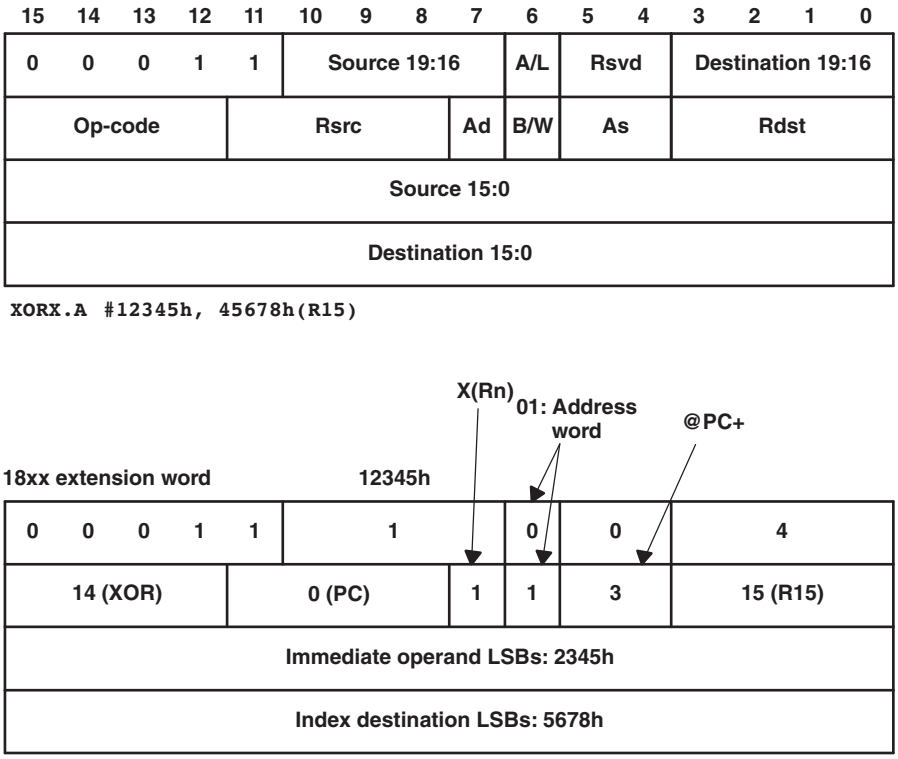


Figure 4-28. Example for Extended Immediate or Indexed Instruction

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	*	*	$\overline{Z}$
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	*	*	*	$\overline{Z}$
ANDX(.B,.A)	src,dst	src .and. dst → dst	0	*	*	$\overline{Z}$

(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.

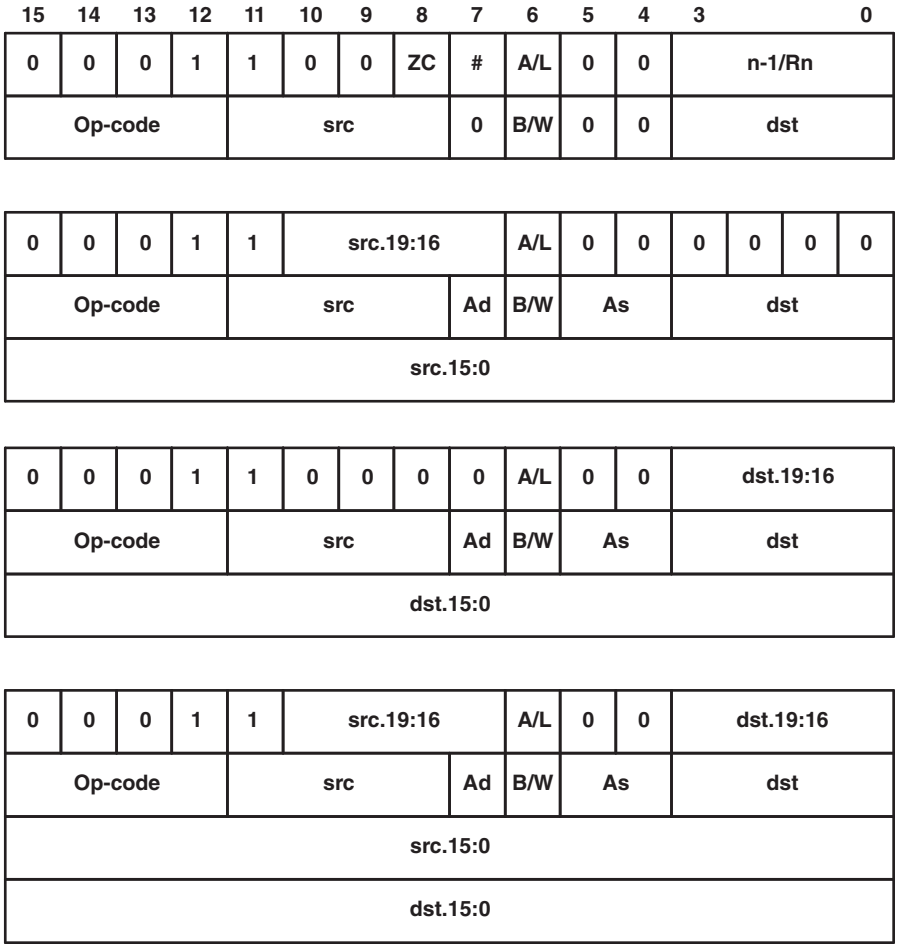


Figure 4-29. Extended Format I Instruction Formats

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.

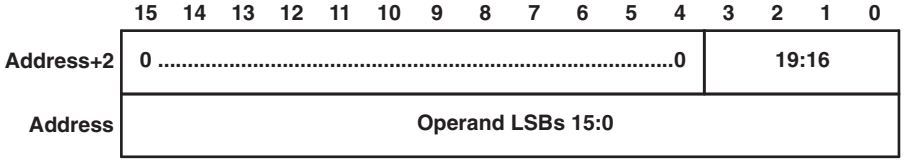


Figure 4-30. 20-Bit Addresses in Memory

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-, or 20-bit source to stack		–	–	–	–
RRCM(.A)	#n,Rdst	Rotate right Rdst n bits through carry (16-, 20-bit register)	1 to 4	0	*	*	*
RRUM(.A)	#n,Rdst	Rotate right Rdst n bits unsigned (16-, 20-bit register)	1 to 4	0	*	*	*
RRAM(.A)	#n,Rdst	Rotate right Rdst n bits arithmetically (16-, 20-bit register)	1 to 4	0	*	*	*
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	0	*	*	*
RRUX(.B,.A)	Rdst	Rotate right dst unsigned (8-, 16-, 20-bit)	1	0	*	*	*
RRAX(.B,.A)	dst	Rotate right dst arithmetically	1	0	*	*	*
SWPBX(.A)	dst	Exchange low byte with high byte	1	–	–	–	–
SXTX(.A)	Rdst	Bit7 → bit8 ... bit19	1	0	*	*	$\overline{Z}$
SXTX(.A)	dst	Bit7 → bit8 ... MSB	1	0	*	*	$\overline{Z}$

(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.

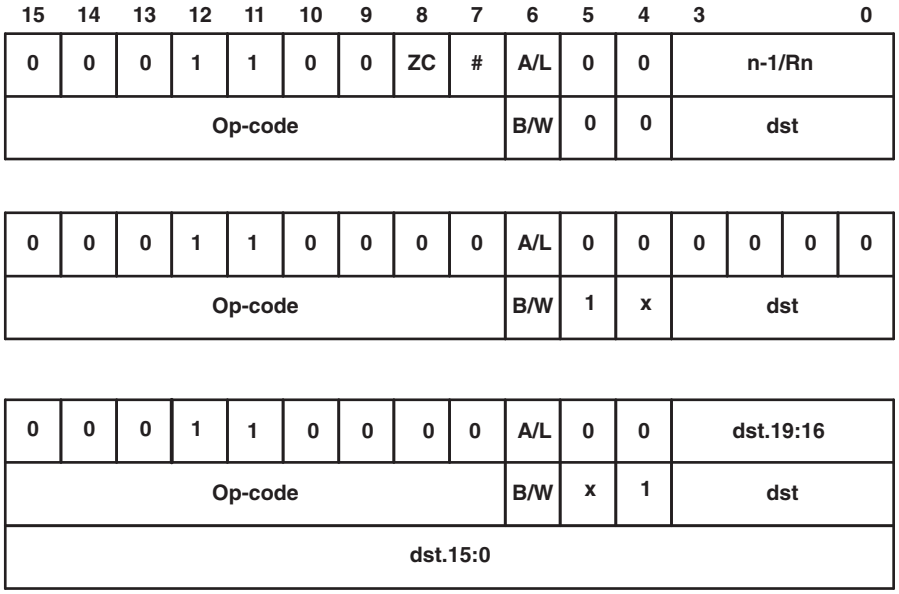


Figure 4-31. Extended Format II Instruction Format

4.5.2.4.1 Extended Format II Instruction Format Exceptions

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

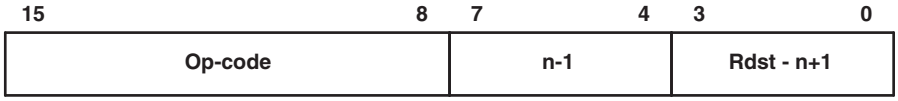


Figure 4-32. PUSHM and POPM Instruction Format

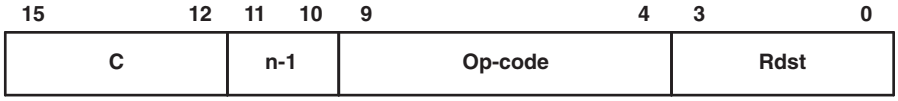


Figure 4-33. RRCM, RRAM, RRUM, and RLAM Instruction Format

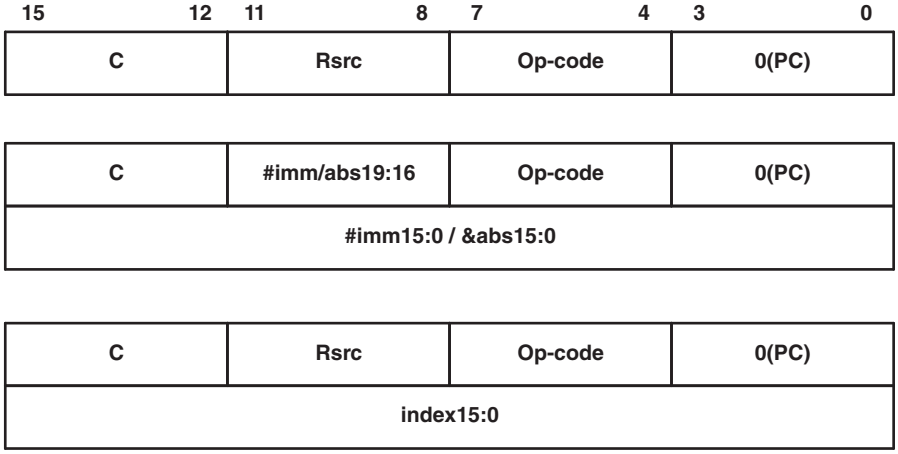


Figure 4-34. BRA Instruction Format

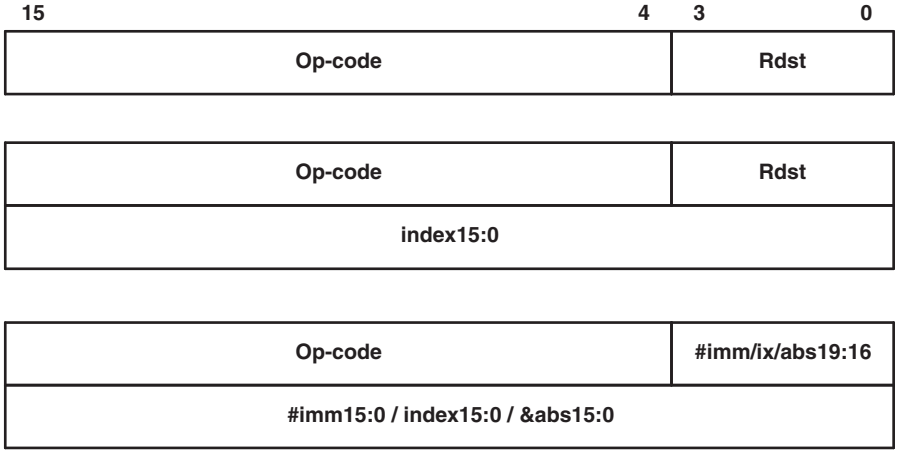


Figure 4-35. CALLA Instruction Format

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.

4.5.2.7.1 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

4.5.2.7.2 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.

4.5.2.7.3 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.

4.6.2.1 ADC

*** 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
```

4.6.2.2 ADD

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
```

4.6.2.3 ADDC

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
```

4.6.2.4 AND

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
```

4.6.2.5 BIC

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
```

4.6.2.6 BIS

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
```

4.6.2.7 BIT

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
```

4.6.2.8 BR, BRANCH

*** 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 (for example #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 (for example table with address
                  ; starting at X). X can be an address or a label
                  ; Core instruction MOV X(R5),PC
                  ; Indirect, indirect R5 + X
```

4.6.2.9 CALL

CALL Call a subroutine in lower 64 K

Syntax `CALL dst`

Operation
dst → tmp 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
```

4.6.2.10 CLR

*** 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
```

4.6.2.11 CLRC

*** CLRC**	Clear carry bit
Syntax	`CLRC`
Operation	0 → 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
```

4.6.2.12 CLRN

*** CLRN** Clear negative bit

Syntax `CLRN`

Operation 0 → 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
```

4.6.2.13 CLRZ

*** CLRZ**	Clear zero bit
Syntax	`CLRZ`
Operation	0 → 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); for example, 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)
```

4.6.2.14 CMP

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
```

4.6.2.15 DADC

*** 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
```

4.6.2.16 DADD

*** 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 = 0

```
CLRC                   ; Clear carry
DADD.B   &BCD,R4       ; Add BCD to R4 decimally.
                         R4: 0,00ddh
```

4.6.2.17 DEC

*** 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     #255,R10
L$1     MOV.B   @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.

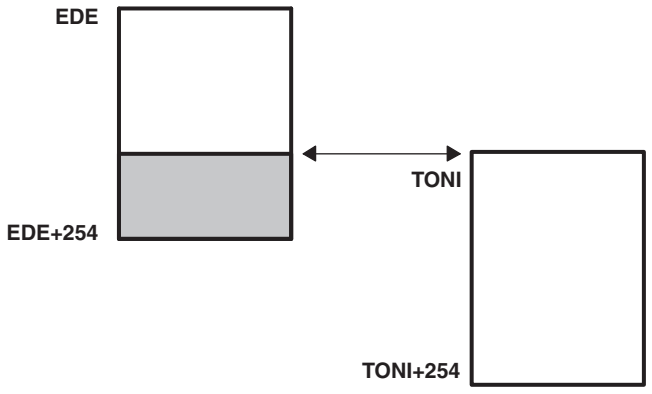


Figure 4-36. Decrement Overlap

4.6.2.18 DECD

*** 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
```

4.6.2.19 DINT

*** 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.

NOTE: Enable and Disable Interrupt

Due to the pipelined CPU architecture, the instruction following the enable interrupt instruction (EINT) is always executed, even if an interrupt service request is pending when the interrupts are enabled.

If the enable interrupt instruction (EINT) is immediately followed by a disable interrupt instruction (DINT), a pending interrupt might not be serviced. Further instructions after DINT might execute incorrectly and result in unexpected CPU execution. It is recommended to always insert at least one instruction between EINT and DINT. Note that any alternative instruction use that sets and immediately clears the CPU status register GIE bit must be considered in the same fashion.

4.6.2.20 EINT

*** 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 and Disable Interrupt

Due to the pipelined CPU architecture, the instruction following the enable interrupt instruction (EINT) is always executed, even if an interrupt service request is pending when the interrupts are enabled.

If the enable interrupt instruction (EINT) is immediately followed by a disable interrupt instruction (DINT), a pending interrupt might not be serviced. Further instructions after DINT might execute incorrectly and result in unexpected CPU execution. It is recommended to always insert at least one instruction between EINT and DINT. Note that any alternative instruction use that sets and immediately clears the CPU status register GIE bit must be considered in the same fashion.

4.6.2.21 INC

*** 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
```

4.6.2.22 INCD

*** 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
```

4.6.2.23 INV

*** 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
```

4.6.2.24 JC, JHS

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
```

4.6.2.25 JEQ, JZ

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.
```

4.6.2.26 JGE

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
```

4.6.2.27 JL

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
```

4.6.2.28 JMP

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
```

4.6.2.29 JN

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.
```

4.6.2.30 JNC, JLO

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
```

4.6.2.31 JNZ, JNE

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.
```

4.6.2.32 MOV

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
```

4.6.2.33 NOP

*** 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.

4.6.2.34 POP

*** 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.

4.6.2.35 PUSH

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
```

4.6.2.36 RET

*** 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
```

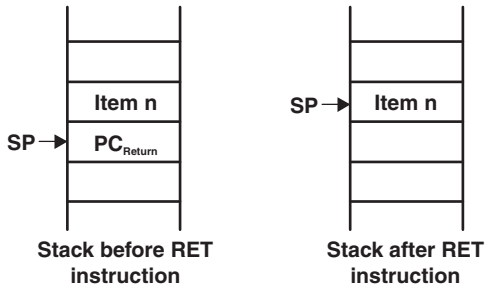


Figure 4-37. Stack After a RET Instruction

4.6.2.37 RETI

RETI Return from interrupt

Syntax `RETI`

Operation

@SP → SR.15:0 Restore saved SR with PC.19:16
SP + 2 → SP

@SP → PC.15:0 Restore saved PC.15:0
SP + 2 → SP 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
```

4.6.2.38 RLA

*** 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.

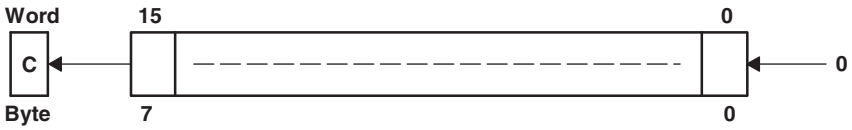


Figure 4-38. Destination Operand—Arithmetic Shift Left

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
```

4.6.2.39 RLC

*** 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).

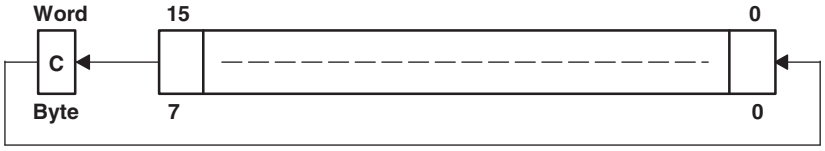


Figure 4-39. Destination Operand—Carry Left Shift

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`

4.6.2.40 RRA

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
```

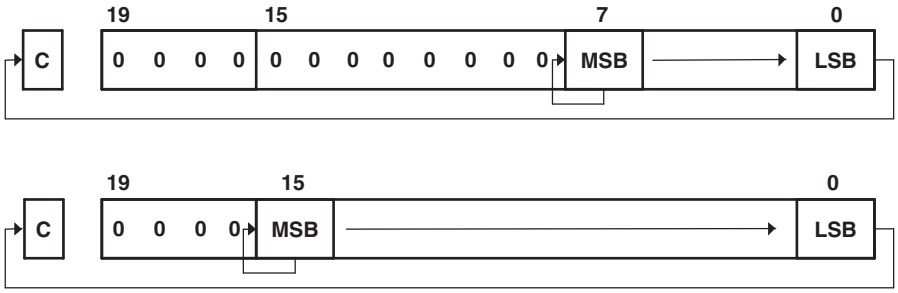


Figure 4-40. Rotate Right Arithmetically RRA.B and RRA.W

4.6.2.41 RRC

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
```

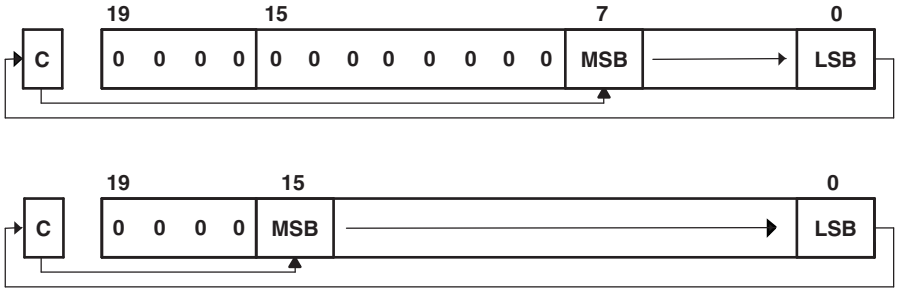


Figure 4-41. Rotate Right Through Carry RRC.B and RRC.W

4.6.2.42 SBC

*** 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

4.6.2.43 SETC

*** 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
```

4.6.2.44 SETN

*** 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.

4.6.2.45 SETZ

*** 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.

4.6.2.46 SUB

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
```

4.6.2.47 SUBC

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
```

4.6.2.48 SWPB

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
```

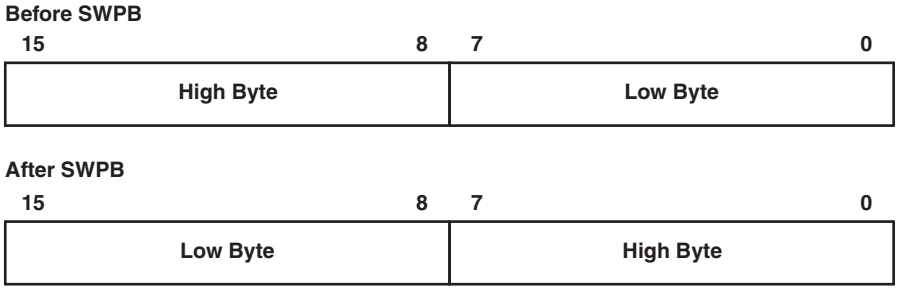


Figure 4-42. Swap Bytes in Memory

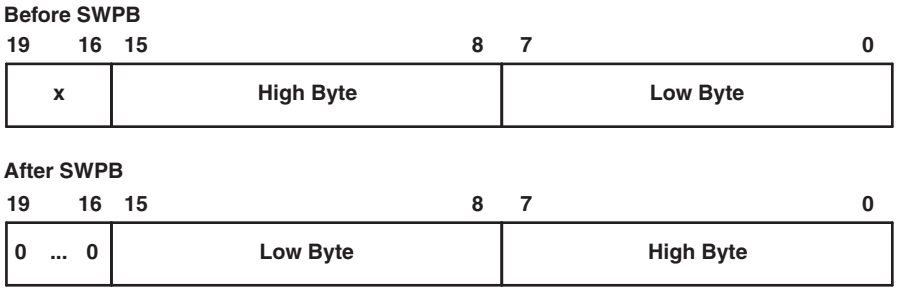


Figure 4-43. Swap Bytes in a Register

4.6.2.49 SXT

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
```

4.6.2.50 TST

*** 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
```

4.6.2.51 XOR

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.

4.6.3.1 ADCX

*** 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
```

4.6.3.2 ADDX

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
```

4.6.3.3 ADDCX

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
```

4.6.3.4 ANDX

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
```

4.6.3.5 BICX

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
```

4.6.3.6 BISX

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
```

4.6.3.7 BITX

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
```

4.6.3.8 CLRX

*** 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
```

4.6.3.9 CMPX

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
```

4.6.3.10 DADCX

*** 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
```

4.6.3.11 DADDX

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
```

4.6.3.12 DECX

*** 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
```

4.6.3.13 DECDX

*** 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
```

4.6.3.14 INCX

*** 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)
```

4.6.3.15 INCDX

*** 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
```

4.6.3.16 INVX

*** 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
```

4.6.3.17 MOVX

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
```

4.6.3.18 POPM

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 ≤ n ≤ 16
	`POPM.W #n,Rdst` or `POPM #n,Rdst`	1 ≤ n ≤ 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
```

4.6.3.19 PUSHM

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 ≤ n ≤ 16

`PUSHM.W #n,Rdst` or `PUSHM #n,Rdst` 1 ≤ n ≤ 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
```

4.6.3.20 POPX

*** 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
```

4.6.3.21 PUSHX

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
```

4.6.3.22 RLAM

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
```

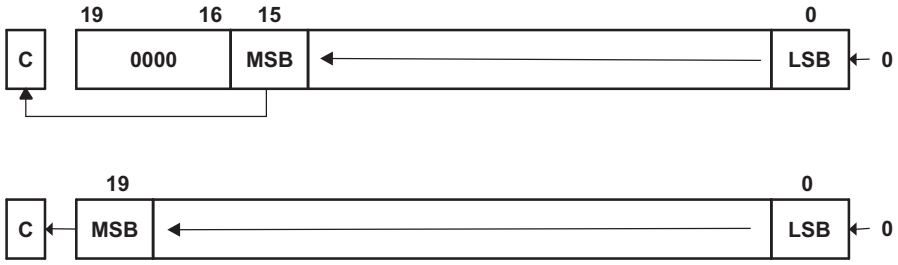


Figure 4-44. Rotate Left Arithmetically—RLAM[.W] and RLAM.A

4.6.3.23 RLAX

*** 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)
```

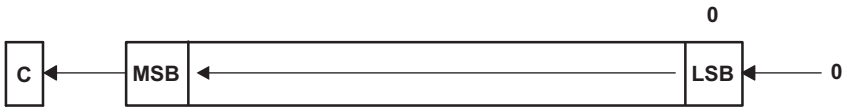


Figure 4-45. Destination Operand-Arithmetic Shift Left

4.6.3.24 RLCX

*** 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)
```

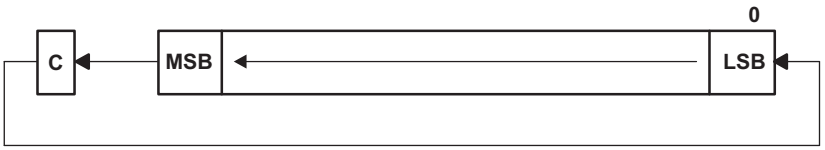


Figure 4-46. Destination Operand-Carry Left Shift

4.6.3.25 RRAM

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 ≤ n ≤ 4

`RRAM.W #n,Rdst` or `RRAM #n,Rdst` 1 ≤ n ≤ 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, or 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
```

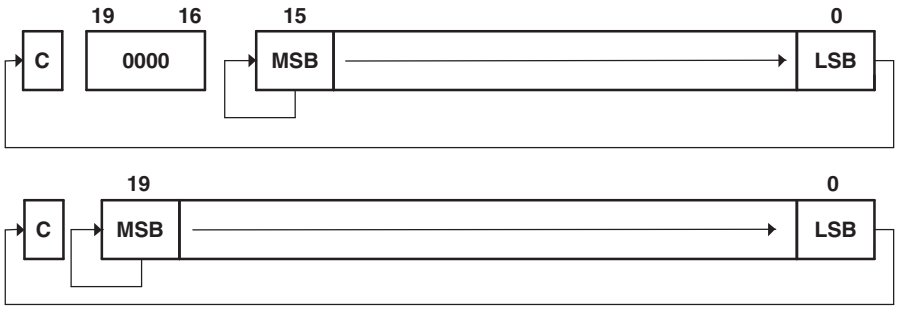


Figure 4-47. Rotate Right Arithmetically RRAM[.W] and RRAM.A

4.6.3.26 RRAX

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
```

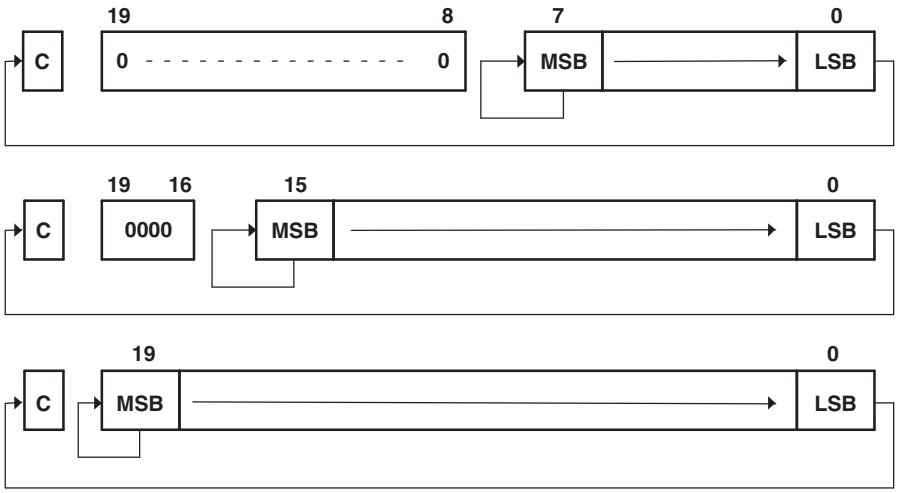


Figure 4-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode

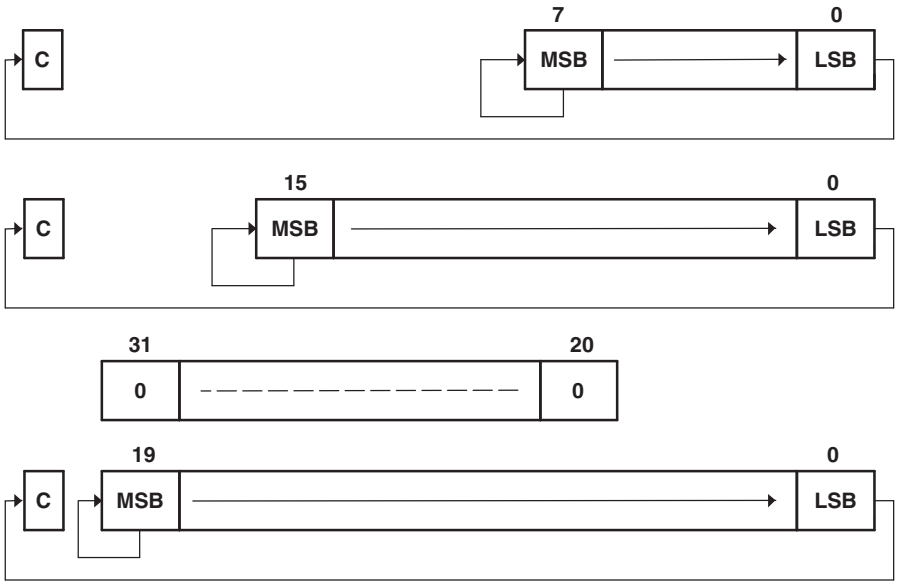


Figure 4-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode

4.6.3.27 RRCM

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 ≤ n ≤ 4
	`RRCM.W #n,Rdst` or `RRCM #n,Rdst`	1 ≤ n ≤ 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
```

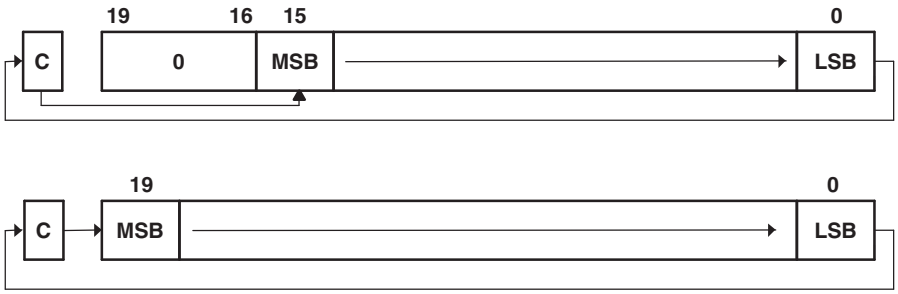


Figure 4-50. Rotate Right Through Carry RRCM[.W] and RRCM.A

4.6.3.28 RRCX

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
```

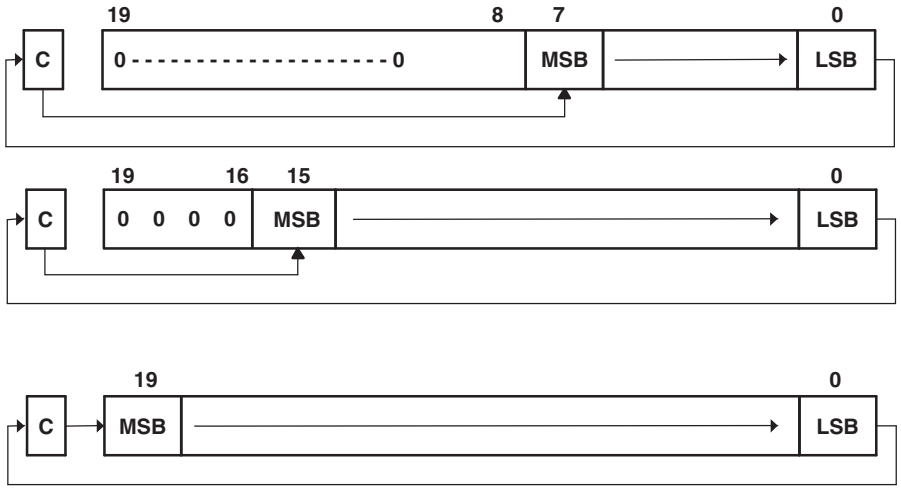


Figure 4-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode

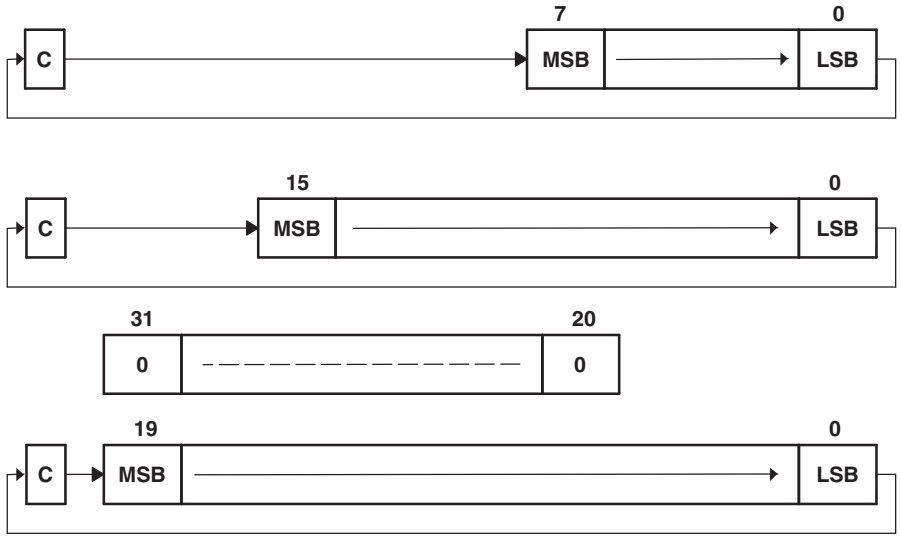


Figure 4-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode

4.6.3.29 RRUM

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 ≤ n ≤ 4

`RRUM.W #n,Rdst` or `RRUM #n,Rdst` 1 ≤ n ≤ 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
```

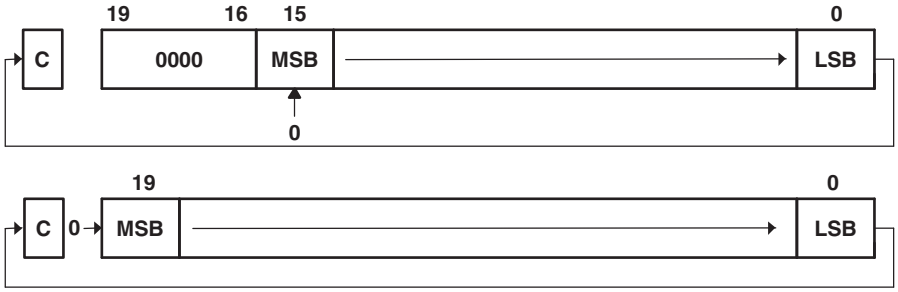


Figure 4-53. Rotate Right Unsigned RRUM[.W] and RRUM.A

4.6.3.30 RRUX

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
```

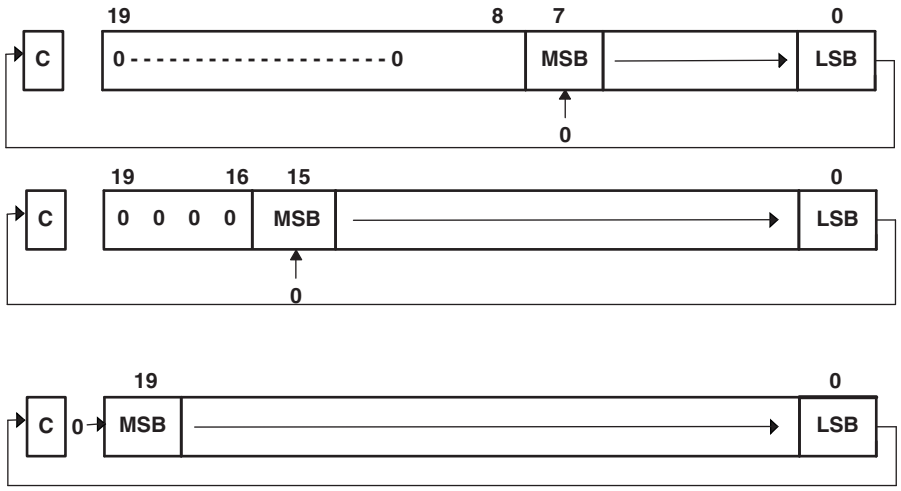


Figure 4-54. Rotate Right Unsigned RRUX(.B,.A) – Register Mode

4.6.3.31 SBCX

*** 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

4.6.3.32 SUBX

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
```

4.6.3.33 SUBCX

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
```

4.6.3.34 SWPBX

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
```

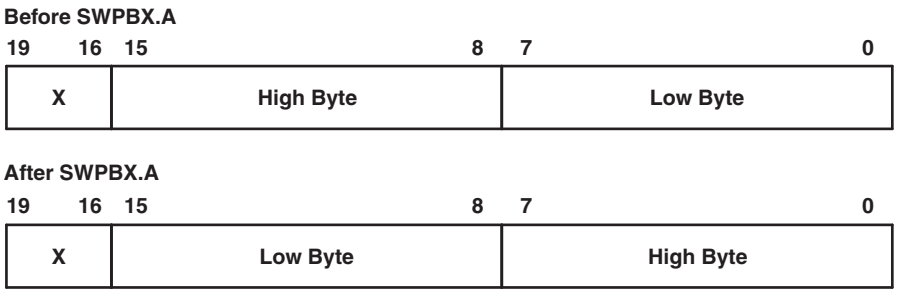


Figure 4-55. Swap Bytes SWPBX.A Register Mode

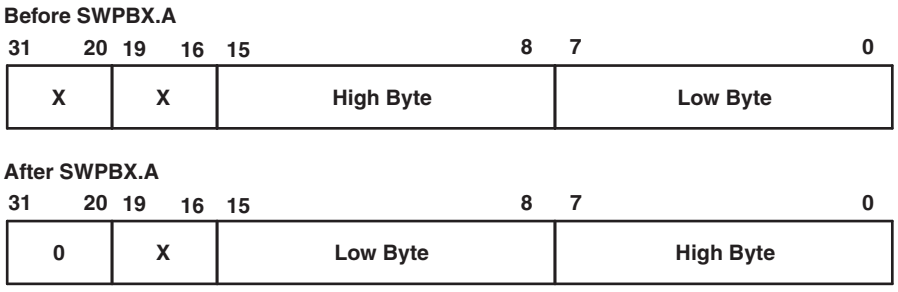


Figure 4-56. Swap Bytes SWPBX.A In Memory

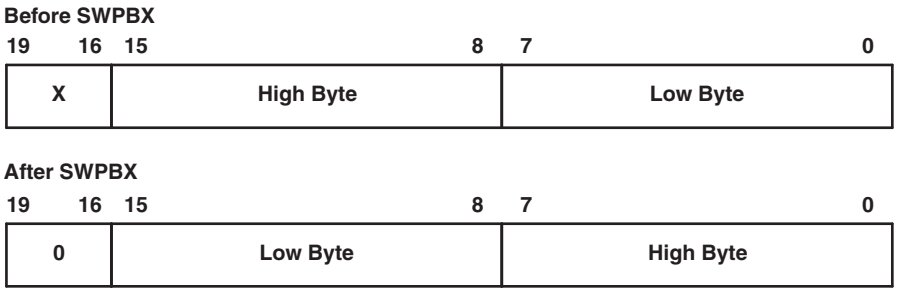


Figure 4-57. Swap Bytes SWPBX[.W] Register Mode

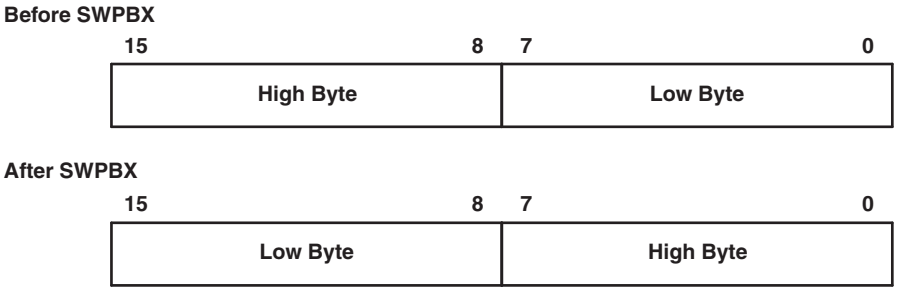


Figure 4-58. Swap Bytes SWPBX[.W] In Memory

4.6.3.35 SXTX

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
```

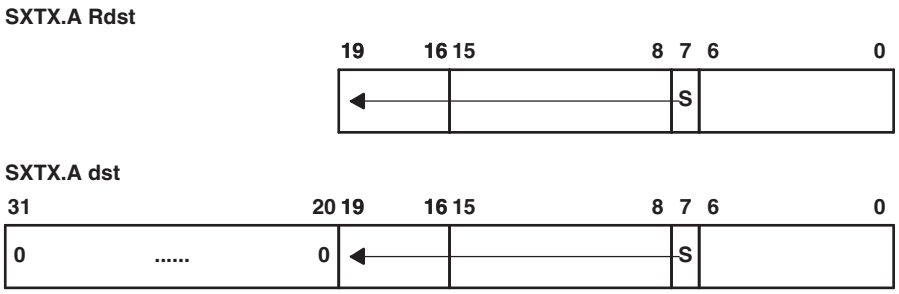


Figure 4-59. Sign Extend SXTX.A

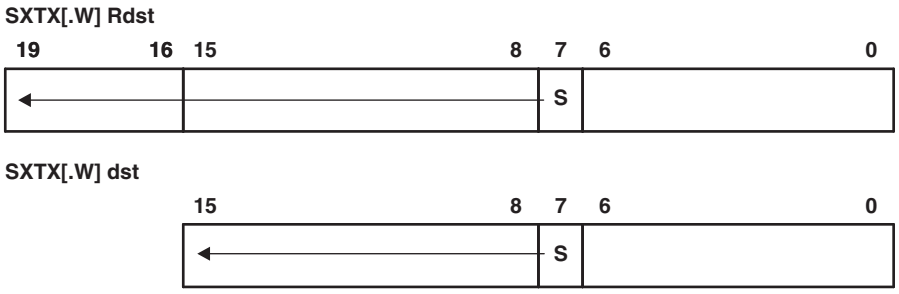


Figure 4-60. Sign Extend SXTX[.W]

4.6.3.36 TSTX

*** 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
```

4.6.3.37 XORX

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.

4.6.4.1 ADDA

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
```

4.6.4.2 BRA

*** 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 software 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) (for example, 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
```

4.6.4.3 CALLA

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 software 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); for example, 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)
```

4.6.4.4 CLRA

*** 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
```

4.6.4.5 CMPA

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           ; R6 = 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
```

4.6.4.6 DECDA

*** 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
```

4.6.4.7 INCDA

*** 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
```

4.6.4.8 MOVA

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
```

4.6.4.9 RETA

*** 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)
```

4.6.4.10 SUBA

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
```

4.6.4.11 TSTA

*** 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
```

FRAM Controller (FRCTL)

This chapter describes the operation of the FRAM memory controller.

Topic **Page**

5.1	FRAM Introduction	251
5.2	FRAM Organization	251
5.3	FRCTL Module Operation	251
5.4	Programming FRAM Memory Devices	252
5.5	Wait State Control	252
5.6	FRAM ECC	253
5.7	FRAM Write Back	253
5.8	FRAM Power Control	253
5.9	FRAM Cache	254
5.10	FRCTL Registers	255

5.1 FRAM Introduction

FRAM memory is a nonvolatile memory that reads and writes like standard SRAM. The MSP430 FRAM memory features include:

- Byte or word write access
- Automatic and programmable wait state control with independent wait state settings for access and cycle times
- Error Correction Code with bit error correction, extended bit error detection and flag indicators
- Cache for fast read
- Power control for disabling FRAM if it is not used

Figure Figure 5-1 shows the block diagram of the FRAM Controller.

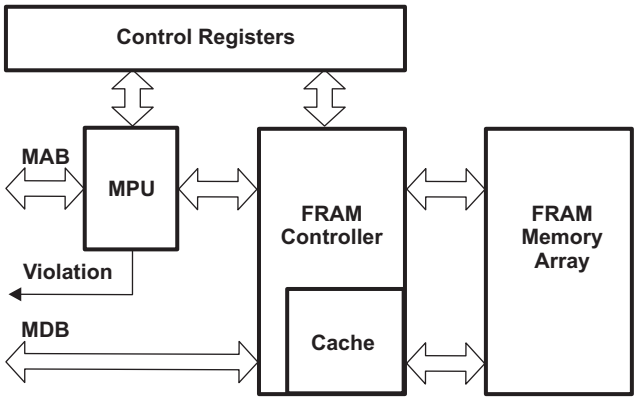


Figure 5-1. FRAM Controller Block Diagram

5.2 FRAM Organization

The FRAM memory can be arranged into segments by the Memory Protection Unit (MPU). See the *Memory Protection Unit* chapter for details. The address space is linear with the exception of the User Information Memory and the Device Descriptor Information (TLV).

5.3 FRCTL Module Operation

The FRAM module can be read in a similar fashion to SRAM and needs no special requirements. Similarly, any writes to unprotected segments can be written in the same fashion as SRAM. All writes to user protected segments are handled as described in the *Memory Protection Unit* chapter.

A FRAM read always requires a write back to the same memory location with the same information read. This write back is part of the FRAM module itself and requires no user interaction. These write backs are different from the normal write access from application code.

The FRAM module has built-in Error Correction Code logic (ECC) that is capable of correcting bit errors and detecting multiple bit errors. Two flags are available that indicate the presence of an error. The CBDIFG is set when a correctable bit error has been detected. If CBDIE is also set, a System NMI event (SYSNMI) occurs. The UBDIFG is set when a multiple bit error which is not correctable has been detected. If UBDIE is also set, a System NMI event (SYSNMI) occurs. Upon correctable or uncorrectable bit errors, the program vectors to the SYSSNIV if the NMI is enabled. If desired, a System Reset event (SYSRST) can be generated by setting the UBDRSTEN bit. If an uncorrectable error is detected, a PUC is initiated and the program vectors to the SYSRSTIV.

5.4 Programming FRAM Memory Devices

There are three options for programming an MSP430 FRAM device. All options support in-system programming.

- Program via JTAG or the Spy-Bi-Wire interface
- Program via the BSL
- Program via a custom solution

5.4.1 Programming FRAM Memory Via JTAG or Spy-Bi-Wire

Devices can be programmed via the JTAG port or the Spy-Bi-Wire port. The JTAG interface requires access to TDI, TDO, TMS, TCK, TEST, ground, and optionally VCC and $\overline{\text{RST}}$/NMI. Spy-Bi-Wire interface requires access to TEST, $\overline{\text{RST}}$/NMI, ground and optionally VCC. For more details, see the *MSP430 Programming Via the JTAG Interface User's Guide* (SLAU320).

5.4.2 Programming FRAM Memory Via Bootstrap Loader (BSL)

Every device contains a BSL stored in ROM. The BSL enables users to read or program the FRAM memory or RAM using a UART serial interface. Access to the FRAM memory via the BSL is protected by a 256-bit user-defined password. For more details, see the *MSP430 Programming Via the Bootstrap Loader User's Guide* (SLAU319).

5.4.3 Programming FRAM Memory Via Custom Solution

The ability of the CPU to write to its own FRAM memory allows for in-system and external custom programming solutions. The user can choose to provide data to the device through any means available (for example, UART or SPI). User-developed software can receive the data and program the FRAM memory. Because this type of solution is developed by the user, it can be completely customized to fit the application needs for programming or updating the FRAM memory.

5.5 Wait State Control

The system clock for the CPU or DMA may exceed the FRAM access and cycle time requirements. For these scenarios, a wait state generator mechanism is implemented. The "Recommended Operating Conditions" of the device-specific data sheet lists the frequency ranges with the required wait state settings. The number of wait states is controlled by the NWAITS[2:0] bits in the FRCTL0 register.

To increase the system clock frequency beyond the maximum frequency allowed by the current wait state setting, the following steps are required:

1. Increase the number of wait states by configuring NWAITS[2:0] according to the target frequency.
2. Increase the frequency to the new target.

To decrease the system clock frequency to a range that supports fewer wait states, the following steps are required:

1. Decrease frequency to the new target.
2. Decrease number of wait states by configuring NWAITS[2:0] according to the new frequency setting.

To ensure memory integrity, a mechanism is implemented that resets the device with a PUC if the system clock frequency and the wait state settings violate the FRAM memory access timing.

NOTE: Wait State Settings

- The device starts with zero wait states.
- Correct wait state settings must be ensured, otherwise a PUC might be generated to avoid erratic FRAM accesses.

5.5.1 Wait State and Cache Hit

The FRAM controller contains a cache with two cache sets. Each of these cache sets contains two lines that are preloaded with four words (64 bits) during one access cycle. An intelligent logic selects one of the cache lines to preload FRAM data and preserves recently accessed data in the other cache. If one of the four words stored in one of the cache lines is requested (a cache hit), no FRAM access occurs; instead, a cache request occurs. No wait state is needed for a cache request, and the data is accessed with full system speed. However, if none of the words that are available in the cache are requested (a cache miss), the wait state controls the CPU to ensure proper FRAM access.

5.6 FRAM ECC

The FRAM supports bit error correction and uncorrectable bit error detection. The UBDIFG FRAM uncorrectable bit error flag is set if an uncorrectable bit error has been detected in the FRAM memory error detection logic. The CBDIFG FRAM correctable bit error flag is set if a correctable bit error has been detected and corrected. UBDRSTEN enable a Power Up Clear (PUC) reset if an uncorrectable bit error is detected, UBDIE enables a NMI event if an uncorrectable bit error is detected. CBDIE enables a NMI event if a marginal correctable bit error is detected and corrected.

5.7 FRAM Write Back

All reads from FRAM requires a write back of the previously read content. This write back is performed under all circumstances without any interaction from a user.

5.8 FRAM Power Control

The FRAM controller can disable the power supply for the FRAM memory array. By setting FRPWR = 0, the FRAM memory array supply is disabled, register accesses in FRAM controller are still possible. Memory accesses pointing into the FRAM address space automatically reset the FRPWR = 1 and re-enable the power supply of the FRAM memory. A second control bit FRLPMPWR is used to delay the power-up of the FRAM memory after LPM exit. With FRLPMPWR = 1, the FRAM is activated directly on exit from LPM. FRLPMPWR = 0 delays the activation of the FRAM to the first access into the FRAM address space. For LPM0, the FRAM power state during LPM0 is determinated and memorized from the previous state in active mode. If a FRAM power is disabled, a memory access automatically inserts wait states to ensure sufficient timing for the FRAM power-up and access. Access to FRAM that can be served from cache do not change the power state of the FRAM power control.

A PUC reset forces the state machine to Active with FRAM enabled.

Figure 5-2 shows the activation flow of the FRAM controller.

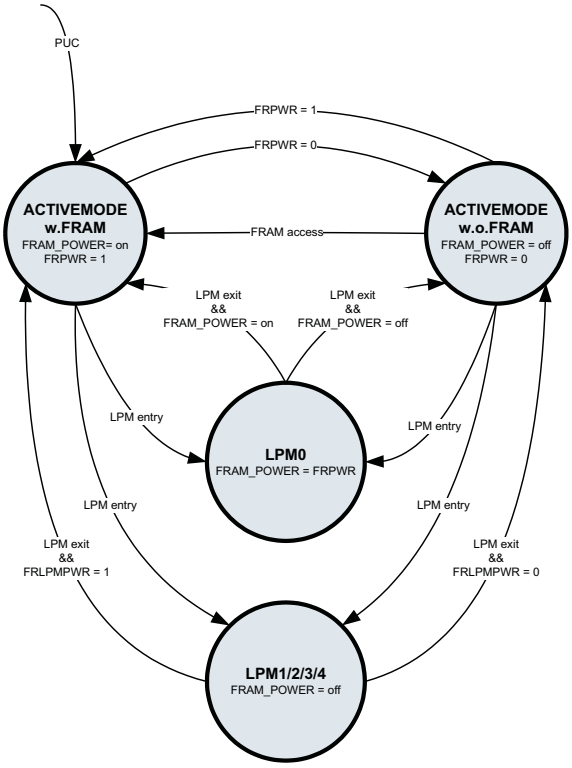


Figure 5-2. FRAM Power Control Diagram

5.9 FRAM Cache

The FRAM controller implements a read cache to provide a speed benefit when running the CPU at higher speeds than the FRAM supports without wait states. The cache implemented is a 2-way associative cache with 4 cache lines of 64 bit size. Memory read accesses on consecutive addresses can be executed without wait states when they are within the same cache line.

5.10 FRCTL Registers

The FRCTL registers and their address offsets are listed in Table 5-1 . The base address of the FRCTL module can be found in the device-specific data sheet.

The password defined in the FRCTL0 register controls access to all FRCTL registers. When the correct password is written, write access to the registers is enabled. The write access is disabled by writing a wrong password in byte mode to the FRCTL upper byte. Word accesses to FRCTL with a wrong password triggers a PUC. A write access to a register other than FRCTL 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 5-1. FRCTL Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	FRCTL0	FRAM Controller Control 0	Read/write	Word	9600h	Section 5.10.1
00h	FRCTL0_L		Read/Write	Byte	00h	
01h	FRCTL0_H		Read/Write	Byte	96h	
04h	GCCTL0	General Control 0	Read/write	Word	0006h	Section 5.10.2
04h	GCCTL0_L		Read/Write	Byte	06h	
05h	GCCTL0_H		Read/Write	Byte	00h	
06h	GCCTL1	General Control 1	Read/write	Word	0000h	Section 5.10.3
06h	GCCTL1_L		Read/Write	Byte	00h	
07h	GCCTL1_H		Read/Write	Byte	00h	

5.10.1 FRCTL0 Register

FRAM Controller Control Register 0

Figure 5-3. FRCTL0 Register

15	14	13	12	11	10	9	8
FRCTLPW							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
Reserved	NWAITS			Reserved			
r-0	rw-[0]	rw-[0]	rw-[0]	r-0	r-0	r-0	r-0

Table 5-2. FRCTL0 Register Description

Bit	Field	Type	Reset	Description
15-8	FRCTLPW	RW	96h	FRCTLPW password. Always reads as 96h. To enable write access to the FRCTL registers, write A5h. A word write of any other value causes a PUC. After a correct password is written and register access is enabled, write a wrong password in byte mode to disable the access. In this case, no PUC is generated.
7	Reserved	R	0h	Reserved. Always read 0.
6-4	NWAITS	RW	0h	Wait state control. Specifies number of wait states (0 to 7) required for an FRAM access (cache miss). 0 implies no wait states.
3	Reserved	R	0h	Reserved. Must be written as 0.
2-0	Reserved	R	0h	Reserved. Always read 0.

5.10.2 GCCTL0 Register

General Control Register 0

Figure 5-4. GCCTL0 Register

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
UBDRSTEN	UBDIE	CBDIE	Reserved	Reserved	FRPWR	FRLPMPWR	Reserved
rw-[0]	rw-[0]	rw-[0]	r-0	r-0	rw-1	rw-1	r-0

Table 5-3. GCCTL0 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always read 0.
7	UBDRSTEN	RW	0h	Enable Power Up Clear (PUC) reset if FRAM uncorrectable bit error detected. The bits UBDRSTEN and UBDIE are mutual exclusive and are not allowed to be set simultaneously. Only one error handling can be selected at one time. 0b = PUC not initiated on uncorrectable bit detection flag. 1b = PUC initiated on uncorrectable bit detection flag. Generates vector in SYSRSTIV.
6	UBDIE	RW	0h	Enable NMI event if uncorrectable bit error detected. The bits UBDRSTEN and UBDIE are mutual exclusive and are not allowed to be set simultaneously. Only one error handling can be selected at one time. 0b = Uncorrectable bit detection interrupt disabled. 1b = Uncorrectable bit detection interrupt enabled. Generates vector in SYSSNIV.
5	CBDIE	RW	0h	Enable NMI event if correctable bit error detected. 0b = Correctable bit detection interrupt disabled. 1b = Correctable bit detection interrupt enabled. Generates vector in SYSSNIV.
4	Reserved	R	0h	Reserved. Always read 0.
3	Reserved	R	0h	Reserved. Always read 0.
2	FRPWR	RW	1h	FRAM power control. Writing to the register enables or disables the FRAM power supply. The read back of the register returns the actual state of the FRAM power supply, also reflecting a possible delay after enabling the power supply. FRPWR = 1 indicates that the FRAM power is up and ready. 0b = FRAM power supply disabled 1b = FRAM power supply enabled
1	FRLPMPWR	RW	1h	Enables FRAM auto power up after LPM 0b = FRAM startup is delayed to the first FRAM access after LPM exit 1b = FRAM is powered up instantly with LPM exit.
0	Reserved	R	0h	Reserved. Always read 0.

5.10.3 GCCTL1 Register

General Control Register 1

Figure 5-5. GCCTL1 Register

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				ACCTEIFG	UBDIFG	CBDIFG	Reserved
r-0	r-0	r-0	r-0	r-0	rw-[0]	rw-[0]	r-0

Table 5-4. GCCTL1 Register Description

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Always read 0.
3	ACCTEIFG	RW	0h	Access time error flag. This flag is set and a reset PUC is generated if a wrong setting for NWAITS is set and the FRAM access time is violated. This bit is cleared by software or by reading the system reset vector word SYSRSTIV if it is the highest pending flag. This bit is write 0 only, write 1 has no effect.
2	UBDIFG	RW	0h	FRAM uncorrectable bit error flag. This interrupt flag is set if an uncorrectable bit error has been detected in the FRAM memory error detection logic. This bit is cleared by software or by reading the system NMI vector word SYSSNIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect. 0b = No interrupt pending 1b = Interrupt pending. Can be cleared by user or by reading SYSSNIV.
1	CBDIFG	RW	0h	FRAM correctable bit error flag. This interrupt flag is set if a correctable bit error has been detected and corrected in the FRAM memory error detection logic. This bit is cleared by software or by reading the system NMI vector word SYSSNIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect. 0b = No interrupt pending 1b = Interrupt pending. Can be cleared by user or by reading SYSSNIV
0	Reserved	R	0h	Reserved. Always read 0.

Memory Protection Unit (MPU)

This chapter describes the operation of the Memory Protection Unit (MPU). The MPU is family specific.

Topic **Page**

6.1 Memory Protection Unit (MPU) Introduction 260
6.2 MPU Segments 261
6.3 MPU Access Management Settings 265
6.4 MPU Violations 266
6.5 MPU Lock 266
6.6 MPU Registers 267

6.1 Memory Protection Unit (MPU) Introduction

The MPU protects against accidental writes to designated read-only memory segments or execution of code from a constant memory segment. Clearing the MPUENA bit disables the MPU, and the complete memory is accessible for read, write, and execute operations. After a BOR, the complete memory is accessible without restrictions to read, write, and execute operations.

The Memory Protection Unit features include:

- Configuration of main memory into three variable-sized segments
- Access rights for each segment can be set independently
- Fixed-size constant user information memory segment with selectable access rights
- Protection of MPU registers by password

NOTE: After BOR, no segmentation is initiated, and the main memory and information memory are accessible by read, write, and execute operations.

Figure 6-1 shows an overview of the Memory Protection Unit.

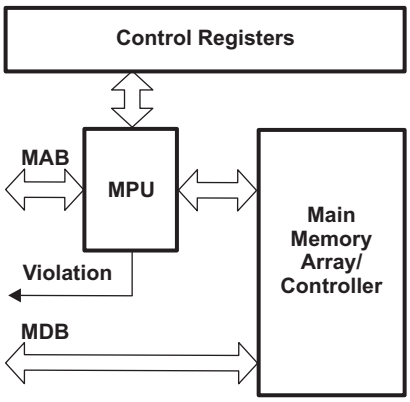


Figure 6-1. Memory Protection Unit Overview

6.2 MPU Segments

6.2.1 Main Memory Segments

The MPU can logically divide the main memory into three segments. The size of each segment is defined by setting the borders between adjacent segments. To configure three segments, a lower border (B1) and a higher border (B2) are positioned by control register bits MPUSEGB1[15:0] and MPUSEGB2[15:0], respectively, in the MPUSBx registers. The position of both borders is limited to the 16 most significant bits of the memory address space. Therefore, the segment borders registers are equivalent to the memory address bus, shifted right by 4 bits (see Figure 6-2). 8 bits of the segment border (MPUSEGBx[13:6]) are user selectable, the remaining bits (MPUSEGBx[15:14] and MPUSEGBx[5:0]) in the border registers are fixed to 0 (see Figure 6-3). This results in a segment size granularity of 1KB for devices with up to 128KB of FRAM.

The beginning of segment 1 is the lowest available address for the main memory as defined in the device-specific data sheet. The lower border (B1) defines the end of segment 1 and the beginning of segment 2. The higher border (B2) defines the end of segment 2 and beginning of segment 3. The end of segment 3 is the highest main memory address as defined in the device-specific data sheet. For example, devices with up to 64KB of FRAM, the highest memory address is 013FFFh. Segment 2 includes the address defined by the lower border (B1) but excludes the higher border (B2).

The address bus (MAB) is analyzed by the MPU using the 16 most significant bits along with the current border settings. If the significant address portion is lower or equal than MPUSEGB1[15:0] and MPUSEGB2[15:0], segment 1 is selected. For significant address bits equal or greater MPUSEGB1[15:0] and lower than MPUSEGB2[15:0], segment 2 is selected. For address values larger than MPUSEGB1[15:0] and MPUSEGB2[15:0], segment 3 is selected.

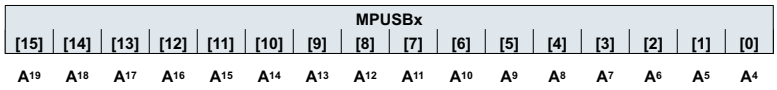


Figure 6-2. Segment Border Register

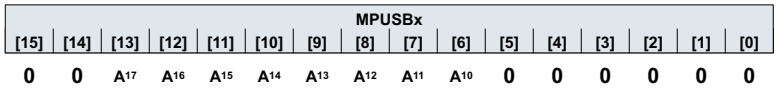


Figure 6-3. Segment Border Register - Fixed Bits

The segmentation of the main memory is shown in Figure 6-4.

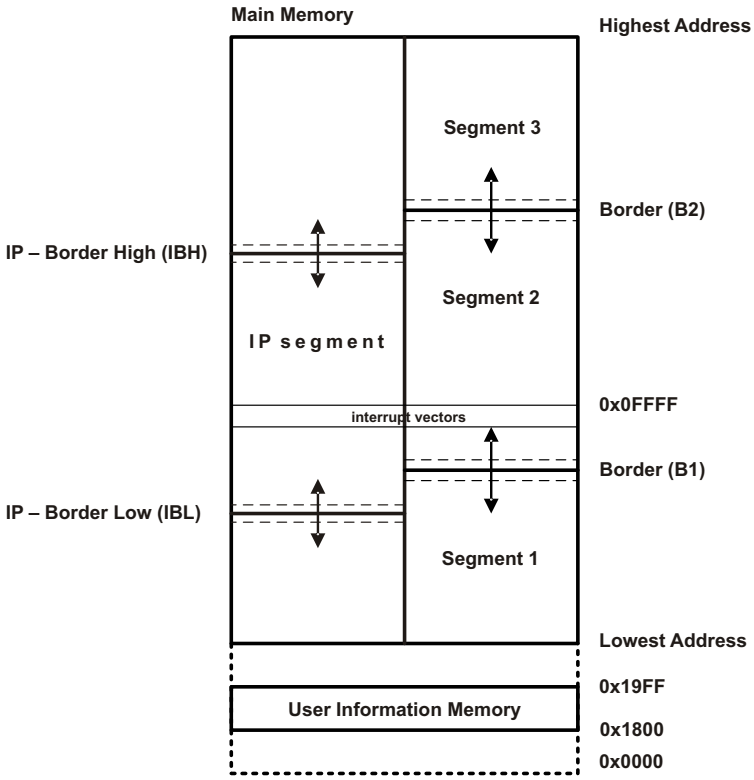


Figure 6-4. Segmentation of Main Memory

6.2.2 IP Encapsulation Segment

The MPU can logically separate an address range in the main memory from unconditional external access. The size of this segment is defined by setting the upper and lower borders of this segment. To configure the segments, a lower (IBL) border and a higher (IBH) border are positioned by control register bits MPUIPSEGB1[15:0] and MPUIPSEGB2[15:0], respectively, in the MPUIPSEGBx register. The position of both borders follows the same mechanism as described in Section 6.2.1 for the main segments.

The beginning of the IP Encapsulation segment (IPE-segment) (IBL) is defined by the lower value of both registers MPUIPSEGB1[15:0] and MPUIPSEGB2[15:0]. The end of the IPE-segment (IBH) is defined by the higher value of both registers MPUIPSEGB1[15:0] and MPUIPSEGB2[15:0]. All memory locations addressed by the 16 most significant bits of the address bus (MAB) equal or above lower border (IBL) and less than IBH are treated as protected.

Only program code executed from the IPE segment can access data stored in this segment. The access rights are evaluated with each code access. Each code access outside of the IP-protected area deactivates the data access into the IPE segment. JTAG or DMA cannot access the IPE segment. The interrupt vector table is always open for read and write accesses (for details see Section 6.4.1).

To execute code from the IPE-segment branch into that segment or call functions stored in that segment. Interrupt service routines can be executed from the IPE-segment, too.

The possible combinations of code execution and memory access types and the resulting access rights are shown in Table 6-1.

An unauthorized access to the IPE-segment returns a value equivalent to the instruction "JMP $" and triggers an interrupt. Additionally, the generation of a PUC can be configured.

Table 6-1. IP Encapsulation Access Rights

IBL ≤ Mem Address < IBH	IBL ≤ Program Counter < IBH	JTAG or DMA Access	CPU Access
0FF80h ≤ Mem Address < 0FFFFh	**-**	**read/write**	**read/write**
false	false	yes	yes
false	true	yes	yes
true	false	no	no
true	true	no	yes

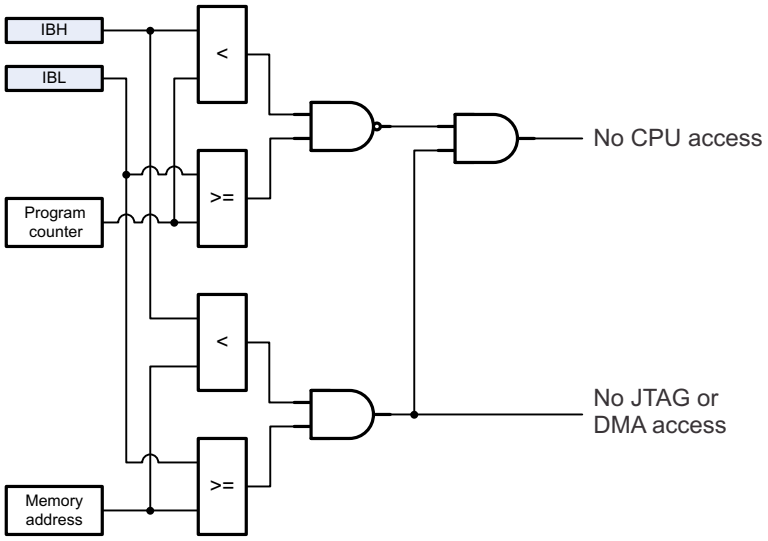


Figure 6-5. IP Encapsulation Access Rights Equivalent Schematic

NOTE: IP Encapsulation area access rights do not override MPU segment rights. The IP Encapsulation rights are evaluated and if the access is granted, access rights as describe in Section 6.3 are applied.

NOTE: **Code fetch from the first 8 bytes in IPE-segment does not enable data access.** The first 8 bytes within the IPE-segment do not enable data access within the IPE-segment if code is executed from that area. The start of an IPE-segment is reserved for a data structure describing the IPE-segment boundaries.

The segmentation of the main memory is shown in Figure 6-4.

6.2.3 Segment Border Setting

Section 6.2.1 describes the procedure of setting borders for segmentation of the main memory. This section describes how the values in MPUSBx[15:0] and MPUIPSEGBx[15:0] bits need to be set to achieve the desired borders for different memory sizes. The bits of the MUSBx[15:0] bits represent the 16 most significant bits of the border address that can be selected.

The setting of the MPUSBx[15:0] bits forms a border between two segments of main memory space. For the following examples, the segment with the higher address range formed by this border is called the higher segment. The segment with the lower address range is called the lower segment.

The lowest address in the higher segment can be calculated with the following formula:

Given:

Segment Border Address (BA) or register value MPUSBx

Hence follows:

MPUSBx = (BA) >> 4

BA = (MPUSBx << 4)

Examples:

Segment border address = 0x0F000 → MPUSBx = (0x0F000 >> 4) = 0x0F00

Segment border address = 0x13000 → MPUSBx = (0x13000 >> 4) = 0x1300

MPUSBx = 0x1100 → segment border address = (0x1100 << 4) = 0x11000

For devices in the memory range of 32k to 128k usable memory, the changeable bits are limited to MPUxBx[13:6] which corresponds with the address bits MAB[17:10]. This leads to a segment size of 1kByte.

Table 6-2. MPU Border Selection Example 64k (004000h to 013FFFh)

Border Address	MPUSBx[15:0]
(outside)	0000h
⋮	⋮
(outside)	03C0h
04000h	0400h
04400h	0440h
04800h	0480h
04C00h	04C0h
05000h	0500h
⋮	⋮
0F000h	0F00h
0F400h	0F40h
0F800h	0F80h
0FC00h	0FC0h
10000h	1000h
10400h	1040h
⋮	⋮
13000h	1300h
13400h	1340h
13800h	1380h
13C00h	13C0h
14000h (top of memory)	1400h
(outside)	1440h
⋮	⋮
(outside)	3F80h
(outside)	3FC0h

NOTE: Depending on the memory size settings for MPUSBx[4:0], the calculation may result in a lower address space than is available. For those settings, a lower segment does not exist, and the higher segment starts with the first available memory address (see memory organization from device-specific data sheet).

6.2.4 IP Encapsulation Border Settings

The setting of the boundaries for the IP Encapsulation segment follows the same principle as the Main Segment settings.

6.2.5 Information Memory

The information memory is a partition of memory that is 512 bytes in size. The information memory can be used to store application-specific information (such as IDs or version numbers), or it can be used for executable code. It is located at address 01800h to 019FFh.

6.3 MPU Access Management Settings

Each segment described in Section 6.2.3 and Section 6.2.5 can have read, write, and execute access rights set independently.

The MPUSAM register allows setting the access rights for the four segments (information memory segment, three main memory segments) . MPUSEGxRE enables read access for segment x, MPUSEGxWE enables write access for segment x, and MPUSEGxXE enables code execution from segment x. JTAG and DMA accesses are treated as read or write data accesses and are evaluated according to the corresponding access bits.

Table 6-3 shows the different settings of MPUSEGxXE, MPUSEGxWE, and MPUSEGxRE. Not all settings lead to a different memory protection. For example, as shown, if the execution bit MPUSEGxXE is set to 1, read access is automatically allowed independent of the setting of MPUSEGxRE. Also, setting the MPUSEGxWE bit to 1 enables the read option.

NOTE: Combinations that are not shown in Table 6-3 should be avoided because they may be used in future versions of the MPU.

Table 6-3. Segment Access Rights

MPUSEGxXE	MPUSEGxWE	MPUSEGxRE	Execute Rights	Write Rights	Read Rights
0	0	0	no	no	no
0	0	1	no	no	yes
0	1	1	no	yes	yes
1	0	1	yes	no	yes
1	1	1	yes	yes	yes

NOTE: Discontinuity instructions at segment boundaries

Do not fill code segments to the last word with program code, because program discontinuity instructions like jump or branch instructions, RET, CALL, ... at a segment boundary can trigger an access right violation.

The CPU prefetches instructions beyond the one currently being executed. The MPU interprets the prefetch as read and instruction fetch accesses. For example if there is a JMP instruction (or any another discontinuity instruction) at the segment boundary, the CPU prefetches from the neighboring segment. This causes an access right violation if instruction fetches are not allowed within the neighboring segment.

6.4 MPU Violations

6.4.1 Interrupt Vector Table and Reset Vector

The interrupt vector table and the reset vector are located at addresses 0FF80h to 0FFFFh. It is possible to define a segment that includes this address space with restricted access rights. If an interrupt or a reset occurs, and this segment is read protected, the MPU automatically allows access to the Interrupt Vector memory space 0FF80h to 0FFFFh. Only the interrupt vector table is accessible. Access to the interrupt routine itself is not automatically enabled.

For IP Encapsulation, the interrupt vector table is always excluded from execute rights. Code fetches at the addresses 0FF80h to 0FFFFh are always denied if IPE-segments include that memory range. Read or write rights are granted depending on MPU segment access management register MPUSAM, if applicable.

NOTE: Only the interrupt table and the reset vector are opened on an interrupt or reset occurrence. If the application protects the segment that contains the interrupt routine itself from execution rights, a violation occurs.

6.4.2 Violation Handling

The handling of access rights violations can be selected for each segment with the MPUSEGxVS bit in the MPUSAM register. By default (MPUSEGxVS = 0), any access right violation causes a Non-Maskable Interrupt (NMI). Setting MPUSEGxVS = 1, causes a PUC to occur. In either case, the illegal instruction on a protected memory segment is not executed. Upon an access rights violation, the data bus content (MDB) is driven with 03FFFh until next valid data is available.

6.5 MPU Lock

The MPU registers can be protected from write access by setting the MPULOCK bit. Write access is not possible on all MPU registers except MPUCTL1, MPUIPC0, and MPUIPSEGBx until a BOR occurs. MPULOCK cannot be cleared manually.

MPUIPLOCK allows to separately lock the MPUIPC0 and MPUIPSEGBx registers. Write access is not possible on these registers until a BOR occurs. MPUIPLOCK cannot be cleared manually.

6.6 MPU Registers

The MPU registers are listed in Table 6-4. The base address of the MPU module can be found in the device-specific data sheet. The address offset of each MPU register is given in Table 6-4. The password defined in the MPUCTL0 register controls access to all MPU 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 MPUCTL0 upper byte. Word accesses to MPUCTL0 with a wrong password triggers a PUC. A write access to a register other than MPUCTL0 while write access is not enabled causes a PUC. This behavior is independent from MPULOCK bit settings. Password write is always enabled to allow consecutive access to MPUCTL1 and independent configuration of MPU and IP-Encapsulation 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 6-4. MPU Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	MPUCTL0	Memory Protection Unit Control 0	Read/write	Word	9600h	Section 6.6.1
00h	MPUCTL0_L		Read/Write	Byte	00h	
01h	MPUCTL0_H		Read/Write	Byte	96h	
02h	MPUCTL1	Memory Protection Unit Control 1	Read/write	Word	0000h	Section 6.6.2
02h	MPUCTL1_L		Read/Write	Byte	00h	
03h	MPUCTL1_H		Read/Write	Byte	00h	
04h	MPUSEGB2	Memory Protection Unit Segmentation Border 2 Register	Read/write	Word	0000h	Section 6.6.3
04h	MPUSEGB2_L		Read/Write	Byte	00h	
05h	MPUSEGB2_H		Read/Write	Byte	00h	
06h	MPUSEGB1	Memory Protection Unit Segmentation Border 1 Register	Read/Write	Word	0000h	Section 6.6.4
06h	MPUSEGB1_L		Read/Write	Byte	00h	
07h	MPUSEGB1_H		Read/Write	Byte	00h	
08h	MPUSAM	Memory Protection Unit Segmentation Access Management Register	Read/write	Word	7777h	Section 6.6.5
08h	MPUSAM_L		Read/Write	Byte	77h	
09h	MPUSAM_H		Read/Write	Byte	77h	
0Ah	MPUIPC0	Memory Protection Unit IP Control 0 Register	Read/Write	Word	0000h	Section 6.6.6
0Ah	MPUIPC0_L		Read/Write	Byte	00h	
0Bh	MPUIPC0_H		Read/Write	Byte	00h	
0Ch	MPUIPSEGB2	Memory Protection Unit IP Encapsulation Segment Border 2 Register	Read/Write	Word	0000h	Section 6.6.7
0Ch	MPUIPSEGB2_L		Read/Write	Byte	00h	
0Dh	MPUIPSEGB2_H		Read/Write	Byte	00h	
0Eh	MPUIPSEGB1	Memory Protection Unit IP Encapsulation Segment Border 1 Register	Read/Write	Word	0000h	Section 6.6.8
0Eh	MPUIPSEGB1_L		Read/Write	Byte	00h	
0Fh	MPUIPSEGB1_H		Read/Write	Byte	00h	

6.6.1 MPUCTL0 Register

Memory Protection Unit Control 0 Register

Figure 6-6. MPUCTL0 Register

15	14	13	12	11	10	9	8
MPUPW							
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			MPUSEGIE	Reserved		MPULOCK	MPUENA
r-0	r-0	r-0	rw-[0]	r-0	r-0	rw-[0]	rw-[0]

Table 6-5. MPUCTL0 Register Description

Bit	Field	Type	Reset	Description
15-8	MPUPW	RW	96h	MPU Password. Always reads as 096h. Must be written as 0A5h; writing any other value with a word write generates a PUC. After a correct password is written and MPU register access is enabled, a wrong password write in byte mode disables the access and no PUC is generated. This behavior is independent from MPULOCK bit settings.
7-5	Reserved	R	0h	Reserved. Always read 0.
4	MPUSEGIE	RW	0h	Enable NMI Event if a Segment violation is detected in any Segment. 0b = Segment violation interrupt disabled 1b = Segment violation interrupt enabled
3-2	Reserved	R	0h	Reserved. Always read 0.
1	MPULOCK	RW	0h	MPU Lock. If this bit is set, access to all MPU Registers except MPUCTL1, MPUIPC0, and MPUIPSEGx are locked and they are read only until a BOR occurs. BOR sets MPULOCK to 0. 0b = Open 1b = Locked
0	MPUENA	RW	0h	MPU Enable. This bit enables the MPU operation. The enable bit can be set any time with word write and a correct password, if MPULOCK is not set 0b = Disabled 1b = Enabled

6.6.2 MPUCTL1 Register

Memory Protection Unit Control 1 Register

Figure 6-7. MPUCTL1 Register

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			MPUSEGIPIFG	MPUSEGIIFG	MPUSEG3IFG	MPUSEG2IFG	MPUSEG1IFG
r-0	r-0	r-0	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]

Table 6-6. MPUCTL1 Register Description

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Always read 0.
4	MPUSEGIPIFG	RW	0h	IP Encapsulation Access Violation Interrupt Flag. This bit is set if an access violation in the IP Encapsulation memory segment is detected. This bit is cleared by software or by reading the reset vector word SYSRSTIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect. 0b = No interrupt pending 1b = Interrupt pending
3	MPUSEGIIFG	RW	0h	User Information Memory Violation Interrupt Flag. This bit is set if an access violation in User Information Memory is detected. This bit is cleared by software or by reading the reset vector word SYSRSTIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect. 0b = No interrupt pending 1b = Interrupt pending
2	MPUSEG3IFG	RW	0h	Main Memory Segment 3 Violation Interrupt Flag. This bit is set if an access violation in Main Memory Segment 3 is detected. This bit is cleared by software or by reading the reset vector word SYSRSTIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect. 0b = No interrupt pending 1b = Interrupt pending
1	MPUSEG2IFG	RW	0h	Main Memory Segment 2 Violation Interrupt Flag. This bit is set if an access violation in Main Memory Segment 2 is detected. This bit is cleared by software or by reading the reset vector word SYSRSTIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect. 0b = No interrupt pending 1b = Interrupt pending
0	MPUSEG1IFG	RW	0h	Main Memory Segment 1 Violation Interrupt Flag. This bit is set if an access violation in Main Memory Segment 1 is detected. This bit is cleared by software or by reading the reset vector word SYSRSTIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect. 0b = No interrupt pending 1b = Interrupt pending

6.6.3 MPUSEGB2 Register

Memory Protection Unit Segmentation Border 2 Register

Figure 6-8. MPUSEGB2 Register

15	14	13	12	11	10	9	8
MPUSEGB2							
r-0	r-0	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]
7	6	5	4	3	2	1	0
MPUSEGB2							
rw-[0]	rw-[0]	r-0	r-0	r-0	r-0	r-0	r-0

Table 6-7. MPUSEGB2 Register Description

Bit	Field	Type	Reset	Description
15-0	MPUSEGB2	RW	0h	MPU Segment Border 2 address line equivalents. MPUSEGB2[15:14] = MPU Segment Border 2 address line 19-18 equivalents. Always read 0. MPUSEGB2[13:6] = MPU Segment Border 2 address lines 17-10. After BOR, the bits are set to 0 (if MPU is enabled and MPUSEGB1 is also 0, only Segment 3 is active). MPUSEGB2[5:0] = MPU Segment Border 2 address line 9-4 equivalents. Always read 0.

6.6.4 MPUSEGB1 Register

Memory Protection Unit Segmentation Border 1 Register

Figure 6-9. MPUSEGB1 Register

15	14	13	12	11	10	9	8
MPUSEGB1							
r-0	r-0	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]
7	6	5	4	3	2	1	0
MPUSEGB1							
rw-[0]	rw-[0]	r-0	r-0	r-0	r-0	r-0	r-0

Table 6-8. MPUSEGB1 Register Description

Bit	Field	Type	Reset	Description
15-0	MPUSEGB1	RW	0h	MPU Segment Border 1 address line equivalents. MPUSEGB1[15:14] = MPU Segment Border 1 address line 19-18 equivalents. Always read 0. MPUSEGB1[13:6] = MPU Segment Border 1 address lines 17-10. After BOR, the bits are set to 0 (if MPU is enabled and MPUSEGB2 is also 0, only Segment 3 is active). MPUSEGB1[5:0] = MPU Segment Border 1 address line 9-4 equivalents. Always read 0.

6.6.5 MPUSAM Register

Memory Protection Unit Segmentation Access Management Register

Figure 6-10. MPUSAM Register

15	14	13	12	11	10	9	8
MPUSEGIVS	MPUSEGIXE	MPUSEGIWE	MPUSEGIRE	MPUSEG3VS	MPUSEG3XE	MPUSEG3WE	MPUSEG3RE
rw-[0]	rw-[1]	rw-[1]	rw-[1]	rw-[0]	rw-[1]	rw-[1]	rw-[1]
7	6	5	4	3	2	1	0
MPUSEG2VS	MPUSEG2XE	MPUSEG2WE	MPUSEG2RE	MPUSEG1VS	MPUSEG1XE	MPUSEG1WE	MPUSEG1RE
rw-[0]	rw-[1]	rw-[1]	rw-[1]	rw-[0]	rw-[1]	rw-[1]	rw-[1]

Table 6-9. MPUSAM Register Description

Bit	Field	Type	Reset	Description
15	MPUSEGIVS	RW	0h	MPU User Information Memory Segment Violation Select. This bit selects if additional to the interrupt flag a PUC must be executed on illegal access to User Information Memory 0b = Violation in User Information Memory asserts the MPUSEGIIFG bit and executes a SNMI if enabled by MPUSEGIE =1 1b = Violation in User Information Memory asserts the MPUSEGIIFG bit and executes a PUC
14	MPUSEGIXE	RW	1h	MPU User Information Memory Segment Execute Enable. If set, this bit enables execution on User Information Memory 0b = Execute code on User Information Memory causes a violation 1b = Execute code on User Information Memory is allowed
13	MPUSEGIWE	RW	1h	MPU User Information Memory Segment Write Enable. If set, this bit enables write access on User Information Memory 0b = Write on User Information Memory causes a violation 1b = Write on User Information Memory is allowed
12	MPUSEGIRE	RW	1h	MPU User Information Memory Segment Read Enable. If set, this bit enables read access on User Information Memory 0b = Read on User Information Memory causes a violation if MPUSEGIWE=MPUSEGIXE=0 1b = Read on User Information Memory is allowed
11	MPUSEG3VS	RW	0h	MPU Main Memory Segment 3 Violation Select. This bit selects if additional to the interrupt flag a PUC must be executed on illegal access to Main Memory segment 3 0b = Violation in Main Memory Segment 3 asserts the MPUSEG3IFG bit and executes a SNMI if enabled by MPUSEGIE =1 1b = Violation in Main Memory Segment 3 asserts the MPUSEG3IFG bit and executes a PUC
10	MPUSEG3XE	RW	1h	MPU Main Memory Segment 3 Execute Enable. If set this bit enables execution on Main Memory segment 3 0b = Execute code on Main Memory Segment 3 causes a violation 1b = Execute code on Main Memory Segment 3 is allowed
9	MPUSEG3WE	RW	1h	MPU Main Memory Segment 3 Write Enable. If set this bit enables write access on Main Memory segment 3 0b = Write on Main Memory Segment 3 causes a violation 1b = Write on Main Memory Segment 3 is allowed
8	MPUSEG3RE	RW	1h	MPU Main Memory Segment 3 Read Enable. If set this bit enables read access on Main Memory segment 3 0b = Read on Main Memory Segment 3 causes a violation if MPUSEG3WE = MPUSEG3XE = 0 1b = Read on Main Memory Segment 3 is allowed

Table 6-9. MPUSAM Register Description (continued)

Bit	Field	Type	Reset	Description
7	MPUSEG2VS	RW	0h	MPU Main Memory Segment 2 Violation Select. This bit selects if additional to the interrupt flag a PUC must be executed on illegal access to Main Memory segment 2 0b = Violation in Main Memory Segment 2 asserts the MPUSEG2IFG bit and executes a SNMI if enabled by MPUSEGIE =1 1b = Violation in Main Memory Segment 2 asserts the MPUSEG2IFG bit and executes a PUC
6	MPUSEG2XE	RW	1h	MPU Main Memory Segment 2 Execute Enable. If set this bit enables execution on Main Memory segment 2 0b = Execute code on Main Memory Segment 2 causes a violation 1b = Execute code on Main Memory Segment 2 is allowed
5	MPUSEG2WE	RW	1h	MPU Main Memory Segment 2 Write Enable. If set this bit enables write access on Main Memory segment 2 0b = Write on Main Memory Segment 2 causes a violation 1b = Write on Main Memory Segment 2 is allowed
4	MPUSEG2RE	RW	1h	MPU Main Memory Segment 2 Read Enable. If set this bit enables read access on Main Memory segment 2 0b = Read on Main Memory Segment 2 causes a violation if MPUSEG2WE = MPUSEG2XE = 0 1b = Read on Main Memory Segment 2 is allowed
3	MPUSEG1VS	RW	0h	MPU Main Memory Segment 1 Violation Select. This bit selects if additional to the interrupt flag a PUC must be executed illegal access to Main Memory segment 1 0b = Violation in Main Memory Segment 1 asserts the MPUSEG1IFG bit and executes a SNMI if enabled by MPUSEGIE = 1 1b = Violation in Main Memory Segment 1 asserts the MPUSEG1IFG bit and executes a PUC
2	MPUSEG1XE	RW	1h	MPU Main Memory Segment 1 Execute Enable. If set this bit enables execution on Main Memory segment 1 0b = Execute code on Main Memory Segment 1 causes a violation 1b = Execute code on Main Memory Segment 1 is allowed
1	MPUSEG1WE	RW	1h	MPU Main Memory Segment 1 Write Enable. If set this bit enables write access on Main Memory segment 1 0b = Write on Main Memory Segment 1 causes a violation 1b = Write on Main Memory Segment 1 is allowed
0	MPUSEG1RE	RW	1h	MPU Main Memory Segment 1 Read Enable. If set this bit enables read access on Main Memory segment 1 0b = Read on Main Memory Segment 1 causes a violation if MPUSEG1WE = MPUSEG1XE = 0 1b = Read on Main Memory Segment 1 is allowed

6.6.6 MPUIPC0 Register

Memory Protection Unit IP Control 0 Register

Figure 6-11. MPUIPC0 Register

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
MPUIPLOCK	MPUIPENA	MPUIPVS	Reserved				
rw[0]	rw-[0]	rw-[0]	r-0	r-0	r-0	r-0	r-0

Table 6-10. MPUIPC0 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always read 0.
7	MPUIPLOCK	RW	0h	MPU IP Encapsulation Lock. If this bit is set, access to MPUIPC0 and MPUIPSEGBx registers is locked, and they are read-only until a BOR occurs. BOR sets the bit to 0. 0b = Open 1b = Locked
6	MPUIPENA	RW	0h	MPU IP Encapsulation Enable. This bit enables the MPU IP Encapsulation operation. The enable bit can be set any time with word write and a correct password, if MPUIPLOCK is not set 0b = Disabled 1b = Enabled
5	MPUIPVS	RW	0h	MPU IP Encapsulation segment Violation Select. This bit selects whether or not a PUC occurs on illegal access to the IPE-segment. 0b = Violation in Main Memory Segment 1 asserts the MPUSEGPIFG bit and executes a SNMI if enabled by MPUSEGIE = 1 1b = Violation in Main Memory Segment 1 asserts the MPUSEGPIFG bit and executes a PUC
4-0	Reserved	R	0h	Reserved. Always read 0.

6.6.7 MPUIPSEGB2 Register

Memory Protection Unit IP Encapsulation Segmentation Border 2 Register

Figure 6-12. MPUIPSEGB2 Register

15	14	13	12	11	10	9	8
MPUIPSEGB2							
r-0	r-0	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]
7	6	5	4	3	2	1	0
MPUIPSEGB2							
rw-[0]	rw-[0]	r-0	r-0	r-0	r-0	r-0	r-0

Table 6-11. MPUIPSEGB2 Register Description

Bit	Field	Type	Reset	Description
15-0	MPUIPSEGB2	RW	0h	MPU IP Segment Border 2 address line equivalents. MPUIPSEGB2[15:14] = MPU IP Segment Border 2 address line 19-18 equivalents. Always read 0. MPUIPSEGB2[13:6] = MPU IP Segment Border 2 address lines 17-10. After BOR, the bits are set to 0 (if MPU is enabled and MPUSEGB1 is also 0, only Segment 3 is active). MPUIPSEGB2[5:0] = MPU IP Segment Border 2 address line 9-4 equivalents. Always read 0.

6.6.8 MPUIPSEGB1 Register

Memory Protection Unit IP Encapsulation Segmentation Border 1 Register

Figure 6-13. MPUIPSEGB1 Register

15	14	13	12	11	10	9	8
MPUIPSEGB1							
r-0	r-0	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]
7	6	5	4	3	2	1	0
MPUIPSEGB1							
rw-[0]	rw-[0]	r-0	r-0	r-0	r-0	r-0	r-0

Table 6-12. MPUIPSEGB1 Register Description

Bit	Field	Type	Reset	Description
15-0	MPUIPSEGB1	RW	0h	MPU Segment Border 1 address line equivalents. MPUIPSEGB1[15:14] = MPU Segment Border 1 address line 19-18 equivalents. Always read 0. MPUIPSEGB1[13:6] = MPU Segment Border 1 address lines 17-10. After BOR, the bits are is set to 0 (if MPU is enabled and MPUSEGB2 is also 0, only Segment 3 is active). MPUIPSEGB1[5:0] = MPU Segment Border 1 address line 9-4 equivalents. Always read 0.

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 278
7.2 DMA Operation 280
7.3 DMA Registers 292

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 can 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 the number of channels that are 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, word, or mixed byte and 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.

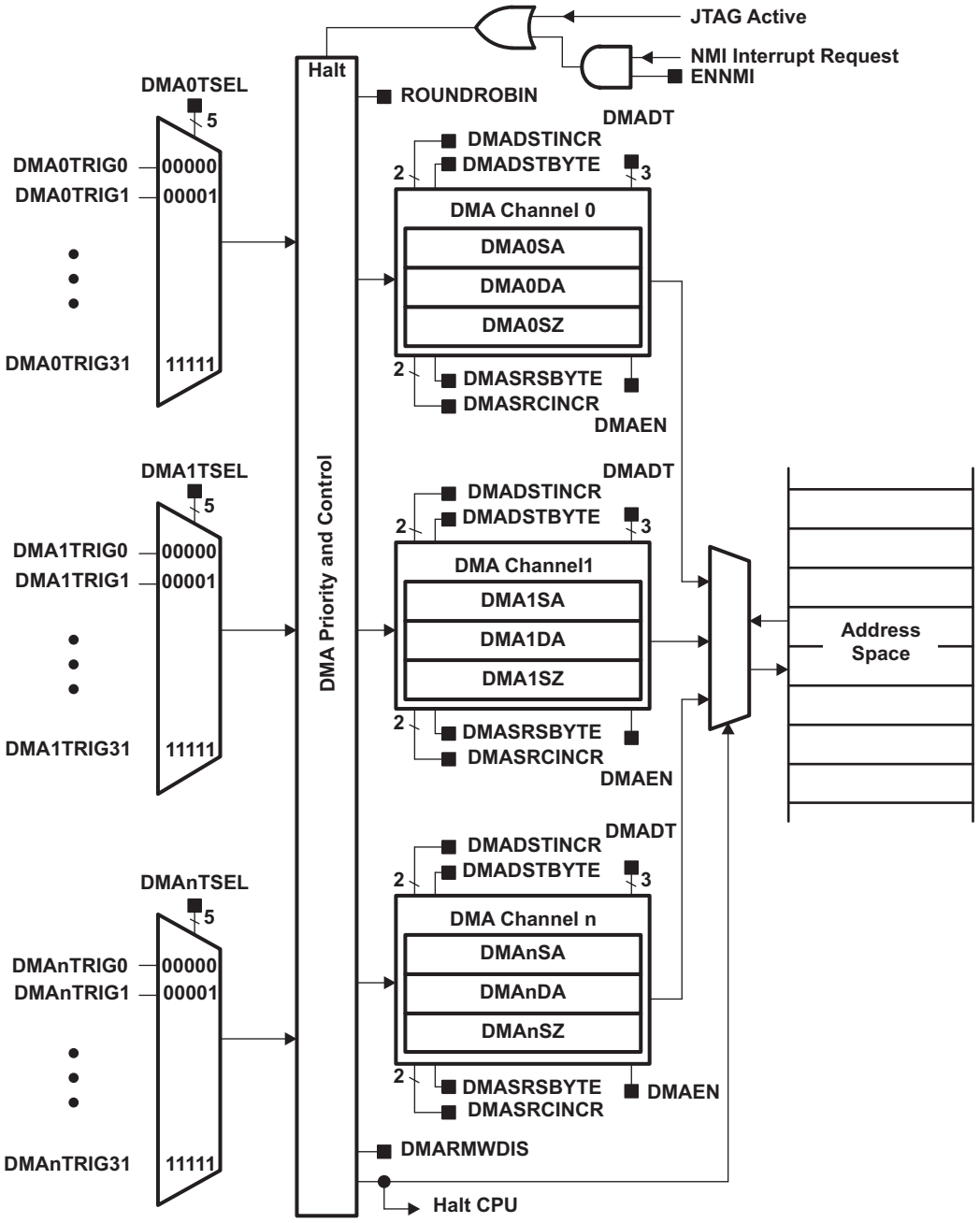


Figure 7-1. DMA Controller Block Diagram

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.

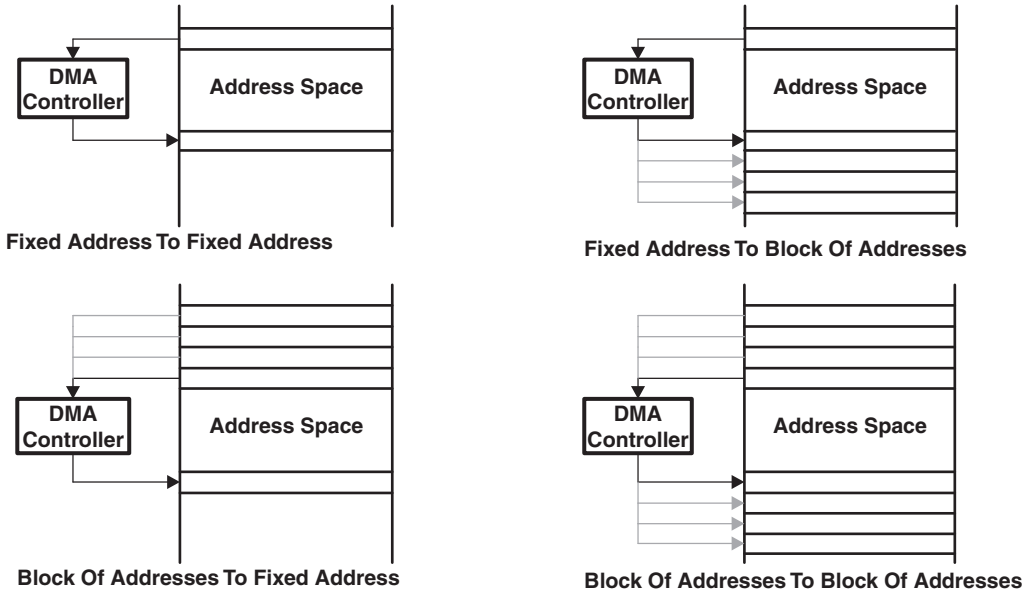


Figure 7-2. DMA Addressing Modes

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 destination locations 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 or word transfer requires a separate trigger. The single transfer state diagram is shown in Figure 7-3.

The DMAxSZ register defines 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.

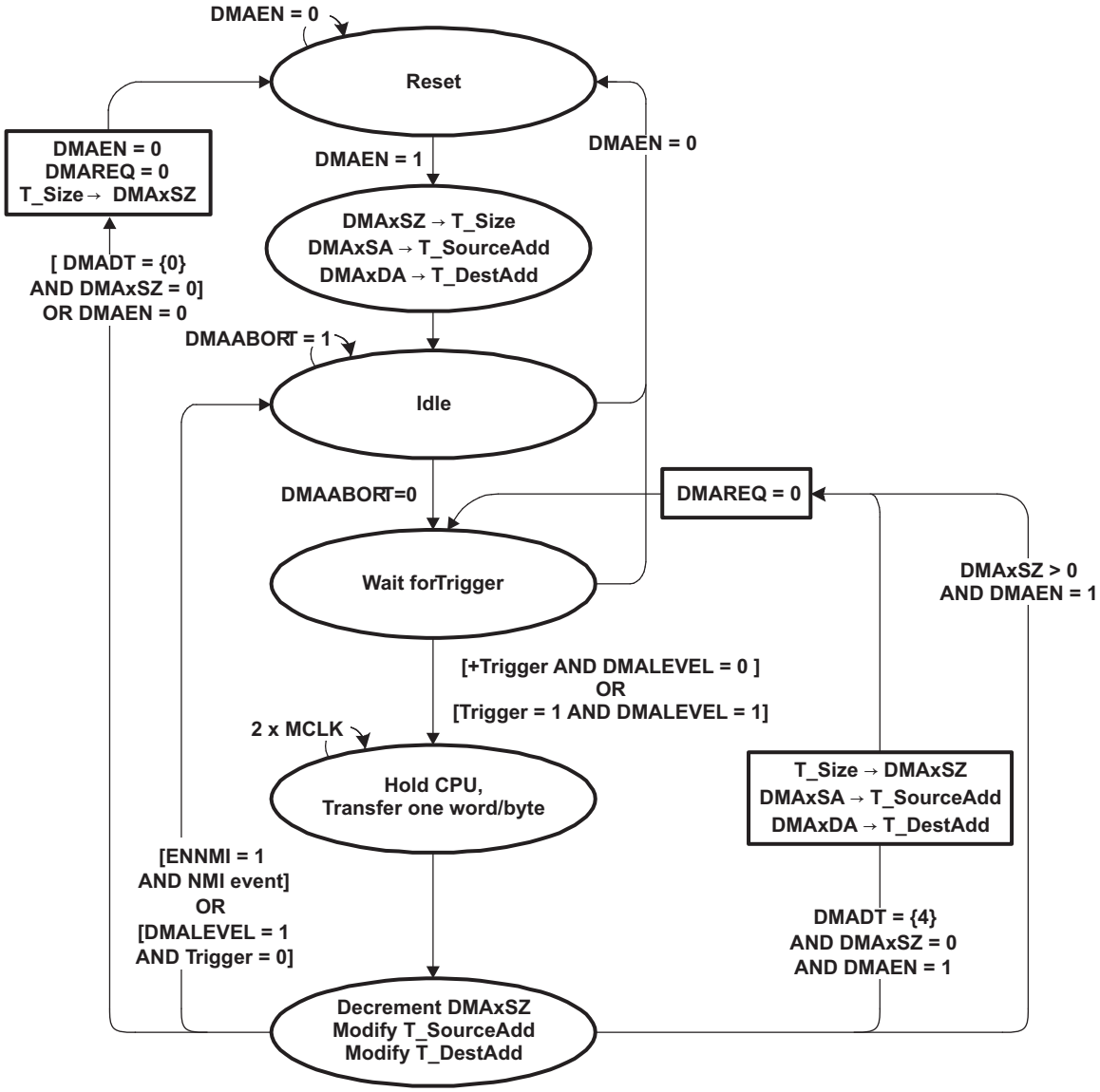


Figure 7-3. DMA Single Transfer State Diagram

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 started, another trigger signal that occurs during the block transfer is ignored. The block transfer state diagram is shown in Figure 7-4.

The DMAxSZ register defines 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 starts another block transfer.

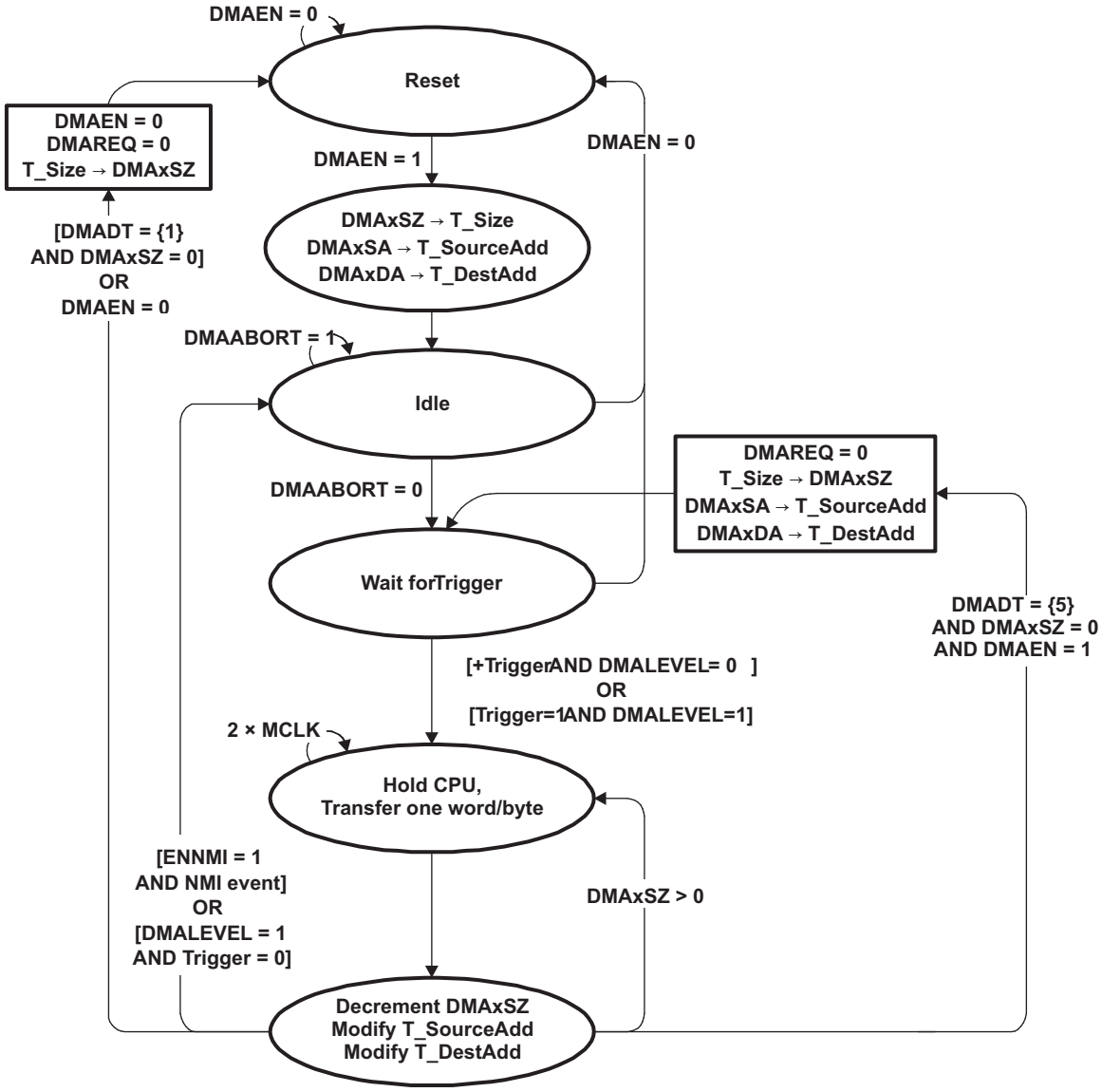


Figure 7-4. DMA Block Transfer State Diagram

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 or 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 defines 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.

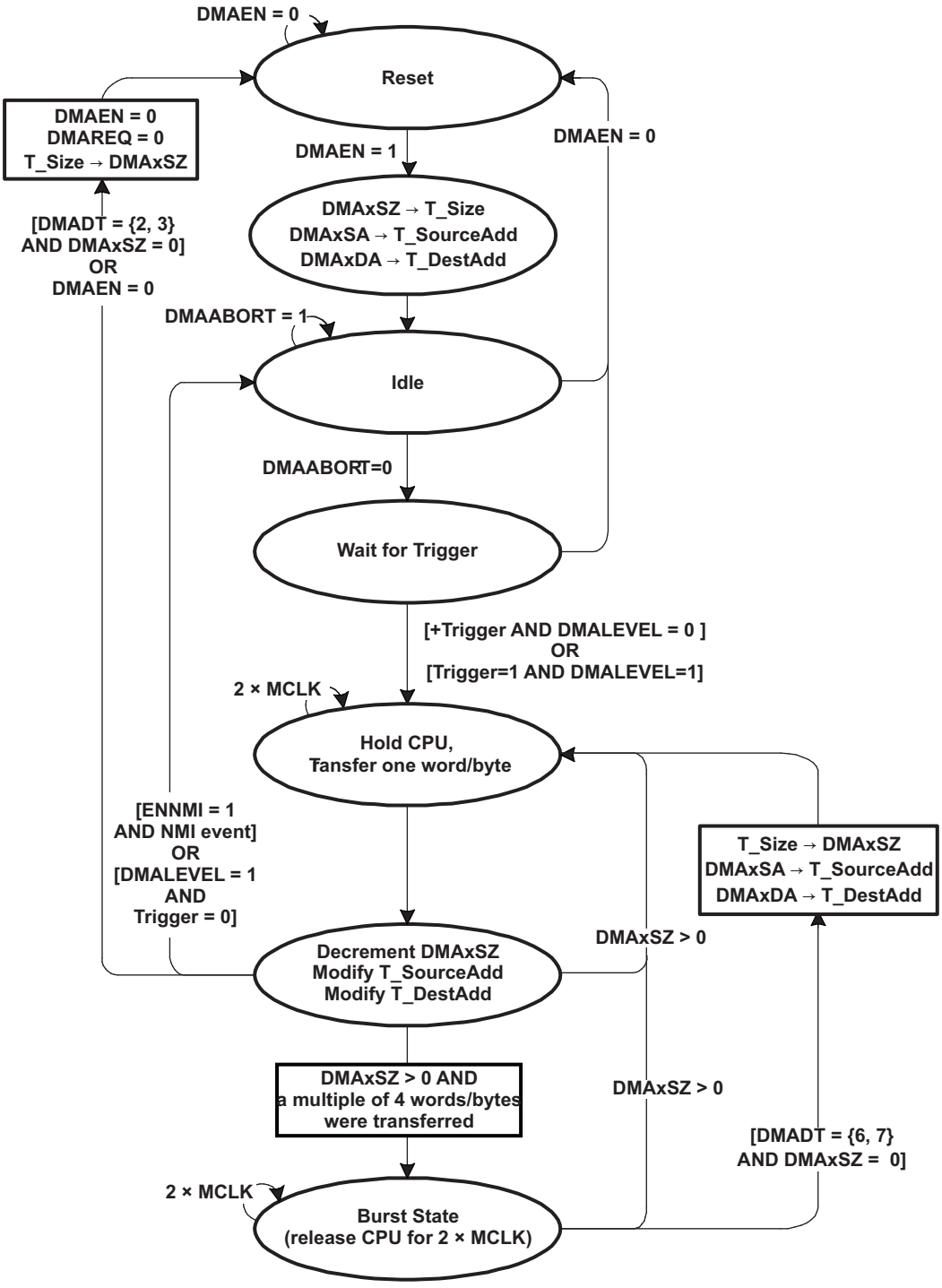


Figure 7-5. DMA Burst-Block Transfer State Diagram

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.

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 does 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.
eUSCI_Ax	A transfer is triggered when eUSCI_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 eUSCI_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.
eUSCI_Bx	A transfer is triggered when eUSCI_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 eUSCI_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.
ADC10_B	A transfer is triggered by an ADC10IFG0 flag. A transfer is triggered when the conversion is completed and the ADC10IFG0 is set. Setting the ADC10IFG0 with software does not trigger a transfer. The ADC10IFG0 flag is automatically reset when the ADC10MEM0 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 or 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 or LPM1 MCLK = DCOCLK	5 MCLK cycles
Low-power mode LPM3 or LPM4 MCLK = DCOCLK	5 MCLK cycles + 5 µs(1)
Low-power mode LPM0 or LPM1 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 eUSCI_B I²C Module With the DMA Controller

The eUSCI_B I²C module provides two trigger sources for the DMA controller. The eUSCI_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 ADC10 With the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data from the ADC10MEM0 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 ADC10 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur. A transfer is triggered when the conversion is completed and the ADC10IFG0 is set. Setting the ADC10IFG0 with software does not trigger a transfer. The ADC10IFG0 flag is automatically reset when the ADC10MEM0 register is accessed by the DMA controller.

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

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	DMACTL0	DMA Control 0	Read/write	Word	0000h	Section 7.3.1
02h	DMACTL1	DMA Control 1	Read/write	Word	0000h	Section 7.3.2
04h	DMACTL2	DMA Control 2	Read/write	Word	0000h	Section 7.3.3
06h	DMACTL3	DMA Control 3	Read/write	Word	0000h	Section 7.3.4
08h	DMACTL4	DMA Control 4	Read/write	Word	0000h	Section 7.3.5
0Eh	DMAIV	DMA Interrupt Vector	Read only	Word	0000h	Section 7.3.10
00h	DMA0CTL	DMA Channel 0 Control	Read/write	Word	0000h	Section 7.3.6
02h	DMA0SA	DMA Channel 0 Source Address	Read/write	Word, double word	undefined	Section 7.3.7
06h	DMA0DA	DMA Channel 0 Destination Address	Read/write	Word, double word	undefined	Section 7.3.8
0Ah	DMA0SZ	DMA Channel 0 Transfer Size	Read/write	Word	undefined	Section 7.3.9
00h	DMA1CTL	DMA Channel 1 Control	Read/write	Word	0000h	Section 7.3.6
02h	DMA1SA	DMA Channel 1 Source Address	Read/write	Word, double word	undefined	Section 7.3.7
06h	DMA1DA	DMA Channel 1 Destination Address	Read/write	Word, double word	undefined	Section 7.3.8
0Ah	DMA1SZ	DMA Channel 1 Transfer Size	Read/write	Word	undefined	Section 7.3.9
00h	DMA2CTL	DMA Channel 2 Control	Read/write	Word	0000h	Section 7.3.6
02h	DMA2SA	DMA Channel 2 Source Address	Read/write	Word, double word	undefined	Section 7.3.7
06h	DMA2DA	DMA Channel 2 Destination Address	Read/write	Word, double word	undefined	Section 7.3.8
0Ah	DMA2SZ	DMA Channel 2 Transfer Size	Read/write	Word	undefined	Section 7.3.9
00h	DMA3CTL	DMA Channel 3 Control	Read/write	Word	0000h	Section 7.3.6
02h	DMA3SA	DMA Channel 3 Source Address	Read/write	Word, double word	undefined	Section 7.3.7
06h	DMA3DA	DMA Channel 3 Destination Address	Read/write	Word, double word	undefined	Section 7.3.8
0Ah	DMA3SZ	DMA Channel 3 Transfer Size	Read/write	Word	undefined	Section 7.3.9
00h	DMA4CTL	DMA Channel 4 Control	Read/write	Word	0000h	Section 7.3.6
02h	DMA4SA	DMA Channel 4 Source Address	Read/write	Word, double word	undefined	Section 7.3.7
06h	DMA4DA	DMA Channel 4 Destination Address	Read/write	Word, double word	undefined	Section 7.3.8
0Ah	DMA4SZ	DMA Channel 4 Transfer Size	Read/write	Word	undefined	Section 7.3.9
00h	DMA5CTL	DMA Channel 5 Control	Read/write	Word	0000h	Section 7.3.6
02h	DMA5SA	DMA Channel 5 Source Address	Read/write	Word, double word	undefined	Section 7.3.7
06h	DMA5DA	DMA Channel 5 Destination Address	Read/write	Word, double word	undefined	Section 7.3.8
0Ah	DMA5SZ	DMA Channel 5 Transfer Size	Read/write	Word	undefined	Section 7.3.9
00h	DMA6CTL	DMA Channel 6 Control	Read/write	Word	0000h	Section 7.3.6
02h	DMA6SA	DMA Channel 6 Source Address	Read/write	Word, double word	undefined	Section 7.3.7
06h	DMA6DA	DMA Channel 6 Destination Address	Read/write	Word, double word	undefined	Section 7.3.8
0Ah	DMA6SZ	DMA Channel 6 Transfer Size	Read/write	Word	undefined	Section 7.3.9

Table 7-4. DMA Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	DMA7CTL	DMA Channel 7 Control	Read/write	Word	0000h	Section 7.3.6
02h	DMA7SA	DMA Channel 7 Source Address	Read/write	Word, double word	undefined	Section 7.3.7
06h	DMA7DA	DMA Channel 7 Destination Address	Read/write	Word, double word	undefined	Section 7.3.8
0Ah	DMA7SZ	DMA Channel 7 Transfer Size	Read/write	Word	undefined	Section 7.3.9

7.3.1 DMACTL0 Register

DMA Control 0 Register

Figure 7-6. DMACTL0 Register

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)

Table 7-5. DMACTL0 Register Description

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12-8	DMA1TSEL	RW	0h	DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA1TRIG0 00001b = DMA1TRIG1 00010b = DMA1TRIG2 ⋮ 11110b = DMA1TRIG30 11111b = DMA1TRIG31
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-0	DMA0TSEL	RW	0h	DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA0TRIG0 00001b = DMA0TRIG1 00010b = DMA0TRIG2 ⋮ 11110b = DMA0TRIG30 11111b = DMA0TRIG31

7.3.2 DMACTL1 Register

DMA Control 1 Register

Figure 7-7. DMACTL1 Register

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)

Table 7-6. DMACTL1 Register Description

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12-8	DMA3TSEL	RW	0h	DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA3TRIG0 00001b = DMA3TRIG1 00010b = DMA3TRIG2 ⋮ 11110b = DMA3TRIG30 11111b = DMA3TRIG31
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-0	DMA2TSEL	RW	0h	DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA2TRIG0 00001b = DMA2TRIG1 00010b = DMA2TRIG2 ⋮ 11110b = DMA2TRIG30 11111b = DMA2TRIG31

7.3.3 DMACTL2 Register

DMA Control 2 Register

Figure 7-8. DMACTL2 Register

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)

Table 7-7. DMACTL2 Register Description

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12-8	DMA5TSEL	RW	0h	DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA5TRIG0 00001b = DMA5TRIG1 00010b = DMA5TRIG2 ⋮ 11110b = DMA5TRIG30 11111b = DMA5TRIG31
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-0	DMA4TSEL	RW	0h	DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA4TRIG0 00001b = DMA4TRIG1 00010b = DMA4TRIG2 ⋮ 11110b = DMA4TRIG30 11111b = DMA4TRIG31

7.3.4 DMACTL3 Register

DMA Control 3 Register

Figure 7-9. DMACTL3 Register

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)

Table 7-8. DMACTL3 Register Description

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12-8	DMA7TSEL	RW	0h	DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA7TRIG0 00001b = DMA7TRIG1 00010b = DMA7TRIG2 ⋮ 11110b = DMA7TRIG30 11111b = DMA7TRIG31
7-5	Reserved	R	0h	Reserved. Always reads as 0.
4-0	DMA6TSEL	RW	0h	DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet for number of channels and trigger assignment. 00000b = DMA6TRIG0 00001b = DMA6TRIG1 00010b = DMA6TRIG2 ⋮ 11110b = DMA6TRIG30 11111b = DMA6TRIG31

7.3.5 DMACTL4 Register

DMA Control 4 Register

Figure 7-10. DMACTL4 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					DMARMWDIS	ROUNDROBIN	ENNMI
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

Table 7-9. DMACTL4 Register Description

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Always reads as 0.
2	DMARMWDIS	RW	0h	Read-modify-write disable. When set, this bit inhibits any DMA transfers from occurring during CPU read-modify-write operations. 0b = DMA transfers can occur during read-modify-write CPU operations. 1b = DMA transfers inhibited during read-modify-write CPU operations
1	ROUNDROBIN	RW	0h	Round robin. This bit enables the round-robin DMA channel priorities. 0b = DMA channel priority is DMA0-DMA1-DMA2 - ..... -DMA7. 1b = DMA channel priority changes with each transfer.
0	ENNMI	RW	0h	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. 0b = NMI does not interrupt DMA transfer 1b = NMI interrupts a DMA transfer

7.3.6 DMAxCTL Register

DMA Channel x Control Register

Figure 7-11. DMAxCTL Register

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
DMADSTBYTE	DMASRCBYTE	DMALEVEL	DMAEN	DMAIFG	DMAIE	DMAABORT	DMAREQ
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 7-10. DMAxCTL Register Description

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14-12	DMADT	RW	0h	DMA transfer mode 000b = Single transfer 001b = Block transfer 010b = Burst-block transfer 011b = Burst-block transfer 100b = Repeated single transfer 101b = Repeated block transfer 110b = Repeated burst-block transfer 111b = Repeated burst-block transfer
11-10	DMADSTINCR	RW	0h	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 or decrements by one. When DMADSTBYTE = 0, the destination address increments or 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. 00b = Destination address is unchanged 01b = Destination address is unchanged 10b = Destination address is decremented 11b = Destination address is incremented
9-8	DMASRCINCR	RW	0h	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 or 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. 00b = Source address is unchanged 01b = Source address is unchanged 10b = Source address is decremented 11b = Source address is incremented
7	DMADSTBYTE	RW	0h	DMA destination byte. This bit selects the destination as a byte or word. 0b = Word 1b = Byte
6	DMASRCBYTE	RW	0h	DMA source byte. This bit selects the source as a byte or word. 0b = Word 1b = Byte
5	DMALEVEL	RW	0h	DMA level. This bit selects between edge-sensitive and level-sensitive triggers. 0b = Edge sensitive (rising edge) 1b = Level sensitive (high level)
4	DMAEN	RW	0h	DMA enable 0b = Disabled 1b = Enabled

Table 7-10. DMAxCTL Register Description (continued)

Bit	Field	Type	Reset	Description
3	DMAIFG	RW	0h	DMA interrupt flag 0b = No interrupt pending 1b = Interrupt pending
2	DMAIE	RW	0h	DMA interrupt enable 0b = Disabled 1b = Enabled
1	DMAABORT	RW	0h	DMA abort. This bit indicates if a DMA transfer was interrupt by an NMI. 0b = DMA transfer not interrupted 1b = DMA transfer interrupted by NMI
0	DMAREQ	RW	0h	DMA request. Software-controlled DMA start. DMAREQ is reset automatically. 0b = No DMA start 1b = Start DMA

7.3.7 DMAxSA Register

DMA Source Address Register

Figure 7-12. DMAxSA Register

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

Table 7-11. DMAxSA Register Description

Bit	Field	Type	Reset	Description
31-20	Reserved	R	0h	Reserved. Always reads as 0.
19-0	DMAxSA	RW	undefined	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 DMAxDA Register

DMA Destination Address Register

Figure 7-13. DMAxDA Register

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

Table 7-12. DMAxDA Register Description

Bit	Field	Type	Reset	Description
31-20	Reserved	R	0h	Reserved. Always reads as 0.
19-0	DMAxDA	RW	undefined	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 DMAxSZ Register

DMA Size Address Register

Figure 7-14. DMAxSZ Register

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

Table 7-13. DMAxSZ Register Description

Bit	Field	Type	Reset	Description
15-0	DMAxSZ	RW	undefined	DMA size. The DMA size register defines the number of byte or 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. 0000h = Transfer is disabled. 0001h = One byte or word is transferred. 0002h = Two bytes or words are transferred. ⋮ FFFFh = 65535 bytes or words are transferred.

7.3.10 DMAIV Register

DMA Interrupt Vector Register

Figure 7-15. DMAIV Register

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

Table 7-14. DMAIV Register Description

Bit	Field	Type	Reset	Description
15-0	DMAIV	R	0h	DMA interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: DMA channel 0; Interrupt Flag: DMA0IFG; Interrupt Priority: Highest 04h = Interrupt Source: DMA channel 1; Interrupt Flag: DMA1IFG 06h = Interrupt Source: DMA channel 2; Interrupt Flag: DMA2IFG 08h = Interrupt Source: DMA channel 3; Interrupt Flag: DMA3IFG 0Ah = Interrupt Source: DMA channel 4; Interrupt Flag: DMA4IFG 0Ch = Interrupt Source: DMA channel 5; Interrupt Flag: DMA5IFG 0Eh = Interrupt Source: DMA channel 6; Interrupt Flag: DMA6IFG 10h = Interrupt Source: DMA channel 7; Interrupt Flag: DMA7IFG; Interrupt Priority: 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 306
8.2 Digital I/O Operation 307
8.3 I/O Configuration 310
8.4 Digital I/O Registers 313

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.

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). 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 by word formats. Port pairs P1 and P2, P3 and P4, P5 and P6, P7 and P8, and so on, are associated with the names PA, PB, PC, PD, and so on, respectively. All port registers are handled in this manner with this naming convention except for the interrupt vector registers, P1IV and P2IV; that is, 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 port PA using byte operations, the upper byte remains unchanged. Similarly, writing to the upper byte of port PA 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 don't care. Ports PB, PC, PD, PE, and PF behave similarly.

Reading port PA using word operations causes all 16 bits to be transferred to the destination. Reading the lower or upper byte of port PA (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 port PA and storing to a general-purpose register using byte operations writes the byte that is transferred 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 fewer 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 or 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 or Pulldown Resistor Enable Registers (PxREN)

Each bit in each PxREN register enables or disables the pullup or 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 or pulldown resistor disabled
- Bit = 1: Pullup or pulldown resistor enabled

Table 8-1 summarizes the use 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 Function Select Registers (PxSEL0, PxSEL1)

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each port pin uses two bits to select the pin function – I/O port or one of the three possible peripheral module function. Table 8-2 shows how to select the various 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.

Table 8-2. I/O Function Selection

PxSEL1	PxSEL0	I/O Function
0	0	General purpose I/O is selected
0	1	Primary module function is selected
1	0	Secondary module function is selected
1	1	Tertiary module function is selected

Setting the PxSEL1 or PxSEL0 bits to a module function 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.

When a port pin is selected as an input to peripheral modules, the input signal to those peripheral modules is a latched representation of the signal at the device pin. While PxSEL1 and PxSEL0 is other than 00, the internal input signal follows the signal at the pin for all connected modules. However, if PxSEL1 and PxSEL0 = 00, the input to the peripherals maintain the value of the input signal at the device pin before the PxSEL1 and PxSEL0 bits were reset.

Because the PxSEL1 and PxSEL0 bits do not reside in contiguous addresses, changing both bits at the same time is not possible. For example, an application might need to change P1.0 from general purpose I/O to the tertiary module function residing on P1.0. Initially, P1SEL1 = 00h and P1SEL0 = 00h. To change the function, it would be necessary to write both P1SEL1 = 01h and P1SEL0 = 01h. This is not possible without first passing through an intermediate configuration, and this configuration may not be desirable from an application standpoint. The PxSELC complement register can be used to handle such situations. The PxSELC register always reads 0. Each set bit of the PxSELC register complements the corresponding respective bit of the PxSEL1 and PxSEL0 registers. In the example, with P1SEL1 = 00h and P1SEL0 = 00h initially, writing P1SELC = 01h causes P1SEL1 = 01h and P1SEL0 = 01h to be written simultaneously.

NOTE: Interrupts are disabled when PxSEL1 = 1 or PxSEL0 = 1

When any PxSEL bit is set, the corresponding pin interrupt function is disabled. Therefore, signals on these pins do not generate interrupts, regardless of the state of the corresponding PxIE bit.

8.2.6 Port Interrupts

At least each pin in ports P1 and P2 have interrupt capability, configured with the PxIFG, PxIE, and PxIES registers. Some devices may contain additional port interrupts besides P1 and P2. See the device-specific data sheet to determine which port interrupts are available.

All Px interrupt flags are prioritized, with PxIFG.0 being the highest, and combined to source a single interrupt vector. The highest priority enabled interrupt generates a number in the PxIV register. This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Px interrupts do not affect the PxIV value. The PxIV registers are word or byte access.

Each PxIFG bit is the interrupt flag for its corresponding I/O pin, and the flag 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 PxOUT, PxDIR, or PxREN can result in setting the corresponding PxIFG flags.

Any access (read or write) of the lower byte of the PxIV register, either word or byte access, 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.

8.2.6.1 P1IV 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 code to handle any other PxIV register is similar.

The numbers at the right margin show the number of CPU cycles that are required 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
```

8.2.6.2 Interrupt Edge Select Registers (PxIES)

Each PxIES bit selects the interrupt edge for the corresponding I/O pin.

- Bit = 0: Respective PxIFG flag is set on a low-to-high transition
- Bit = 1: Respective PxIFG flag is set on 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 → 1	0	May be set
0 → 1	1	Unchanged
1 → 0	0	Unchanged
1 → 0	1	May be set

8.2.6.3 Interrupt Enable Registers (PxIE)

Each PxIE bit enables the associated PxIFG interrupt flag.

- Bit = 0: The interrupt is disabled
- Bit = 1: The interrupt is enabled

8.3 I/O Configuration

8.3.1 Configuration After Reset

After a BOR reset, all port pins are high-impedance with Schmitt triggers and their module functions disabled to prevent any cross currents. The application must initialize all port pins including unused ones (Section 8.3.2) as input high impedance, input with pulldown, input with pullup, output high, or output low according to the application needs by configuring PxDIR, PxREN, PxOUT, and PxIES accordingly. This initialization takes effect as soon as the LOCKLPM5 bit in the PM5CTL register (described in the PMM chapter) is cleared; until then, the I/Os remain in their high-impedance state with Schmitt trigger inputs disabled. Note that this is usually the same I/O initialization that is required after a wake-up from LPMx.5. After clearing LOCKLPM5 all interrupt flags should be cleared (note, this is different to the wake-up from LPMx.5 flow). Then port interrupts can be enabled by setting the corresponding PxIE bits.

After a POR or PUC reset all port pins are configured as inputs with their module function being disabled. Also here to prevent floating inputs all port pins including unused ones (Section 8.3.2) should be configured according to the application needs as early as possible during the initialization procedure.

Note, the same I/O initialization procedure can be used for all reset cases and wake-up from LPMx.5 - except for PxIFG:

1. Initialize Ports: PxDIR, PxREN, PxOUT, and PxIES
2. Clear LOCKLPM5
3. If not wake-up from LPMx.5: clear all PxIFGs to avoid erroneous port interrupts
4. Enable port interrupts in PxIE

8.3.2 Configuration of Unused Port Pins

To prevent a floating input and to reduce power consumption, unused I/O pins should be configured as I/O function, output direction, and left unconnected on the PC board. The value of the PxOUT bit is don't care, because the pin is unconnected. Alternatively, the integrated pullup or pulldown resistor can be enabled by setting the PxREN bit of the unused pin to prevent a floating input. See the *System Resets, Interrupts, and Operating Modes, System Control Module (SYS)* chapter for termination of unused pins.

NOTE: Configuring port PJ and shared JTAG pins:

The application should make sure 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.

8.3.3 Configuration for LPMx.5 Low-Power Modes

NOTE: See , *Entering and Exiting Low-Power Modes LPMx.5*, in the *System Resets, Interrupts, and Operating Modes, System Control Module (SYS)* chapter for details about LPMx.5 low-power modes.

See the device-specific data sheet to determine which LPMx.5 low-power modes are available and which modules can operate in LPM3.5, if any.

With regard to the digital I/O, the following description is applicable to both LPM3.5 and LPM4.5.

Upon entering LPMx.5 (LPM3.5 or LPM4.5) the LDO of the PMM module is disabled, which removes the supply voltage from the core of the device. This causes all I/O register configurations to be lost, thus 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 achieve the lowest possible power consumption in LPMx.5, and to prevent an uncontrolled input or output I/O state in the application. The application has complete control of the I/O pin conditions that are necessary to prevent unwanted spurious activity upon entry and exit from LPMx.5.

Before entering LPMx.5 the following operations are required for the I/Os:

(a) Set all I/Os to general-purpose I/Os (PxSEL0 = 000h and PxSEL1 = 000h) and configure as needed. Each I/O can be set to input high impedance, input with pulldown, input with pullup, output high, or output low. 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.

(b) 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 and wakeup capability. Not all inputs with interrupt capability offer wakeup from LPMx.5. See the device-specific data sheet for availability. To wake up the device, a port pin must 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. Last, the PxIE for the port must be enabled, as well as the general interrupt enable.

NOTE: It is not possible to wakeup from a port interrupt if its respective port interrupt flag is already asserted. It is recommended that the flag be cleared prior to entering LPMx.5. It is also recommended that GIE = 1 be set prior to entry into LPMx.5. Any pending flags in this case could then be serviced prior to LPMx.5 entry.

This completes the operations required for the I/Os prior to entering LPMx.5.

During LPMx.5 the I/O pin states are held and locked based on the settings prior to LPMx.5 entry. Note that only the pin conditions are retained. All other port configuration register settings such as PxDIR, PxREN, PxOUT, PxIES, and PxIE contents are lost.

Upon exit from LPMx.5, all peripheral registers are set to their default conditions but the I/O pins remain locked while LOCKLPM5 remains set. Keeping the I/O pins locked ensures that all pin conditions remain stable when entering the active mode, regardless of the default I/O register settings.

When back in active mode, the I/O configuration and I/O interrupt configuration such as PxDIR, PxREN, PxOUT, and PxIES should be restored to the values prior to entering LPMx.5. 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.

After enabling the I/O interrupts by configuring PxIE, the I/O interrupt that caused the wakeup can be serviced as indicated by the PxIFG flags. These flags can be used directly, or the corresponding PxIV register may be used. 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 are set, and it cannot be determined which port caused the I/O wakeup.

8.4 Digital I/O Registers

The digital I/O registers are listed in Table 8-3. 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 8-3.

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 8-3. Digital I/O Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
0Eh	P1IV	Port 1 Interrupt Vector	Read only	Word	0000h	Section 8.4.1
0Eh	P1IV_L		Read only	Byte	00h	
0Fh	P1IV_H		Read only	Byte	00h	
1Eh	P2IV	Port 2 Interrupt Vector	Read only	Word	0000h	Section 8.4.2
1Eh	P2IV_L		Read only	Byte	00h	
1Fh	P2IV_H		Read only	Byte	00h	
2Eh	P3IV	Port 3 Interrupt Vector	Read only	Word	0000h	Section 8.4.3
2Eh	P3IV_L		Read only	Byte	00h	
2Fh	P3IV_H		Read only	Byte	00h	
3Eh	P4IV	Port 4 Interrupt Vector	Read only	Word	0000h	Section 8.4.4
3Eh	P4IV_L		Read only	Byte	00h	
3Fh	P4IV_H		Read only	Byte	00h	
00h	P1IN or PAIN_L	Port 1 Input	Read only	Byte	undefined	Section 8.4.5
02h	P1OUT or PAOUT_L	Port 1 Output	Read/write	Byte	undefined	Section 8.4.6
04h	P1DIR or PADIR_L	Port 1 Direction	Read/write	Byte	00h	Section 8.4.7
06h	P1REN or PAREN_L	Port 1 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Ah	P1SEL0 or PASEL0_L	Port 1 Select 0	Read/write	Byte	00h	Section 8.4.9
0Ch	P1SEL1 or PASEL1_L	Port 1 Select 1	Read/write	Byte	00h	Section 8.4.10
16h	P1SELC or PASELC_L	Port 1 Complement Selection	Read/write	Byte	00h	Section 8.4.11
18h	P1IES or PAIES_L	Port 1 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Ah	P1IE or PAIE_L	Port 1 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Ch	P1IFG or PAIFG_L	Port 1 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
01h	P2IN or PAIN_H	Port 2 Input	Read only	Byte	undefined	Section 8.4.5
03h	P2OUT or PAOUT_H	Port 2 Output	Read/write	Byte	undefined	Section 8.4.6
05h	P2DIR or PADIR_H	Port 2 Direction	Read/write	Byte	00h	Section 8.4.7
07h	P2REN or PAREN_H	Port 2 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Bh	P2SEL0 or PASEL0_H	Port 2 Select 0	Read/write	Byte	00h	Section 8.4.9
0Dh	P2SEL1 or PASEL1_H	Port 2 Select 1	Read/write	Byte	00h	Section 8.4.10
17h	P2SELC or PASELC_L	Port 2 Complement Selection	Read/write	Byte	00h	Section 8.4.11
19h	P2IES or PAIES_H	Port 2 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Bh	P2IE or PAIE_H	Port 2 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Dh	P2IFG or PAIFG_H	Port 2 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14
00h	P3IN or PBIN_L	Port 3 Input	Read only	Byte	undefined	Section 8.4.5
02h	P3OUT or PBOUT_L	Port 3 Output	Read/write	Byte	undefined	Section 8.4.6
04h	P3DIR or PBDIR_L	Port 3 Direction	Read/write	Byte	00h	Section 8.4.7
06h	P3REN or PBREN_L	Port 3 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Ah	P3SEL0 or PBSEL0_L	Port 3 Select 0	Read/write	Byte	00h	Section 8.4.9
0Ch	P3SEL1 or PBSEL1_L	Port 3 Select 1	Read/write	Byte	00h	Section 8.4.10
16h	P3SELC or PBSELC_L	Port 3 Complement Selection	Read/write	Byte	00h	Section 8.4.11
18h	P3IES or PBIES_L	Port 3 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Ah	P3IE or PBIE_L	Port 3 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Ch	P3IFG or PBIFG_L	Port 3 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
01h	P4IN or PBIN_H	Port 4 Input	Read only	Byte	undefined	Section 8.4.5
03h	P4OUT or PBOUT_H	Port 4 Output	Read/write	Byte	undefined	Section 8.4.6
05h	P4DIR or PBDIR_H	Port 4 Direction	Read/write	Byte	00h	Section 8.4.7
07h	P4REN or PBREN_H	Port 4 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Bh	P4SEL0 or PBSEL0_H	Port 4 Select 0	Read/write	Byte	00h	Section 8.4.9
0Dh	P4SEL1 or PBSEL1_H	Port 4 Select 1	Read/write	Byte	00h	Section 8.4.10
17h	P4SELC or PBSELC_L	Port 4 Complement Selection	Read/write	Byte	00h	Section 8.4.11
19h	P4IES or PBIES_H	Port 4 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Bh	P4IE or PBIE_H	Port 4 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Dh	P4IFG or PBIFG_H	Port 4 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14
00h	P5IN or PCIN_L	Port 5 Input	Read only	Byte	undefined	Section 8.4.5
02h	P5OUT or PCOUT_L	Port 5 Output	Read/write	Byte	undefined	Section 8.4.6
04h	P5DIR or PCDIR_L	Port 5 Direction	Read/write	Byte	00h	Section 8.4.7
06h	P5REN or PCREN_L	Port 5 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Ah	P5SEL0 or PCSEL0_L	Port 5 Select 0	Read/write	Byte	00h	Section 8.4.9
0Ch	P5SEL1 or PCSEL1_L	Port 5 Select 1	Read/write	Byte	00h	Section 8.4.10
16h	P5SELC or PCSELC_L	Port 5 Complement Selection	Read/write	Byte	00h	Section 8.4.11
18h	P5IES or PCIES_L	Port 5 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Ah	P5IE or PCIE_L	Port 5 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Ch	P5IFG or PCIFG_L	Port 5 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
01h	P6IN or PCIN_H	Port 6 Input	Read only	Byte	undefined	Section 8.4.5
03h	P6OUT or PCOUT_H	Port 6 Output	Read/write	Byte	undefined	Section 8.4.6
05h	P6DIR or PCDIR_H	Port 6 Direction	Read/write	Byte	00h	Section 8.4.7
07h	P6REN or PCREN_H	Port 6 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Bh	P6SEL0 or PCSEL0_H	Port 6 Select 0	Read/write	Byte	00h	Section 8.4.9
0Dh	P6SEL1 or PCSEL1_H	Port 6 Select 1	Read/write	Byte	00h	Section 8.4.10
17h	P6SELC or PCSELC_L	Port 6 Complement Selection	Read/write	Byte	00h	Section 8.4.11
19h	P6IES or PCIES_H	Port 6 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Bh	P6IE or PCIE_H	Port 6 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Dh	P6IFG or PCIFG_H	Port 6 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14
00h	P7IN or PDIN_L	Port 7 Input	Read only	Byte	undefined	Section 8.4.5
02h	P7OUT or PDOUT_L	Port 7 Output	Read/write	Byte	undefined	Section 8.4.6
04h	P7DIR or PDDIR_L	Port 7 Direction	Read/write	Byte	00h	Section 8.4.7
06h	P7REN or PDREN_L	Port 7 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Ah	P7SEL0 or PDSEL0_L	Port 7 Select 0	Read/write	Byte	00h	Section 8.4.9
0Ch	P7SEL1 or PDSEL1_L	Port 7 Select 1	Read/write	Byte	00h	Section 8.4.10
16h	P7SELC or PDSELC_L	Port 7 Complement Selection	Read/write	Byte	00h	Section 8.4.11
18h	P7IES or PDIES_L	Port 7 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Ah	P7IE or PDIE_L	Port 7 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Ch	P7IFG or PDIFG_L	Port 7 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
01h	P8IN or PDIN_H	Port 8 Input	Read only	Byte	undefined	Section 8.4.5
03h	P8OUT or PDOUT_H	Port 8 Output	Read/write	Byte	undefined	Section 8.4.6
05h	P8DIR or PDDIR_H	Port 8 Direction	Read/write	Byte	00h	Section 8.4.7
07h	P8REN or PDREN_H	Port 8 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Bh	P8SEL0 or PDSEL0_H	Port 8 Select 0	Read/write	Byte	00h	Section 8.4.9
0Dh	P8SEL1 or PDSEL1_H	Port 8 Select 1	Read/write	Byte	00h	Section 8.4.10
17h	P8SELC or PDSELC_L	Port 8 Complement Selection	Read/write	Byte	00h	Section 8.4.11
19h	P8IES or PDIES_H	Port 8 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Bh	P8IE or PDIE_H	Port 8 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Dh	P8IFG or PDIFG_H	Port 8 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14
00h	P9IN or PEIN_L	Port 9 Input	Read only	Byte	undefined	Section 8.4.5
02h	P9OUT or PEOUT_L	Port 9 Output	Read/write	Byte	undefined	Section 8.4.6
04h	P9DIR or PEDIR_L	Port 9 Direction	Read/write	Byte	00h	Section 8.4.7
06h	P9REN or PEREN_L	Port 9 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Ah	P9SEL0 or PESEL0_L	Port 9 Select 0	Read/write	Byte	00h	Section 8.4.9
0Ch	P9SEL1 or PESEL1_L	Port 9 Select 1	Read/write	Byte	00h	Section 8.4.10
16h	P9SELC or PESELC_L	Port 9 Complement Selection	Read/write	Byte	00h	Section 8.4.11
18h	P9IES or PEIES_L	Port 9 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Ah	P9IE or PEIE_L	Port 9 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Ch	P9IFG or PEIFG_L	Port 9 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
01h	P10IN or PEIN_H	Port 10 Input	Read only	Byte	undefined	Section 8.4.5
03h	P10OUT or PEOUT_H	Port 10 Output	Read/write	Byte	undefined	Section 8.4.6
05h	P10DIR or PEDIR_H	Port 10 Direction	Read/write	Byte	00h	Section 8.4.7
07h	P10REN or PEREN_H	Port 10 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Bh	P10SEL0 or PESEL0_H	Port 10 Select 0	Read/write	Byte	00h	Section 8.4.9
0Dh	P10SEL1 or PESEL1_H	Port 10 Select 1	Read/write	Byte	00h	Section 8.4.10
17h	P10SELC or PESELC_L	Port 10 Complement Selection	Read/write	Byte	00h	Section 8.4.11
19h	P10IES or PEIES_H	Port 10 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Bh	P10IE or PEIE_H	Port 10 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Dh	P10IFG or PEIFG_H	Port 10 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14
00h	P11IN or PFIN_L	Port 11 Input	Read only	Byte	undefined	Section 8.4.5
02h	P11OUT or PFOUT_L	Port 11 Output	Read/write	Byte	undefined	Section 8.4.6
04h	P11DIR or PFDIR_L	Port 11 Direction	Read/write	Byte	00h	Section 8.4.7
06h	P11REN or PFREN_L	Port 11 Resistor Enable	Read/write	Byte	00h	Section 8.4.8
0Ah	P11SEL0 or PFSEL0_L	Port 11 Select 0	Read/write	Byte	00h	Section 8.4.9
0Ch	P11SEL1 or PFSEL1_L	Port 11 Select 1	Read/write	Byte	00h	Section 8.4.10
16h	P11SELC or PFSELC_L	Port 11 Complement Selection	Read/write	Byte	00h	Section 8.4.11
18h	P11IES or PFIES_L	Port 11 Interrupt Edge Select	Read/write	Byte	undefined	Section 8.4.12
1Ah	P11IE or PFIE_L	Port 11 Interrupt Enable	Read/write	Byte	00h	Section 8.4.13
1Ch	P11IFG or PFIFG_L	Port 11 Interrupt Flag	Read/write	Byte	00h	Section 8.4.14

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	PAIN	Port A Input	Read only	Word	undefined	
00h	PAIN_L		Read only	Byte	undefined	
01h	PAIN_H		Read only	Byte	undefined	
02h	PAOUT	Port A Output	Read/write	Word	undefined	
02h	PAOUT_L		Read/write	Byte	undefined	
03h	PAOUT_H		Read/write	Byte	undefined	
04h	PADIR	Port A Direction	Read/write	Word	0000h	
04h	PADIR_L		Read/write	Byte	00h	
05h	PADIR_H		Read/write	Byte	00h	
06h	PAREN	Port A Resistor Enable	Read/write	Word	0000h	
06h	PAREN_L		Read/write	Byte	00h	
07h	PAREN_H		Read/write	Byte	00h	
0Ah	PASEL0	Port A Select 0	Read/write	Word	0000h	
0Ah	PASEL0_L		Read/write	Byte	00h	
0Bh	PASEL0_H		Read/write	Byte	00h	
0Ch	PASEL1	Port A Select 1	Read/write	Word	0000h	
0Ch	PASEL1_L		Read/write	Byte	00h	
0Dh	PASEL1_H		Read/write	Byte	00h	
16h	PASELC	Port A Complement Select	Read/write	Word	0000h	
16h	PASELC_L		Read/write	Byte	00h	
17h	PASELC_H		Read/write	Byte	00h	
18h	PAIES	Port A Interrupt Edge Select	Read/write	Word	undefined	
18h	PAIES_L		Read/write	Byte	undefined	
19h	PAIES_H		Read/write	Byte	undefined	
1Ah	PAIE	Port A Interrupt Enable	Read/write	Word	0000h	
1Ah	PAIE_L		Read/write	Byte	00h	
1Bh	PAIE_H		Read/write	Byte	00h	
1Ch	PAIFG	Port A Interrupt Flag	Read/write	Word	0000h	
1Ch	PAIFG_L		Read/write	Byte	00h	
1Dh	PAIFG_H		Read/write	Byte	00h	

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	PBIN	Port B Input	Read only	Word	undefined	
00h	PBIN_L		Read only	Byte	undefined	
01h	PBIN_H		Read only	Byte	undefined	
02h	PBOUT	Port B Output	Read/write	Word	undefined	
02h	PBOUT_L		Read/write	Byte	undefined	
03h	PBOUT_H		Read/write	Byte	undefined	
04h	PBDIR	Port B Direction	Read/write	Word	0000h	
04h	PBDIR_L		Read/write	Byte	00h	
05h	PBDIR_H		Read/write	Byte	00h	
06h	PBREN	Port B Resistor Enable	Read/write	Word	0000h	
06h	PBREN_L		Read/write	Byte	00h	
07h	PBREN_H		Read/write	Byte	00h	
0Ah	PBSEL0	Port B Select 0	Read/write	Word	0000h	
0Ah	PBSEL0_L		Read/write	Byte	00h	
0Bh	PBSEL0_H		Read/write	Byte	00h	
0Ch	PBSEL1	Port B Select 1	Read/write	Word	0000h	
0Ch	PBSEL1_L		Read/write	Byte	00h	
0Dh	PBSEL1_H		Read/write	Byte	00h	
16h	PBSELC	Port B Complement Select	Read/write	Word	0000h	
16h	PBSELC_L		Read/write	Byte	00h	
17h	PBSELC_H		Read/write	Byte	00h	
18h	PBIES	Port B Interrupt Edge Select	Read/write	Word	undefined	
18h	PBIES_L		Read/write	Byte	undefined	
19h	PBIES_H		Read/write	Byte	undefined	
1Ah	PBIE	Port B Interrupt Enable	Read/write	Word	0000h	
1Ah	PBIE_L		Read/write	Byte	00h	
1Bh	PBIE_H		Read/write	Byte	00h	
1Ch	PBIFG	Port B Interrupt Flag	Read/write	Word	0000h	
1Ch	PBIFG_L		Read/write	Byte	00h	
1Dh	PBIFG_H		Read/write	Byte	00h	

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	PCIN	Port C Input	Read only	Word	undefined	
00h	PCIN_L		Read only	Byte	undefined	
01h	PCIN_H		Read only	Byte	undefined	
02h	PCOUT	Port C Output	Read/write	Word	undefined	
02h	PCOUT_L		Read/write	Byte	undefined	
03h	PCOUT_H		Read/write	Byte	undefined	
04h	PCDIR	Port C Direction	Read/write	Word	0000h	
04h	PCDIR_L		Read/write	Byte	00h	
05h	PCDIR_H		Read/write	Byte	00h	
06h	PCREN	Port C Resistor Enable	Read/write	Word	0000h	
06h	PCREN_L		Read/write	Byte	00h	
07h	PCREN_H		Read/write	Byte	00h	
0Ah	PCSEL0	Port C Select 0	Read/write	Word	0000h	
0Ah	PCSEL0_L		Read/write	Byte	00h	
0Bh	PCSEL0_H		Read/write	Byte	00h	
0Ch	PCSEL1	Port C Select 1	Read/write	Word	0000h	
0Ch	PCSEL1_L		Read/write	Byte	00h	
0Dh	PCSEL1_H		Read/write	Byte	00h	
16h	PCSELC	Port C Complement Select	Read/write	Word	0000h	
16h	PCSELC_L		Read/write	Byte	00h	
17h	PCSELC_H		Read/write	Byte	00h	
18h	PCIES	Port C Interrupt Edge Select	Read/write	Word	undefined	
18h	PCIES_L		Read/write	Byte	undefined	
19h	PCIES_H		Read/write	Byte	undefined	
1Ah	PCIE	Port C Interrupt Enable	Read/write	Word	0000h	
1Ah	PCIE_L		Read/write	Byte	00h	
1Bh	PCIE_H		Read/write	Byte	00h	
1Ch	PCIFG	Port C Interrupt Flag	Read/write	Word	0000h	
1Ch	PCIFG_L		Read/write	Byte	00h	
1Dh	PCIFG_H		Read/write	Byte	00h	

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	PDIN	Port D Input	Read only	Word	undefined	
00h	PDIN_L		Read only	Byte	undefined	
01h	PDIN_H		Read only	Byte	undefined	
02h	PDOUT	Port D Output	Read/write	Word	undefined	
02h	PDOUT_L		Read/write	Byte	undefined	
03h	PDOUT_H		Read/write	Byte	undefined	
04h	PDDIR	Port D Direction	Read/write	Word	0000h	
04h	PDDIR_L		Read/write	Byte	00h	
05h	PDDIR_H		Read/write	Byte	00h	
06h	PDREN	Port D Resistor Enable	Read/write	Word	0000h	
06h	PDREN_L		Read/write	Byte	00h	
07h	PDREN_H		Read/write	Byte	00h	
0Ah	PDSEL0	Port D Select 0	Read/write	Word	0000h	
0Ah	PDSEL0_L		Read/write	Byte	00h	
0Bh	PDSEL0_H		Read/write	Byte	00h	
0Ch	PDSEL1	Port D Select 1	Read/write	Word	0000h	
0Ch	PDSEL1_L		Read/write	Byte	00h	
0Dh	PDSEL1_H		Read/write	Byte	00h	
16h	PDSELC	Port D Complement Select	Read/write	Word	0000h	
16h	PDSELC_L		Read/write	Byte	00h	
17h	PDSELC_H		Read/write	Byte	00h	
18h	PDIES	Port D Interrupt Edge Select	Read/write	Word	undefined	
18h	PDIES_L		Read/write	Byte	undefined	
19h	PDIES_H		Read/write	Byte	undefined	
1Ah	PDIE	Port D Interrupt Enable	Read/write	Word	0000h	
1Ah	PDIE_L		Read/write	Byte	00h	
1Bh	PDIE_H		Read/write	Byte	00h	
1Ch	PDIFG	Port D Interrupt Flag	Read/write	Word	0000h	
1Ch	PDIFG_L		Read/write	Byte	00h	
1Dh	PDIFG_H		Read/write	Byte	00h	

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	PEIN	Port E Input	Read only	Word	undefined	
00h	PEIN_L		Read only	Byte	undefined	
01h	PEIN_H		Read only	Byte	undefined	
02h	PEOUT	Port E Output	Read/write	Word	undefined	
02h	PEOUT_L		Read/write	Byte	undefined	
03h	PEOUT_H		Read/write	Byte	undefined	
04h	PEDIR	Port E Direction	Read/write	Word	0000h	
04h	PEDIR_L		Read/write	Byte	00h	
05h	PEDIR_H		Read/write	Byte	00h	
06h	PEREN	Port E Resistor Enable	Read/write	Word	0000h	
06h	PEREN_L		Read/write	Byte	00h	
07h	PEREN_H		Read/write	Byte	00h	
0Ah	PESEL0	Port E Select 0	Read/write	Word	0000h	
0Ah	PESEL0_L		Read/write	Byte	00h	
0Bh	PESEL0_H		Read/write	Byte	00h	
0Ch	PESEL1	Port E Select 1	Read/write	Word	0000h	
0Ch	PESEL1_L		Read/write	Byte	00h	
0Dh	PESEL1_H		Read/write	Byte	00h	
16h	PESELC	Port E Complement Select	Read/write	Word	0000h	
16h	PESELC_L		Read/write	Byte	00h	
17h	PESELC_H		Read/write	Byte	00h	
18h	PEIES	Port E Interrupt Edge Select	Read/write	Word	undefined	
18h	PEIES_L		Read/write	Byte	undefined	
19h	PEIES_H		Read/write	Byte	undefined	
1Ah	PEIE	Port E Interrupt Enable	Read/write	Word	0000h	
1Ah	PEIE_L		Read/write	Byte	00h	
1Bh	PEIE_H		Read/write	Byte	00h	
1Ch	PEIFG	Port E Interrupt Flag	Read/write	Word	0000h	
1Ch	PEIFG_L		Read/write	Byte	00h	
1Dh	PEIFG_H		Read/write	Byte	00h	

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	PFIN	Port F Input	Read only	Word	undefined	
00h	PFIN_L		Read only	Byte	undefined	
01h	PFIN_H		Read only	Byte	undefined	
02h	PFOUT	Port F Output	Read/write	Word	undefined	
02h	PFOUT_L		Read/write	Byte	undefined	
03h	PFOUT_H		Read/write	Byte	undefined	
04h	PFDIR	Port F Direction	Read/write	Word	0000h	
04h	PFDIR_L		Read/write	Byte	00h	
05h	PFDIR_H		Read/write	Byte	00h	
06h	PFREN	Port F Resistor Enable	Read/write	Word	0000h	
06h	PFREN_L		Read/write	Byte	00h	
07h	PFREN_H		Read/write	Byte	00h	
0Ah	PFSEL0	Port F Select 0	Read/write	Word	0000h	
0Ah	PFSEL0_L		Read/write	Byte	00h	
0Bh	PFSEL0_H		Read/write	Byte	00h	
0Ch	PFSEL1	Port F Select 1	Read/write	Word	0000h	
0Ch	PFSEL1_L		Read/write	Byte	00h	
0Dh	PFSEL1_H		Read/write	Byte	00h	
16h	PFSELC	Port F Complement Select	Read/write	Word	0000h	
16h	PFSELC_L		Read/write	Byte	00h	
17h	PFSELC_H		Read/write	Byte	00h	
18h	PFIES	Port F Interrupt Edge Select	Read/write	Word	undefined	
18h	PFIES_L		Read/write	Byte	undefined	
19h	PFIES_H		Read/write	Byte	undefined	
1Ah	PFIE	Port F Interrupt Enable	Read/write	Word	0000h	
1Ah	PFIE_L		Read/write	Byte	00h	
1Bh	PFIE_H		Read/write	Byte	00h	
1Ch	PFIFG	Port F Interrupt Flag	Read/write	Word	0000h	
1Ch	PFIFG_L		Read/write	Byte	00h	
1Dh	PFIFG_H		Read/write	Byte	00h	

Table 8-3. Digital I/O Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	PJIN	Port J Input	Read only	Word	undefined	
00h	PJIN_L		Read only	Byte	undefined	
01h	PJIN_H		Read only	Byte	undefined	
02h	PJOUT	Port J Output	Read/write	Word	undefined	
02h	PJOUT_L		Read/write	Byte	undefined	
03h	PJOUT_H		Read/write	Byte	undefined	
04h	PJDIR	Port J Direction	Read/write	Word	0000h	
04h	PJDIR_L		Read/write	Byte	00h	
05h	PJDIR_H		Read/write	Byte	00h	
06h	PJREN	Port J Resistor Enable	Read/write	Word	0000h	
06h	PJREN_L		Read/write	Byte	00h	
07h	PJREN_H		Read/write	Byte	00h	
0Ah	PJSEL0	Port J Select 0	Read/write	Word	0000h	
0Ah	PJSEL0_L		Read/write	Byte	00h	
0Bh	PJSEL0_H		Read/write	Byte	00h	
0Ch	PJSEL1	Port J Select 1	Read/write	Word	0000h	
0Ch	PJSEL1_L		Read/write	Byte	00h	
0Dh	PJSEL1_H		Read/write	Byte	00h	
16h	PJSELC	Port J Complement Select	Read/write	Word	0000h	
16h	PJSELC_L		Read/write	Byte	00h	
17h	PJSELC_H		Read/write	Byte	00h	

8.4.1 P1IV Register

Port 1 Interrupt Vector Register

Figure 8-1. P1IV Register

15	14	13	12	11	10	9	8
P1IV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
P1IV							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

Table 8-4. P1IV Register Description

Bit	Field	Type	Reset	Description
15-0	P1IV	R	0h	Port 1 interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Port 1.0 interrupt; Interrupt Flag: P1IFG.0; Interrupt Priority: Highest 04h = Interrupt Source: Port 1.1 interrupt; Interrupt Flag: P1IFG.1 06h = Interrupt Source: Port 1.2 interrupt; Interrupt Flag: P1IFG.2 08h = Interrupt Source: Port 1.3 interrupt; Interrupt Flag: P1IFG.3 0Ah = Interrupt Source: Port 1.4 interrupt; Interrupt Flag: P1IFG.4 0Ch = Interrupt Source: Port 1.5 interrupt; Interrupt Flag: P1IFG.5 0Eh = Interrupt Source: Port 1.6 interrupt; Interrupt Flag: P1IFG.6 10b = Interrupt Source: Port 1.7 interrupt; Interrupt Flag: P1IFG.7; Interrupt Priority: Lowest

8.4.2 P2IV Register

Port 2 Interrupt Vector Register

Figure 8-2. P2IV Register

15	14	13	12	11	10	9	8
P2IV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
P2IV							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

Table 8-5. P2IV Register Description

Bit	Field	Type	Reset	Description
15-0	P2IV	R	0h	Port 2 interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Port 2.0 interrupt; Interrupt Flag: P2IFG.0; Interrupt Priority: Highest 04h = Interrupt Source: Port 2.1 interrupt; Interrupt Flag: P2IFG.1 06h = Interrupt Source: Port 2.2 interrupt; Interrupt Flag: P2IFG.2 08h = Interrupt Source: Port 2.3 interrupt; Interrupt Flag: P2IFG.3 0Ah = Interrupt Source: Port 2.4 interrupt; Interrupt Flag: P2IFG.4 0Ch = Interrupt Source: Port 2.5 interrupt; Interrupt Flag: P2IFG.5 0Eh = Interrupt Source: Port 2.6 interrupt; Interrupt Flag: P2IFG.6 10b = Interrupt Source: Port 2.7 interrupt; Interrupt Flag: P2IFG.7; Interrupt Priority: Lowest

8.4.3 P3IV Register

Port 3 Interrupt Vector Register

Figure 8-3. P3IV Register

15	14	13	12	11	10	9	8
P3IV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
P3IV							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

Table 8-6. P3IV Register Description

Bit	Field	Type	Reset	Description
15-0	P3IV	R	0h	Port 3 interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Port 3.0 interrupt; Interrupt Flag: P3IFG.0; Interrupt Priority: Highest 04h = Interrupt Source: Port 3.1 interrupt; Interrupt Flag: P3IFG.1 06h = Interrupt Source: Port 3.2 interrupt; Interrupt Flag: P3IFG.2 08h = Interrupt Source: Port 3.3 interrupt; Interrupt Flag: P3IFG.3 0Ah = Interrupt Source: Port 3.4 interrupt; Interrupt Flag: P3IFG.4 0Ch = Interrupt Source: Port 3.5 interrupt; Interrupt Flag: P3IFG.5 0Eh = Interrupt Source: Port 3.6 interrupt; Interrupt Flag: P3IFG.6 10b = Interrupt Source: Port 3.7 interrupt; Interrupt Flag: P3IFG.7; Interrupt Priority: Lowest

8.4.4 P4IV Register

Port 4 Interrupt Vector Register

Figure 8-4. P4IV Register

15	14	13	12	11	10	9	8
P4IV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
P4IV							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

Table 8-7. P4IV Register Description

Bit	Field	Type	Reset	Description
15-0	P4IV	R	0h	Port 4 interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Port 4.0 interrupt; Interrupt Flag: P4IFG.0; Interrupt Priority: Highest 04h = Interrupt Source: Port 4.1 interrupt; Interrupt Flag: P4IFG.1 06h = Interrupt Source: Port 4.2 interrupt; Interrupt Flag: P4IFG.2 08h = Interrupt Source: Port 4.3 interrupt; Interrupt Flag: P4IFG.3 0Ah = Interrupt Source: Port 4.4 interrupt; Interrupt Flag: P4IFG.4 0Ch = Interrupt Source: Port 4.5 interrupt; Interrupt Flag: P4IFG.5 0Eh = Interrupt Source: Port 4.6 interrupt; Interrupt Flag: P4IFG.6 10b = Interrupt Source: Port 4.7 interrupt; Interrupt Flag: P4IFG.7; Interrupt Priority: Lowest

8.4.5 PxIN Register

Port x Input Register

Figure 8-5. PxIN Register

7	6	5	4	3	2	1	0
PxIN							
r	r	r	r	r	r	r	r

Table 8-8. PxIN Register Description

Bit	Field	Type	Reset	Description
7-0	PxIN	R	Undefined	Port x input 0b = Input is low 1b = Input is high

8.4.6 PxOUT Register

Port x Output Register

Figure 8-6. PxOUT Register

7	6	5	4	3	2	1	0
PxOUT							
rw	rw	rw	rw	rw	rw	rw	rw

Table 8-9. PxOUT Register Description

Bit	Field	Type	Reset	Description
7-0	PxOUT	RW	Undefined	Port x output When I/O configured to output mode: 0b = Output is low. 1b = Output is high. When I/O configured to input mode and pullups/pulldowns enabled: 0b = Pulldown selected 1b = Pullup selected

8.4.7 PxDIR Register

Port x Direction Register

Figure 8-7. PxDIR Register

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

Table 8-10. P1DIR Register Description

Bit	Field	Type	Reset	Description
7-0	PxDIR	RW	0h	Port x direction 0b = Port configured as input 1b = Port configured as output

8.4.8 PxREN Register

Port x Pullup or Pulldown Resistor Enable Register

Figure 8-8. PxREN Register

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

Table 8-11. PxREN Register Description

Bit	Field	Type	Reset	Description
7-0	PxREN	RW	0h	Port x pullup or pulldown resistor enable. When the port is configured as an input, setting this bit enables or disables the pullup or pulldown. 0b = Pullup or pulldown disabled 1b = Pullup or pulldown enabled

8.4.9 PxSEL0 Register

Port x Function Selection Register 0

Figure 8-9. PxSEL0 Register

7	6	5	4	3	2	1	0
PxSEL0							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 8-12. PxSEL0 Register Description

Bit	Field	Type	Reset	Description
7-0	PxSEL0	RW	0h	Port function selection. Each bit corresponds to one channel on Port x. The values of each bit position in PxSEL1 and PxSEL0 are combined to specify the function. For example, if P1SEL1.5 = 1 and P1SEL0.5 = 0, then the secondary module function is selected for P1.5. See PxSEL1 for the definition of each value.

8.4.10 PxSEL1 Register

Port x Function Selection Register 1

Figure 8-10. PxSEL1 Register

7	6	5	4	3	2	1	0
PxSEL1							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 8-13. PxSEL1 Register Description

Bit	Field	Type	Reset	Description
7-0	PxSEL1	RW	0h	Port function selection. Each bit corresponds to one channel on Port x. The values of each bit position in PxSEL1 and PxSEL0 are combined to specify the function. For example, if P1SEL1.5 = 1 and P1SEL0.5 = 0, then the secondary module function is selected for P1.5. 00b = General-purpose I/O is selected 01b = Primary module function is selected 10b = Secondary module function is selected 11b = Tertiary module function is selected

8.4.11 PxSELC Register

Port x Complement Selection

Figure 8-11. PxSELC Register

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

Table 8-14. PxSELC Register Description

Bit	Field	Type	Reset	Description
7-0	PxSELC	RW	0h	Port selection complement. Each bit that is set in PxSELC complements the corresponding respective bit of both the PxSEL1 and PxSEL0 registers; that is, for each bit set in PxSELC, the corresponding bits in both PxSEL1 and PxSEL0 are both changed at the same time. Always reads as 0.

8.4.12 PxIES Register

Port x Interrupt Edge Select Register

Figure 8-12. PxIES Register

7	6	5	4	3	2	1	0
PxIES							
rw	rw	rw	rw	rw	rw	rw	rw

Table 8-15. PxIES Register Description

Bit	Field	Type	Reset	Description
7-0	PxIES	RW	Undefined	Port x interrupt edge select 0b = PxIFG flag is set with a low-to-high transition 1b = PxIFG flag is set with a high-to-low transition

8.4.13 PxIE Register

Port x Interrupt Enable Register

Figure 8-13. PxIE Register

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

Table 8-16. PxIE Register Description

Bit	Field	Type	Reset	Description
7-0	PxIE	RW	0h	Port x interrupt enable 0b = Corresponding port interrupt disabled 1b = Corresponding port interrupt enabled

8.4.14 PxIFG Register

Port x Interrupt Flag Register

Figure 8-14. PxIFG Register

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

Table 8-17. PxIFG Register Description

Bit	Field	Type	Reset	Description
7-0	PxIFG	RW	0h	Port x interrupt flag 0b = No interrupt is pending. 1b = Interrupt is pending.

Capacitive Touch IO

This chapter describes the functionality of the Capacitive Touch IOs and related control.

Topic **Page**

9.1 Capacitive Touch IO Introduction 333
9.2 Capacitive Touch IO Operation 334
9.3 CapTouch Registers 335

9.1 Capacitive Touch IO Introduction

The Capacitive Touch IO module allows implementation of a simple capacitive touch sense application. The module uses the integrated pullup and pulldown resistors and an external capacitor to form an oscillator by feeding back the inverted input voltage sensed by the input Schmitt triggers to the pullup and pulldown control. Figure 9-1 shows the capacitive touch IO principle

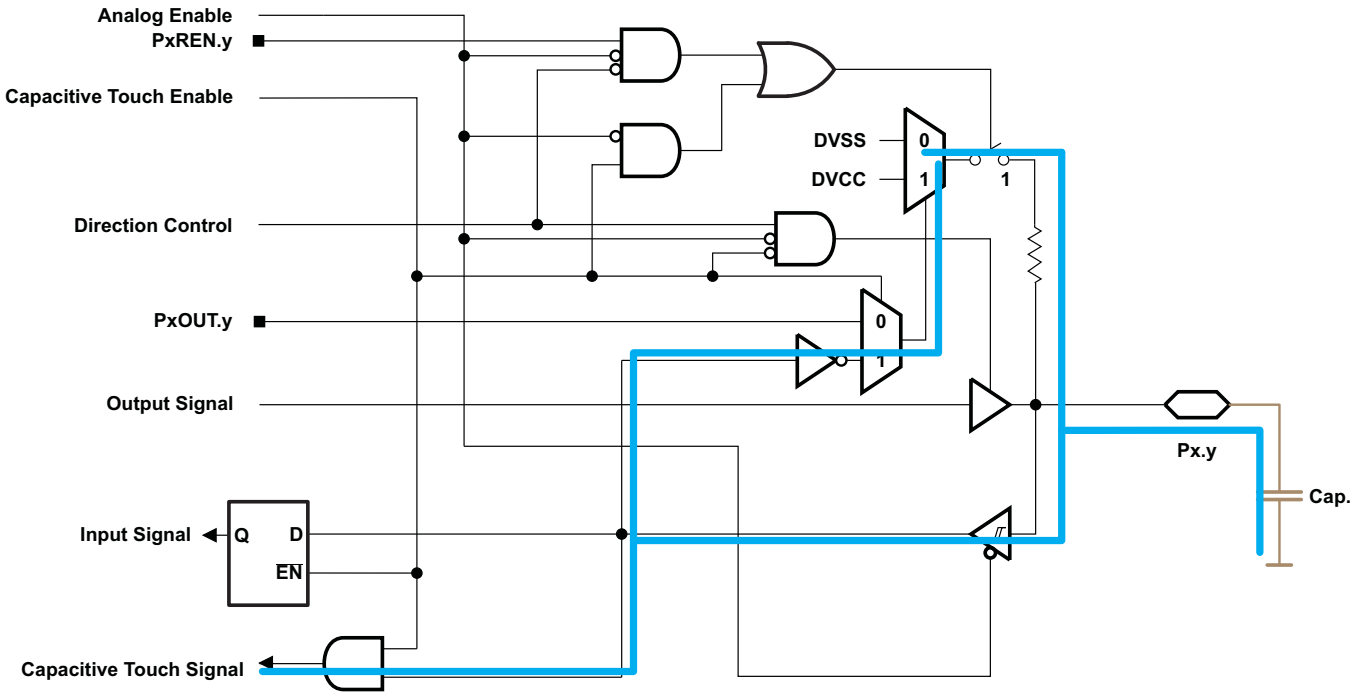


Figure 9-1. Capacitive Touch IO Principle

Figure 9-2 shows the block diagram of the Capacitive Touch IO module.

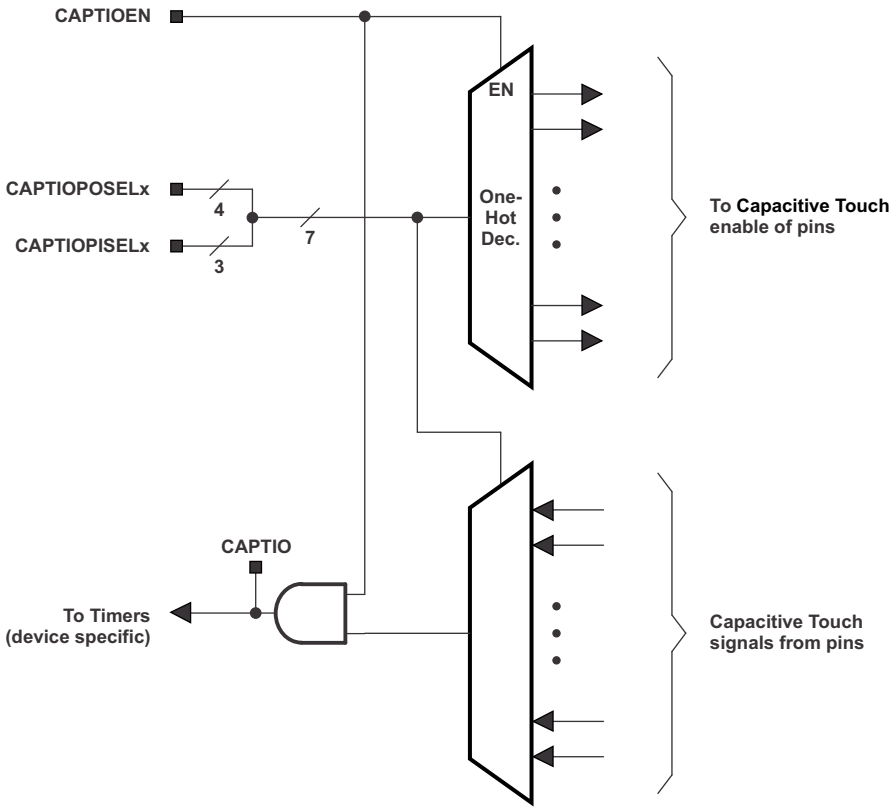


Figure 9-2. Capacitive Touch IO Block Diagram

9.2 Capacitive Touch IO Operation

Enable the Capacitive Touch IO functionality with CAPTIOEN = 1 and select a port pin using CAPTIOPOSELx and CAPTIOPISELx. The selected port pin is switched into the Capacitive Touch state, and the resulting oscillating signal is provided to be measured by a timer. The connected timers are device-specific (see the device-specific data sheet).

It is possible to scan to successive port pins by incrementing the low byte of the Capacitive Touch IO control register CAPTIOCTL_L by 2.

9.3 CapTouch Registers

The Capacitive Touch IO registers and their address offsets are listed in Table 9-1. In a given device, multiple Capacitive Touch IO registers might be available. The base address of each Capacitive Touch IO module can be found in the device-specific data sheet.

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 9-1. CapTouch Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
0Eh	CAPTIOxCTL	Capacitive Touch IO x control register	Read/write	Word	0000h	Section 9.3.1
0Eh	CAPTIOxCTL_L		Read/write	Byte	00h	
0Fh	CAPTIOxCTL_H		Read/write	Byte	00h	

9.3.1 CAPTIOxCTL Register (offset = 0Eh) [reset = 0000h]

Capacitive Touch IO x Control Register

Figure 9-3. CAPTIOxCTL Register

15	14	13	12	11	10	9	8
Reserved						CAPTIO	CAPTIOEN
r0	r0	r0	r0	r0	r0	r-0	rw-0
7	6	5	4	3	2	1	0
CAPTIOPOSELx				CAPTIOPISELx			Reserved
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r0

Table 9-2. CAPTIOxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads 0.
9	CAPTIO	R	0h	Capacitive Touch IO state. Reports the current state of the selected Capacitive Touch IO. Reads 0, if Capacitive Touch IO disabled. 0b = Curent state 0 or Capacitive Touch IO is disabled 1b = Current state 1
8	CAPTIOEN	RW	0h	Capacitive Touch IO enable 0b = All Capacitive Touch IOs are disabled. Signal toward timers is 0. 1b = Selected Capacitive Touch IO is enabled
7-4	CAPTIOPOSELx	RW	0h	Capacitive Touch IO port select. Selects port Px. Selecting a port pin that is not available on the device in use gives unpredictible results. 0000b = Px = PJ 0001b = Px = P1 0010b = Px = P2 0011b = Px = P3 0100b = Px = P4 0101b = Px = P5 0110b = Px = P6 0111b = Px = P7 1000b = Px = P8 1001b = Px = P9 1010b = Px = P10 1011b = Px = P11 1100b = Px = P12 1101b = Px = P13 1110b = Px = P14 1111b = Px = P15
3-1	CAPTIOPISELx	RW	0h	Capacitive Touch IO pin select. Selects the pin within selected port Px (see CAPTIOPOSELx). Selecting a port pin that is not available on the device in use gives unpredictible results. 000b = Px.0 001b = Px.1 010b = Px.2 011b = Px.3 100b = Px.4 101b = Px.5 110b = Px.6 111b = Px.7
0	Reserved	R	0h	Reserved. Always reads 0.

AES256 Accelerator

The AES256 accelerator module performs Advanced Encryption Standard (AES) encryption or decryption in hardware. It supports key lengths of 128 bits, 192 bits, and 256 bits. This chapter describes the AES256 accelerator.

Topic **Page**

10.1 AES Accelerator Introduction **338**
10.2 AES Accelerator Operation **339**
10.3 AES Accelerator Registers **357**

10.1 AES Accelerator Introduction

The AES accelerator module performs encryption and decryption of 128-bit data with 128-, 192-, or 256-bit keys according to the advanced encryption standard (AES) (FIPS PUB 197) in hardware.

The AES accelerator features are:

- AES encryption
  - 128 bit in 168 cycles
  - 192 bit in 204 cycles
  - 256 bit in 234 cycles
- AES decryption
  - 128 bit in 168 cycles
  - 192 bit in 206 cycles
  - 256 bit in 234 cycles
- On-the-fly key expansion for encryption and decryption
- Offline key generation for decryption
- Shadow register storing the initial key for all key lengths
- DMA support for ECB, CBC, OFB, and CFB cipher modes
- Byte and word access to key, input data, and output data
- AES ready interrupt flag

The AES accelerator block diagram is shown in Figure 10-1.

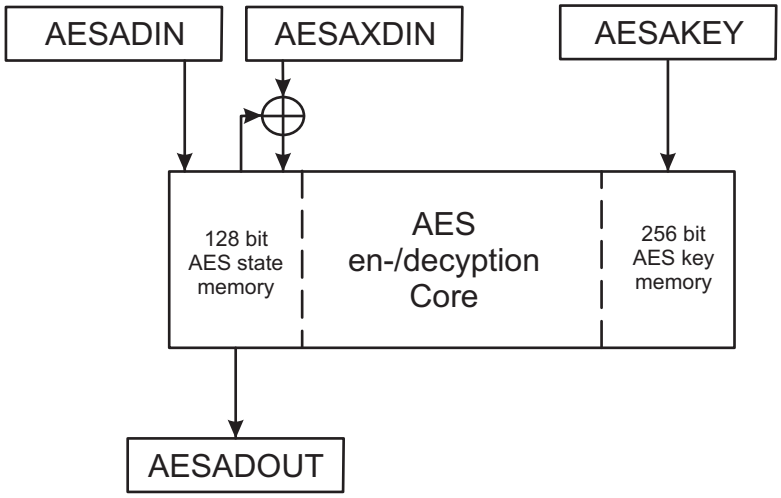


Figure 10-1. AES Accelerator Block Diagram

10.2 AES Accelerator Operation

The AES accelerator is configured with user software. The bit AESKLx determines if AES128, AES192, or AES256 is going to be performed. There are four different operation modes available, selectable by the AESOPx bits (see Table 10-1).

Table 10-1. AES Operation Modes Overview

AESOPx	AESKLx	Operation	Clock Cycles
00	00	AES128 encryption	168
	01	AES192 encryption	204
	10	AES256 encryption	234
01	00	AES128 decryption (with initial roundkey) is performed	215
	01	AES192 decryption (with initial roundkey) is performed	255
	10	AES256 decryption (with initial roundkey) is performed	292
10	00	AES128 encryption key schedule is performed	53
	01	AES192 encryption key schedule is performed	57
	10	AES256 encryption key schedule is performed	68
11	00	AES128 (with last roundkey) decryption is performed	168
	01	AES192 (with last roundkey) decryption is performed	206
	10	AES256 (with last roundkey) decryption is performed	234

The execution time of the different modes of operation is shown in Table 10-1. While the AES module is operating, the AESBUSY bit is 1. As soon as the operation has finished, the AESRDYIFG bit is set.

Internally, the AES algorithm's operations are performed on a two-dimensional array of bytes called the State. The State consists of four rows of bytes, each containing four bytes, independently if AES128, AES192, or AES256 is performed. The input is assigned to the State array as shown in Figure 10-2, with in[0] being the first data byte written into one of the AES accelerators input registers (AESADIN, AESAXDIN, and AESXIN). The encrypt or decrypt operations are then conducted on the State array, after which its final values can be read from the output with out[0] being the first data byte read from the AES accelerator data output register (AESADOUT).

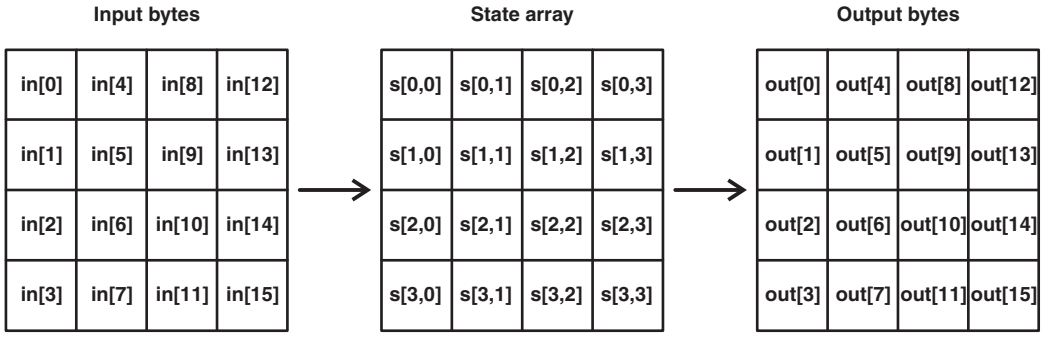


Figure 10-2. AES State Array Input and Output

If an encryption is to be performed, the initial state is called plaintext. If a decryption is to be performed, the initial state is called ciphertext.

The module allows word and byte access to all data registers—AESAKEY, AESADIN, AESAXDIN, AESAXIN, and AESADOUT. Word and byte access cannot be mixed while reading from or writing into one of the registers. However, it is possible to write one of the registers using byte access and another using word access.

NOTE: General Access Restrictions

While the AES accelerator is busy (AESBUSY = 1):

- AESADOUT always reads as zero.
- The AESDOUTCNTx counter, the AESDOUTRD flag, and the AESDINWR flag are reset.
- Any attempt to change AESOPx, AESKLx, AESDINWR, or AESKEYWR is ignored.
- Writing to AESAKEY, AESADIN, AESAXDIN, or AESAXIN aborts the current operation, the complete module is reset (except for the AESRDYIE, AESOPx, and AESKLx), and the AES error flag AESERRFG is set.

AESADIN, AESAXDIN, AESAXIN, and AESAKEY are write-only registers and always read as zero.

Writing data into AESADIN, AESAXDIN, or AESAXIN influences the content of the corresponding output data; for example, writing in[0] alters out[0], writing in[1] alters out[1], and so on. However, interleaved operation is possible; for example, first reading out[0] and then writing in[0], and continuing with reading out[1] and then writing in[1], and so on. This interleaved operation must be either byte or word access on in[x] and out[x].

Access Restriction With Cipher Modes Enabled (AESCMEN = 1)

When cipher modes are enabled (AESCMEN = 1) and a cipher block operation is being processed (AESBLKCNTx > 0), writes to the following bits are ignored, independent of AESBUSY: AESCMEN, AESCMx, AESKLx, AESOPx, and AESBLKCNTx. Writing to AESAKEY aborts the cipher block mode operation if AESBUSY = 1, and the complete module is reset (except for AESRDYIE, AESOPx, and AESKLx).

10.2.1 Load the Key (128-Bit, 192-Bit, or 256-Bit Keylength)

The key can be loaded by writing to the AESAKEY register or by setting AESKEYWR. Depending on the selected keylength (AESKLx), a different number of bits must be loaded:

- If AESKLx = 00, the 128-bit key must be loaded using either 16 byte-writes or 8 word-writes to AESAKEY.
- If AESKLx = 01, the 192-bit key must be loaded using either 24 byte-writes or 12 word-writes to AESAKEY.
- If AESKLx = 10, the 256-bit key must be loaded using either 32 byte-writes or 16 word-writes to AESAKEY.

The key memory is reset after changing the AESKLx.

If a key was loaded previously without changing AESOPx, the AESKEYWR flag is cleared with the first write access to AESAKEY.

If the conversion is triggered without writing a new key, the last key is used. The key must always be written before writing the data.

10.2.2 Load the Data (128-Bit State)

The state can be loaded by writing to AESADIN, AESAXDIN, or AESAXIN with 16 byte writes or 8 word writes. Do not mix byte and word mode when writing the state. Writing to a mixture of AESADIN, AESAXDIN, and AESAXIN using the same byte or word data format is allowed. When the 16th byte or 8th word of the state is written, AESDINWR is set.

When writing to AESADIN, the corresponding byte or word of the state is overwritten. If AESADIN is used to write the last byte or word of the state, encryption or decryption starts automatically.

When writing to AESAXDIN, the corresponding byte or word is XORed with the current byte or word of the state. If AESAXDIN is used to write the last byte or word of the state, encryption or decryption starts automatically.

Writing to AESAXIN has the same behavior as writing to AESAXDIN: the corresponding byte or word is XORed with the current byte or word of the state; however, writing the last byte or word of the state using AESAXIN does not start encryption or decryption.

10.2.3 Read the Data (128-Bit State)

The state can be read if AESBUSY = 0 using 16 byte reads or 8 word reads from AESADOUT. When all 16 bytes are read, the AESDOUTRD flag indicates completion.

10.2.4 Trigger an Encryption or Decryption

The AES module's encrypt or decrypt operations are triggered if the state was completely written via AESADIN or AESAXDIN registers. Alternatively, the bit AESDINWR can be set to trigger an operation if AESCMEN = 0.

10.2.5 Encryption

Figure 10-3 shows the encryption process with the cipher being a series of transformations that convert the plaintext written into the AESADIN register to a ciphertext that can be read from the AESADOUT register using the cipher key provided via the AESAKEY register.

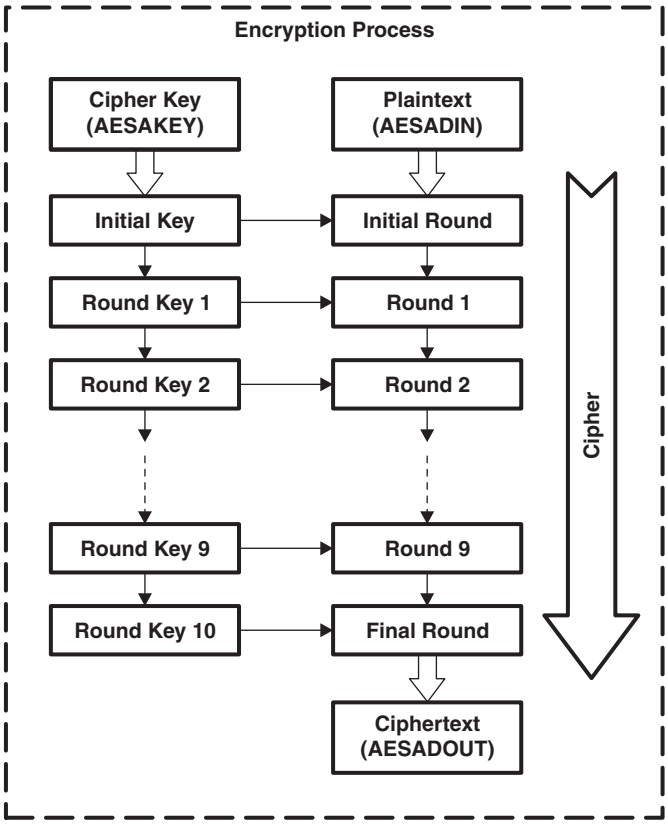


Figure 10-3. AES Encryption Process for 128-Bit Key

To perform encryption:

1. Set AESOPx = 00 to select encryption. Changing the AESOPx bits clears the AESKEYWR flag, and a new key must be loaded in the next step.
2. Load the key as described in Section 10.2.1.
3. Load the state (data) as described in Section 10.2.2. After the data is loaded, the AES module starts the encryption.
4. After the encryption is ready, the result can be read from AESADOUT as described in Section 10.2.3.
5. To encrypt additional data with the same key loaded in step 2, write the new data into AESADIN after the results of the operation on the previous data were read from AESADOUT. When an additional 16 data bytes are written, the module automatically starts the encryption using the key loaded in step 2.

   When implementing, for example, the output feedback (OFB) cipher block chaining mode, setting the AESDINWR flag triggers the next encryption, and the module starts the encryption using the output data from the previous encryption as the input data.

10.2.6 Decryption

Figure 10-4 shows the decryption process with the inverse cipher being a series of transformations that convert the ciphertext written into the AESADIN register to a plaintext that can be read from the AESADOUT register using the cipher key provided via the AESAKEY register.

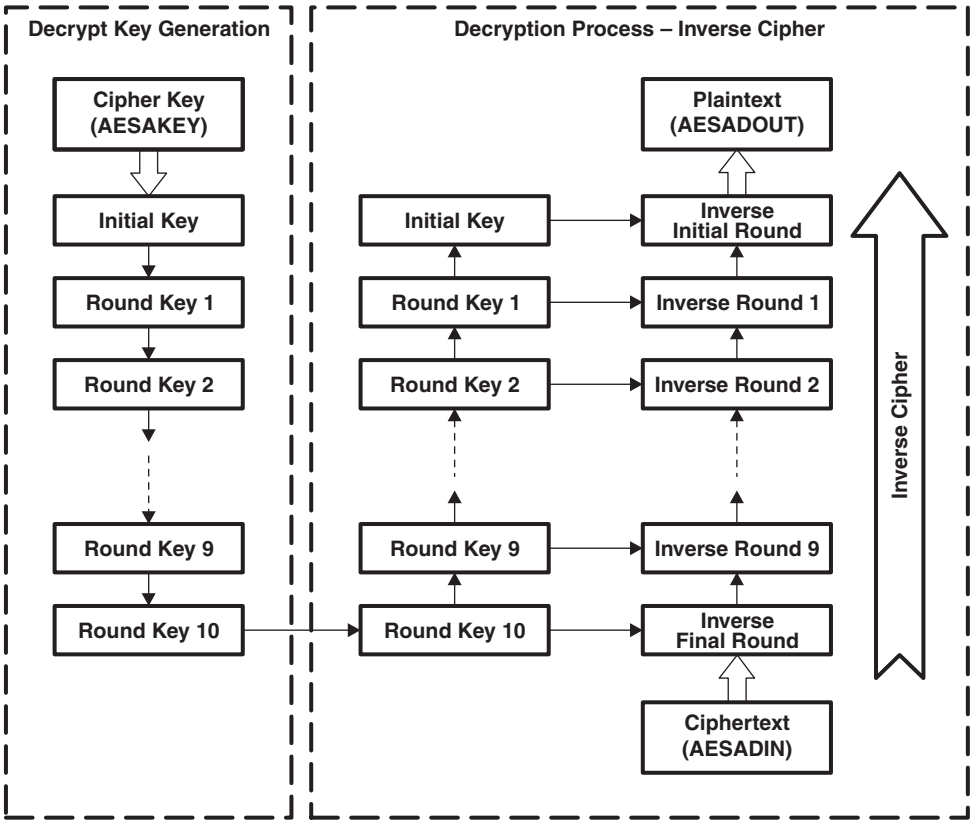


Figure 10-4. AES Decryption Process Using AESOPx = 01 for 128-bit key

The steps to perform decryption are:

1. Set AESOPx = 01 to select decryption using the same key used for encryption. Set AESOPx = 11 if the first-round key required for decryption (the last roundkey) is already generated and will be loaded in step 2. Changing the AESOPx bits clears the AESKEYWR flag, and a new key must be loaded in step 2.
2. Load the key according to Section 10.2.1.
3. Load the state (data) according to Section 10.2.2. After the data is loaded, the AES module starts the decryption.
4. After the decryption is ready, the result can be read from AESADOUT according to Section 10.2.3.
5. If additional data should be decrypted with the same key loaded in step 2, new data can be written into AESADIN after the results of the operation on the previous data were read from AESADOUT. When additional 16 data bytes are written, the module automatically starts the decryption using the key loaded in step 2.

10.2.7 Decryption Key Generation

Figure 10-5 shows the decryption process with a pregenerated decryption key. In this case, the decryption key is calculated first with AESOPx = 10, then the precalculated key can be used together with the decryption operation AESOPx = 11.

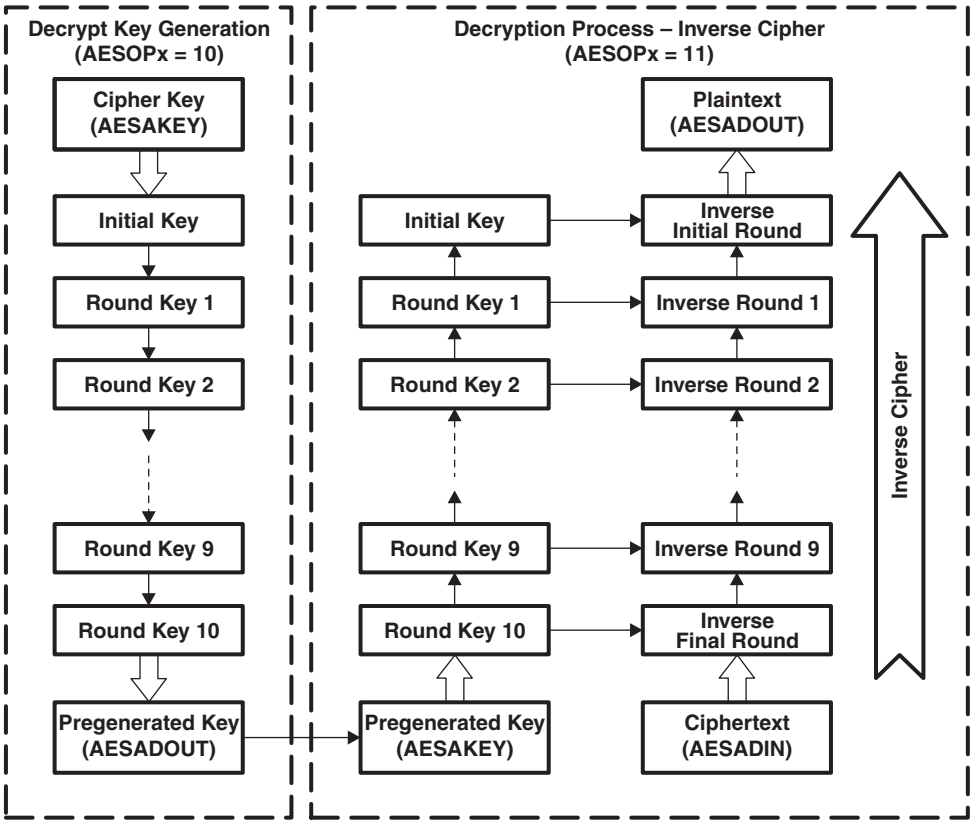


Figure 10-5. AES Decryption Process using AESOPx = 10 and 11 for 128-bit key

To generate the decryption key independent from the actual decryption:

1. Set AESOPx = 10 to select decryption key generation. Changing the AESOPx bits clears the AESKEYWR flag, and a new key must be loaded in step 2.
2. Load the key as described in Section 10.2.1. The generation of the first round key required for decryption is started immediately.
3. While the AES module is performing the key generation, the AESBUSY bit is 1. 53 CPU clock cycles are required to complete the key generation for a 128-bit key (for other key lengths, see Table 10-1). After its completion, the AESRDYIFG is set, and the result can be read from AESADOUT. When all 16 bytes are read, the AESDOUTRD flag indicates completion.

   The AESRDYIFG flag is cleared when reading AESADOUT or writing to AESAKEY or AESADIN.
4. If data should be decrypted with the generated key, AESOPx must be set to 11. Then the generated key must be loaded or, if it was just generated with AESOPx = 10, set the AESKEYWR flag by software to indicate that the key is already valid.
5. See Section 10.2.6 for instructions on the decryption steps, starting from step 3 (load data).

10.2.8 AES Key Buffer

The AES128, AES192, or AES256 algorithm operates not only on the state, but also on the key. To avoid the need of reloading the key for each encryption or decryption, a key buffer is included in the AES accelerator.

10.2.9 Using the AES Accelerator With Low-Power Modes

The AES accelerator module provides automatic clock activation for MCLK for use with low-power modes. When the AES accelerator is busy, it automatically activates MCLK, regardless of the control-bit settings for the clock source. The clock remains active until the AES accelerator completes its operation.

The interrupt flag AESRDYIFG is reset after a PUC or with AESSWRST = 1. AESRDYIE is reset after a PUC but is not reset by AESSWRST = 1.

10.2.10 AES Accelerator Interrupts

The AESRDYIFG interrupt flag is set when the AES module completes the selected operation on the provided data. An interrupt request is generated if AESRDYIE and GIE are also set. AESRDYIFG is automatically reset if the AES interrupt is serviced, if AESADOUT is read, or if AESADIN or AESAKEY are written. AESRDYIFG is reset after a PUC or with AESSWRST = 1. AESRDYIE is reset after a PUC but is not reset by AESSWRST = 1.

10.2.11 DMA Operation and Implementing Block Cipher Modes

DMA operation, meaning the implementation of the ciphermodes Electronic code book (ECB), Cipher block chaining (CBC), Output feedback (OFB), and Cipher feedback (CFB) using the DMA, supports easy and fast encryption and decryption of more than 128 bits.

When DMA ciphermode support is enabled by setting the AESCMEN bit, the AES256 module triggers 'AES trigger 0', 'AES trigger 1', and 'AES trigger 2' (also called 'AES trigger 0-2') in a certain order to execute different block cipher modes together with the DMA module.

For example, when using ECB encryption with AESCMEN = 1, 'AES trigger 0' is triggered eight times for DMA word access to read out AESADOUT, and then 'AES trigger 1' is triggered eight times to fill the next data into AESADIN.

Table 10-2 shows the behavior of the 'AES trigger 0-2' for the different ciphermodes selected by AESCMx.

Table 10-2. 'AES trigger 0-2' Operation When AESCMEN = 1

AESCMx	AESOPx	'AES trigger 0'	'AES trigger 1'	'AES trigger 2'
00 ECB	00 encryption	Set after encryption ready, set again until 128 bit are read from AESADOUT	Set to load the first block and set after 'AES trigger 0' was served the last time, set again until 128 bit are written to AESADIN	not set
	01 or 11 decryption	Set after decryption ready, set again until 128 bit are read from AESADOUT	Set to load the first block and set after 'AES trigger 0' was served the last time, set again until 128 bit are written to AESADIN	not set
01 CBC	00 encryption	Set after encryption ready, set again until 128 bit are read from AESADOUT	Set after 'AES trigger 0' was served the last time, set again until 128 bit are written to AESAXDIN	not set
	01 or 11 decryption	Set after decryption ready, set again until 128 bit are written to from AESAXIN	Set after 'AES trigger 0' was served the last time, set again until 128 bit are read from AESADOUT	Set after 'AES trigger 1' was served the last time, set again until 128 bit are written to AESADIN
10 OFB	00 encryption	Set after encryption ready, set again until 128 bit are written to AESAXIN	Set after 'AES trigger 0' was served the last time, set again until 128 bit are read from AESADOUT	Set after 'AES trigger 1' was served the last time, set again until 128 bit are written to AESAXDIN
	01 or 11 decryption	Set after decryption ready, set again until 128 bit are written to AESAXIN	Set after 'AES trigger 0' was served the last time, set again until 128 bit are read from AESADOUT	Set after 'AES trigger 1' was served the last time, set again until 128 bit are written to AESAXDIN

Table 10-2. 'AES trigger 0-2' Operation When AESCMEN = 1 (continued)

AESCMx	AESOPx	'AES trigger 0'	'AES trigger 1'	'AES trigger 2'
11 CFB	00 encryption	Set after encryption ready, set again until 128 bit are written to AESAXIN	Set after 'AES trigger 0' was served the last time, set again until 128 bit are read from AESADOUT	not set
	01 or 11 decryption	Set after decryption ready, set again until 128 bit are written to AESAXIN	Set after 'AES trigger 0' was served the last time, set again until 128 bit are read from AESADOUT	Set after 'AES trigger 1' was served the last time, set again until 128 bit are written to AESADIN

The retriggering of the 'AES trigger 0-2' until 128-bit of data are written or read from the corresponding register supports both byte and word access for writing and reading the state via the DMA.

For AESCMEN = 0, no DMA triggers are generated.

The following sections explain the configuration of the AES module for automatic cipher mode execution using DMA.

It is assumed that the key is written by software (or by a separate DMA transfer) before writing the first block to the AES state. The key shadow register always restores the original key, so that there is no need to reload it. The AESAKEY register should not be written after AESBLKCNTx is written to a non-zero value.

The number of blocks to be encrypted or decrypted must be programmed into the AESBLKCNTx bits prior to writing the first data. Writing a non-zero value into AESBLKCNTx starts the cipher mode sequence and, thus, AESBLKCNTx must be written after the DMA channels are configured.

Throughout these sections, the different DMA channels are called DMA_A, DMA_B, and so on. In the figures, these letters appear in dotted circles showing which operation is going to be executed by which DMA channel. The DMA counter must be loaded with a multiple of 8 for word mode or a multiple of 16 for byte mode. The DMA priorities of DMA_A, DMA_B, and DMA_C do not play any role but static DMA priorities must be enabled. The DMA triggers must be configured as level-sensitive triggers.

10.2.11.1 Electronic Codebook (ECB) Mode

The electronic codebook block ciphermode is the most simple cipher mode. The data is divided into 128-bit blocks and encrypted and decrypted separately.

The disadvantage of the ECB is that the same 128-bit plaintext is always encrypted to the same ciphertext, whereas the other modes encrypt each block differently, partly dependent on the already executed encryptions.

10.2.11.1.1 ECB Encryption

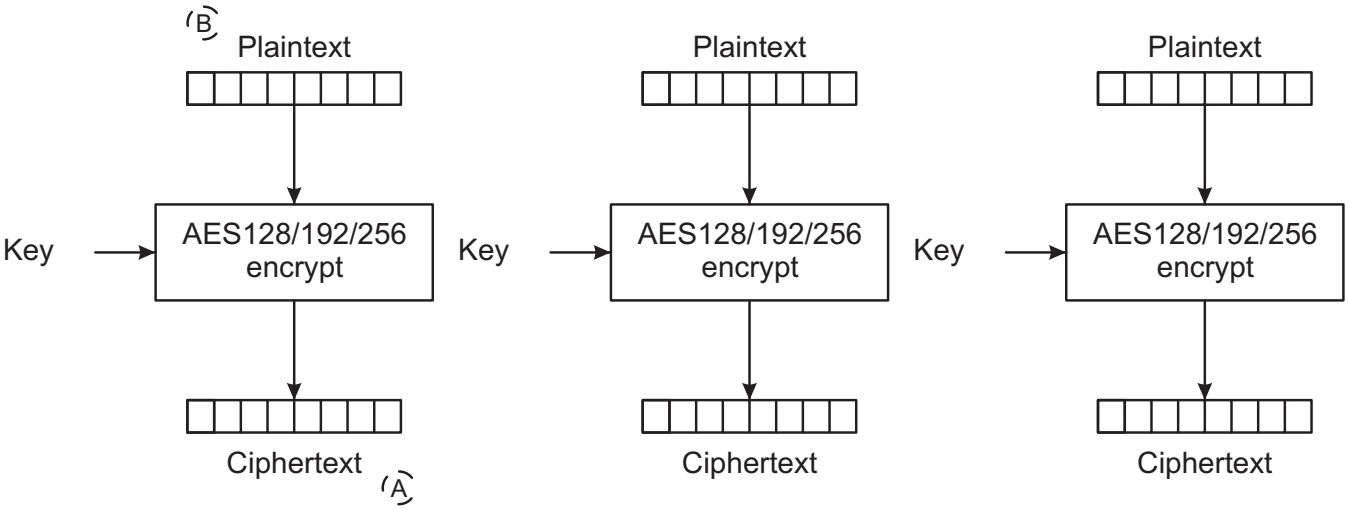


Figure 10-6. ECB encryption

To implement the ECB encryption without CPU interaction, two DMA channels are needed. Static DMA priorities must be enabled. The DMA triggers must be configured as level-sensitive triggers.

Table 10-3. AES and DMA Configuration for ECB Encryption

AES CMEN	AES CMx	AES OPx	DMA_A Triggered by 'AES trigger 0'	DMA_B Triggered by 'AES trigger 1'
1	00	00	Read ciphertext from AESADOUT	Write plaintext to AESADIN, which also triggers the next encryption

The following pseudo code snippet shows the implementation of the ECB encryption in software:

```
ECB_Encryption(key, plaintext, ciphertext, num_blocks)
// Pseudo Code
{
    Configure AES for block cipher:
        AESCMEN= 1; AESCMx= ECB; AESOPx= 00;

    Write key into AESAKEY;
    Setup DMA:
        DMA0: Triggered by AES trigger 0,
              Source: AESADOUT,  Destination: ciphertext, Size: num_blocks*8 words
        DMA1: Triggered by AES trigger 1,
              Source: plaintext, Destination: AESADIN,    Size: num_blocks*8 words

    Start encryption:
        AESBLKCNT= num_blocks;

    End of encryption: DMA0IFG=1
}
```

10.2.11.1.2 ECB Decryption

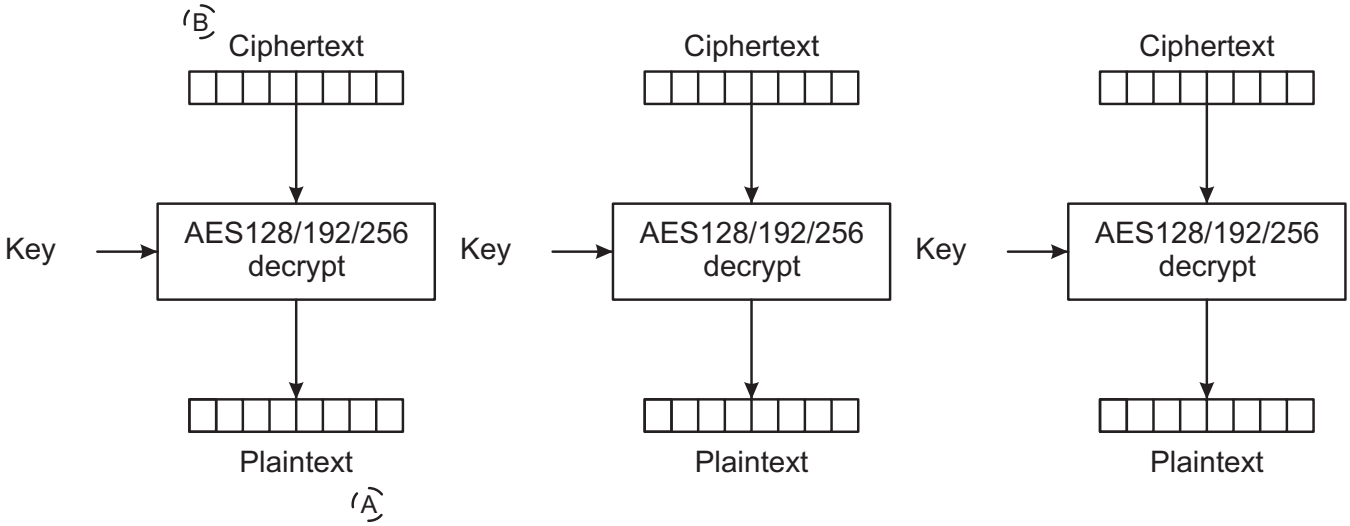


Figure 10-7. ECB decryption

To implement the ECB decryption without CPU interaction, two DMA channels are needed. Static DMA priorities must be enabled. The DMA triggers must be configured as level-sensitive triggers.

Table 10-4. AES DMA Configuration for ECB Decryption

AES CMEN	AES CMx	AES OPx	DMA_A Triggered by 'AES trigger 0'	DMA_B Triggered by 'AES trigger 1'
1	00	01 or 11	Read plaintext from AESADOUT	Write ciphertext to AESADIN, which also triggers the next decryption

The following pseudo code snippet shows the implementation of the ECB decryption in software:

```
ECB_Decryption(key, plaintext, ciphertext, num_blocks)
// Pseudo Code
{
    Generate Decrypt Key
        Configure AES:
            AESCMEN= 0; AESOPx= 10;
         Write key into AESAKEY;
         Wait until key generation completed.

    Configure AES for block cipher:
        AESCMEN= 1; AESCMx= ECB; AESOPx= 11;
        AESKEYWR= 1; // Use previously generated key

    Setup DMA:
        DMA0: Triggered by AES trigger 0,
              Source: AESADOUT, Destination: plaintext, Size: num_blocks*8 words
        DMA1: Triggered by AES trigger 1,
              Source: ciphertext, Destination: AESADIN, Size: num_blocks*8 words

    Start decryption:
        AESBLKCNT= num_blocks;

    End of decryption: DMA0IFG=1
}
```

10.2.11.2 Cipher Block Chaining (CBC) Mode

The cipher block chaining ciphermode always performs an XOR on the ciphertext of the previous block with the current block. Therefore, the encryption of each block depends not only on the key but also on the previous encryption.

10.2.11.2.1 CBC Encryption

For encryption, the initialization vector must be loaded by software (or by a separate DMA transfer) into AESXIN before the DMA can be enabled to write the first 16 bytes of the plaintext into AESAXDIN

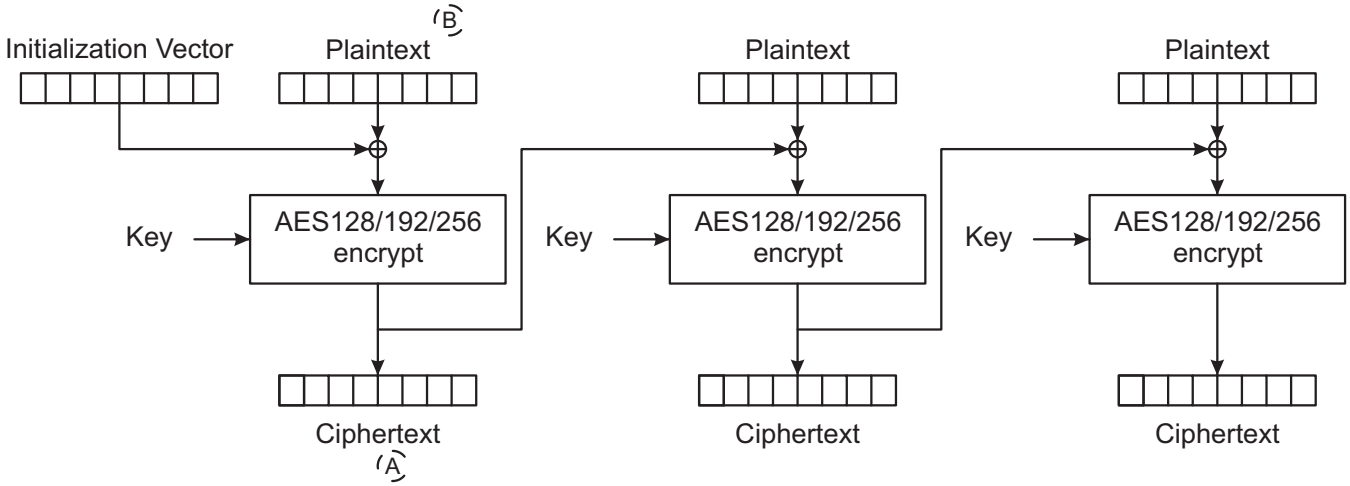


Figure 10-8. CBC Encryption

To implement the CBC encryption without CPU interaction, two DMA channels are needed. Static DMA priorities must be enabled. The DMA triggers must be configured as level-sensitive triggers.

Table 10-5. AES and DMA Configuration for CBC Encryption

AES CMEN	AES CMx	AES OPx	DMA_A Triggered by 'AES trigger 0'	DMA_B Triggered by 'AES trigger 1'
1	01	00	Read ciphertext from AESADOUT	Write plaintext to AESAXDIN, which also triggers the next encryption

The following pseudo code snippet shows the implementation of the CBC encryption in software:

```
CBC_Encryption(key, IV, plaintext, ciphertext, num_blocks)
// Pseudo Code
{
    Reset AES Module (clears internal state memory):
        AESSWRST= 1;
    Configure AES for block cipher:
        AESCMEN= 1; AESCMx= CBC; AESOPx= 00;

    Write key into AESAKEY;
    Write IV into AESAXIN; // Does not trigger encryption.
                           // Assumes that state is reset (=> XORing with Zeros).
    Setup DMA:
        DMA0: Triggered by AES trigger 0,
              Source: AESADOUT,  Destination: ciphertext, Size: num_blocks*8 words
        DMA1: Triggered by AES trigger 1,
              Source: plaintext, Destination: AESAXDIN,   Size: num_blocks*8 words

    Start encryption:
        AESBLKCNT= num_blocks;
```

```
        End of encryption: DMA0IFG=1
}
```

10.2.11.2.2 CBC Decryption

For CBC decryption, the first block of data needs some special handling because the output must be XORed with the Initialization Vector. For that purpose, the DMA triggered by 'AES trigger 0' must be configured to read the data from the Initialization Vector first and then must be reconfigured to read from the ciphertext.

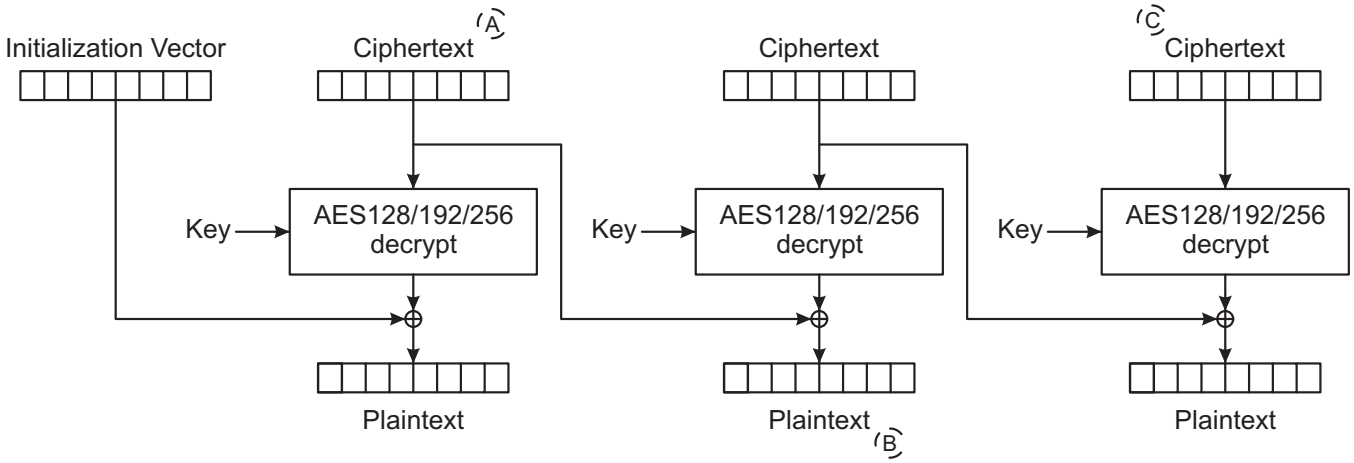


Figure 10-9. CBC Decryption

To implement the CBC decryption without CPU interaction, three DMA channels are needed. Static DMA priorities must be enabled. The DMA triggers must be configured as level-sensitive triggers.

Table 10-6. AES and DMA Configuration for CBC Decryption

AES CMEN	AES CMx	AES OPx	DMA_A Triggered by 'AES trigger 0'	DMA_B Triggered by 'AES trigger 1'	DMA_C Triggered by 'AES trigger 2'
1	01	01 or 11	Write the previous ciphertext block to AESAXIN	Read plaintext from AESADOUT	Write next plaintext to AESADIN, which also triggers the next decryption

The following pseudo code snippet shows the implementation of the CBC decryption in software:

```
CBC_Decryption(key, IV, plaintext, ciphertext, num_blocks)
// Pseudo Code
{
    Generate Decrypt Key:
        Configure AES:
            AESCMEN= 0; AESOPx= 10;
         Write key into AESAKEY;
         Wait until key generation completed;

    Configure AES for block cipher:
        AESCMEN= 1; AESCMx= CBC; AESOPx= 11;
        AESKEYWR= 1; // Use previously generated key

    Setup DMA:
        DMA0: Triggered by AES trigger 0,
              Source: IV,         Destination: AESAXIN,   Size: 8 words
        DMA1: Triggered by AES trigger 1,
              Source: AESADOUT,   Destination: plaintext, Size: num_blocks*8 words
        DMA2: Triggered by AES trigger 2,
              Source: ciphertext, Destination: AESADIN,   Size: num_blocks*8 words
```

```
        Start decryption:
            AESBLKCNT= num_blocks;

        Wait until first block is decrypted: DMA0IFG=1;

        Setup DMA0 for further blocks:
           DMA0: // Write previous cipher text into AES module
                 Triggered by AES trigger 0,
                 Source: ciphertext, Destination: AESAXIN,   Size: (num_blocks-1)*8 words

        End of decryption: DMA1IFG=1
    }
```

10.2.11.3 Output Feedback (OFB) Mode

The output feedback ciphermode continuously encrypts the initialization vector. The ciphertext is generated by XORing the corresponding plaintext with the encrypted initialization vector.

The initialization vector must be loaded by software (or by a separate DMA transfer).

10.2.11.3.1 OFB Encryption

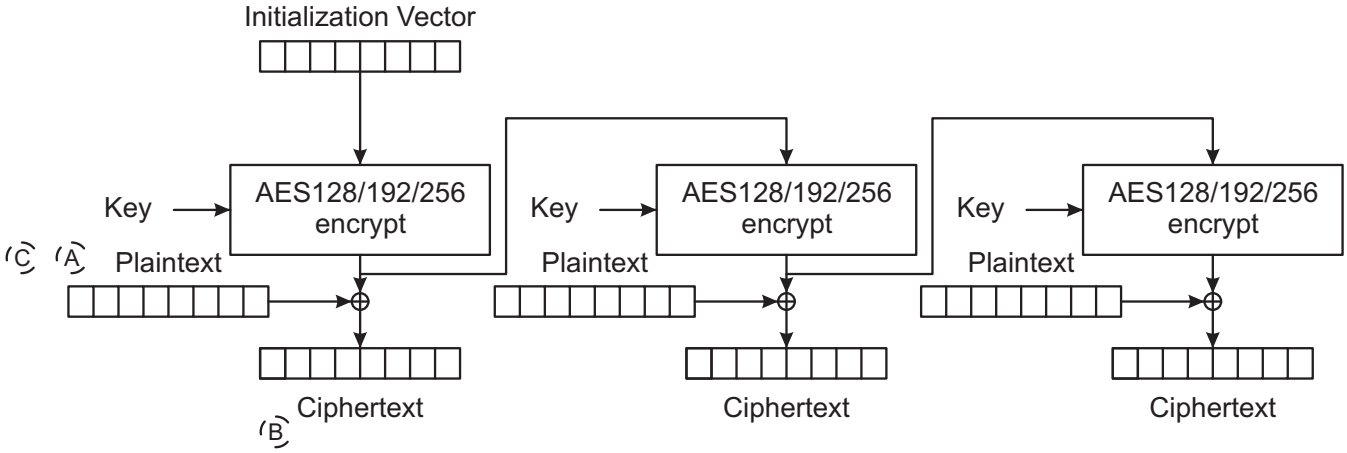


Figure 10-10. OFB Encryption

To implement the OFB encryption without CPU interaction, three DMA channels are needed. Static DMA priorities must be enabled. The DMA triggers must be configured as level-sensitive triggers.

Table 10-7. AES and DMA Configuration for OFB Encryption

AES CMEN	AES CMx	AES OPx	DMA_A Triggered by 'AES trigger 0'	DMA_B Triggered by 'AES trigger 1'	DMA_C Triggered by 'AES trigger 2'
1	10	00	Write the plaintext of the current block to AESAXIN	Read ciphertext from AESADOUT	Write the plaintext of the current block to AESAXDIN, which also triggers the next encryption

The following pseudo code snippet shows the implementation of the OFB encryption in software:

```
OFB_Encryption(Key, IV, plaintext, ciphertext, num_blocks)
// Pseudo Code
{
    Reset AES Module (clears internal state memory):
        AESSWRST= 1;
    Configure AES:
        AESCMEN= 1; AESCMx= OFB; AESOPx= 00;

    Write Key into AESAKEY;
    Write IV into AESAXIN; // Does not trigger encryption.
                           // Assumes that state is reset (=> XORing with Zeros).
    Setup DMA:
        DMA0: Triggered by AES trigger 0,
              Source: plaintext, Destination: AESAXIN,    Size: num_blocks*8 words
        DMA1: Triggered by AES trigger 1,
              Source: AESADOUT,  Destination: ciphertext, Size: num_blocks*8 words
        DMA2: Triggered by AES trigger 2,
              Source: plaintext, Destination: AESAXDIN,   Size: num_blocks*8 words

    Start encryption:
        AESBLKCNT= num_blocks;
        Trigger encryption by setting AESDINWR= 1;
```

```
    End of encryption: DMA1IFG=1
}
```

10.2.11.3.2 OFB Decryption

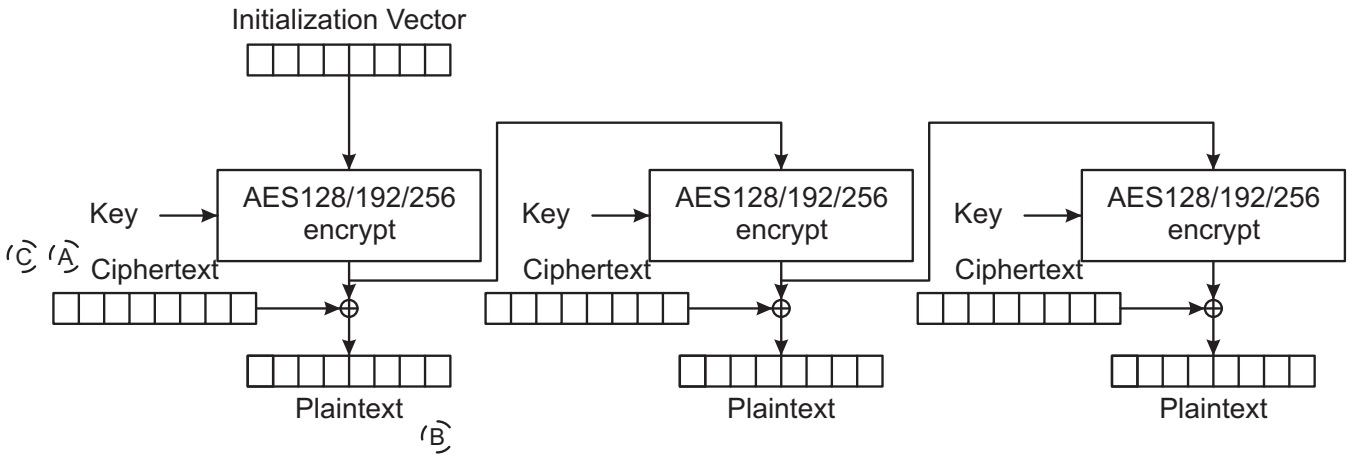


Figure 10-11. OFB Decryption

To implement the OFB decryption without CPU interaction, three DMA channels are needed. Static DMA priorities must be enabled. The DMA triggers must be configured as level-sensitive triggers.

Table 10-8. AES and DMA Configuration for OFB Decryption

AES CMEN	AES CMx	AES OPx	DMA_A Triggered by 'AES trigger 0'	DMA_B Triggered by 'AES trigger 1'	DMA_C Triggered by 'AES trigger 2'
1	10	01 or 11 (1)	Write the ciphertext of the current block to AESAXIN	Read plaintext from AESADOUT	Write the ciphertext of the current block to AESAXDIN, which also triggers the next encryption

(1) Note, in this cipher mode, the decryption also uses AES encryption on block level, thus the key used for decryption is identical with the key used for encryption; therefore, no decryption key generation is required.

The following pseudo code snippet shows the implementation of the OFB decryption in software:

```
OFB_Decryption(Key, IV, plaintext, ciphertext, num_blocks)
// Pseudo Code
{
    Reset AES Module (clears internal state memory):
        AESSWRST= 1;
    Configure AES:
        AESCMEN= 1; AESCMx= OFB; AESOPx= 01;

    Write Key into AESAKEY;
    Write IV into AESAXIN; // Does not trigger encryption.
                           // Assumes that state is reset (=> XORing with Zeros).
    Setup DMA:
       DMA0: Triggered by AES trigger 0,
             Source: ciphertext, Destination: AESAXIN,   Size: num_blocks*8 words
       DMA1: Triggered by AES trigger 1,
             Source: AESADOUT,   Destination: plaintext, Size: num_blocks*8 words
       DMA2: Triggered by AES trigger 2,
             Source: ciphertext, Destination: AESAXDIN,  Size: num_blocks*8 words

    Start decryption:
        AESBLKCNT= num_blocks;
        Trigger decryption by setting AESDINWR= 1;

    End of decryption: DMA1IFG=1
```

```
}
```

10.2.11.4 Cipher Feedback (CFB) Mode

In the cipher feedback ciphermode, the plaintext of the new block is XORed to the last encryption result. The result of the encryption is the input for the new encryption.

The initialization vector must be loaded by software (or by a separate DMA transfer).

10.2.11.4.1 CFB Encryption

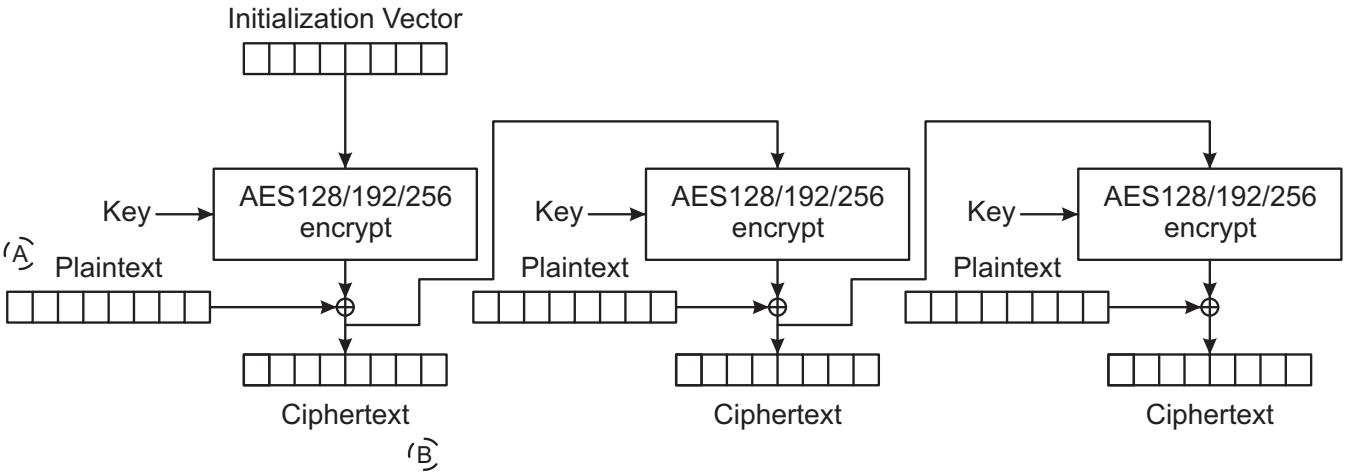


Figure 10-12. CFB Encryption

To implement the CFB encryption without CPU interaction, two DMA channels are needed. Static DMA priorities must be enabled. The DMA triggers must be configured as level-sensitive triggers.

Table 10-9. AES and DMA Configuration for CFB Encryption

AES CMEN	AES CMx	AES OPx	DMA_A Triggered by 'AES trigger 0'	DMA_B Triggered by 'AES trigger 1'
1	11	00	Write the plaintext of the current block to AESAXIN	Read the ciphertext from AESADOUT, which also triggers the next encryption

The following pseudo code snippet shows the implementation of the CFB encryption in software:

```
CFB_Encryption(Key, IV, plaintext, ciphertext, num_blocks)
// Pseudo Code
{
    Reset AES Module (clears internal state memory):
        AESSWRST= 1;
    Configure AES:
        AESCMEN= 1; AESCMx= CFB; AESOPx= 00;

    Write Key into AESAKEY;
    Write IV into AESAXIN; // Does not trigger encryption.
                           // Assumes that state is reset (=> XORing with Zeros).
    Setup DMA:
        DMA0: Triggered by AES trigger 0,
              Source: plaintext, Destination: AESAXIN,    Size: num_blocks*8 words
        DMA1: Triggered by AES trigger 1,
              Source: AESADOUT,  Destination: ciphertext, Size: num_blocks*8 words

    Start encryption:
        AESBLKCNT= num_blocks;
        Trigger encryption by setting AESDINWR= 1;

    End of encryption: DMA1IFG=1
}
```

10.2.11.4.2 CFB Decryption

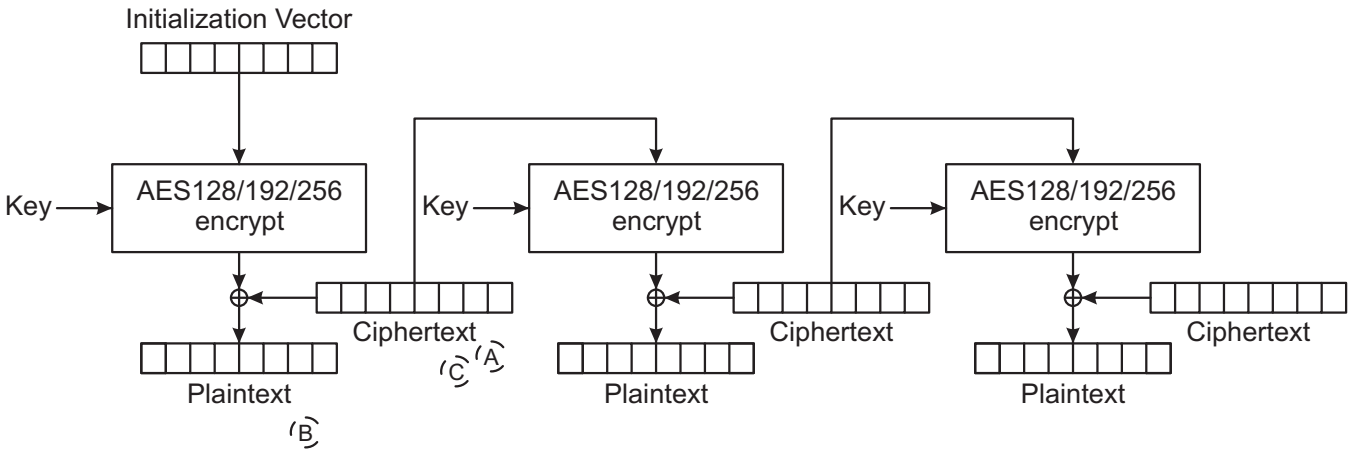


Figure 10-13. CFB Decryption

To implement the CFB decryption without CPU interaction, three DMA channels are needed. Static DMA priorities must be enabled. The DMA triggers must be configured as level-sensitive triggers.

Table 10-10. AES and DMA Configuration for CFB Decryption

AES CMEN	AES CMx	AES OPx	DMA_A Triggered by 'AES trigger 0'	DMA_B Triggered by 'AES trigger 1'	DMA_C Triggered by 'AES trigger 2'
1	11	01 or 11 (1)	Write the ciphertext of the current block to AESAXIN	Read the plaintext from AESADOUT	Write the ciphertext of the current block to AESADIN, which also triggers the next encryption

(1) Note, in this cipher mode, the decryption also uses AES encryption on block level thus the key used for decryption is identical with the key used for encryption; therefore, no decryption key generation is required.

The following pseudo code snippets shows the implementation of the CFB encryption and decryption in software:

```
CFB_Decryption(Key, IV, plaintext, ciphertext, num_blocks)
// Pseudo Code
{
    Reset AES Module (clears internal state memory):
        AESSWRST= 1;
    Configure AES:
        AESCMEN= 1; AESCMx= CFB; AESOPx= 01;

    Write Key into AESAKEY;
    Write IV into AESAXIN; // Does not trigger encryption.
                           // Assumes that state is reset (=> XORing with Zeros).
    Setup DMA:
        DMA0: Triggered by AES trigger 0,
              Source: ciphertext, Destination: AESAXIN,   Size: num_blocks*8 words
        DMA1: Triggered by AES trigger 1,
              Source: AESADOUT,   Destination: plaintext, Size: num_blocks*8 words
        DMA2: Triggered by AES trigger 2,
              Source: ciphertext, Destination: AESADIN,   Size: num_blocks*8 words

    Start decryption:
        AESBLKCNT= num_blocks;
        Trigger decryption by setting AESDINWR= 1;

    End of decryption: DMA1IFG=1
}
```

10.3 AES Accelerator Registers

Table 10-11 shows the memory-mapped registers for the AES256 module with their address offsets. See the device-specific data sheet for the base memory address of these registers. All other register offset addresses not listed in Table 10-11 should be considered as reserved locations, and the register contents should not be modified.

Table 10-11. AES256 Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	AESACTL0	AES accelerator control register 0	Read/write	Word	00h	Section 10.3.1
02h	AESACTL1	AES accelerator control register 1	Read/write	Word	00h	Section 10.3.2
04h	AESASTAT	AES accelerator status register	Read only	Word	00h	Section 10.3.3
06h	AESAKEY	AES accelerator key register	Read/write	Word	00h	Section 10.3.4
08h	AESADIN	AES accelerator data in register	Write only	Word	00h	Section 10.3.5
0Ah	AESADOUT	AES accelerator data out register	Read/write	Word	00h	Section 10.3.6
0Ch	AESAXDIN	AES accelerator XORed data in register	Write only	Word	00h	Section 10.3.7
0Eh	AESAXIN	AES accelerator XORed data in register (no trigger)	Write only	Word	00h	Section 10.3.8

10.3.1 AESACTL0 Register

AES Accelerator Control Register 0

Figure 10-14. AESACTL0 Register

15	14	13	12	11	10	9	8
AESCMEN	Reserved		AESRDYIE	AESERRFG	Reserved		AESRDYIFG
rw-0	r0	r0	rw-0	rw-0	r0	r0	rw-0
7	6	5	4	3	2	1	0
AESSWRST	AESCMx		Reserved	AESKLx		AESOPx	
rw-0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0

Writes are ignored when AESCMEN = 1 and AESBLKCNTx > 0.

Table 10-12. AESACTL0 Register Description

Bit	Field	Type	Reset	Description
15	AESCMEN	RW	0h	AESCMEN enables the support of the ciphermodes ECB, CBC, OFB and CFB together with the DMA. Writes are ignored when AESCMEN = 1 and AESBLKCNTx > 0. 0 = No DMA triggers are generated 1 = DMA ciphermode support operation is enabled and the corresponding DMA triggers are generated.
14-13	Reserved	R	0h	Reserved
12	AESRDYIE	RW	0h	AES ready interrupt enable. AESRDYIE is not reset by AESSWRST = 1. 0b = Interrupt disabled 1b = Interrupt enabled
11	AESERRFG	RW	0h	AES error flag. AESAKEY or AESADIN were written while an AES operation was in progress. The bit must be cleared by software. 0b = No error 1b = Error occurred
10-9	Reserved	R	0h	Reserved
8	AESRDYIFG	RW	0h	AES ready interrupt flag. Set when the selected AES operation was completed and the result can be read from AESADOUT. Automatically cleared when AESADOUT is read or AESAKEY or AESADIN is written. 0b = No interrupt pending 1b = Interrupt pending
7	AESSWRST	RW	0h	AES software reset. Immediately resets the complete AES accelerator module even when busy except for the AESRDYIE, the AESKLx and the AESOPx bits. It also clears the (internal) state memory. The AESSWRST bit is automatically reset and is always read as zero. 0b = No reset 1b = Reset AES accelerator module
6-5	AESCMx	RW	0h	AES cipher mode select. These bits are ignored for AESCMEN = 0. Writes are ignored when AESCMEN = 1 and AESBLKCNTx > 0. 00b = ECB 01b = CBC 10b = OFB 11b = CFB
4	Reserved	R	0h	Reserved
3-2	AESKLx	RW	0h	AES key length. These bits define which of the 3 AES standards is performed. The AESKLx bits are not reset by AESSWRST = 1. Writes are ignored when AESCMEN = 1 and AESBLKCNTx > 0. 00b = AES128. The key size is 128 bit. 01b = AES192. The key size is 192 bit. 10b = AES256. The key size is 256 bit. 11b = Reserved

Table 10-12. AESACTL0 Register Description (continued)

Bit	Field	Type	Reset	Description
1-0	AESOPx	RW	0h	AES operation. The AESOPx bits are not reset by AESSWRST = 1. Writes are ignored when AESCMEN = 1 and AESBLKCNTx > 0. 00b = Encryption 01b = Decryption. The provided key is the same key used for encryption. 10b = Generate first round key required for decryption. 11b = Decryption. The provided key is the first round key required for decryption.

10.3.2 AESACTL1 Register

AES Accelerator Control Register 1

Figure 10-15. AESACTL1 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
AESBLKCNTx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Writes are ignored when AESCMEN = 1 and AESBLKCNTx > 0.

Table 10-13. AESACTL1 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads 0.
7-0	AESBLKCNTx	RW	0h	Cipher Block Counter. Number of blocks to be encrypted or decrypted with block cipher modes enabled (AESCMEN = 1). Ignored if AESCMEN = 0. The block counter decrements with each performed encryption or decryption. Writes are ignored when AESCMEN = 1 and AESBLKCNTx > 0.

10.3.3 AESASTAT Register

AES Accelerator Status Register

Figure 10-16. AESASTAT Register

15	14	13	12	11	10	9	8
AESDOUTCNTx				AESDINCNTx			
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
AESKEYCNTx				AESDOUTRD	AESDINWR	AEKEYWR	AESBUSY
r-0	r-0	r-0	r-0	r-0	rw-0	rw-0	r-0

Table 10-14. AESASTAT Register Description

Bit	Field	Type	Reset	Description
15-12	AESDOUTCNTx	R	0h	Bytes read via AESADOUT. Reset when AESDOUTRD is reset. If AESDOUTCNTx = 0 and AESDOUTRD = 0, no bytes were read. If AESDOUTCNTx = 0 and AESDOUTRD = 1, all bytes were read.
11-8	AESDINCNTx	R	0h	Bytes written via AESADIN, AESAXDIN or AESAXIN. Reset when AESDINWR is reset. If AESDINCNTx = 0 and AESDINWR = 0, no bytes were written. If AESDINCNTx = 0 and AESDINWR = 1, all bytes were written.
7-4	AESKEYCNTx	R	0h	Bytes written via AESAKEY for AESKLx=00, words written via AESAKEY if AESKLx = 01, 10, 11. Reset when AESKEYWR is reset. If AESKEYCNTx = 0 and AESKEYWR = 0, no bytes were written. If AESKEYCNTx = 0 and AESKEYWR = 1, all bytes were written.
3	AESDOUTRD	R	0h	All 16 bytes read from AESADOUT. AESDOUTRD is reset by PUC, AESSWRST, an error condition, changing AESOPx, changing AESKLx, when the AES accelerator is busy, and when the output data is read again. 0 = Not all bytes read 1 = All bytes read
2	AESDINWR	RW	0h	All 16 bytes written to AESADIN, AESAXDIN or AESAXIN. This bit can be modified by software only if AESCMEN=0. Changing its state by software also resets the AESDINCNTx bits. AESDINWR is reset by PUC, AESSWRST, an error condition, changing AESOPx, changing AESKLx, the start to (over)write the data, and when the AES accelerator is busy. Because it is reset when AESOPx or AESKLx is changed it can be set by software again to indicate that the current data is still valid. 0 = Not all bytes written 1 = All bytes written
1	AESKEYWR	RW	0h	All 16 bytes written to AESAKEY. This bit can be modified by software but it must not be reset by software (1→0) if AESCMEN=1. Changing its state by software also resets the AESKEYCNTx bits. AESKEYWR is reset by PUC, AESSWRST, an error condition, changing AESOPx, changing AESKLx, and the start to (over)write a new key. Because it is reset when AESOPx is changed it can be set by software again to indicate that the loaded key is still valid 0 = Not all bytes written 1 = All bytes written
0	AESBUSY	R	0h	AES accelerator module busy; encryption, decryption, or key generation in progress. 0 = Not busy 1 = Busy

10.3.4 AESAKEY Register

AES Accelerator Key Register

Figure 10-17. AESAKEY Register

15	14	13	12	11	10	9	8
AESKEY1x							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0
7	6	5	4	3	2	1	0
AESKEY0x							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0

Table 10-15. AESAKEY Register Description

Bit	Field	Type	Reset	Description
15-8	AESKEY1x	W	0h	AES key byte n+1 when AESAKEY is written as word. Do not use these bits for byte access. Do not mix word and byte access. Always reads as zero. The key is reset by PUC or by AESSWRST = 1.
7-0	AESKEY0x	W	0h	AES key byte n when AESAKEY is written as word. AES next key byte when AESAKEY_L is written as byte. Do not mix word and byte access. Always reads as zero. The key is reset by PUC or by AESSWRST = 1.

10.3.5 AESADIN Register

AES Accelerator Data In Register

Figure 10-18. AESADIN Register

15	14	13	12	11	10	9	8
AESDIN1x							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0
7	6	5	4	3	2	1	0
AESDIN0x							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0

Table 10-16. AESADIN Register Description

Bit	Field	Type	Reset	Description
15-8	AESDIN1x	W	0h	AES data in byte n+1 when AESADIN is written as word. Do not use these bits for byte access. Do not mix word and byte access. Always reads as zero.
7-0	AESDIN0x	W	0h	AES data in byte n when AESADIN is written as word. AES next data in byte when AESADIN_L is written as byte. Do not mix word and byte access. Always reads as zero.

10.3.6 AESADOUT Register

AES Accelerator Data Out Register

Figure 10-19. AESADOUT Register

15	14	13	12	11	10	9	8
AESDOUT1x							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
AESDOUT0x							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

Table 10-17. AESADOUT Register Description

Bit	Field	Type	Reset	Description
15-8	AESDOUT1x	R	0h	AES data out byte n+1 when AESADOUT is read as word. Do not use these bits for byte access. Do not mix word and byte access.
7-0	AESDOUT0x	R	0h	AES data out byte n when AESADOUT is read as word. AES next data out byte when AESADOUT_L is read as byte. Do not mix word and byte access.

10.3.7 AESAXDIN Register

AES Accelerator XORed Data In Register

Figure 10-20. AESAXDIN Register

15	14	13	12	11	10	9	8
AESXDIN1x							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0
7	6	5	4	3	2	1	0
AESXDIN0x							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0

Table 10-18. AESAXDIN Register Description

Bit	Field	Type	Reset	Description
15-8	AESXDIN1x	W	0h	AES data in byte n+1 when AESAXDIN is written as word. Do not use these bits for byte access. Do not mix word and byte access. Always reads as zero.
7-0	AESXDIN0x	W	0h	AES data in byte n when AESAXDIN is written as word. AES next data in byte when AESAXDIN_L is written as byte. Do not mix word and byte access. Always reads as zero.

10.3.8 AESAXIN Register

AES Accelerator XORed Data In Register (No Trigger)

Figure 10-21. AESAXIN Register

15	14	13	12	11	10	9	8
AESXIN1x							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0
7	6	5	4	3	2	1	0
AESXIN0x							
w-0	w-0	w-0	w-0	w-0	w-0	w-0	w-0

Table 10-19. AESAXIN Register Description

Bit	Field	Type	Reset	Description
15-8	AESXIN1x	W	0h	AES data in byte n+1 when AESAXIN is written as word. Do not use these bits for byte access. Do not mix word and byte access. Always reads as zero.
7-0	AESXIN0x	W	0h	AES data in byte n when AESAXIN is written as word. AES next data in byte when AESAXIN_L is written as byte. Do not mix word and byte access. Always reads as zero.

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.

Topic **Page**

11.1 Cyclic Redundancy Check (CRC) Module Introduction 368
11.2 CRC Standard and Bit Order 368
11.3 CRC Checksum Generation 369
11.4 CRC Registers 372

11.1 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 11-1). The CRC signature is based on the polynomial given in the CRC-CCITT-BR polynomial (see Equation 10) .

$$f(x) = x^{16} + x^{12} + x^5 + 1 \quad (10)$$

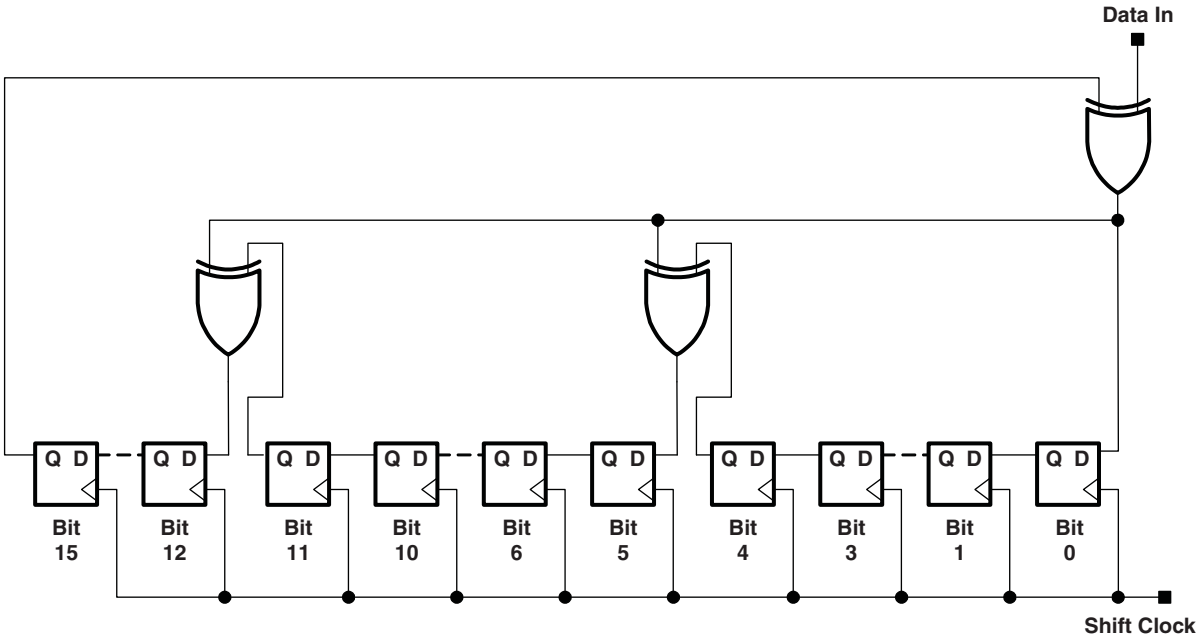


Figure 11-1. LFSR Implementation of CRC-CCITT Standard, Bit 0 is the MSB of the Result

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.

11.2 CRC Standard and Bit Order

The definitions of the various CRC standards were done in the era of main frame computers, and by convention bit 0 was treated as the MSB. Today, as in most microcontrollers such as the MSP430, bit 0 normally denotes the LSB. In , the bit convention shown is as given in the original standards i.e. bit 0 is the MSB. The fact that bit 0 is treated for some as LSB, and for others as MSB, continues to cause confusion. The CRC16 module therefore provides a bit reversed register pair for CRC16 operations to support both conventions.

11.3 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 of 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.

11.3.1 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 (for example, the DMA) can transfer data to the CRCDI or CRCDIRB register (for example, 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 checksum 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.

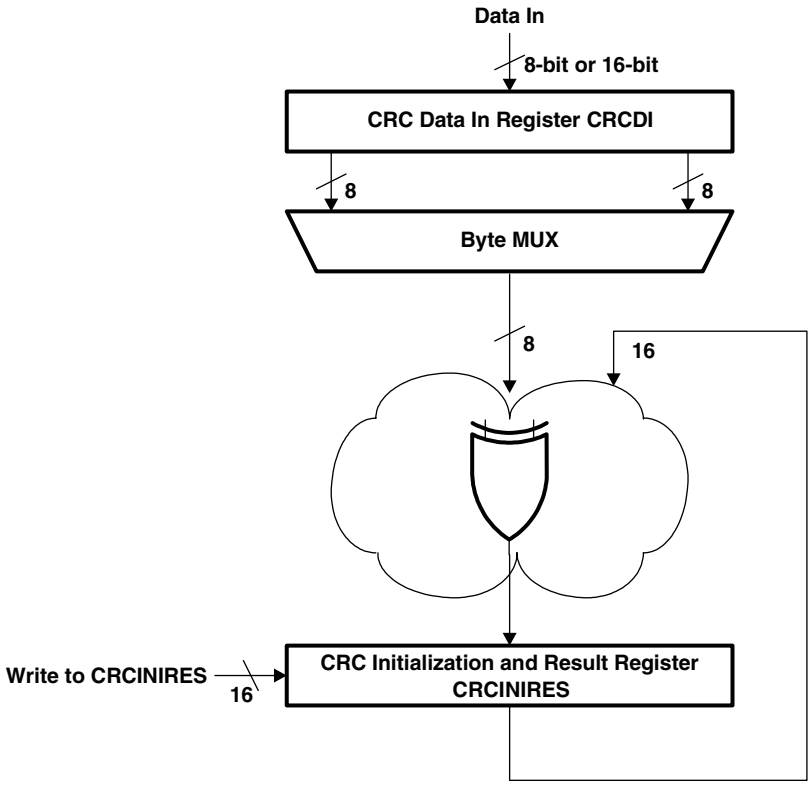


Figure 11-2. Implementation of CRC-CCITT Using the CRCDI and CRCINIRES Registers

11.3.2 Assembler Examples

Example 11-1 demonstrates the operation of the on-chip CRC.

Example 11-1. General Assembler Example

```
   ...
   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
```

The details of the implemented CRC algorithm are shown by the data sequences in Example 11-2 using word or byte accesses and the CRC data-in as well as the CRC data-in reverse byte registers.

Example 11-2. Reference Data Sequence

```
   ...
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
```

11.4 CRC Registers

The CRC module registers are listed in Table 11-1. The base address can be found in the device-specific data sheet. 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. CRC Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	CRCDI	CRC Data In	Read/write	Word	0000h	Section 11.4.1
00h	CRCDI_L		Read/write	Byte	00h	
01h	CRCDI_H		Read/write	Byte	00h	
02h	CRCDIRB	CRC Data In Reverse Byte	Read/write	Word	0000h	Section 11.4.2
02h	CRCDIRB_L		Read/write	Byte	00h	
03h	CRCDIRB_H		Read/write	Byte	00h	
04h	CRCINIRES	CRC Initialization and Result	Read/write	Word	FFFFh	Section 11.4.3
04h	CRCINIRES_L		Read/write	Byte	FFh	
05h	CRCINIRES_H		Read/write	Byte	FFh	
06h	CRCRESR	CRC Result Reverse	Read only	Word	FFFFh	Section 11.4.4
06h	CRCRESR_L		Read/write	Byte	FFh	
07h	CRCRESR_H		Read/write	Byte	FFh	

11.4.1 CRCDI Register

CRC Data In Register

Figure 11-3. CRCDI Register

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

Table 11-2. CRCDI Register Description

Bit	Field	Type	Reset	Description
15-0	CRCDI	RW	0h	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.

11.4.2 CRCDIRB Register

CRC Data In Reverse Register

Figure 11-4. CRCDIRB Register

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

Table 11-3. CRCDIRB Register Description

Bit	Field	Type	Reset	Description
15-0	CRCDIRB	RW	0h	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.

11.4.3 CRCINIRES Register

CRC Initialization and Result Register

Figure 11-5. CRCINIRES Register

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

Table 11-4. CRCINIRES Register Description

Bit	Field	Type	Reset	Description
15-0	CRCINIRES	RW	FFFFh	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.

11.4.4 CRCRESR Register

CRC Reverse Result Register

Figure 11-6. CRCRESR Register

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
CRCRESR							
r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1

Table 11-5. CRCRESR Register Description

Bit	Field	Type	Reset	Description
15-0	CRCRESR	R	FFFFh	CRC reverse result. This register holds the current CRC result (according to the CRC-CCITT standard). The order of bits is reverse (for example, 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**

12.1 WDT_A Introduction **376**
12.2 WDT_A Operation **378**
12.3 WDT_A Registers **380**

12.1 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 12-1.

NOTE: Watchdog timer powers up active.

After a PUC, the WDT_A module is automatically configured in the watchdog mode with an initial approximately 32-ms reset interval using the SMCLK. The user must set up or halt the WDT_A before the initial reset interval expires.

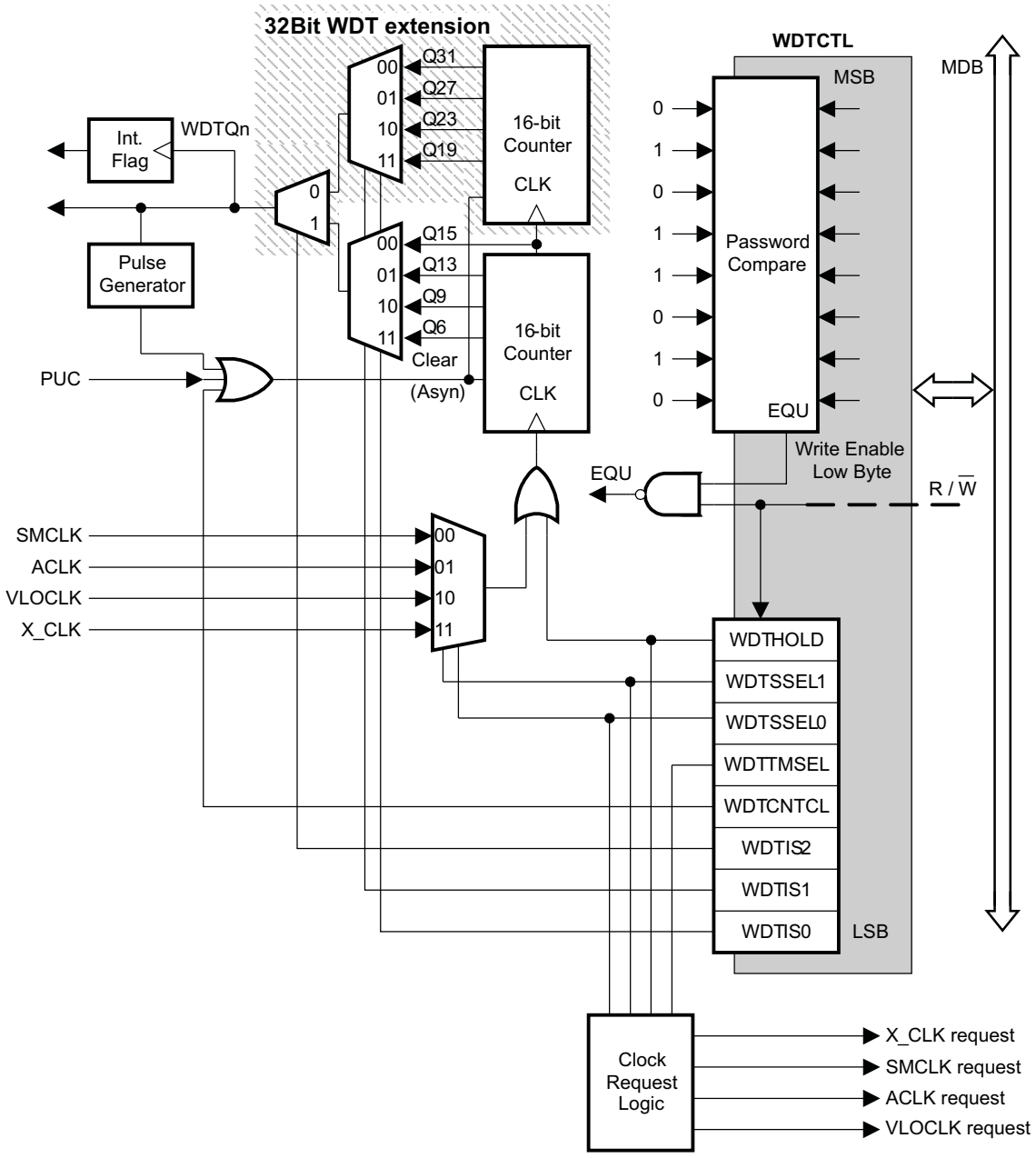


Figure 12-1. Watchdog Timer Block Diagram

12.2 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. A write to WDTCTL with any value other than 05Ah in the upper byte is a password violation and causes 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.

12.2.1 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.

12.2.2 Watchdog Mode

After a PUC condition, the WDT module is configured in the watchdog mode with an initial 32-ms (approximate) reset interval using the SMCLK. The user must set up, halt, or clear the watchdog timer before this initial reset interval expires, 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.

12.2.3 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.

12.2.4 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.

12.2.5 Fail-Safe Features

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.

In watchdog mode the WDT_A prevents LPMx.5 because in LPMx.5 the WDT_A cannot operate.

If SMCLK or ACLK fails as the WDT_A clock source, LFMODOSC clock is automatically selected as the WDT_A clock source.

When the WDT_A module is used in interval timer mode, there are no fail-safe features.

12.2.6 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.

Any write operation to WDTCTL must be a word operation with 05Ah (WDTPW) in the upper byte (see Example 12-1).

Example 12-1. Writes to WDTCTL

```
; 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
```

12.3 WDT_A Registers

The watchdog timer module registers are listed in Table 12-1. The base address for the watchdog timer module registers and special function registers (SFRs) can be found in the device-specific data sheets. The address offset is given in Table 12-1.

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 12-1. WDT_A Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
0Ch	WDTCTL	Watchdog Timer Control	Read/write	Word	6904h	Section 12.3.1
0Ch	WDTCTL_L		Read/write	Byte	04h	
0Dh	WDTCTL_H		Read/write	Byte	69h	

12.3.1 WDTCTL Register

Watchdog Timer Control Register

Figure 12-2. WDTCTL Register

15	14	13	12	11	10	9	8
WDTPW							
rw	rw	rw	rw	rw	rw	rw	rw
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

Table 12-2. WDTCTL Register Description

Bit	Field	Type	Reset	Description
15-8	WDTPW	RW	69h	Watchdog timer password. Always read as 069h. Must be written as 05Ah, or a PUC is generated.
7	WDTHOLD	RW	0h	Watchdog timer hold. This bit stops the watchdog timer. Setting WDTHOLD = 1 when the WDT is not in use conserves power. 0b = Watchdog timer is not stopped 1b = Watchdog timer is stopped
6-5	WDTSSEL	RW	0h	Watchdog timer clock source select 00b = SMCLK 01b = ACLK 10b = VLOCLK 11b = X_CLK , same as VLOCLK if not defined differently in data sheet
4	WDTTMSEL	RW	0h	Watchdog timer mode select 0b = Watchdog mode 1b = Interval timer mode
3	WDTCNTCL	RW	0h	Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset. 0b = No action 1b = WDTCNT = 0000h
2-0	WDTIS	RW	0h	Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag or generate a PUC. 000b = Watchdog clock source / (2^(31)) (18:12:16 at 32.768 kHz) 001b = Watchdog clock source /(2^(27)) (01:08:16 at 32.768 kHz) 010b = Watchdog clock source /(2^(23)) (00:04:16 at 32.768 kHz) 011b = Watchdog clock source /(2^(19)) (00:00:16 at 32.768 kHz) 100b = Watchdog clock source /(2^(15)) (1 s at 32.768 kHz) 101b = Watchdog clock source / (2^(13)) (250 ms at 32.768 kHz) 110b = Watchdog clock source / (2^(9)) (15.625 ms at 32.768 kHz) 111b = Watchdog clock source / (2^(6)) (1.95 ms at 32.768 kHz)

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**

13.1 Timer_A Introduction **383**
13.2 Timer_A Operation **385**
13.3 Timer_A Registers **397**

13.1 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 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_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.

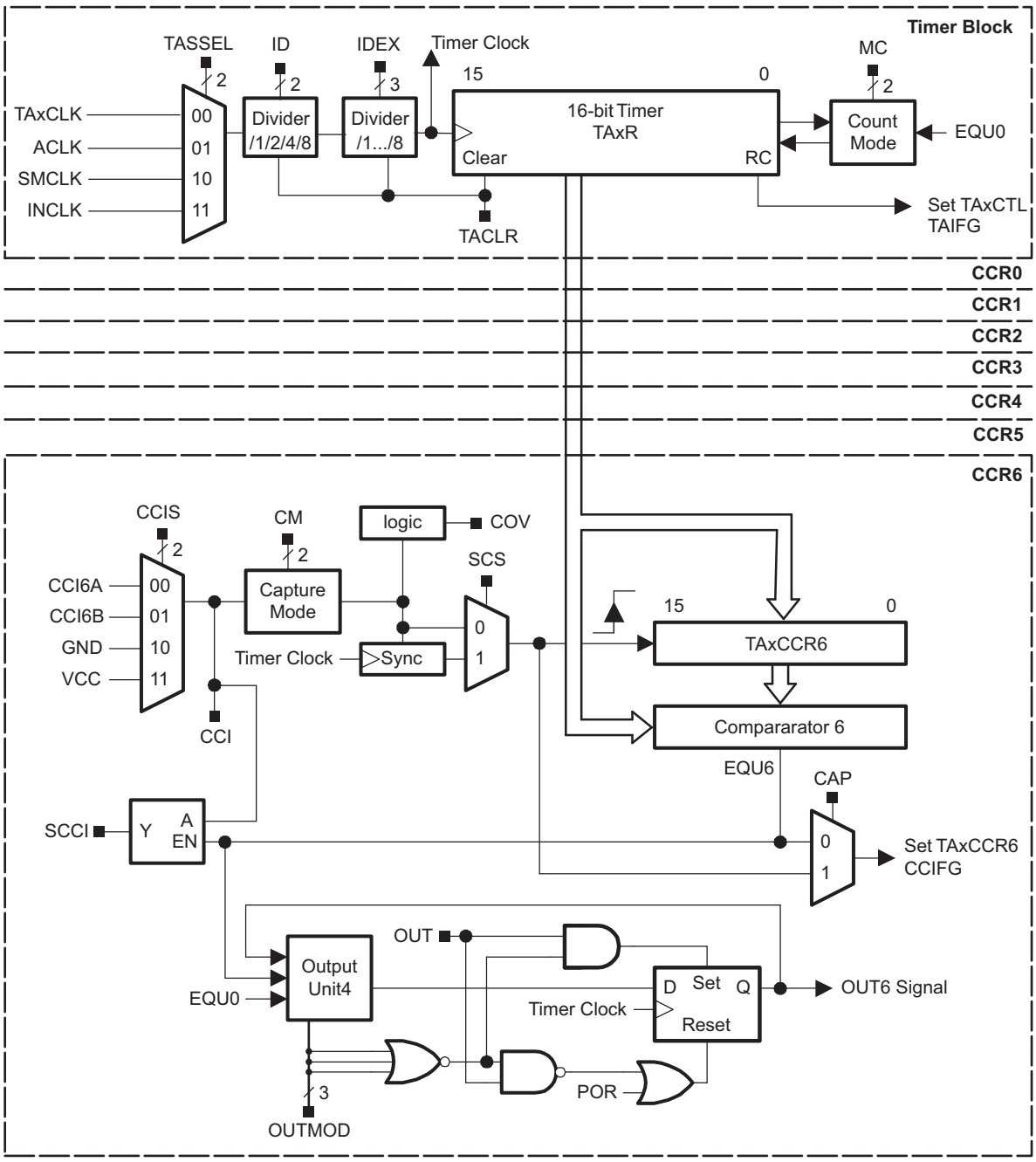


Figure 13-1. Timer_A Block Diagram

13.2 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.

13.2.1 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.

13.2.1.1 Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally via TAxCLK or INCLK. 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 TAIDEX bits. The timer clock divider logic is reset when TACLR is set.

NOTE: Timer_A dividers

After programming ID or TAIDEX bits, set the TACLR bit. This clears the contents of TAxR and resets the clock divider logic to a defined state. 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 TAIDEX 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 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.

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 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.

13.2.3.1 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 13-2). 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.

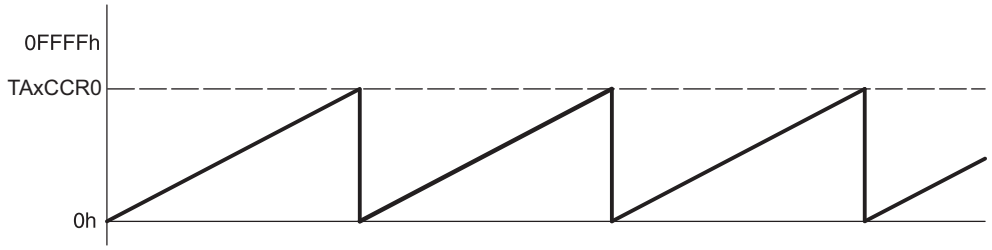


Figure 13-2. Up Mode

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 13-3 shows the flag set cycle.

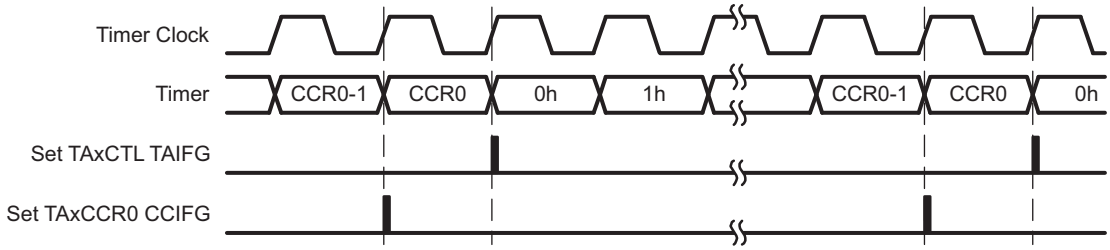


Figure 13-3. Up Mode Flag Setting

13.2.3.1.1 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.

13.2.3.2 Continuous Mode

In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in Figure 13-4. The capture/compare register TAxCCR0 works the same way as the other capture/compare registers.

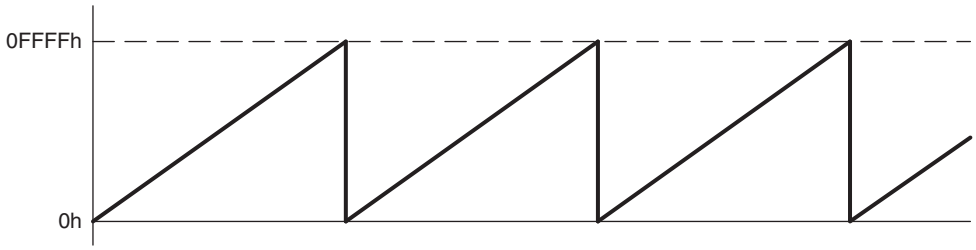


Figure 13-4. Continuous Mode

The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. Figure 13-5 shows the flag set cycle.

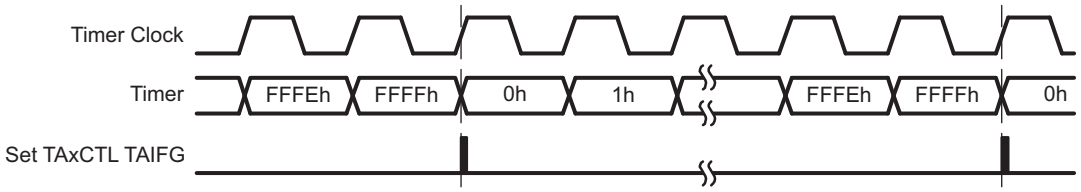


Figure 13-5. Continuous Mode Flag Setting

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 TAxCCRn register 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. 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.

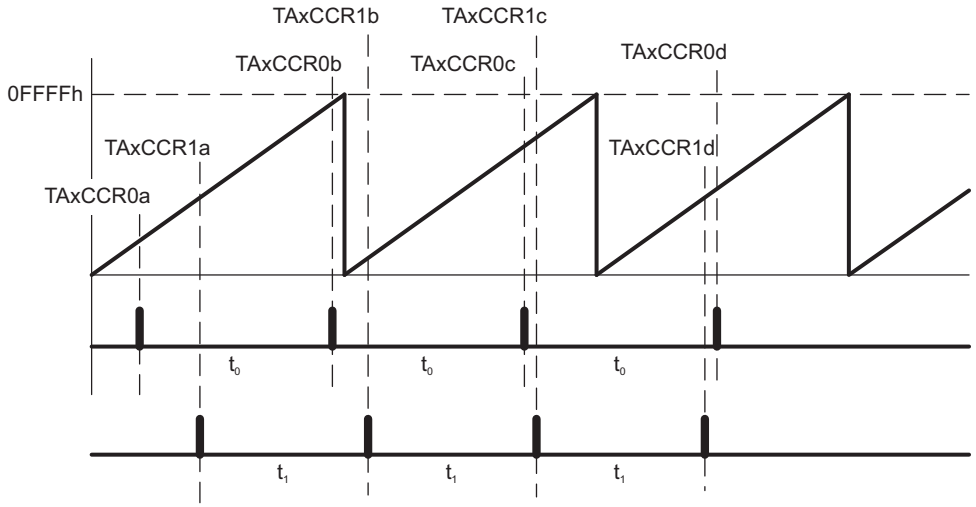


Figure 13-6. Continuous Mode Time Intervals

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.

13.2.3.4 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 13-7). The period is twice the value in TAxCCR0.

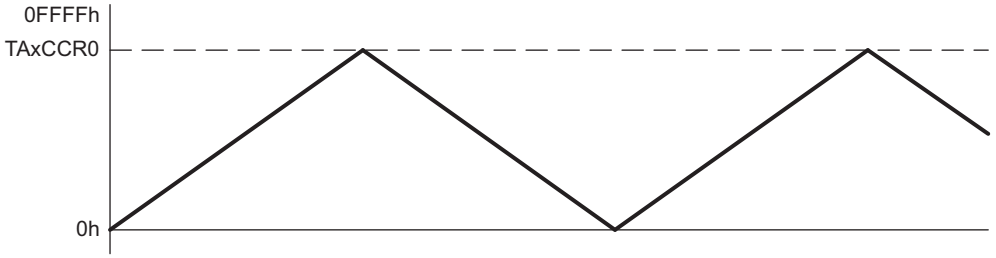


Figure 13-7. Up/Down Mode

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 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 13-8 shows the flag set cycle.

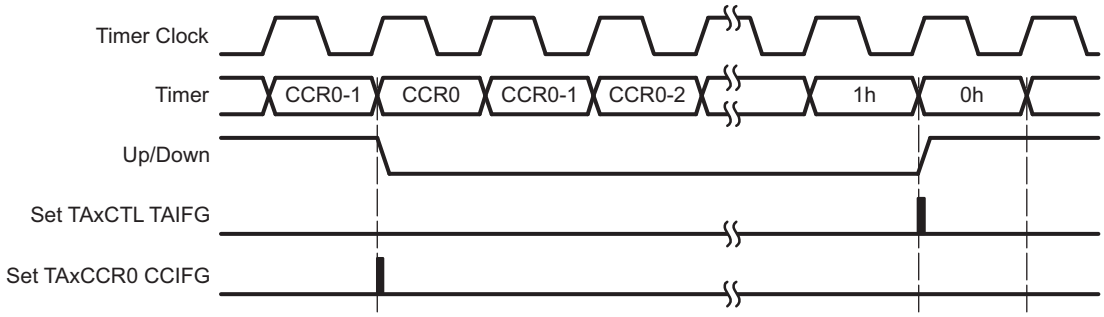


Figure 13-8. Up/Down Mode Flag Setting

13.2.3.4.1 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 effect 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.

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_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 13-9, the t_{dead} is:

$t_{dead} = t_{timer} \times (TAxCCR1 – TAxCCR2)$

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.

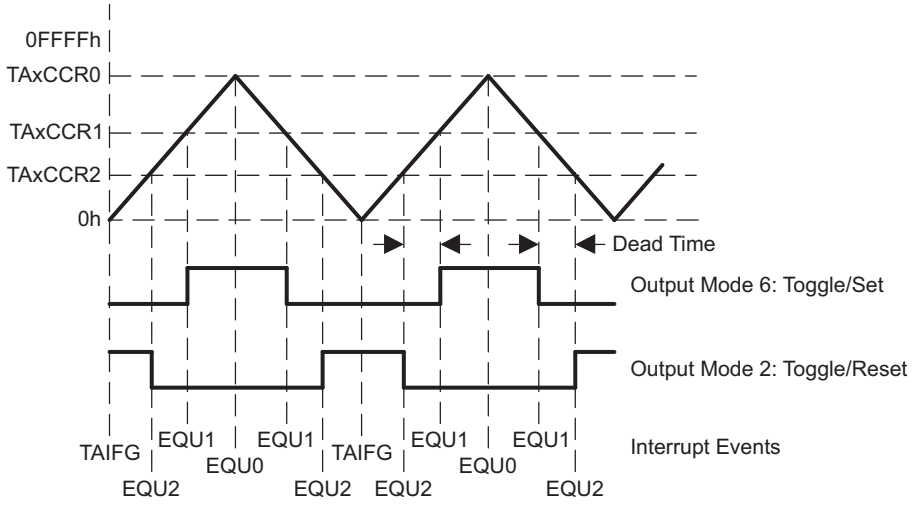


Figure 13-9. Output Unit in Up/Down Mode

13.2.4 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.

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 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 13-10).

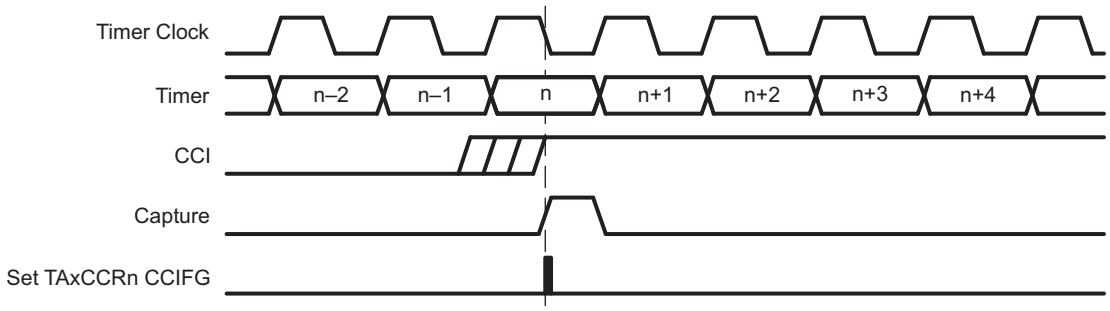


Figure 13-10. Capture Signal (SCS = 1)

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 as shown in Figure 13-11. COV must be reset with software.

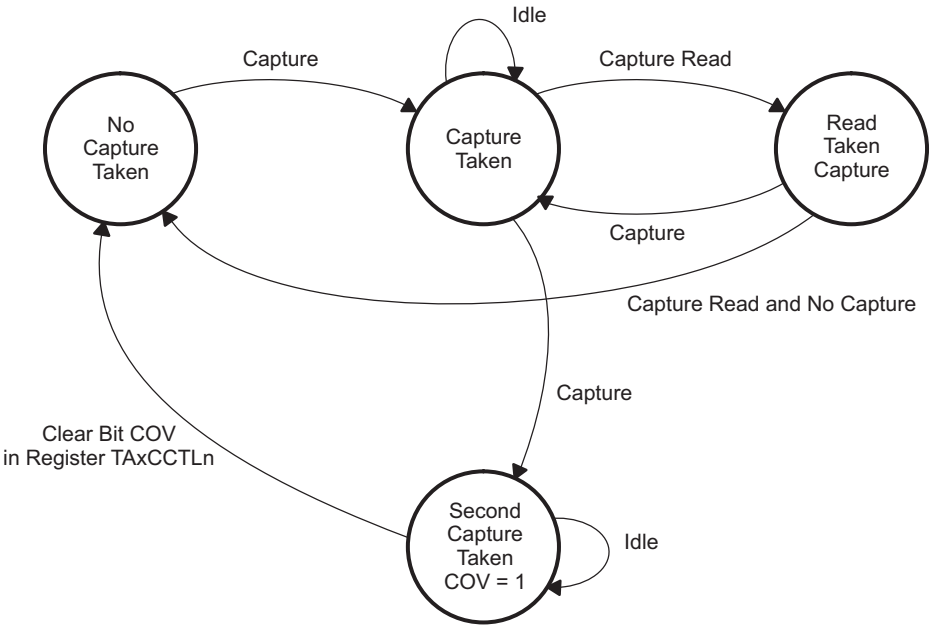


Figure 13-11. Capture Cycle

13.2.4.1.1 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.

13.2.4.2 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.

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.

13.2.5.1 Output Modes

The output modes are defined by the OUTMOD bits and are described in Table 13-2. 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 13-2. 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.

13.2.5.1.1 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 13-12 using TAxCCR0 and TAxCCR1.

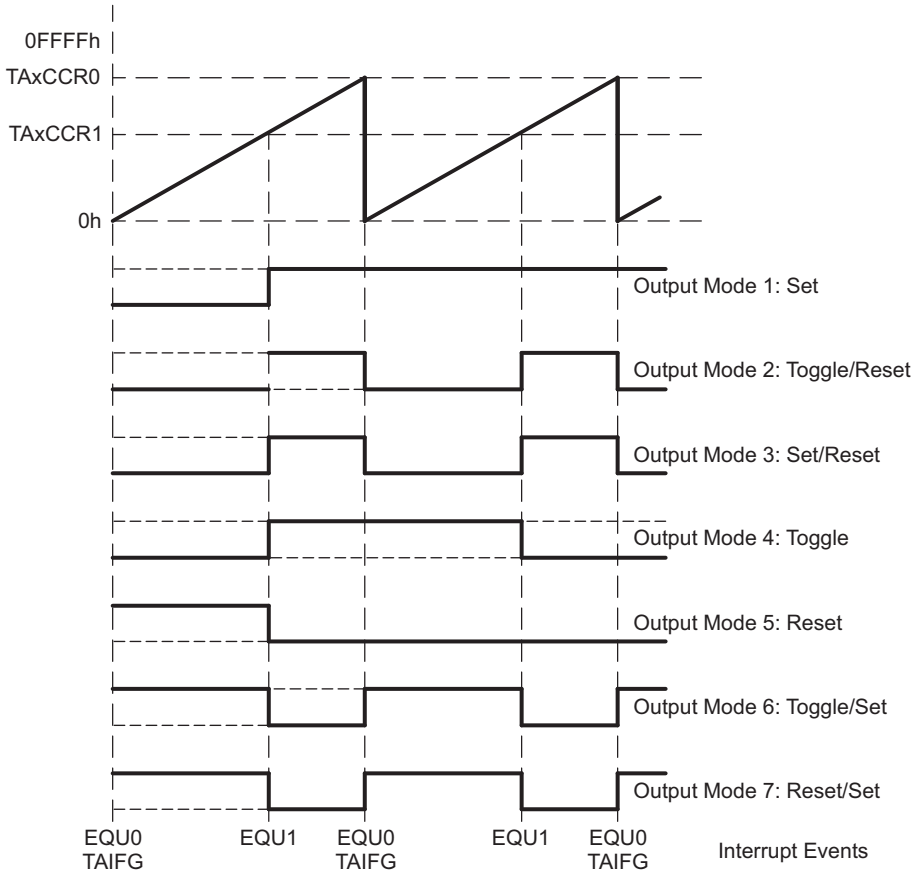


Figure 13-12. Output Example – Timer in Up Mode

13.2.5.1.2 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 13-13 using TAxCCR0 and TAxCCR1.

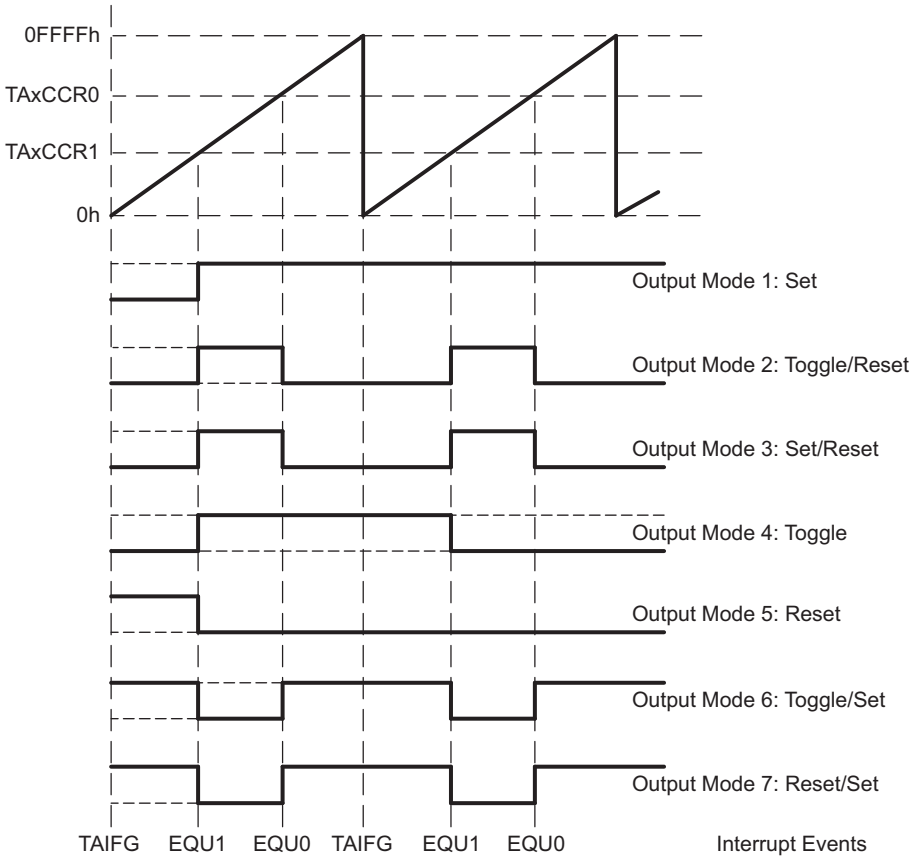


Figure 13-13. Output Example – Timer in Continuous Mode

13.2.5.1.3 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 13-14 using TAxCCR0 and TAxCCR2.

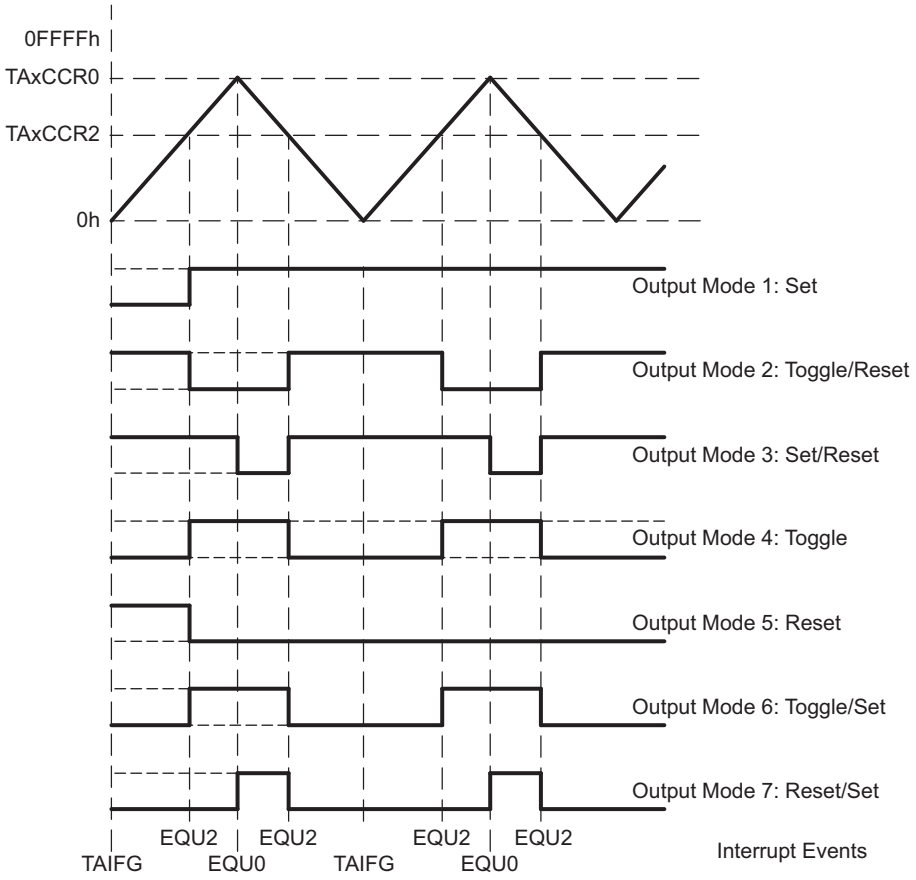


Figure 13-14. Output Example – Timer in Up/Down Mode

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,&TA0CCTL1      ; Set output mode=7
BIC    #OUTMOD,&TA0CCTL1       ; Clear unwanted bits
```

13.2.6 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.

13.2.6.1 TAxCCR0 Interrupt

The TAxCCR0 CCIFG flag has the highest Timer_A interrupt priority and has a dedicated interrupt vector as shown in Figure 13-15. The TAxCCR0 CCIFG flag is automatically reset when the TAxCCR0 interrupt request is serviced.

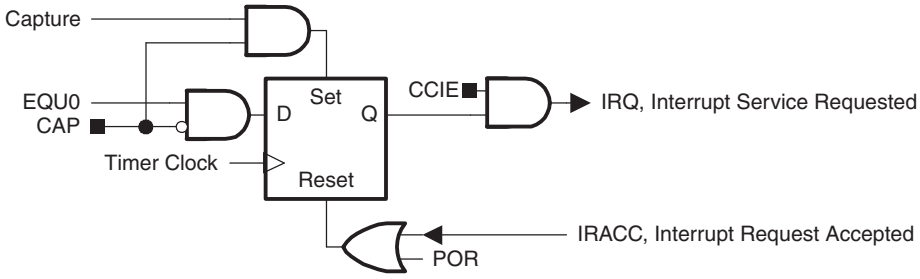


Figure 13-15. Capture/Compare TAxCCR0 Interrupt Flag

13.2.6.2 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.

13.2.6.2.1 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
```

13.3 Timer_A Registers

Timer_A registers are listed in Table 13-3 for the largest configuration available. The base address can be found in the device-specific data sheet.

Table 13-3. Timer_A Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TAxCTL	Timer_Ax Control	Read/write	Word	0000h	Section 13.3.1
02h	TAxCCTL0	Timer_Ax Capture/Compare Control 0	Read/write	Word	0000h	Section 13.3.3
04h	TAxCCTL1	Timer_Ax Capture/Compare Control 1	Read/write	Word	0000h	Section 13.3.3
06h	TAxCCTL2	Timer_Ax Capture/Compare Control 2	Read/write	Word	0000h	Section 13.3.3
08h	TAxCCTL3	Timer_Ax Capture/Compare Control 3	Read/write	Word	0000h	Section 13.3.3
0Ah	TAxCCTL4	Timer_Ax Capture/Compare Control 4	Read/write	Word	0000h	Section 13.3.3
0Ch	TAxCCTL5	Timer_Ax Capture/Compare Control 5	Read/write	Word	0000h	Section 13.3.3
0Eh	TAxCCTL6	Timer_Ax Capture/Compare Control 6	Read/write	Word	0000h	Section 13.3.3
10h	TAxR	Timer_Ax Counter	Read/write	Word	0000h	Section 13.3.2
12h	TAxCCR0	Timer_Ax Capture/Compare 0	Read/write	Word	0000h	Section 13.3.4
14h	TAxCCR1	Timer_Ax Capture/Compare 1	Read/write	Word	0000h	Section 13.3.4
16h	TAxCCR2	Timer_Ax Capture/Compare 2	Read/write	Word	0000h	Section 13.3.4
18h	TAxCCR3	Timer_Ax Capture/Compare 3	Read/write	Word	0000h	Section 13.3.4
1Ah	TAxCCR4	Timer_Ax Capture/Compare 4	Read/write	Word	0000h	Section 13.3.4
1Ch	TAxCCR5	Timer_Ax Capture/Compare 5	Read/write	Word	0000h	Section 13.3.4
1Eh	TAxCCR6	Timer_Ax Capture/Compare 6	Read/write	Word	0000h	Section 13.3.4
2Eh	TAxIV	Timer_Ax Interrupt Vector	Read only	Word	0000h	Section 13.3.5
20h	TAxEX0	Timer_Ax Expansion 0	Read/write	Word	0000h	Section 13.3.6

13.3.1 TAxCTL Register

Timer_Ax Control Register

Figure 13-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						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		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Table 13-4. TAxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit resets TAxR, the timer clock divider logic, and the count direction. The TACLR bit is automatically reset and is always read as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

13.3.2 TAxR Register

Timer_Ax Counter Register

Figure 13-17. TAxR Register

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)

Table 13-5. TAxR Register Description

Bit	Field	Type	Reset	Description
15-0	TAxR	RW	0h	Timer_A register. The TAxR register is the count of Timer_A.

13.3.3 TAxCCTLn Register

Timer_Ax Capture/Compare Control n Register

Figure 13-18. TAxCCTLn Register

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Reserved	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)

Table 13-6. TAxCCTLn Register Description

Bit	Field	Type	Reset	Description
15-14	CM	RW	0h	Capture mode 00b = No capture 01b = Capture on rising edge 10b = Capture on falling edge 11b = Capture on both rising and falling edges
13-12	CCIS	RW	0h	Capture/compare input select. These bits select the TAxCCR0 input signal. See the device-specific data sheet for specific signal connections. 00b = CCIxA 01b = CCIxB 10b = GND 11b = VCC
11	SCS	RW	0h	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0b = Asynchronous capture 1b = Synchronous capture
10	SCCI	RW	0h	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit.
9	Reserved	R	0h	Reserved. Reads as 0.
8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode
7-5	OUTMOD	RW	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 because EQUx = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set
4	CCIE	RW	0h	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0b = Interrupt disabled 1b = Interrupt enabled
3	CCI	R	0h	Capture/compare input. The selected input signal can be read by this bit.
2	OUT	RW	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high

Table 13-6. TAxCCTLn Register Description (continued)

Bit	Field	Type	Reset	Description
1	COV	RW	0h	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0b = No capture overflow occurred 1b = Capture overflow occurred
0	CCIFG	RW	0h	Capture/compare interrupt flag 0b = No interrupt pending 1b = Interrupt pending

13.3.4 TAxCCRn Register

Timer_A Capture/Compare n Register

Figure 13-19. TAxCCRn Register

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

Table 13-7. TAxCCRn Register Description

Bit	Field	Type	Reset	Description
15-0	TAxCCR0	RW	0h	Compare mode: TAxCCRn holds the data for the comparison to the timer value in the Timer_A Register, TAR. Capture mode: The Timer_A Register, TAR, is copied into the TAxCCRn register when a capture is performed.

13.3.5 TAxIV Register

Timer_Ax Interrupt Vector Register

Figure 13-20. TAxIV Register

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

Table 13-8. TAxIV Register Description

Bit	Field	Type	Reset	Description
15-0	TAIV	R	0h	Timer_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Capture/compare 1; Interrupt Flag: TAxCCR1 CCIFG; Interrupt Priority: Highest 04h = Interrupt Source: Capture/compare 2; Interrupt Flag: TAxCCR2 CCIFG 06h = Interrupt Source: Capture/compare 3; Interrupt Flag: TAxCCR3 CCIFG 08h = Interrupt Source: Capture/compare 4; Interrupt Flag: TAxCCR4 CCIFG 0Ah = Interrupt Source: Capture/compare 5; Interrupt Flag: TAxCCR5 CCIFG 0Ch = Interrupt Source: Capture/compare 6; Interrupt Flag: TAxCCR6 CCIFG 0Eh = Interrupt Source: Timer overflow; Interrupt Flag: TAxCTL TAIFG; Interrupt Priority: Lowest

13.3.6 TAxEX0 Register

Timer_Ax Expansion 0 Register

Figure 13-21. TAxEX0 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					TAIDEX(1)		
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

(1) After programming TAIDEX bits and configuration of the timer, set TACLR bit to ensure proper reset of the timer divider logic.

Table 13-9. TAxEX0 Register Description

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Reads as 0.
2-0	TAIDEX	RW	0h	Input divider expansion. These bits along with the ID bits select the divider for the input clock. 000b = Divide by 1 001b = Divide by 2 010b = Divide by 3 011b = Divide by 4 100b = Divide by 5 101b = Divide by 6 110b = Divide by 7 111b = Divide by 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 operaand use of the Timer_B module.

Topic **Page**

14.1 Timer_B Introduction **405**
14.2 Timer_B Operation **407**
14.3 Timer_B Registers **420**

14.1 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 14-1.

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.

14.1.1 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.

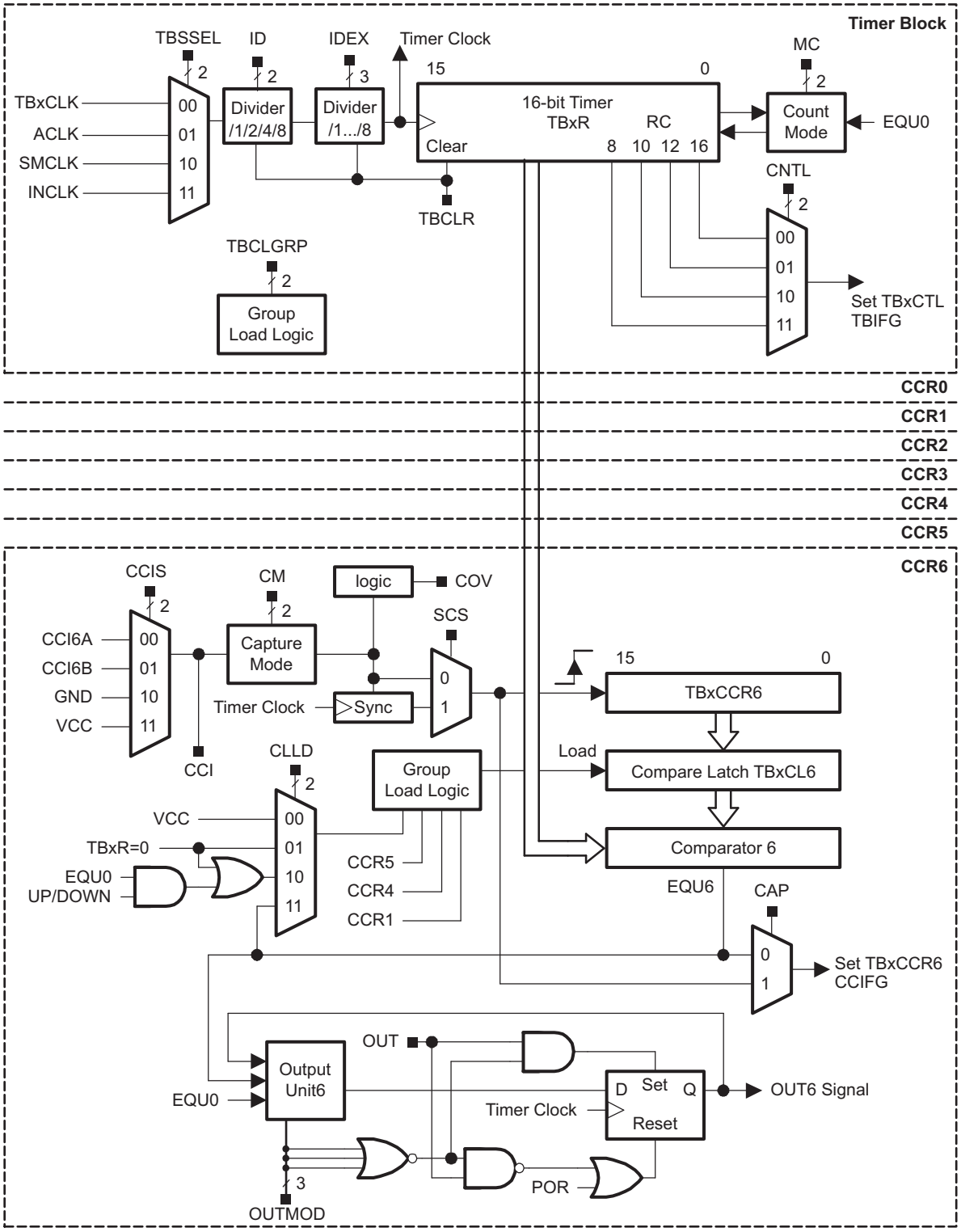


Figure 14-1. Timer_B Block Diagram

14.2 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.

14.2.1 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.

14.2.1.1 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.

14.2.1.2 Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally via TBxCLK or INCLK. 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 TBIDEX bits. The timer clock divider logic is reset when TBCLR is set.

NOTE: Timer_B dividers

After programming ID or TBIDEX bits, set the TBCLR bit. This clears the contents of TBxR and resets the clock divider logic to a defined state. 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 TBIDEX bits.

14.2.2 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.

14.2.3 Timer Mode Control

The timer has four modes of operation: stop, up, continuous, and up/down (see Table 14-1). The operating mode is selected with the MC bits.

Table 14-1. 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.

14.2.3.1 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 14-2). 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.

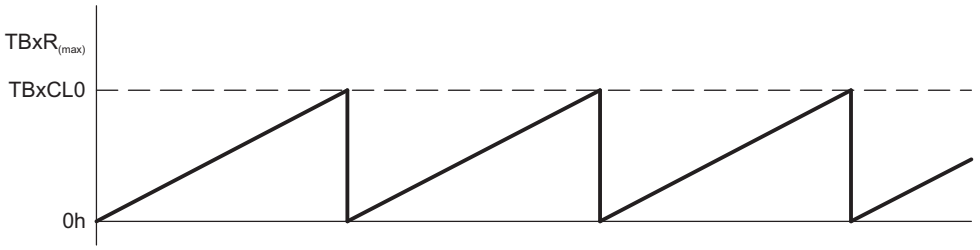


Figure 14-2. Up Mode

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 14-3 shows the flag set cycle.

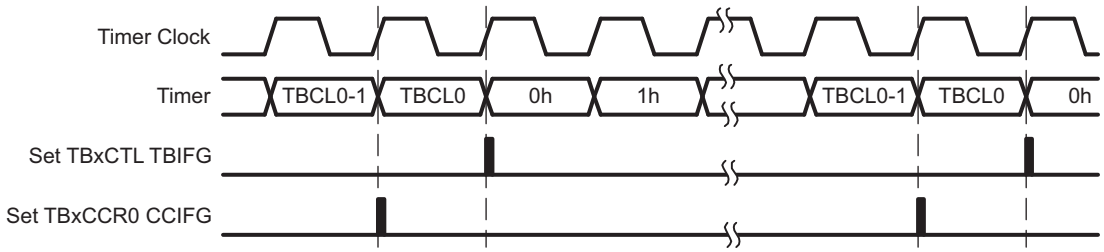


Figure 14-3. Up Mode Flag Setting

14.2.3.1.1 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.

14.2.3.2 Continuous Mode

In continuous mode, the timer repeatedly counts up to $TBxR_{(max)}$ and restarts from zero (see Figure 14-4). The compare latch TBxCL0 works the same way as the other capture/compare registers.

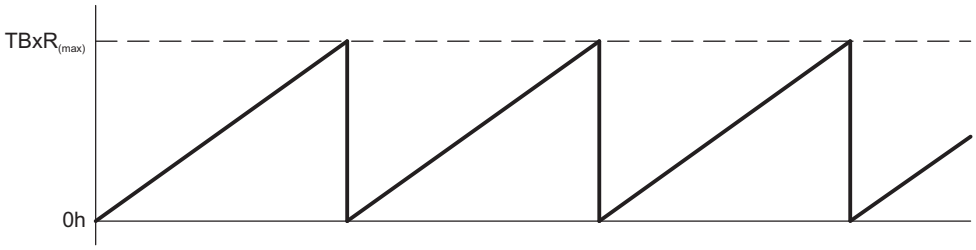


Figure 14-4. Continuous Mode

The TBIFG interrupt flag is set when the timer *counts* from $TBxR_{(max)}$ to zero. Figure 14-5 shows the flag set cycle.

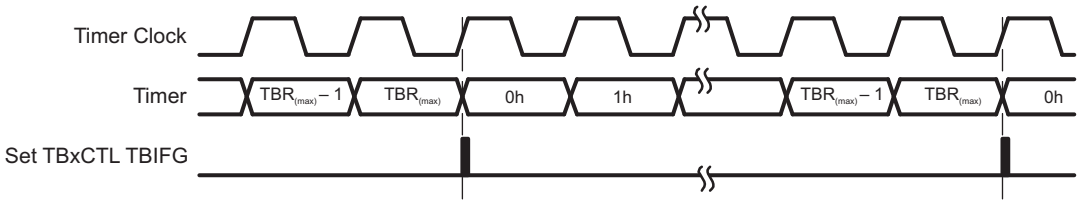


Figure 14-5. Continuous Mode Flag Setting

14.2.3.3 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 14-6 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.

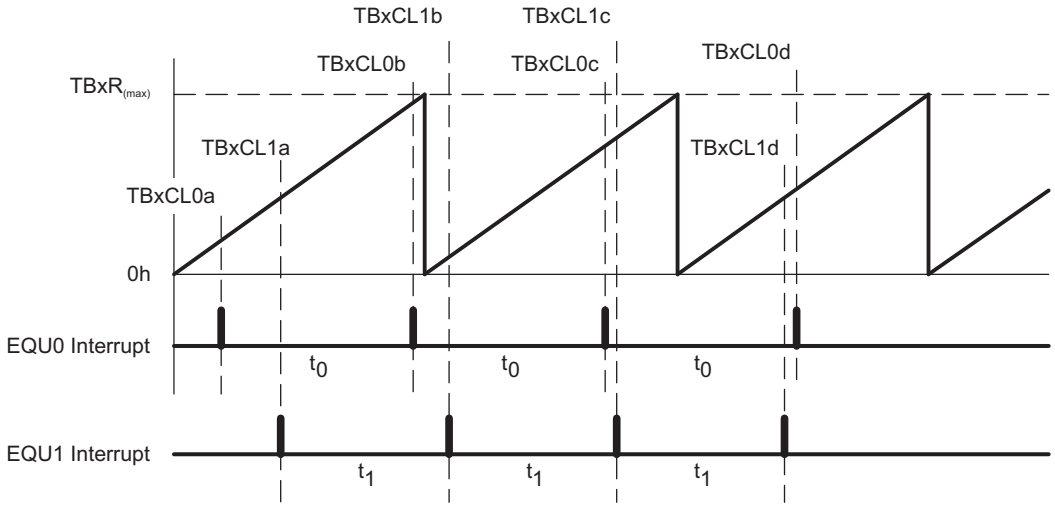


Figure 14-6. Continuous Mode Time Intervals

Time intervals can be produced with other modes as well, where TBxCL0 is used as the period register. Their handling is more complex, because 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.

14.2.3.4 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 14-7). 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.

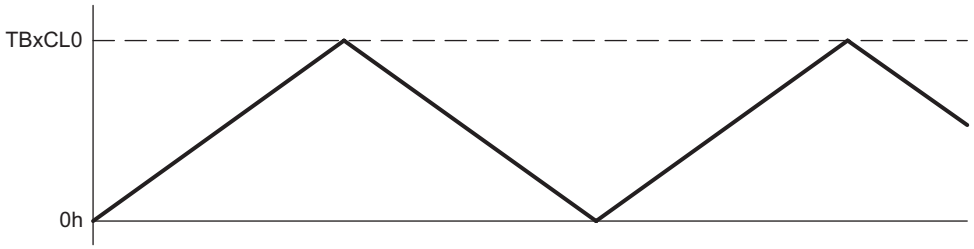


Figure 14-7. Up/Down Mode

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 14-8 shows the flag set cycle.

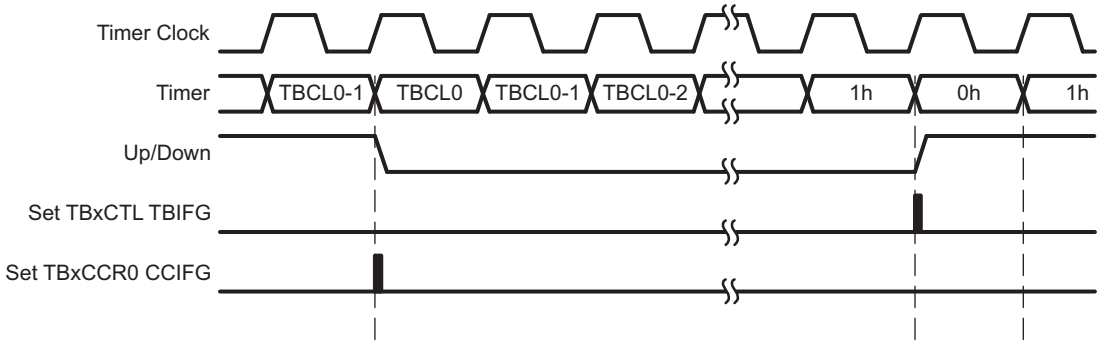


Figure 14-8. Up/Down Mode Flag Setting

14.2.3.4.1 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.

14.2.3.5 Use of Up/Down Mode

The up/down mode supports applications that require dead times between output signals (see Section 14.2.5). 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 14-9, 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.

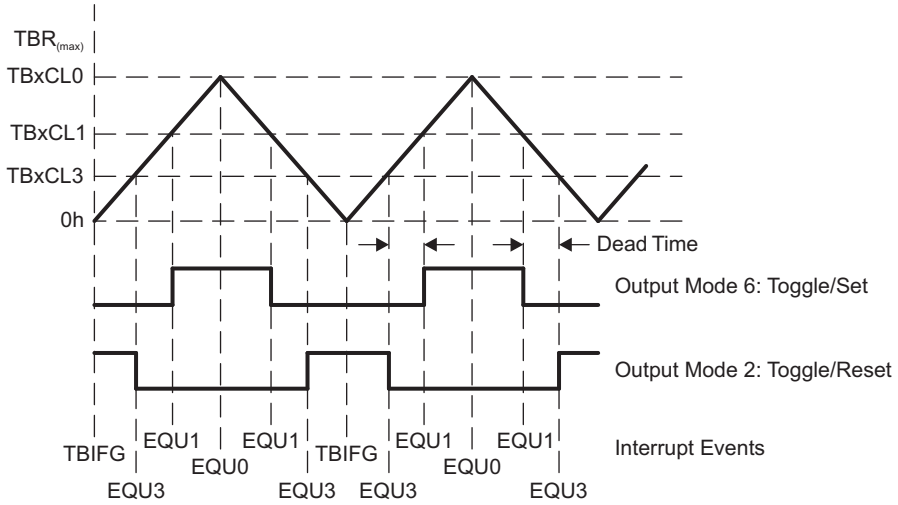


Figure 14-9. Output Unit in Up/Down Mode

14.2.4 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.

14.2.4.1 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. 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 14-10).

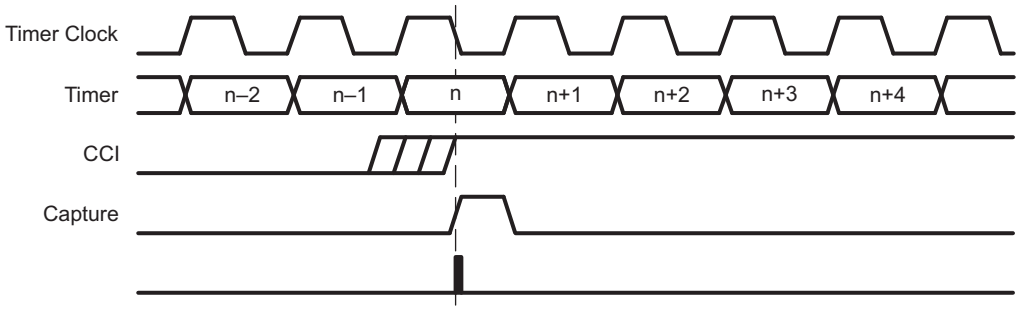


Figure 14-10. Capture Signal (SCS = 1)

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 14-11). COV must be reset with software.

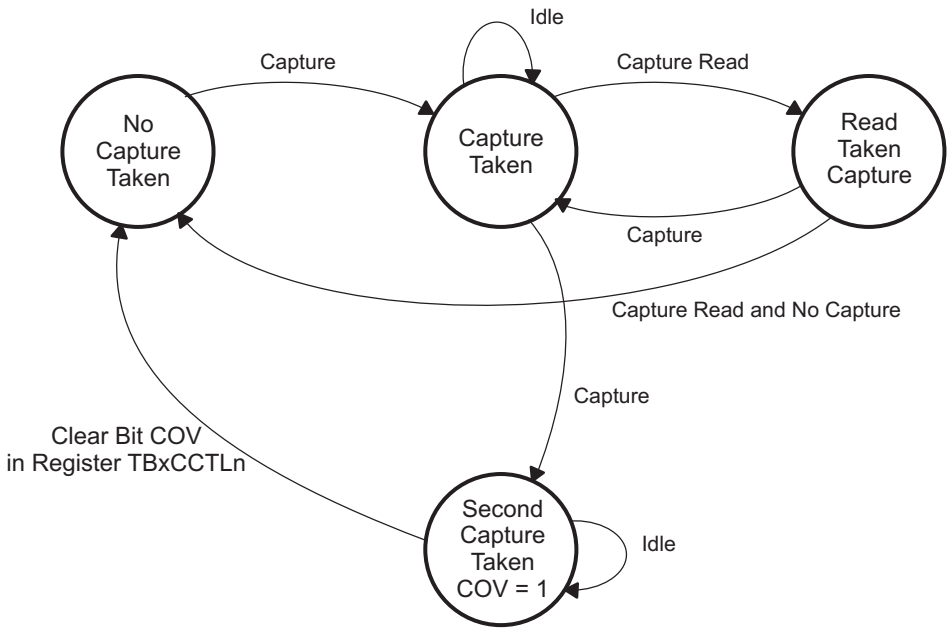


Figure 14-11. Capture Cycle

14.2.4.1.1 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.

14.2.4.2 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.

14.2.4.2.1 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 14-2.

Table 14-2. 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.

14.2.4.2.2 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 14-3). 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 14-3. Compare Latch Operating Modes

TBCLGRPx	Grouping	Update Control
00	None	Individual
01	TBxCL1+TBxCL2 TBxCL3+TBxCL4 TBxCL5+TBxCL6	TBxCCR1 TBxCCR3 TBxCCR5
10	TBxCL1+TBxCL2+TBxCL3 TBxCL4+TBxCL5+TBxCL6	TBxCCR1 TBxCCR4
11	TBxCL0+TBxCL1+TBxCL2+TBxCL3+TBxCL4+TBxCL5+TBxCL6	TBxCCR1

14.2.5 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.

14.2.5.1 Output Modes

The output modes are defined by the OUTMOD bits and are described in Table 14-4. 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 14-4. 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.

14.2.5.1.1 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 14-12 using TBxCL0 and TBxCL1.

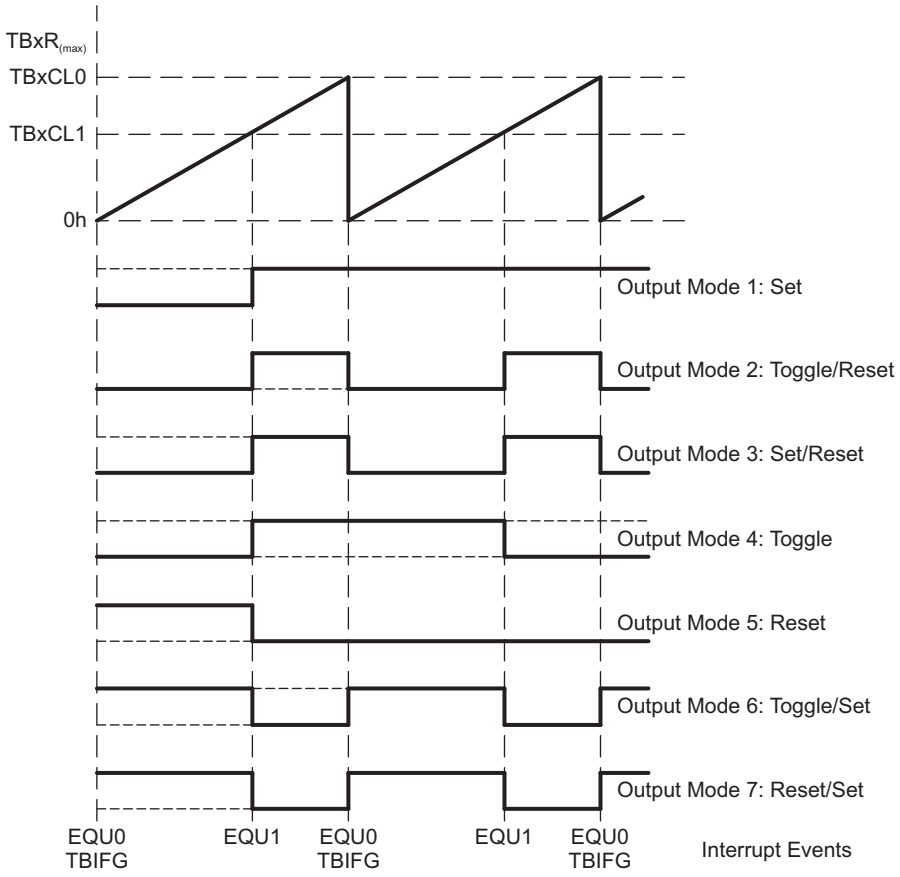


Figure 14-12. Output Example – Timer in Up Mode

14.2.5.1.2 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 14-13 using TBxCL0 and TBxCL1.

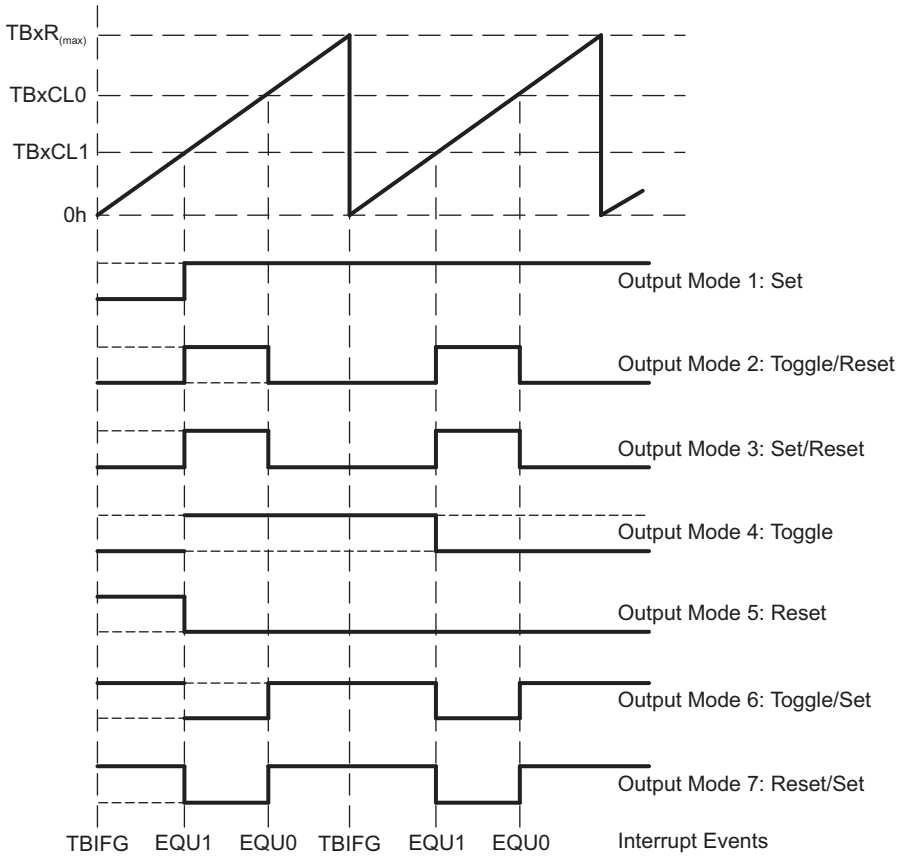


Figure 14-13. Output Example – Timer in Continuous Mode

14.2.5.1.3 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 14-14 using TBxCL0 and TBxCL3.

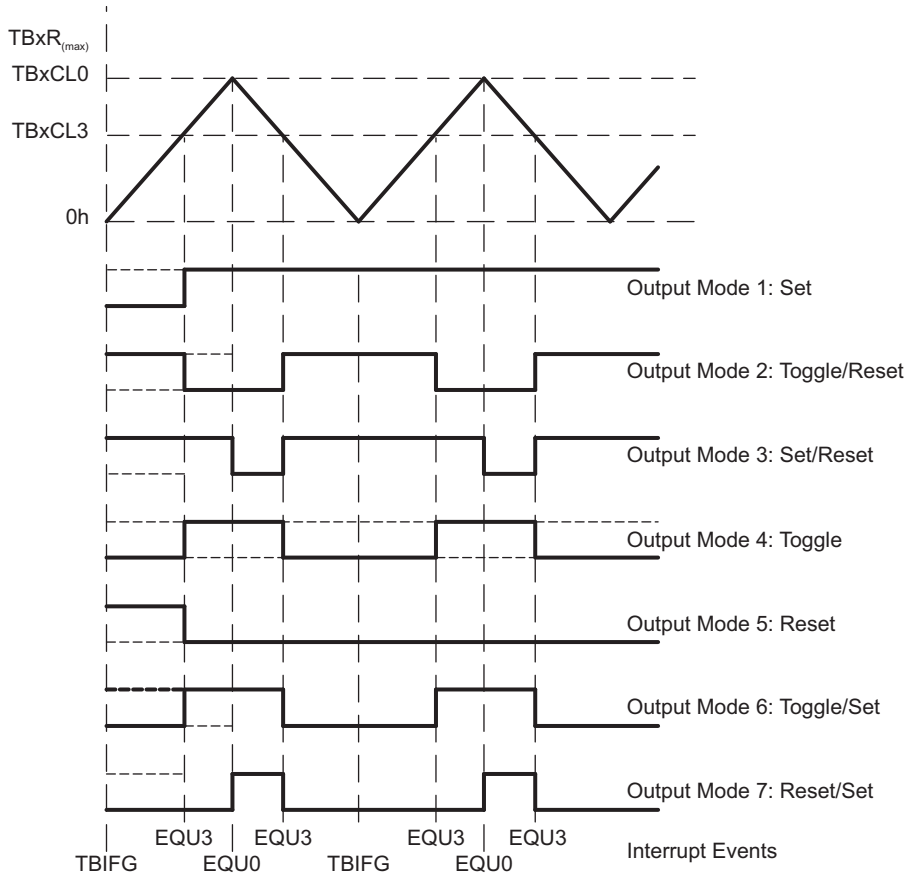


Figure 14-14. Output Example – Timer in Up/Down Mode

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
```

14.2.6 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.

14.2.6.1 TBxCCR0 Interrupt Vector

The TBxCCR0 CCIFG flag has the highest Timer_B interrupt priority and has a dedicated interrupt vector (see Figure 14-15). The TBxCCR0 CCIFG flag is automatically reset when the TBxCCR0 interrupt request is serviced.

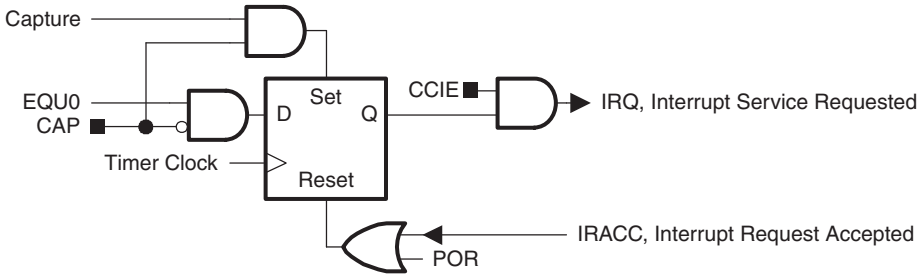


Figure 14-15. Capture/Compare TBxCCR0 Interrupt Flag

14.2.6.2 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.

14.2.6.3 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
```

14.3 Timer_B Registers

The Timer_B registers are listed in Table 14-5. The base address can be found in the device-specific data sheet. The address offset is listed in Table 14-5.

Table 14-5. Timer_B Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TBxCTL	Timer_B Control	Read/write	Word	0000h	Section 14.3.1
02h	TBxCCTL0	Timer_B Capture/Compare Control 0	Read/write	Word	0000h	Section 14.3.3
04h	TBxCCTL1	Timer_B Capture/Compare Control 1	Read/write	Word	0000h	Section 14.3.3
06h	TBxCCTL2	Timer_B Capture/Compare Control 2	Read/write	Word	0000h	Section 14.3.3
08h	TBxCCTL3	Timer_B Capture/Compare Control 3	Read/write	Word	0000h	Section 14.3.3
0Ah	TBxCCTL4	Timer_B Capture/Compare Control 4	Read/write	Word	0000h	Section 14.3.3
0Ch	TBxCCTL5	Timer_B Capture/Compare Control 5	Read/write	Word	0000h	Section 14.3.3
0Eh	TBxCCTL6	Timer_B Capture/Compare Control 6	Read/write	Word	0000h	Section 14.3.3
10h	TBxR	Timer_B Counter	Read/write	Word	0000h	Section 14.3.2
12h	TBxCCR0	Timer_B Capture/Compare 0	Read/write	Word	0000h	Section 14.3.4
14h	TBxCCR1	Timer_B Capture/Compare 1	Read/write	Word	0000h	Section 14.3.4
16h	TBxCCR2	Timer_B Capture/Compare 2	Read/write	Word	0000h	Section 14.3.4
18h	TBxCCR3	Timer_B Capture/Compare 3	Read/write	Word	0000h	Section 14.3.4
1Ah	TBxCCR4	Timer_B Capture/Compare 4	Read/write	Word	0000h	Section 14.3.4
1Ch	TBxCCR5	Timer_B Capture/Compare 5	Read/write	Word	0000h	Section 14.3.4
1Eh	TBxCCR6	Timer_B Capture/Compare 6	Read/write	Word	0000h	Section 14.3.4
2Eh	TBxIV	Timer_B Interrupt Vector	Read only	Word	0000h	Section 14.3.5
20h	TBxEX0	Timer_B Expansion 0	Read/write	Word	0000h	Section 14.3.6

14.3.1 TBxCTL Register

Timer_B x Control Register

Figure 14-16. TBxCTL Register

15	14	13	12	11	10	9	8
Reserved	TBCLGRPx		CNTL		Reserved	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		Reserved	TBCLR	TBIE	TBIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Table 14-6. TBxCTL Register Description

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14-13	TBCLGRP	RW	0h	TBxCLn group 00b = Each TBxCLn latch loads independently. 01b = 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 10b = TBxCL1+TBxCL2+TBxCL3 (TBxCCR1 CLLD bits control the update); TBxCL4+TBxCL5+TBxCL6 (TBxCCR4 CLLD bits control the update); TBxCL0 independent 11b = TBxCL0+TBxCL1+TBxCL2+TBxCL3+TBxCL4+TBxCL5+TBxCL6 (TBxCCR1 CLLD bits control the update)
12-11	CNTL	RW	0h	Counter length 00b = 16-bit, TBxR(max) = 0FFFFh 01b = 12-bit, TBxR(max) = 0FFFh 10b = 10-bit, TBxR(max) = 03FFh 11b = 8-bit, TBxR(max) = 0FFh
10	Reserved	R	0h	Reserved. Always reads as 0.
9-8	TBSSEL	RW	0h	Timer_B clock source select 00b = TBxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits, along with the TBIDEX bits, select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MC = 00h when Timer_B is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TBxCL0 10b = Continuous mode: Timer counts up to the value set by CNTL 11b = Up/down mode: Timer counts up to TBxCL0 and down to 0000h
3	Reserved	R	0h	Reserved. Always reads as 0.
2	TBCLR	RW	0h	Timer_B clear. Setting this bit resets TBxR, the timer clock divider logic, and the count direction. The TBCLR bit is automatically reset and is always read as zero.
1	TBIE	RW	0h	Timer_B interrupt enable. This bit enables the TBIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled

Table 14-6. TBxCTL Register Description (continued)

Bit	Field	Type	Reset	Description
0	TBIFG	RW	0h	Timer_B interrupt flag 0b = No interrupt pending 1b = Interrupt pending

14.3.2 TBxR Register

Timer_B x Counter Register

Figure 14-17. TBxR Register

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)

Table 14-7. TBxR Register Description

Bit	Field	Type	Reset	Description
15-0	TBxR	RW	0h	Timer_B register. The TBxR register is the count of Timer_B.

14.3.3 TBxCCTLn Register

Timer_B x Capture/Compare Control Register n

Figure 14-18. TBxCCTLn Register

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)

Table 14-8. TBxCCTLn Register Description

Bit	Field	Type	Reset	Description
15-14	CM	RW	0h	Capture mode 00b = No capture 01b = Capture on rising edge 10b = Capture on falling edge 11b = Capture on both rising and falling edges
13-12	CCIS	RW	0h	Capture/compare input select. These bits select the TBxCCRn input signal. See the device-specific data sheet for specific signal connections. 00b = CCIxA 01b = CCIxB 10b = GND 11b = VCC
11	SCS	RW	0h	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0b = Asynchronous capture 1b = Synchronous capture
10-9	CLLD	RW	0h	Compare latch load. These bits select the compare latch load event. 00b = TBxCLn loads on write to TBxCCRn 01b = TBxCLn loads when TBxR counts to 0 10b = TBxCLn loads when TBxR counts to 0 (up or continuous mode). TBxCLn loads when TBxR counts to TBxCL0 or to 0 (up/down mode). 11b = TBxCLn loads when TBxR counts to TBxCLn
8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode
7-5	OUTMOD	RW	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TBxCL0 because EQUn = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set
4	CCIE	RW	0h	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0b = Interrupt disabled 1b = Interrupt enabled
3	CCI	R	Undef	Capture/compare input. The selected input signal can be read by this bit.

Table 14-8. TBxCCTLn Register Description (continued)

Bit	Field	Type	Reset	Description
2	OUT	RW	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high
1	COV	RW	0h	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0b = No capture overflow occurred 1b = Capture overflow occurred
0	CCIFG	RW	0h	Capture/compare interrupt flag 0b = No interrupt pending 1b = Interrupt pending

14.3.4 TBxCCRn Register

Timer_B x Capture/Compare Register n

Figure 14-19. TBxCCRn Register

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

Table 14-9. TBxCCRn Register Description

Bit	Field	Type	Reset	Description
15-0	TBxCCRn	RW	0h	Timer_B capture/compare register. Compare mode: TBxCCRn holds the data for the comparison to the timer value in the Timer_B Register, TBR. Capture mode: The Timer_B Register, TBR, is copied into the TBxCCRn register when a capture is performed.

14.3.5 TBxIV Register

Timer_B x Interrupt Vector Register

Figure 14-20. TBxIV Register

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

Table 14-10. TBxIV Register Description

Bit	Field	Type	Reset	Description
15-0	TBIV	R	0h	Timer_B interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Capture/compare 1; Interrupt Flag: TBxCCR1 CCIFG; Interrupt Priority: Highest 04h = Interrupt Source: Capture/compare 2; Interrupt Flag: TBxCCR2 CCIFG 06h = Interrupt Source: Capture/compare 3; Interrupt Flag: TBxCCR3 CCIFG 08h = Interrupt Source: Capture/compare 4; Interrupt Flag: TBxCCR4 CCIFG 0Ah = Interrupt Source: Capture/compare 5; Interrupt Flag: TBxCCR5 CCIFG 0Ch = Interrupt Source: Capture/compare 6; Interrupt Flag: TBxCCR6 CCIFG 0Eh = Interrupt Source: Timer overflow; Interrupt Flag: TBxCTL TBIFG; Interrupt Priority: Lowest

14.3.6 TBxEX0 Register

Timer_B x Expansion Register 0

Figure 14-21. TBxEX0 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					TBIDEX(1)		
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

(1) After programming TAIDEX bits and configuration of the timer, set TACLR bit to ensure proper reset of the timer divider logic.

Table 14-11. TBxEX0 Register Description

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	TBIDEX	RW	0h	Input divider expansion. These bits along with the ID bits select the divider for the input clock. 000b = Divide by 1 001b = Divide by 2 010b = Divide by 3 011b = Divide by 4 100b = Divide by 5 101b = Divide by 6 110b = Divide by 7 111b = Divide by 8

Real-Time Clock B (RTC_B)

The real-time clock RTC_B module provides clock counters with calendar mode, a flexible programmable alarm, and calibration. Note that the RTC_B supports only calendar mode and not counter mode. The RTC_B also support operation in LPMx.5. See the device-specific data sheet for the supported features. This chapter describes the RTC_B module.

Topic **Page**

15.1 Real-Time Clock RTC_B Introduction **430**
15.2 RTC_B Operation **432**
15.3 RTC_B Registers **437**

15.1 Real-Time Clock RTC_B Introduction

The RTC_B module provides configurable clock counters.

RTC_B features include:

- Real-time clock and calendar mode providing seconds, minutes, hours, day of week, day of month, month, and year (including leap year correction)

  Note that only the calendar mode is supported by RTC_B; the counter mode that is available in some other RTC modules is not supported.
- Interrupt capability
- Selectable BCD or binary format
- Programmable alarms
- Calibration logic for time offset correction
- Operation in LPMx.5

The RTC_B block diagram for devices supporting LPMx.5 is shown in Figure 15-1.

NOTE: Real-time clock initialization

Most RTC_B module registers have no initial condition. These registers must be configured by user software before use.

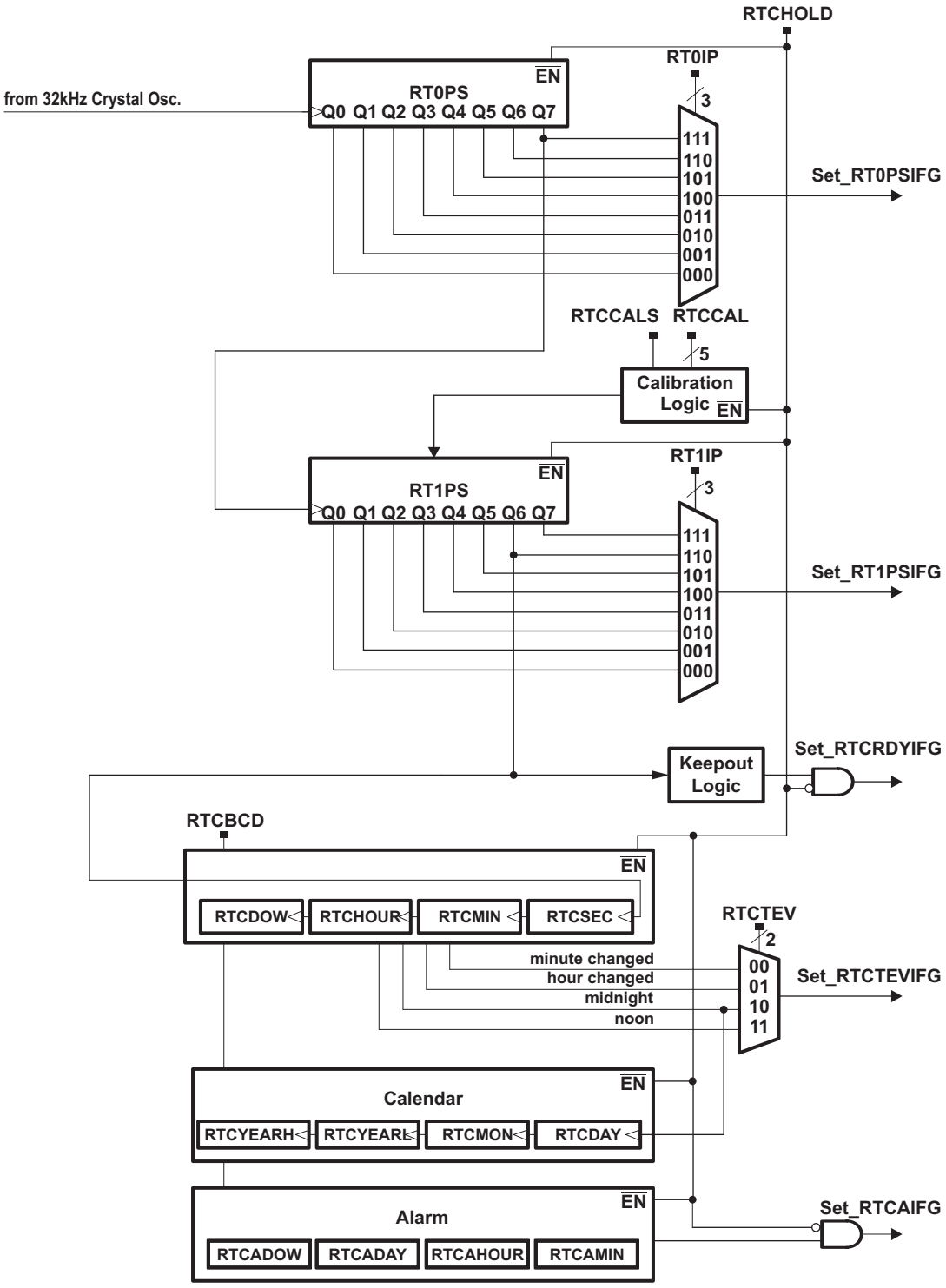


Figure 15-1. RTC_B Block Diagram

15.2 RTC_B Operation

The RTC_B 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.

15.2.1 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_B. The low-frequency oscillator must be operated at 32768 Hz (nominal) for proper RTC_B operation. RT0PS is sourced from the low-frequency oscillator XT1. The output of RT0PS / 256 (Q7) is used to source RT1PS. RT1PS is further divider and the /128 output sources the real-time clock counter registers providing the required 1-second time interval.

When RTCBCD = 1, BCD format is selected for the calendar registers.It is possible to switch between BCD and hexadecimal format while the RTC is counting.

Setting RTCHOLD halts the real-time counters and prescale counters, RT0PS, and RT1PS.

15.2.2 Real-Time Clock Alarm Function

The RTC_B 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.

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 (that is, at 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 RTCAIFG 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, and so on.
- 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 RTCAIFG 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 RTCAIFG is set when 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 RTCAIFG is set when 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 RTCAIFG is set when the time count transitions from 06:29:59 to 06:30:00 and the RTCDAY equals 5.

NOTE: Setting the alarm

Prior to setting an initial alarm, all alarm registers including the AE bits should be cleared.

To prevent potential erroneous alarm conditions from occurring, the alarms should be disabled by clearing the RTCAIE, RTCAIFG, and AE bits prior to writing initial or new time values to the RTC time registers.

NOTE: Invalid alarm settings

Invalid alarm settings are not checked via hardware. It is the user's responsibility that valid alarm settings are entered.

NOTE: Invalid time and date values

Writing of invalid date or time information or data values outside the legal ranges specified in the RTCSEC, RTCMIN, RTCHOUR, RTCDAY, RTCDOW, RTCYEAR, RTCAMIN, RTCAHOUR, RTCADAY, and RTCADOW registers can result in unpredictable behavior.

15.2.3 Reading or Writing Real-Time Clock Registers

Because the system clock may in fact be asynchronous to the RTC_B clock source, special care must be used when accessing the real-time clock registers.

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 keep-out window is provided. The keep-out window is centered approximately 128/32768 seconds around the update transition. The read-only RTCRDY bit is reset during the keep-out window period and set outside the keep-out 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 utilize 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 it can be reset with software.

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, or RTCYEAR 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 second during a write. Writing of data outside the legal ranges or invalid time stamp combinations results in unpredictable behavior.

15.2.4 Real-Time Clock Interrupts

Six sources for interrupts are available, namely RT0PSIFG, RT1PSIFG, RTCRDYIFG, RTCTEVIFG, RTCAIFG, and RTCOFIFG. 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_B 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. RT0PS is sourced with low-frequency oscillator clock 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 be used to generate interrupt intervals selectable by the RT1IP bits. 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.

NOTE: Changing RT0IP or RT1IP

Changing the settings of the interrupt interval bits RT0IP or RT1IP while the corresponding pre-scaler is running or is stopped in a non-zero state can result in setting the corresponding interrupt flags.

The RTCOFIFG bit flags a failure of the 32-kHz crystal oscillator. Its main purpose is to wake up the CPU from LPM3.5 if an oscillator failure occurs.

15.2.4.1 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_HND    ; Vector 6: RTCAIFG          5
           JMP RT0PSIFG_HND   ; Vector 8: RT0PSIFG         5
           JMP RT1PSIFG_HND   ; Vector A: RT1PSIFG         5
           JMP RTCOFIFG_HND   ; Vector C: RTCOFIFG         5
           RETI               ; Vector E: Reserved         5

RTCRDYIFG_HND                 ; Vector 2: RTCRDYIFG Flag
           ...                ; Task starts here
           RETI               ; Back to main program       5

RTCTEVIFG_HND                 ; Vector 4: RTCTEVIFG Flag
           ...                ; Task starts here
           RETI               ; Back to main program       5

RTCAIFG_HND                   ; Vector 6: RTCAIFG Flag
           ...                ; Task starts here
           RETI               ; Back to main program       5

RT0PSIFG_HND                  ; Vector 8: RT0PSIFG Flag
```

```
            ...                 ; Task starts here
            RETI                ; Back to main program      5

RT1PSIFG_HND                    ; Vector A: RT1PSIFG Flag
            ...                 ; Task starts here
            RETI                ; Back to main program      5

RTCOFIFG_HND                    ; Vector C: RTCOFIFG Flag
            ...                 ; Task starts here
            RETI                ; Back to main program      5
```

15.2.5 Real-Time Clock Calibration

The RTC_B module has calibration logic that allows for adjusting the crystal frequency in approximately +4-ppm or –2-ppm steps, allowing for higher time keeping accuracy from standard crystals. The RTCCALx bits are used to adjust the frequency. When RTCCALS is set, each RTCCALx LSB causes a ≈ +4-ppm adjustment. When RTCCALS is cleared, each RTCCALx LSB causes a ≈ –2-ppm adjustment.

Calibration is accomplished by periodically adjusting the RT1PS counter based on the RTCCALS and RTCCALx settings. The RT0PS divides the nominal 37268-Hz low-frequency (LF) crystal clock input by 256. A 60-minute period has 32768 cycles/sec × 60 sec/min × 60 min = 117964800 cycles. Therefore, a –2-ppm reduction in frequency (down calibration) approximately equates to adding an additional 256 cycles every 117964800 cycles (256/117964800 = 2.17 ppm). This is accomplished by holding the RT1PS counter for one additional clock of the RT0PS output within a 60-minute period. Similarly, a +4-ppm increase in frequency (up calibration) approximately equates to removing 512 cycles every 117964800 cycle (512/117964800 = 4.34 ppm). This is accomplished by incrementing the RT1PS counter for two additional clocks of the RT0PS output within a 60-minute period. Each RTCCALx calibration bit causes either 256 LF crystal clock cycles to be added every 60 minutes or 512 LF crystal clock cycles to be subtracted every 60 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. RTCCALF bits can be used to select the frequency rate of the output signal, either no signal, 512 Hz, 256 Hz, or 1 Hz.

The basic flow to calibrate the frequency is as follows:

1. Configure the RTCCLK pin.
2. Measure the RTCCLK output signal with an appropriate resolution frequency counter ; that is, within the resolution required.
3. Compute the absolute error in ppm: Absolute error (ppm) = $|10^6 (f_{MEASURED} - f_{RTCCLK})/f_{RTCCLK}|$, where f_{RTCCLK} is the expected frequency of 512 Hz, 256 Hz, or 1 Hz.
4. Adjust the frequency by performing the following:
   (a) If the frequency is too low, set RTCCALS = 1 and apply the appropriate RTCCALx bits, where RTCCALx = (Absolute Error) / 4.34 rounded to the nearest integer
   (b) If the frequency is too high, clear RTCCALS = 0 and apply the appropriate RTCCALx bits, where RTCCALx = (Absolute Error) / 2.17 rounded to the nearest integer

For example, assume that RTCCLK is configured to output at a frequency of 512 Hz. The measured RTCCLK is 511.9658 Hz. This frequency error is approximately 66.8 ppm too low. To increase the frequency by 66.8 ppm, RTCCALS would be set, and RTCCALx would be set to 15 (66.8 / 4.34). Similarly, assume that 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 11 (24.4 / 2.17).

The calibration corrects only initial offsets and does not adjust for temperature and aging effects. These effects can be handled by periodically measuring temperature and using the crystal's characteristic curve to adjust the ppm based on temperature, as required.

NOTE: Minimum Possible Calibration

The minimal calibration possible is -4 ppm or +8 ppm. For example, setting RTCCALS = 0 and RTCCAL = 0h would result in a -4 ppm decrease in frequency. Similarly, setting RTCCALS = 1 and RTCCAL = 0h would result in a +8 ppm increase in frequency.

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, because these output frequencies are generated prior to the calibration logic. The 1-Hz output frequency is affected by changes in the calibration settings. Because the frequency change is small and infrequent over a very long time interval, it can be difficult to observe.

15.2.6 Real-Time Clock Operation in LPMx.5 Low-Power Mode

The regulator of the Power Management Module (PMM) is disabled upon entering LPMx.5, which causes most of the RTC_B configuration registers to be lost; only the counters are retained. Table 15-1 lists the retained registers in LPMx.5. Also the configuration of the interrupts is stored so that the configured interrupts can cause a wakeup upon exit from LPMx.5. Interrupt flags that are set prior to entering LPM3.5 are cleared upon entering LPM3.5 (Note: this can only happen if the corresponding interrupt is not enabled). The interrupt flags RTCTEVIFG, RTCAIFG, RT1PSIFG, and RTCOFIFG can be used as RTC_B wake-up interrupt sources. After restoring the configuration registers (and clearing LOCKLPM5) the interrupts can be serviced as usual. The detailed flow is as follows:

1. Set all I/Os to general purpose I/Os and configure as needed. Optionally configure input interrupt pins for wake-up. Configure RTC_B interrupts for wake-up (set RTCTEVIE, RTCAIE, RT1PSIE, or RTCOFIE. If the alarm interrupt is also used as wake-up event, the alarm registers must be configured as needed).
2. Enter LPMx.5 with LPMx.5 entry sequence.

```
MOV #PMMKEY + PMMREGOFF, &PMMCTL0 ; Open PMM registers for write and set PMMREGOFF
                                  ;
BIS #LPM4,SR                      ; Enter LPMx.5 when PMMREGOFF is set
```

3. LOCKLPM5 is automatically set by hardware upon entering LPMx.5, the core voltage regulator is disabled, and all clocks are disabled except for the 32-kHz crystal oscillator clock if the RTC is enabled with RTCHOLD = 0.
4. An LPMx.5 wake-up event, such as an edge on a wake-up input pin, are an RTC_B interrupt event and start the BOR entry sequence together with the core voltage regulator. All peripheral registers are set to their default conditions. The I/O pin state remains locked as well as the interrupt configuration for the RTC_B.
5. The device can be configured. The I/O configuration and the RTC_B interrupt configuration that was not retained during LPMx.5 should be restored to the values prior to entering LPMx.5. Then the LOCKLPM5 bit can be cleared, this releases the I/O pin conditions as well as the RTC_B interrupt configuration.
6. After enabling I/O and RTC_B interrupts, the interrupt that caused the wake-up can be serviced.
7. To re-enter LPMx.5, the LOCKLPM5 bit must be cleared prior to re-entry, otherwise LPMx.5 is not entered.

If the RTC is enabled (RTCHOLD = 0), the 32-kHz oscillator remains active during LPMx.5. The fault detection also remains functional. If a fault occurs during LPMx.5 and the RTCOFIE was set before entering LPMx.5, a wake-up event is issued.

15.3 RTC_B Registers

The RTC_B module registers are listed in Table 15-1. This table also lists the retention during LPMx.5. Registers that are not retained during LPMx.5 must be restored after exit from LPMx.5. The base address for the RTC_B module registers can be found in the device-specific data sheet. The address offsets are given in Table 15-1.

NOTE: Most 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 15-1. RTC_B Registers

Offset	Acronym	Register Name	Type	Access	Reset	LPMx.5 and Backup Operation
00h	RTCCTL01	Real-Time Clock Control 0, 1	Read/write	Word	7000h	not retained
00h	RTCCTL0 or RTCCTL01_L	Real-Time Clock Control 0	Read/write	Byte	00h	not retained
01h	RTCCTL1 or RTCCTL01_H	Real-Time Clock Control 1	Read/write	Byte	70h	not retained
02h	RTCCTL23	Real-Time Clock Control 2, 3	Read/write	Word	0000h	retained
02h	RTCCTL2 or RTCCTL23_L	Real-Time Clock Control 2	Read/write	Byte	00h	retained
03h	RTCCTL3 or RTCCTL23_H	Real-Time Clock Control 3	Read/write	Byte	00h	retained
08h	RTCPS0CTL	Real-Time Prescale Timer 0 Control	Read/write	Word	0000h	not retained
08h	RTCPS0CTLL or RTCPS0CTL_L		Read/write	Byte	00h	not retained
09h	RTCPS0CTLH or RTCPS0CTL_H		Read/write	Byte	00h	not retained
0Ah	RTCPS1CTL	Real-Time Prescale Timer 1 Control	Read/write	Word	0000h	not retained
0Ah	RTCPS1CTLL or RTCPS1CTL_L		Read/write	Byte	00h	not retained
0Bh	RTCPS0CTLH or RTCPS0CTL_H		Read/write	Byte	00h	not retained
0Ch	RTCPS	Real-Time Prescale Timer 0, 1 Counter	Read/write	Word	none	retained
0Ch	RT0PS or RTCPS_L	Real-Time Prescale Timer 0 Counter	Read/write	Byte	none	retained
0Dh	RT1PS or RTCPS_H	Real-Time Prescale Timer 1 Counter	Read/write	Byte	none	retained
0Eh	RTCIV	Real Time Clock Interrupt Vector	Read	Word	0000h	not retained
10h	RTCTIM0	Real-Time Clock Seconds, Minutes	Read/write	Word	undefined	retained
10h	RTCSEC or RTCTIM0_L	Real-Time Clock Seconds	Read/write	Byte	undefined	retained
11h	RTCMIN or RTCTIM0_H	Real-Time Clock Minutes	Read/write	Byte	undefined	retained
12h	RTCTIM1	Real-Time Clock Hour, Day of Week	Read/write	Word	undefined	retained
12h	RTCHOUR or RTCTIM1_L	Real-Time Clock Hour	Read/write	Byte	undefined	retained
13h	RTCDOW or RTCTIM1_H	Real-Time Clock Day of Week	Read/write	Byte	undefined	retained

Table 15-1. RTC_B Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	LPMx.5 and Backup Operation
14h	RTCDATE	Real-Time Clock Date	Read/write	Word	undefined	retained
14h	RTCDAY or RTCDATE_L	Real-Time Clock Day of Month	Read/write	Byte	undefined	retained
15h	RTCMON or RTCDATE_H	Real-Time Clock Month	Read/write	Byte	undefined	retained
16h	RTCYEAR	Real-Time Clock Year(1)	Read/write	Word	undefined	retained
18h	RTCAMINHR	Real-Time Clock Minutes, Hour Alarm	Read/write	Word	undefined	retained
18h	RTCAMIN or RTCAMINHR_L	Real-Time Clock Minutes Alarm	Read/write	Byte	undefined	retained
19h	RTCAHOUR or RTCAMINHR_H	Real-Time Clock Hours Alarm	Read/write	Byte	undefined	retained
1Ah	RTCADOWDAY	Real-Time Clock Day of Week, Day of Month Alarm	Read/write	Word	undefined	retained
1Ah	RTCADOW or RTCADOWDAY_L	Real-Time Clock Day of Week Alarm	Read/write	Byte	undefined	retained
1Bh	RTCADAY or RTCADOWDAY_H	Real-Time Clock Day of Month Alarm	Read/write	Byte	undefined	retained
1Ch	BIN2BCD	Binary-to-BCD Conversion Register	Read/write	Word	00h	not retained
1Eh	BCD2BIN	BCD-to-Binary Conversion Register	Read/write	Word	00h	not retained

(1) Do not access the RTCYEAR register in byte mode.

15.3.1 RTCCTL0 Register

Real-Time Clock Control 0 Register

Figure 15-2. RTCCTL0 Register

7	6	5	4	3	2	1	0
RTCOFIE[1]	RTCTEVIE[1]	RTCAIE[1]	RTCRDYIE	RTCOFIFG	RTCTEVIFG	RTCAIFG	RTCRDYIFG
rw-0	rw-0	rw-0	rw-0	rw-(0)	rw-(0)	rw-(0)	rw-(0)

(1) The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

Table 15-2. RTCCTL0 Register Description

Bit	Field	Type	Reset	Description
7	RTCOFIE	RW	0h	32-kHz crystal oscillator fault interrupt enable. This interrupt can be used as LPMx.5 wake-up event. 0b = Interrupt not enabled 1b = Interrupt enabled (LPMx.5 wake-up enabled)
6	RTCTEVIE	RW	0h	Real-time clock time event interrupt enable. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0b = Interrupt not enabled 1b = Interrupt enabled (LPMx.5 wake-up enabled)
5	RTCAIE	RW	0h	Real-time clock alarm interrupt enable. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0b = Interrupt not enabled 1b = Interrupt enabled (LPMx.5 wake-up enabled)
4	RTCRDYIE	RW	0h	Real-time clock ready interrupt enable. 0b = Interrupt not enabled 1b = Interrupt enabled
3	RTCOFIFG	RW	0h	32-kHz crystal oscillator fault interrupt flag. This interrupt can be used as LPMx.5 wake-up event. It also indicates a clock failure during backup operation. 0b = No interrupt pending 1b = Interrupt pending. A 32-kHz crystal oscillator fault occurred after last reset.
2	RTCTEVIFG	RW	0h	Real-time clock time event interrupt flag. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0b = No time event occurred 1b = Time event occurred
1	RTCAIFG	RW	0h	Real-time clock alarm interrupt flag. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0b = No time event occurred 1b = Time event occurred
0	RTCRDYIFG	RW	0h	Real-time clock ready interrupt flag 0b = RTC cannot be read safely 1b = RTC can be read safely

15.3.2 RTCCTL1 Register

Real-Time Clock Control Register 1

Figure 15-3. RTCCTL1 Register

7	6	5	4	3	2	1	0
RTCBCD	RTCHOLD(1)	Reserved	RTCRDY	Reserved		RTCTEVx(1)	
rw-(0)	rw-(1)	r1	r-(1)	r0	r0	rw-(0)	rw-(0)

(1) The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

Table 15-3. RTCCTL1 Register Description

Bit	Field	Type	Reset	Description
7	RTCBCD	RW	0h	Real-time clock BCD select. Selects BCD counting for real-time clock. 0b = Binary-hexadecimal code selected 1b = BCD Binary coded decimal (BCD) code selected
6	RTCHOLD	RW	1h	Real-time clock hold 0b = Real-time clock is operational. 1b = The calendar is stopped as well as the prescale counters, RT0PS, and RT1PS.
5	Reserved	R	1h	Reserved. Always read as 1.
4	RTCRDY	RW	1h	Real-time clock ready 0b = RTC time values in transition 1b = RTC time values safe for reading. This bit indicates when the real-time clock time values are safe for reading.
3-2	Reserved	R	0h	Reserved. Always read as 0.
1-0	RTCTEVx	RW	0h	Real-time clock time interrupt event 00b = Minute changed 01b = Hour changed 10b = Every day at midnight (00:00) 11b = Every day at noon (12:00)

15.3.3 RTCCTL2 Register

Real-Time Clock Control 2 Register

Figure 15-4. RTCCTL2 Register

7	6	5	4	3	2	1	0
RTCCALS	Reserved	RTCCALx					
rw-(0)	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 15-4. RTCCTL2 Register Description

Bit	Field	Type	Reset	Description
7	RTCCALS	RW	0h	Real-time clock calibration sign 0b = Frequency adjusted down 1b = Frequency adjusted up
6	Reserved	R	0h	Reserved. Always read as 0.
5-0	RTCCALx	RW	0h	Real-time clock calibration. Each LSB represents approximately +4-ppm (RTCCALS = 1) or a –2-ppm (RTCCALS = 0) adjustment in frequency.

15.3.4 RTCCTL3 Register

Real-Time Clock Control 3 Register

Figure 15-5. RTCCTL3 Register

7	6	5	4	3	2	1	0
Reserved						RTCCALFx	
r0	r0	r0	r0	r0	r0	rw-(0)	rw-(0)

Table 15-5. RTCCTL3 Register Description

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always read as 0.
1-0	RTCCALFx	RW	0h	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. 00b = No frequency output to RTCCLK pin 01b = 512 Hz 10b = 256 Hz 11b = 1 Hz

15.3.5 RTCSEC Register – Hexadecimal Format

Real-Time Clock Seconds Register – Hexadecimal Format

Figure 15-6. RTCSEC Register

7	6	5	4	3	2	1	0
0	0	Seconds					
r-0	r-0	rw	rw	rw	rw	rw	rw

Table 15-6. RTCSEC Register Description

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always reads as 0.
5-0	Seconds	RW	undefined	Seconds. Valid values are 0 to 59.

15.3.6 RTCSEC Register – BCD Format

Real-Time Clock Seconds Register – BCD Format

Figure 15-7. RTCSEC Register

7	6	5	4	3	2	1	0
0	Seconds – high digit			Seconds – low digit			
r-0	rw	rw	rw	rw	rw	rw	rw

Table 15-7. RTCSEC Register Description

Bit	Field	Type	Reset	Description
7	0	R	0h	Always reads as 0.
6-4	Seconds – high digit	RW	undefined	Seconds – high digit. Valid values are 0 to 5.
3-0	Seconds – low digit	RW	undefined	Seconds – low digit. Valid values are 0 to 9.

15.3.7 RTCMIN Register – Hexadecimal Format

Real-Time Clock Minutes Register – Hexadecimal Format

Figure 15-8. RTCMIN Register

7	6	5	4	3	2	1	0
0	0	Minutes					
r-0	r-0	rw	rw	rw	rw	rw	rw

Table 15-8. RTCMIN Register Description

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always reads as 0.
5-0	Minutes	RW	undefined	Minutes. Valid values are 0 to 59.

15.3.8 RTCMIN Register – BCD Format

Real-Time Clock Minutes Register – BCD Format

Figure 15-9. RTCMIN Register

7	6	5	4	3	2	1	0
0	Minutes – high digit			Minutes – low digit			
r-0	rw	rw	rw	rw	rw	rw	rw

Table 15-9. RTCMIN Register Description

Bit	Field	Type	Reset	Description
7	0	R	0h	Always reads as 0.
6-4	Minutes – high digit	RW	undefined	Minutes – high digit. Valid values are 0 to 5.
3-0	Minutes – low digit	RW	undefined	Minutes – low digit. Valid values are 0 to 9.

15.3.9 RTCHOUR Register – Hexadecimal Format

Real-Time Clock Hours Register – Hexadecimal Format

Figure 15-10. RTCHOUR Register

7	6	5	4	3	2	1	0
0	0	0	Hours				
r-0	r-0	r-0	rw	rw	rw	rw	rw

Table 15-10. RTCHOUR Register Description

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always reads as 0.
4-0	Hours	RW	undefined	Hours. Valid values are 0 to 23.

15.3.10 RTCHOUR Register – BCD Format

Real-Time Clock Hours Register – BCD Format

Figure 15-11. RTCHOUR Register

7	6	5	4	3	2	1	0
0	0	Hours – high digit		Hours – low digit			
r-0	r-0	rw	rw	rw	rw	rw	rw

Table 15-11. RTCHOUR Register Description

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always reads as 0.
5-4	Hours – high digit	RW	undefined	Hours – high digit. Valid values are 0 to 2.
3-0	Hours – low digit	RW	undefined	Hours – low digit. Valid values are 0 to 9.

15.3.11 RTCDOW Register

Real-Time Clock Day of Week Register

Figure 15-12. RTCDOW Register

7	6	5	4	3	2	1	0
0	0	0	0	0	Day of week		
r-0	r-0	r-0	r-0	r-0	rw	rw	rw

Table 15-12. RTCDOW Register Description

Bit	Field	Type	Reset	Description
7-3	0	R	0h	Always reads as 0.
2-0	Day of week	RW	undefined	Day of week. Valid values are 0 to 6.

15.3.12 RTCDAY Register – Hexadecimal Format

Real-Time Clock Day of Month Register – Hexadecimal Format

Figure 15-13. RTCDAY Register

7	6	5	4	3	2	1	0
0	0	0	Day of month				
r-0	r-0	r-0	rw	rw	rw	rw	rw

Table 15-13. RTCDAY Register Description

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always reads as 0.
4-0	Day of month	RW	undefined	Day of month. Valid values are 1 to 31.

15.3.13 RTCDAY Register – BCD Format

Real-Time Clock Day of Month Register – BCD Format

Figure 15-14. RTCDAY Register

7	6	5	4	3	2	1	0
0	0	Day of month – high digit		Day of month – low digit			
r-0	r-0	rw	rw	rw	rw	rw	rw

Table 15-14. RTCDAY Register Description

Bit	Field	Type	Reset	Description
7-6	0	R	0h	Always reads as 0.
5-4	Day of month – high digit	RW	undefined	Day of month – high digit. Valid values are 0 to 3.
3-0	Day of month – low digit	RW	undefined	Day of month – low digit. Valid values are 0 to 9.

15.3.14 RTCMON Register – Hexadecimal Format

Real-Time Clock Month Register – Hexadecimal Format

Figure 15-15. RTCMON Register

7	6	5	4	3	2	1	0
0	0	0	0	Month			
r-0	r-0	r-0	r-0	rw	rw	rw	rw

Table 15-15. RTCMON Register Description

Bit	Field	Type	Reset	Description
7-4	0	R	0h	Always reads as 0.
3-0	Month	RW	undefined	Month. Valid values are 1 to 12.

15.3.15 RTCMON Register – BCD Format

Real-Time Clock Month Register

Figure 15-16. RTCMON Register

7	6	5	4	3	2	1	0
0	0	0	Month – high digit	Month – low digit			
r-0	r-0	r-0	rw	rw	rw	rw	rw

Table 15-16. RTCMON Register Description

Bit	Field	Type	Reset	Description
7-5	0	R	0h	Always reads as 0.
4	Month – high digit	RW	undefined	Month – high digit. Valid values are 0 or 1.
3-0	Month – low digit	RW	undefined	Month – low digit. Valid values are 0 to 9.

15.3.16 RTCYEAR Register – Hexadecimal Format

Real-Time Clock Year Register – Hexadecimal Format

Figure 15-17. RTCYEAR Register

15	14	13	12	11	10	9	8
0	0	0	0	Year – high byte			
r-0	r-0	r-0	r-0	rw	rw	rw	rw
7	6	5	4	3	2	1	0
Year – low byte							
rw	rw	rw	rw	rw	rw	rw	rw

Table 15-17. RTCYEAR Register Description

Bit	Field	Type	Reset	Description
15-12	0	R	0h	Always reads as 0.
11-8	Year – high byte	RW	undefined	Year – high byte. Valid values of Year are 0 to 4095.
7-0	Year – low byte	RW	undefined	Year – low byte. Valid values of Year are 0 to 4095.

15.3.17 RTCYEAR Register – BCD Format

Real-Time Clock Year Register – BCD Format

Figure 15-18. RTCYEAR Register

15	14	13	12	11	10	9	8
0	Century – high digit			Century – low digit			
r-0	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
Decade				Year – lowest digit			
rw	rw	rw	rw	rw	rw	rw	rw

Table 15-18. RTCYEAR Register Description

Bit	Field	Type	Reset	Description
15	0	R	0h	Always reads as 0.
14-12	Century – high digit	RW	undefined	Century – high digit . Valid values are 0 to 4.
11-8	Century – low digit	RW	undefined	Century – low digit. Valid values are 0 to 9.
7-4	Decade	RW	undefined	Decade. Valid values are 0 to 9.
3-0	Year – lowest digit	RW	undefined	Year – lowest digit. Valid values are 0 to 9.

15.3.18 RTCAMIN Register – Hexadecimal Format

Real-Time Clock Minutes Alarm Register – Hexadecimal Format

Figure 15-19. RTCAMIN Register

7	6	5	4	3	2	1	0
AE	0	Minutes					
rw	r-0	rw	rw	rw	rw	rw	rw

Table 15-19. RTCAMIN Register Description

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6	0	R	0h	Always reads as 0.
5-0	Minutes	RW	undefined	Minutes. Valid values are 0 to 59.

15.3.19 RTCAMIN Register – BCD Format

Real-Time Clock Minutes Alarm Register – BCD Format

Figure 15-20. RTCAMIN Register

7	6	5	4	3	2	1	0
AE	Minutes – high digit			Minutes – low digit			
rw	rw	rw	rw	rw	rw	rw	rw

Table 15-20. RTCAMIN Register Description

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6-4	Minutes – high digit	RW	undefined	Minutes – high digit. Valid values are 0 to 5.
3-0	Minutes – low digit	RW	undefined	Minutes – low digit. Valid values are 0 to 9.

15.3.20 RTCAHOUR Register – Hexadecimal Format

Real-Time Clock Hours Alarm Register – Hexadecimal Format

Figure 15-21. RTCAHOUR Register

7	6	5	4	3	2	1	0
AE	0	0	Hours				
rw	r-0	r-0	rw	rw	rw	rw	rw

Table 15-21. RTCAHOUR Register Description

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6-5	0	R	0h	Always reads as 0.
4-0	Hours	RW	undefined	Hours. Valid values are 0 to 23.

15.3.21 RTCAHOUR Register – BCD Format

Real-Time Clock Hours Alarm Register – BCD Format

Figure 15-22. RTCAHOUR Register

7	6	5	4	3	2	1	0
AE	0	Hours – high digit		Hours – low digit			
rw	r-0	rw	rw	rw	rw	rw	rw

Table 15-22. RTCAHOUR Register Description

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6	0	R	0h	Always reads as 0.
5-4	Hours – high digit	RW	undefined	Hours – high digit. Valid values are 0 to 2.
3-0	Hours – low digit	RW	undefined	Hours – low digit. Valid values are 0 to 9.

15.3.22 RTCADOW Register

Real-Time Clock Day of Week Alarm Register

Figure 15-23. RTCADOW Register

7	6	5	4	3	2	1	0
AE	0	0	0	0	Day of week		
rw	r-0	r-0	r-0	r-0	rw	rw	rw

Table 15-23. RTCADOW Register Description

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6-3	0	R	0h	Always reads as 0.
2-0	Day of week	RW	undefined	Day of week. Valid values are 0 to 6.

15.3.23 RTCADAY Register – Hexadecimal Format

Real-Time Clock Day of Month Alarm Register – Hexadecimal Format

Figure 15-24. RTCADAY Register

7	6	5	4	3	2	1	0
AE	0	0	Day of month				
rw	r-0	r-0	rw	rw	rw	rw	rw

Table 15-24. RTCADAY Register Description

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6-5	0	R	0h	Always reads as 0.
4-0	Day of month	RW	undefined	Day of month. Valid values are 1 to 31.

15.3.24 RTCADAY Register – BCD Format

Real-Time Clock Day of Month Alarm Register – BCD Format

Figure 15-25. RTCADAY Register

7	6	5	4	3	2	1	0
AE	0	Day of month – high digit		Day of month – low digit			
rw	r-0	rw	rw	rw	rw	rw	rw

Table 15-25. RTCADAY Register Description

Bit	Field	Type	Reset	Description
7	AE	RW	undefined	Alarm enable 0b = This alarm register is enabled 1b = This alarm register is disabled
6	0	R	0h	Always reads as 0.
5-4	Day of month – high digit	RW	undefined	Day of month – high digit. Valid values are 0 to 3.
3-0	Day of month – low digit	RW	undefined	Day of month – low digit. Valid values are 0 to 9.

15.3.25 RTCPS0CTL Register

Real-Time Clock Prescale Timer 0 Control Register

Figure 15-26. RTCPS0CTL 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			RT0IPx(1)			RT0PSIE	RT0PSIFG
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-(0)

(1) The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

Table 15-26. RTCPS0CTL Register Description

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Always reads as 0.
4-2	RT0IPx	RW	0h	Prescale timer 0 interrupt interval 000b = Divide by 2 001b = Divide by 4 010b = Divide by 8 011b = Divide by 16 100b = Divide by 32 101b = Divide by 64 110b = Divide by 128 111b = Divide by 256
1	RT0PSIE	RW	0h	Prescale timer 0 interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled
0	RT0PSIFG	RW	0h	Prescale timer 0 interrupt flag 0b = No time event occurred 1b = Time event occurred

15.3.26 RTCPS1CTL Register

Real-Time Clock Prescale Timer 1 Control Register

Figure 15-27. RTCPS1CTL 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			RT1IPx(1)			RT1PSIE(1)	RT1PSIFG
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-(0)

(1) The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits themselves; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

Table 15-27. RTCPS1CTL Register Description

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Always reads as 0.
4-2	RT1IPx	RW	0h	Prescale timer 1 interrupt interval 000b = Divide by 2 001b = Divide by 4 010b = Divide by 8 011b = Divide by 16 100b = Divide by 32 101b = Divide by 64 110b = Divide by 128 111b = Divide by 256
1	RT1PSIE	RW	0h	Prescale timer 1 interrupt enable 0b = Interrupt not enabled 1b = Interrupt enabled (LPMx.5 wake-up enabled)
0	RT1PSIFG	RW	0h	Prescale timer 1 interrupt flag. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0b = No time event occurred 1b = Time event occurred

15.3.27 RTCPS0 Register

Real-Time Clock Prescale Timer 0 Counter Register

Figure 15-28. RTCPS0 Register

7	6	5	4	3	2	1	0
RT0PS							
rw	rw	rw	rw	rw	rw	rw	rw

Table 15-28. RTCPS0 Register Description

Bit	Field	Type	Reset	Description
7-0	RT0PS	RW	undefined	Prescale timer 0 counter value

15.3.28 RTCPS1 Register

Real-Time Clock Prescale Timer 1 Counter Register

Figure 15-29. RTCPS1 Register

7	6	5	4	3	2	1	0
RT1PS							
rw	rw	rw	rw	rw	rw	rw	rw

Table 15-29. RTCPS1 Register Description

Bit	Field	Type	Reset	Description
7-0	RT1PS	RW	undefined	Prescale timer 1 counter value

15.3.29 RTCIV Register

Real-Time Clock Interrupt Vector Register

Figure 15-30. RTCIV Register

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

Table 15-30. RTCIV Register Description

Bit	Field	Type	Reset	Description
15-0	RTCIVx	R	0h	Real-time clock interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: RTC ready; Interrupt Flag: RTCRDYIFG; Interrupt Priority: Highest 04h = Interrupt Source: RTC interval timer; Interrupt Flag: RTCTEVIFG 06h = Interrupt Source: RTC user alarm; Interrupt Flag: RTCAIFG 08h = Interrupt Source: RTC prescaler 0; Interrupt Flag: RT0PSIFG 0Ah = Interrupt Source: RTC prescaler 1; Interrupt Flag: RT1PSIFG 0Ch = Interrupt Source: RTC oscillator failure; Interrupt Flag: RTCOFIFG 0Eh = Reserved; Interrupt Priority: Lowest

15.3.30 BIN2BCD Register

Binary-to-BCD Conversion Register

Figure 15-31. BIN2BCD Register

15	14	13	12	11	10	9	8
BIN2BCDx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
BIN2BCDx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 15-31. BIN2BCD Register Description

Bit	Field	Type	Reset	Description
15-0	BIN2BCDx	RW	0h	Read: 16-bit BCD conversion of previously written 12-bit binary number Write: 12-bit binary number to be converted

15.3.31 BCD2BIN Register

BCD-to-Binary Conversion Register

Figure 15-32. BCD2BIN Register

15	14	13	12	11	10	9	8
BCD2BINx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
BCD2BINx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 15-32. BCD2BIN Register Description

Bit	Field	Type	Reset	Description
15-0	BCD2BINx	RW	0h	Read: 12-bit binary conversion of previously written 16-bit BCD number Write: 16-bit BCD number to be converted

32-Bit Hardware Multiplier (MPY32)

This chapter describes the 32-bit hardware multiplier (MPY32). The MPY32 module is implemented in all devices.

Topic **Page**

16.1 32-Bit Hardware Multiplier (MPY32) Introduction 458
16.2 MPY32 Operation 460
16.3 MPY32 Registers 472

16.1 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 16-1.

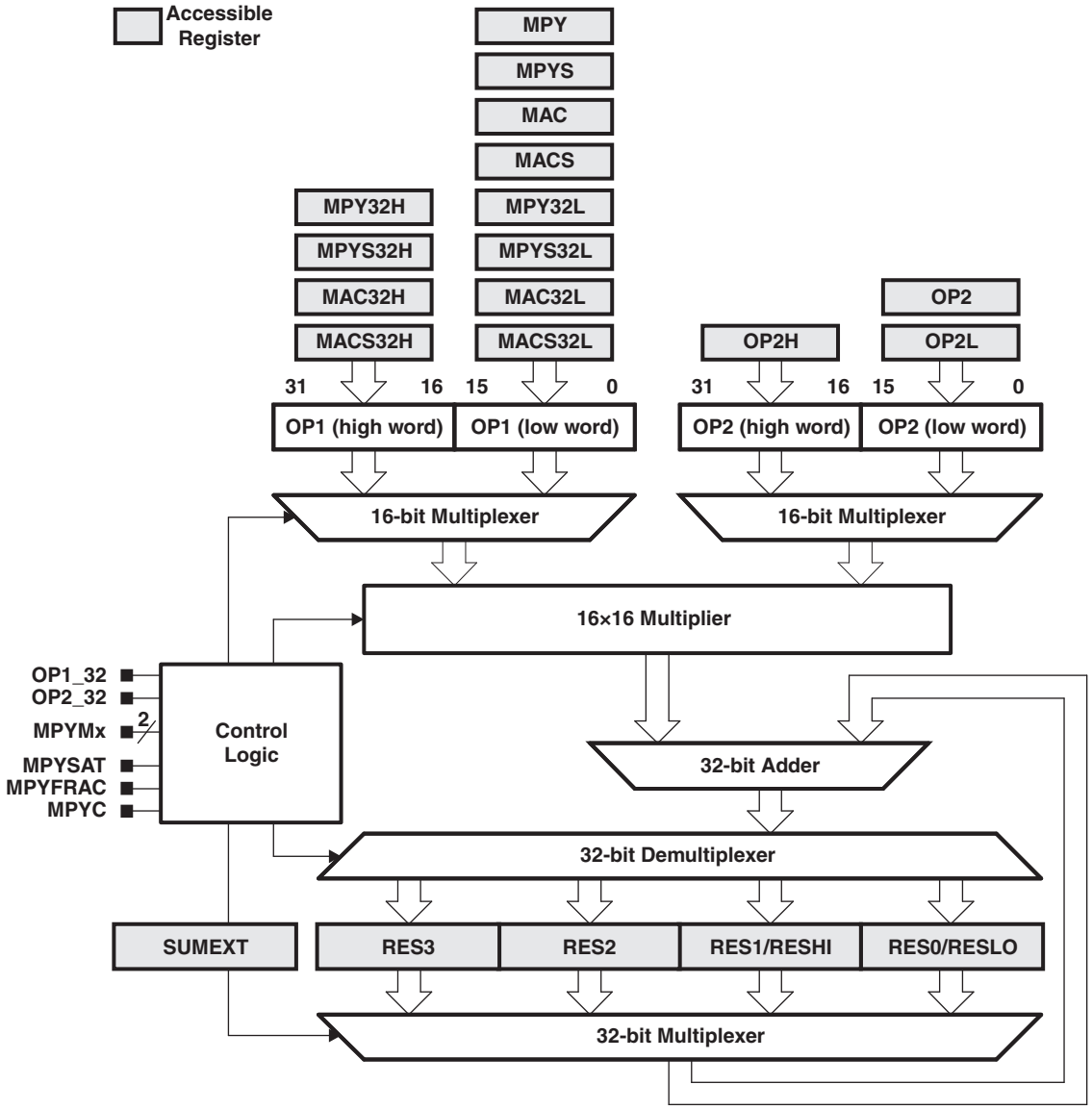


Figure 16-1. MPY32 Block Diagram

16.2 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 16-1 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 16-1. 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

16.2.1 Operand Registers

Operand one (OP1) has 12 registers (see Table 16-2) 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 16-2. 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 16-3. 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 operands 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 16-1 for how many CPU cycles are needed until a certain result register is ready and valid for each of the different modes.

16.2.2 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 16-4. 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 16-4 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 16-4. 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

16.2.2.1 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 (that is, 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.

16.2.3 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
```

16.2.4 Fractional Numbers

The MPY32 provides support for fixed-point signal processing. In fixed-point signal processing, fractional number are numbers that have a fixed number of digits after (and sometimes also before) the radix point. To classify different ranges of binary fixed-point numbers, a Q-format is used. Different Q-formats represent different locations of the radix point. Figure 16-2 shows the format of a signed Q15 number using 16 bits. Every bit after the radix point has a resolution of 1/2, and 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 ≈ 1.0 for the signed Q15 format with 16 bits.

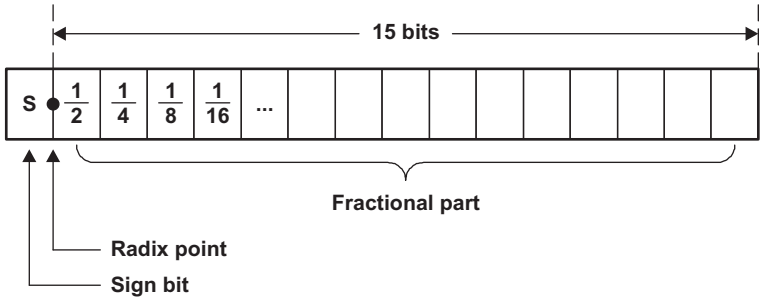


Figure 16-2. Q15 Format Representation

The range can be increased by shifting the radix point to the right as shown in Figure 16-3. The signed Q14 format with 16 bits gives a range from –2.0 to 1.999938965 ≈ 2.0.

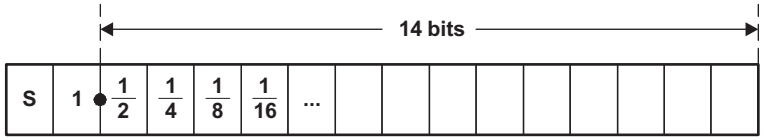


Figure 16-3. Q14 Format Representation

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.

16.2.4.1 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 registers 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 16-5. 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

16.2.4.2 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 registers is not modified when MPYSAT = 1. When the result is accessed using software, the value is automatically adjusted to provide 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; that is, 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.

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 16-6. 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 16-4 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, that is, the result that is read with fractional mode disabled.

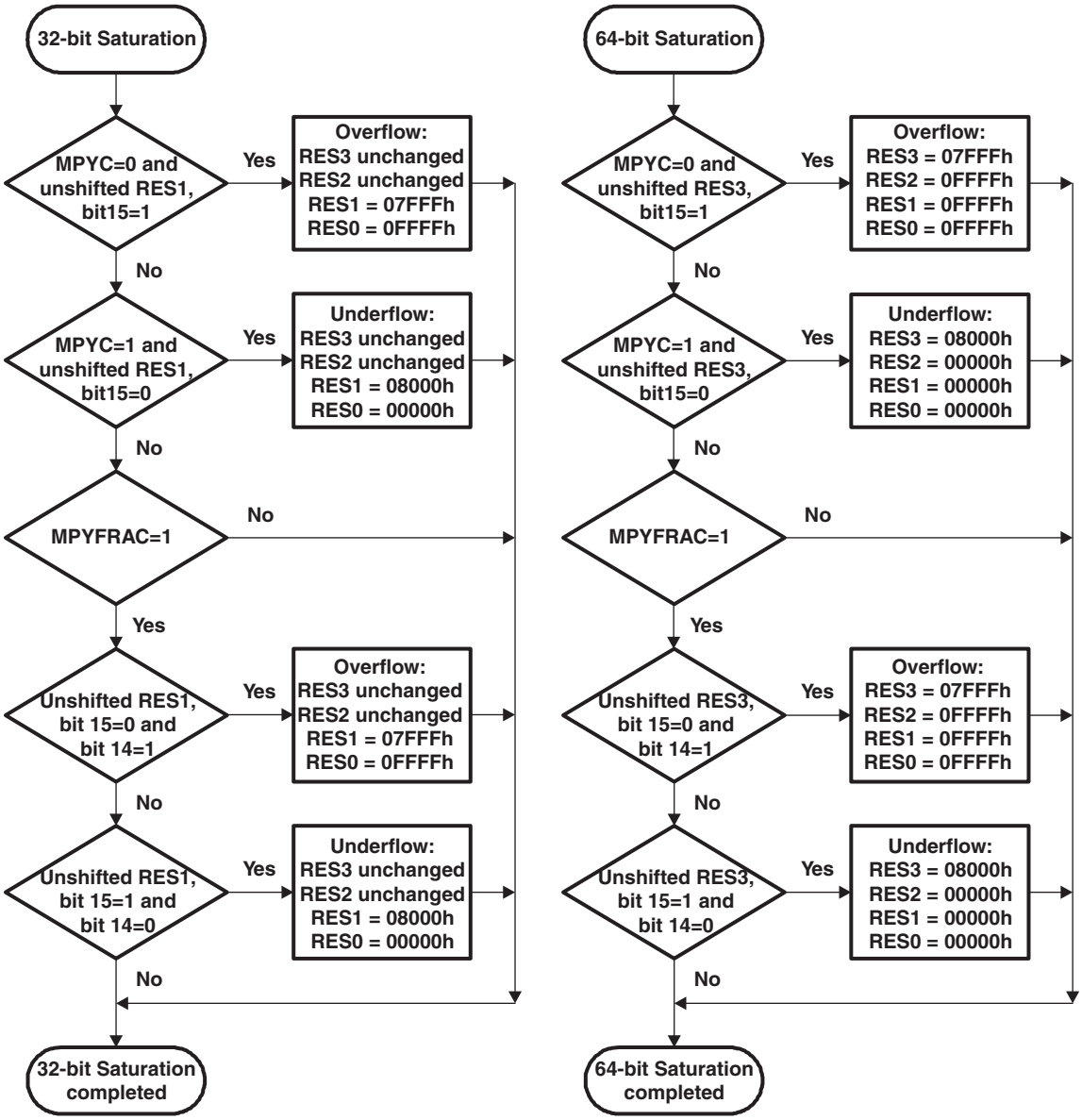


Figure 16-4. Saturation Flow Chart

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 16-4.

NOTE: Validity of saturated result

The saturated result is valid only if the registers RES0 to RES3, the size of OP1 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.

16.2.5 Putting It All Together

Figure 16-5 shows the complete multiplication flow, depending on the various selectable modes for the MPY32 module.

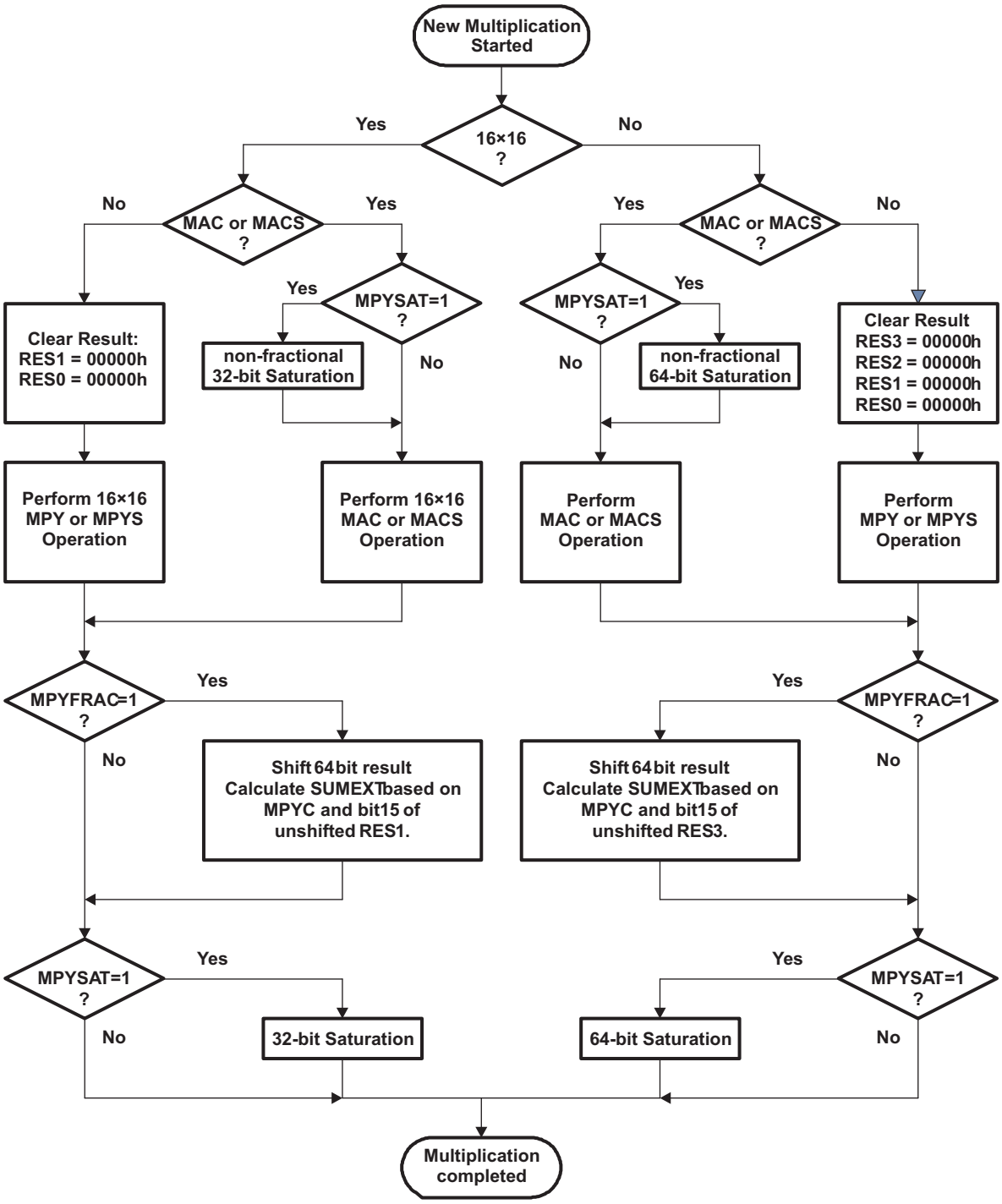


Figure 16-5. Multiplication Flow Chart

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 or results with 32-bit operands or results. User software must address these points during use when mixing these operations. The following code 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 shows.

```
   ;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.

16.2.6 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 16-1, 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
```

16.2.7 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
```

16.2.7.1 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
```

16.2.8 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).

16.3 MPY32 Registers

MPY32 registers are listed in Table 16-7. The base address can be found in the device-specific data sheet. The address offsets are listed in Table 16-7.

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 16-7. MPY32 Registers

Offset	Acronym	Register Name	Type	Access	Reset
00h	MPY	16-bit operand one – multiply	Read/write	Word	Undefined
00h	MPY_L		Read/write	Byte	Undefined
01h	MPY_H		Read/write	Byte	Undefined
00h	MPY_B	8-bit operand one – multiply	Read/write	Byte	Undefined
02h	MPYS	16-bit operand one – signed multiply	Read/write	Word	Undefined
02h	MPYS_L		Read/write	Byte	Undefined
03h	MPYS_H		Read/write	Byte	Undefined
02h	MPYS_B	8-bit operand one – signed multiply	Read/write	Byte	Undefined
04h	MAC	16-bit operand one – multiply accumulate	Read/write	Word	Undefined
04h	MAC_L		Read/write	Byte	Undefined
05h	MAC_H		Read/write	Byte	Undefined
04h	MAC_B	8-bit operand one – multiply accumulate	Read/write	Byte	Undefined
06h	MACS	16-bit operand one – signed multiply accumulate	Read/write	Word	Undefined
06h	MACS_L		Read/write	Byte	Undefined
07h	MACS_H		Read/write	Byte	Undefined
06h	MACS_B	8-bit operand one – signed multiply accumulate	Read/write	Byte	Undefined
08h	OP2	16-bit operand two	Read/write	Word	Undefined
08h	OP2_L		Read/write	Byte	Undefined
09h	OP2_H		Read/write	Byte	Undefined
08h	OP2_B	8-bit operand two	Read/write	Byte	Undefined
0Ah	RESLO	16x16-bit result low word	Read/write	Word	Undefined
0Ah	RESLO_L		Read/write	Byte	Undefined
0Ch	RESHI	16x16-bit result high word	Read/write	Word	Undefined
0Eh	SUMEXT	16x16-bit sum extension register	Read	Word	Undefined
10h	MPY32L	32-bit operand 1 – multiply – low word	Read/write	Word	Undefined
10h	MPY32L_L		Read/write	Byte	Undefined
11h	MPY32L_H		Read/write	Byte	Undefined
12h	MPY32H	32-bit operand 1 – multiply – high word	Read/write	Word	Undefined
12h	MPY32H_L		Read/write	Byte	Undefined
13h	MPY32H_H		Read/write	Byte	Undefined
12h	MPY32H_B	24-bit operand 1 – multiply – high byte	Read/write	Byte	Undefined
14h	MPYS32L	32-bit operand 1 – signed multiply – low word	Read/write	Word	Undefined
14h	MPYS32L_L		Read/write	Byte	Undefined
15h	MPYS32L_H		Read/write	Byte	Undefined
16h	MPYS32H	32-bit operand 1 – signed multiply – high word	Read/write	Word	Undefined
16h	MPYS32H_L		Read/write	Byte	Undefined
17h	MPYS32H_H		Read/write	Byte	Undefined
16h	MPYS32H_B	24-bit operand 1 – signed multiply – high byte	Read/write	Byte	Undefined
18h	MAC32L	32-bit operand 1 – multiply accumulate – low word	Read/write	Word	Undefined

Table 16-7. MPY32 Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset
18h	MAC32L_L		Read/write	Byte	Undefined
19h	MAC32L_H		Read/write	Byte	Undefined
1Ah	MAC32H	32-bit operand 1 – multiply accumulate – high word	Read/write	Word	Undefined
1Ah	MAC32H_L		Read/write	Byte	Undefined
1Bh	MAC32H_H		Read/write	Byte	Undefined
1Ah	MAC32H_B	24-bit operand 1 – multiply accumulate – high byte	Read/write	Byte	Undefined
1Ch	MACS32L	32-bit operand 1 – signed multiply accumulate – low word	Read/write	Word	Undefined
1Ch	MACS32L_L		Read/write	Byte	Undefined
1Dh	MACS32L_H		Read/write	Byte	Undefined
1Eh	MACS32H	32-bit operand 1 – signed multiply accumulate – high word	Read/write	Word	Undefined
1Eh	MACS32H_L		Read/write	Byte	Undefined
1Fh	MACS32H_H		Read/write	Byte	Undefined
1Eh	MACS32H_B	24-bit operand 1 – signed multiply accumulate – high byte	Read/write	Byte	Undefined
20h	OP2L	32-bit operand 2 – low word	Read/write	Word	Undefined
20h	OP2L_L		Read/write	Byte	Undefined
21h	OP2L_H		Read/write	Byte	Undefined
22h	OP2H	32-bit operand 2 – high word	Read/write	Word	Undefined
22h	OP2H_L		Read/write	Byte	Undefined
23h	OP2H_H		Read/write	Byte	Undefined
22h	OP2H_B	24-bit operand 2 – high byte	Read/write	Byte	Undefined
24h	RES0	32x32-bit result 0 – least significant word	Read/write	Word	Undefined
24h	RES0_L		Read/write	Byte	Undefined
26h	RES1	32x32-bit result 1	Read/write	Word	Undefined
28h	RES2	32x32-bit result 2	Read/write	Word	Undefined
2Ah	RES3	32x32-bit result 3 – most significant word	Read/write	Word	Undefined
2Ch	MPY32CTL0	MPY32 control register 0	Read/write	Word	Undefined
2Ch	MPY32CTL0_L		Read/write	Byte	Undefined
2Dh	MPY32CTL0_H		Read/write	Byte	00h

The registers listed in Table 16-8 are treated equally.

Table 16-8. 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

16.3.1 MPY32CTL0 Register

32-Bit Hardware Multiplier Control 0 Register

Figure 16-6. MPY32CTL0 Register

15	14	13	12	11	10	9	8
Reserved						MPYDLY32	MPYDLYWRTEN
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

Table 16-9. MPY32CTL0 Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9	MPYDLY32	RW	0h	Delayed write mode 0b = Writes are delayed until 64-bit result (RES0 to RES3) is available. 1b = Writes are delayed until 32-bit result (RES0 to RES1) is available.
8	MPYDLYWRTEN	RW	0h	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. 0b = Writes are not delayed. 1b = Writes are delayed.
7	MPYOP2_32	RW	0h	Multiplier bit width of operand 2 0b = 16 bits 1b = 32 bits
6	MPYOP1_32	RW	0h	Multiplier bit width of operand 1 0b = 16 bits 1b = 32 bits
5-4	MPYMx	RW	0h	Multiplier mode 00b = MPY – Multiply 01b = MPYS – Signed multiply 10b = MAC – Multiply accumulate 11b = MACS – Signed multiply accumulate
3	MPYSAT	RW	0h	Saturation mode 0b = Saturation mode disabled 1b = Saturation mode enabled
2	MPYFRAC	RW	0h	Fractional mode 0b = Fractional mode disabled 1b = Fractional mode enabled
1	Reserved	RW	0h	Reserved. Always reads as 0.
0	MPYC	RW	0h	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. 0b = No carry for result 1b = Result has a carry

REF_A

The REF_A 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, or comparators. This chapter describes the REF_A module.

Topic **Page**

17.1 REF_A Introduction **476**
17.2 Principle of Operation **477**
17.3 REF_A Registers **479**

17.1 REF_A 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. The heart of the reference system is the bandgap from which all other references are derived by unity or noninverting gain stages. The REFGEN subsystem consists of the bandgap, the bandgap bias, and the noninverting buffer stage, which generates the three primary voltage reference available in the system, namely 1.2 V, 2.0 V, and 2.5 V. In addition, when enabled, a buffered bandgap voltage is available.

Features of the REF_A include:

- Centralized factory-trimmed bandgap with excellent PSRR, temperature coefficient, and accuracy
- 1.2-V, 2.0-V, or 2.5-V user selectable internal references
- Buffered bandgap voltage available to rest of system
- Power saving features
- Hardware reference request and reference ready signals for bandgap and variable reference voltages for save operation

Figure 17-1 shows an example block diagram of the REF_A module. The example shown here is for a device with an ADC, a DAC, an LCD, and two Comparators.

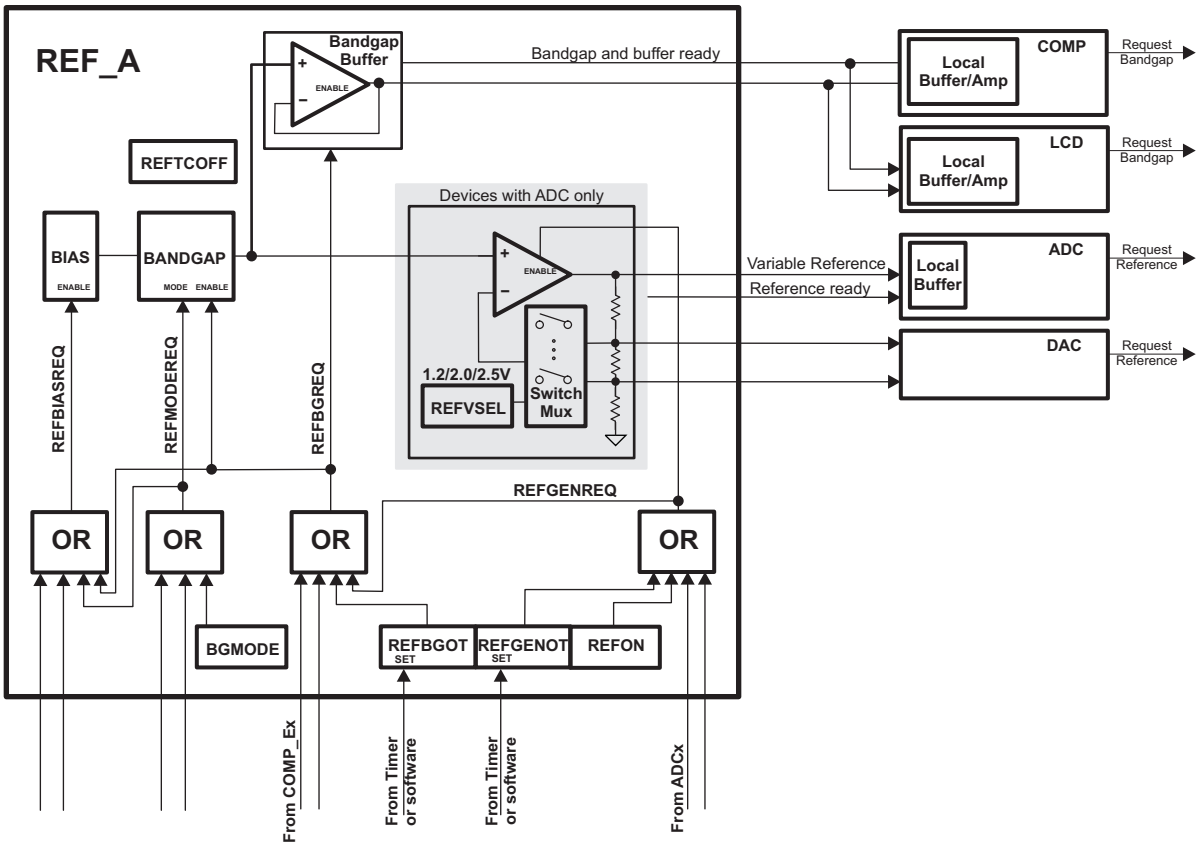


Figure 17-1. REF_A Block Diagram

17.2 Principle of Operation

The REF_A module provides all of the necessary voltage references that can be used by various peripheral modules throughout the system.

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 through a noninverting amplifier stage, namely 1.2 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 1.2 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. Additionally, the REFGEN supports the voltage references that are required for the DAC12 module, when available. Lastly, the REFGEN subsystem also includes the temperature sensor circuitry, which is derived from the bandgap. The temperature sensor is used by an ADC to measure a voltage proportional to temperature.

17.2.1 Low-Power Operation

The REF_A 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 can request static mode or sampled mode via their own individual request lines. In this way, the particular module determines which mode is appropriate for its proper operation and performance. Any one active module that requests static mode causes all other modules to use static mode, even if another module is requesting sampled mode. In other words, static mode always has higher priority over sampled mode.

17.2.2 Reference System Requests

There are three basic reference system requests that are used by the reference system. Each module can use 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 automatically select the proper request.

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. When the REFGENREQ is asserted, the REFGEN subsystem is enabled. After the specified settling time, the variable reference line voltage is stable and ready for use. The REFVSEL settings determine which voltage is generated on the variable reference line.

After the specified settling time of the REFGEN subsystem, the REF_A module sets the REFGENRDY signal. This signal can be used by each module to wait, for example, before a conversion is started after a REFGENREQ was set. The generation of the reference voltage can be triggered by a timer or by software to make sure that the reference voltage is ready when a module requires it.

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. When the REFBGREQ is asserted, the bandgap and its bias circuitry and local buffer are enabled, if not already enabled by a prior request.

After the specified settling time of the REFBGREQ subsystem, the REF_A module sets the REFBGRDY signal. This signal can be used by each module to delay operation while the bandgap reference voltage is settling. The generation of the buffered bandgap voltage can be triggered by a timer or by software to make sure that the reference voltage is ready when a module requires it.

The REFMODEREQ request signal configures the bandgap and its bias circuitry to operate in either sampled or static mode of operation. The REFMODEREQ signal represents a logical AND of individual requests coming from the various analog modules. A REFMODEREQ occurs only if a module's REFGENREQ or REFBGQ is also asserted, otherwise it is a don't 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. Because REMODEREQ is a logical AND of all individual requests, any modules that request static mode cause the bandgap to operate in static mode. The BGMODE bit can be read for use as an indicator of static or sampled mode of operation.

17.2.2.1 REFBGACT, REFGENACT, REFGENBUSY

Any module that is using the variable reference line causes REFGENACT to be set inside the REFCTL register. This bit is read only and indicates to the user whether the REFGEN is active or off. Similarly, the REFBGACT is active any time one or more modules are actively using the bandgap reference line and, therefore, indicates to the user whether the REFBG is active or off.

The REFGENBUSY signal, when asserted, indicates that a module is using the reference and that no changes should be made to the reference settings. For example, during an active ADC12_B conversion, the reference voltage level should not be changed. REFGENBUSY is asserted when there is an active ADC12_B conversion (ENC = 1) or when the DAC12_TBD 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. Note that there is no such protection for the DAC12_TBD if the ADC12_B legacy control bits are used for the reference control. If the user changes the ADC12_B settings and the DAC12_TBD is using the reference, the DAC12_TBD conversion is affected.

17.2.2.2 ADC12_B

For devices that contain an ADC12_B module, there are two buffers. The larger buffer can be used to drive the reference voltage, which is present on the variable reference line. This buffer has larger power consumption to drive larger DC loads that may be present outside the device. The large buffer is enabled continuously when REFON = 1 and REFOUT =1. 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 through an internal analog switch. This makes sure that 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.

17.2.2.3 LCD Modules

On devices that contain an LCD module, this module requires a reference to generate the proper LCD voltages. The bandgap reference line from the REFGEN subsystem is used for this purpose. Enabling the LCD module in a mode that requires a reference voltage causes a REFBGREQ from the LCD module to be asserted. The buffered bandgap is made available on the bandgap reference line for use inside the LCD module.

17.3 REF_A Registers

The REF_A registers are listed in Table 17-1. The base address can be found in the device specific datasheet. The address offset is listed in Table 17-1.

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 17-1. REF_A Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	REFCTL0	REFCTL0	Read/write	Word	0000h	Section 17.3.1
00h	REFCTL0_L		Read/write	Byte	00h	
01h	REFCTL0_H		Read/write	Byte	00h	

17.3.1 REFCTL0 Register (offset = 00h) [reset = 0000h]

REF_A Control Register 0

Figure 17-2. REFCTL0 Register

15	14	13	12	11	10	9	8
Reserved		REFBGRDY	REFGENRDY	BGMODE	REFGENBUSY	REFBGACT	REFGENACT
r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)
7	6	5	4	3	2	1	0
REFBGOT	REFGENOT	REFVSEL		REFTCOFF	Reserved	REFOUT	REFON
rw-0	rw-0	rw-(0)	rw-(0)	rw-(0)	r0	rw-(0)	rw-(0)

Can be modified only when REFGENBUSY = 0.

Table 17-2. REFCTL0 Register Description

Bit	Field	Type	Reset	Description
15-14	Reserved	R	0h	Reserved. Always reads as 0.
13	REFBGRDY	R	0h	Buffered bandgap voltage is ready to be used. Both the bandgap and the bandgap buffer are active, and the reference voltage is settled for use by the comparator and the LCD. 0b = Buffered bandgap voltage is not ready to be used 1b = Buffered bandgap voltage is ready to be used
12	REFGENRDY	R	0h	Variable reference voltage ready status. Variable reference voltage is ready to be used. Both the bandgap and the reference voltage amplifier are active and the variable reference voltage is settled; for example, for use by the ADC. 0b = Reference voltage output is not ready to be used 1b = Reference voltage output is ready to be used
11	BGMODE	R	0h	Bandgap mode. Read only. 0b = Static mode 1b = Sampled mode
10	REFGENBUSY	R	0h	Reference generator busy. Read only. 0b = Reference generator not busy 1b = Reference generator busy
9	REFBGACT	R	0h	Reference bandgap active. Read only. 0b = Reference bandgap buffer not active 1b = Reference bandgap buffer active
8	REFGENACT	R	0h	Reference generator active. Read only. 0b = Reference generator not active 1b = Reference generator active
7	REFBGOT	RW	0h	Bandgap and bandgap buffer one-time trigger. If written with a 1, the generation of the buffered bandgap voltage is started. When the bandgap buffer voltage request is set, this bit is cleared by hardware. 0b = No trigger 1b = Generation of the bandgap voltage is started by writing 1 or by a hardware trigger
6	REFGENOT	RW	0h	Reference generator one-time trigger. If written with a 1, the generation of the variable reference voltage is started. When the reference voltage request is set, this bit is cleared by hardware. 0b = No trigger 1b = Generation of the reference voltage is started by writing 1 or by a hardware trigger

Table 17-2. REFCTL0 Register Description (continued)

Bit	Field	Type	Reset	Description
5-4	REFVSEL	RW	0h	Reference voltage level select. Can be modified only when REFGENBUSY = 0. 00b = 1.2 V available when reference requested or REFON = 1 01b = 2.0 V available when reference requested or REFON = 1 10b = 2.5 V available when reference requested or REFON = 1 11b = 2.5 V available when reference requested or REFON = 1
3	REFTCOFF	RW	0h	Temperature sensor disable. The temperature sensor is disabled if the ADC on the device is not enabled independent of this control bit. Can be modified only when REFGENBUSY = 0. 0b = Temperature sensor enabled 1b = Temperature sensor disabled to save power
2	Reserved	R	0h	Reserved. Always reads as 0.
1	REFOUT	RW	0h	Reference output buffer. On devices with an ADC10_A, this bit must be written with 0. Can be modified only when REFGENBUSY = 0. 0b = Reference output not available externally 1b = Reference output available externally
0	REFON	RW	0h	Reference enable. Can be modified only when REFGENBUSY = 0. 0b = Disables reference if no other reference requests are pending 1b = Enables reference

ADC12_B

The ADC12_B module is a high-performance 12-bit analog-to-digital converter (ADC). This chapter describes the operation of the ADC12_B module.

Topic **Page**

18.1 ADC12_B Introduction **483**
18.2 ADC12_B Operation **485**
18.3 ADC12_B Registers **499**

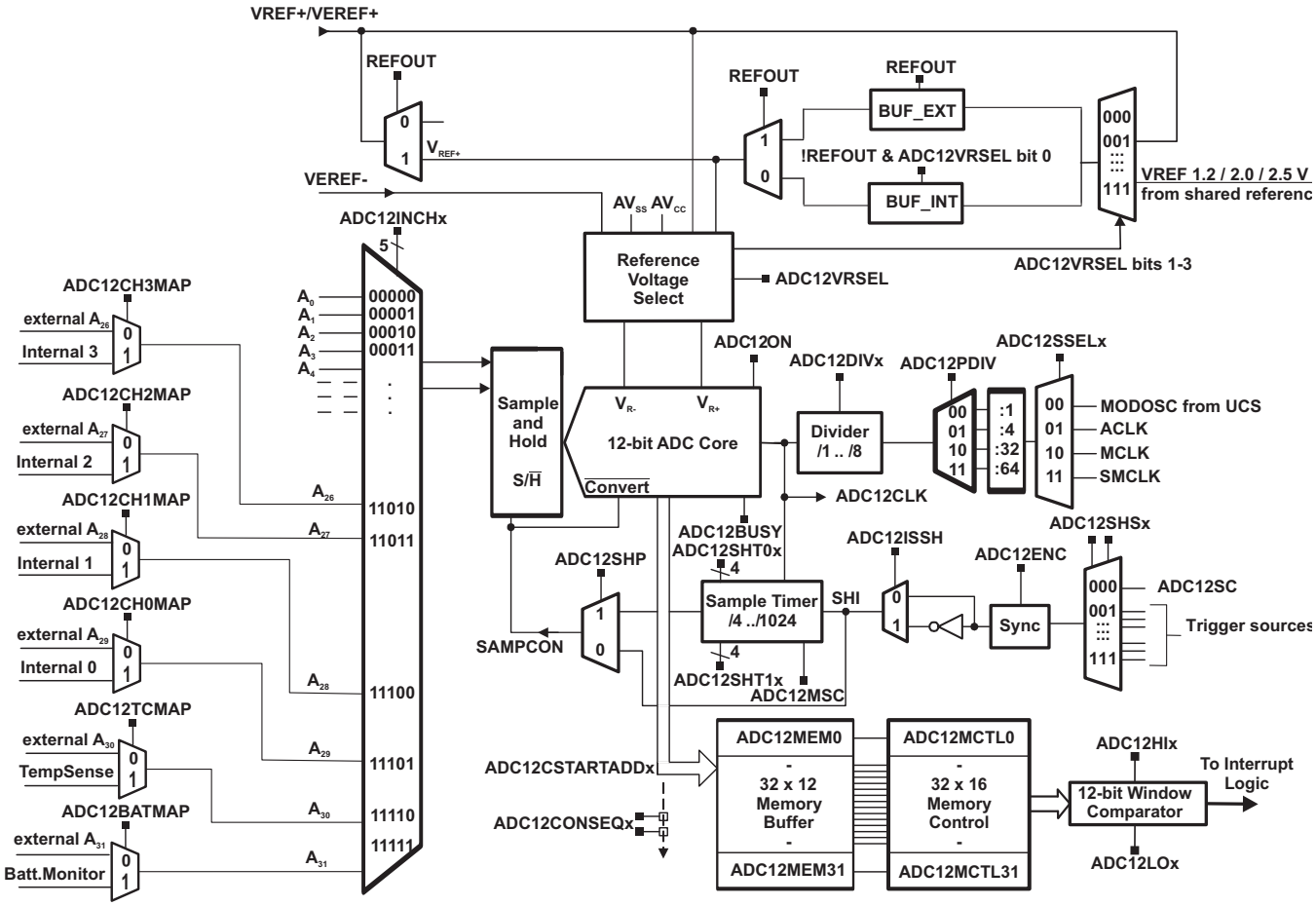
18.1 ADC12_B Introduction

The ADC12_B module supports fast 12-bit analog-to-digital conversions. The module implements a 12-bit SAR core, sample select control, and up to 32 independent conversion-and-control buffers. The conversion-and-control buffer allows up to 32 independent analog-to-digital converter (ADC) samples to be converted and stored without any CPU intervention.

ADC12_B features include:

- 200-ksps maximum conversion rate at maximum resolution of 12 bits
- 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 (1.2 V, 2.0 V, or 2.5 V) with option to make available externally
- Software-selectable internal or external reference
- Up to 32 individually configurable external input channels with single-ended or differential input selection available
- Internal conversion channels for internal temperature sensor and 1/2 × AV_{CC} and four more internal channels available on select devices (see device data sheet for availability as well as function)
- 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
- Interrupt vector register for fast decoding of 38 ADC interrupts
- 32 conversion-result storage registers
- Window comparator for low-power monitoring of input signals of conversion-result registers

Figure 18-1 shows the block diagram of ADC12_B. The reference generation is located in the reference module (REF) (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.

C See the device-specific data sheet for Internal Channel 0-3 availability and function.

D REFOUT bit is part of the Reference module registers.

Figure 18-1. ADC12_B Block Diagram

18.2 ADC12_B Operation

The ADC12_B module is configured with user software. The following sections describe the setup and operation of the ADC12_B.

18.2.1 12-Bit ADC Core

The ADC core converts an analog input to its 12-bit digital representation. The core uses two programmable and 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 is 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.

Equation 11 shows the conversion formula for the ADC result N_{ADC} for single-ended mode.

$$N_{ADC} = 4096 \times \frac{(V_{in+} + \frac{1}{2}LSB) - V_{R-}}{V_{R+} - V_{R-}}, 1LSB = \frac{V_{R+} - V_{R-}}{4096} \quad (11)$$

Equation 12 shows the conversion formula for the ADC result N_{ADC} for differential mode.

$$N_{ADC} = \left(2048 \times \frac{V_{in+} - V_{in-} + \frac{1}{2}LSB}{V_{R+} - V_{R-}} \right) + 2048 \qquad \text{Where, 1 LSB} = \frac{V_{R+} - V_{R-}}{2048} \quad (12)$$

Equation 13 describes the input voltage at which the ADC output saturates for singled-ended mode.

$$V_{in+} = V_{R+} - V_{R-} - 1.5 \text{ LSB} \quad (13)$$

Equation 14 describes the input voltage at which the ADC output saturates for differential mode.

$$V_{in+} - V_{in-} = V_{R+} - V_{R-} - 1.5 \text{ LSB} \quad (14)$$

Four control registers configure the ADC12_B core: ADC12CTL0, ADC12CTL1, ADC12CTL2, and ADC12CTL3. The ADC12ON bit enables or disables the core. The ADC12_B can be turned off when it is not in use to save power. If the ADC12ON bit is set to 0 during a conversion, the conversion is abruptly exited and the module is powered down. With few exceptions, an application can modify the ADC12_B control bits only when ADC12ENC = 0. ADC12ENC must be set to 1 before any conversion can take place.

The conversion results are always stored in binary unsigned format. For differential input, this means that an offset of 2048 is added to the result to make the number positive. The data format bit ADC12DF in ADC12CTL2 allows the user to read the conversion results as binary unsigned or signed binary (2s complement).

18.2.1.1 Conversion Clock Selection

The ADC12CLK operates as the conversion clock and also generates the sampling period when the pulse sampling mode is selected. The ADC12SSELx bit selects the ADC12_B source clock. SMCLK, MCLK, ACLK, and the MODOSC are the possible ADC12CLK sources. The ADC12PDIV bits set the initial divider on the input clock (1, 4, 32, or 64), and then ADC12DIV bits set an additional divider of 1 to 8.

The user must ensure that the clock that is used 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.

18.2.2 ADC12_B Inputs and Multiplexer

Up to 32 external and up to 6 internal analog signals are selected as the channel for conversion by the analog input multiplexer; the number of channels that are available is device specific and is shown in the device-specific data sheet. The input multiplexer is a break-before-make type to reduce input-to-input noise injection that can result from channel switching (see Figure 18-2). The input multiplexer is also a T-switch to minimize the coupling between channels. Channels that are not selected are isolated from the ADC, and the intermediate node is connected to analog ground (AVSS), so that the stray capacitance is grounded to eliminate crosstalk.

The ADC12_B 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.

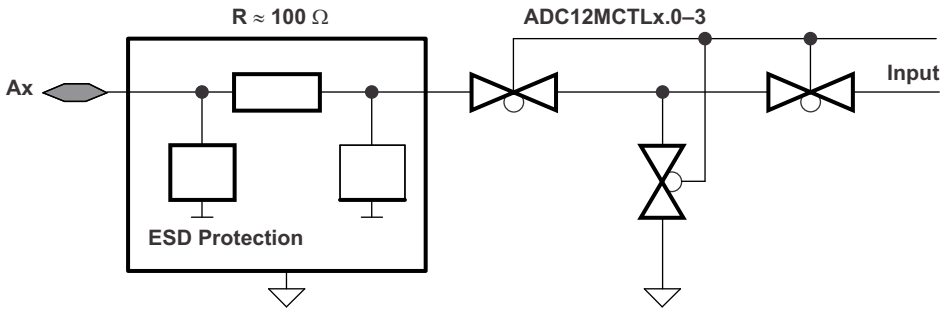


Figure 18-2. Analog Multiplexer

18.2.2.1 Analog Port Selection

The ADC12_B 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 part of the port pin eliminates the parasitic current flow and, therefore, reduces overall current consumption. The PySELx bits can 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_B function
```

18.2.3 Voltage References

The ADC12_B module may use an on-chip shared reference module that supplies three selectable voltage levels of 1.2 V, 2.0 V, and 2.5 V (see the reference module for proper configuration details) to supply V_{R+} and V_{R-}. These reference voltages may be used internally and externally on pin VREF+. Alternatively, external references may be supplied for V_{R+} and V_{R-} through pins VREF+/VEREF+ and VEREF-, respectively.

18.2.4 Auto Power Down

The ADC12_B is designed for low-power applications. When the ADC12_B 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.

18.2.5 Sample Frequency Mode Selection

The ADC12PWRMD bit optimizes the ADC12_B power consumption at two different sample rate ranges. Select the lowest sample frequency that meets or exceeds the application requirements.

18.2.6 Sample and Conversion Timing

A rising edge of the sample input signal (SHI) initiates an analog-to-digital conversion. The SHSx bits select the source for SHI and include the following:

- ADC12SC bit
- Up to seven other sources that may include timer output (see to the device-specific data sheet for available sources).

The ADC12_B supports 8-bit, 10-bit, and 12-bit resolution modes, and the ADC12RES bits select the current mode. The analog-to-digital conversion requires 10, 12, and 14 ADC12CLK cycles, respectively. The ADC12ISSH bit can invert the polarity of the SHI signal source. 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. Control bit ADC12SHP defines the sample-timing method, either extended sample mode or pulse mode. See the device-specific data sheet for timers that are available for SHI sources.

18.2.6.1 Extended Sample Mode

ADC12SHP = 0 selects the extended sample mode. The SHI signal directly controls SAMPCON and defines the length of the sample period t_{sample}. If an ADC local reference buffer is used, the user should assert the sample trigger, wait for the ADC12RDYIFG flag to be set (which indicates that the ADC12_B local reference buffer is settled, and the flag does not occur if the sample trigger has not been asserted), and then keep the sample trigger asserted for the desired sample period before de-asserting. Alternately, if a local reference buffer is used, the user may assert the sample trigger for the desired sample time plus the maximum time for the reference and buffers to settle (reference and buffer settling times are provided in the device-specific data sheet). An ADC local reference buffer is used when ADC12VRSEL= 0001, 0011, 0101, 0111, 1001, 1011, 1101, or 1111. When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the conversion after synchronization with ADC12CLK plus one clock cycle (see Figure 18-3).

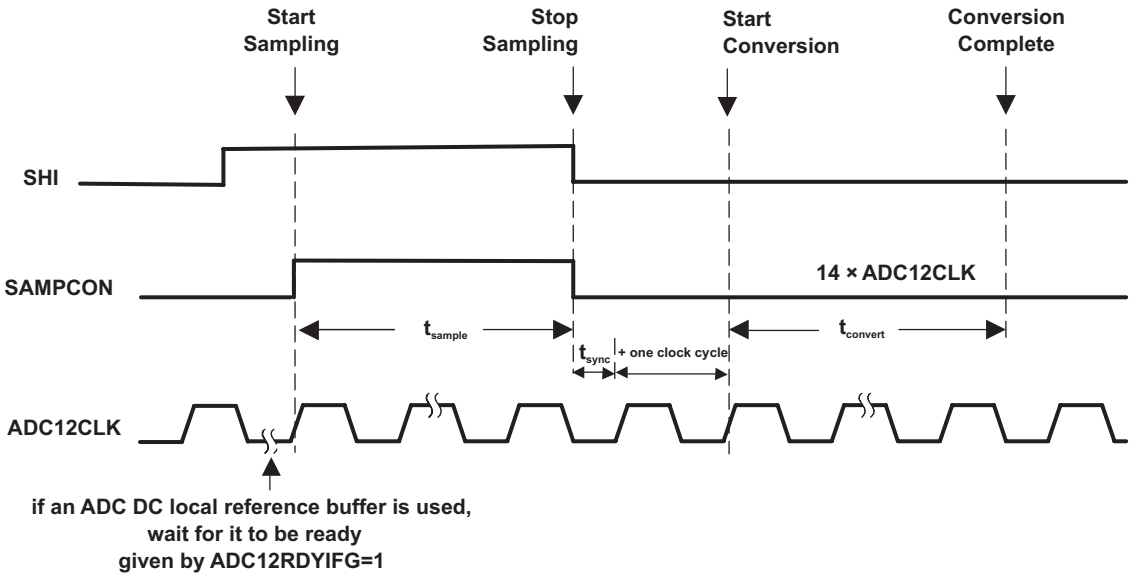


Figure 18-3. Extended Sample Mode in 12-Bit Mode

18.2.6.2 Pulse Sample Mode

ADC12SHP = 1 selects the pulse sample mode. The SHI signal triggers 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 while waiting for reference and ADC local reference buffer to settle(if the internal reference is used), synchronization with AD12CLK plus one clock cycle, and for the programmed interval t_{sample}. The total sampling time is t_{sample} plus the time for ADC12RDYIFG to go high (if an ADC local reference buffer is used) plus t_{sync}, where t_{sync} is the time to sync to the ADC12CLK, plus one clock cycle (see Figure 18-4).

The ADC12SHTx bits select the sampling time in 4x multiples of ADC12CLK. ADC12SHT0x selects the sampling time for ADC12MCTL8 to ADC12MCTL23, and ADC12SHT1x selects the sampling time for ADC12MCTL0 to ADC12MCTL7 and ADC12MCTL24 to ADC12MCTL31.

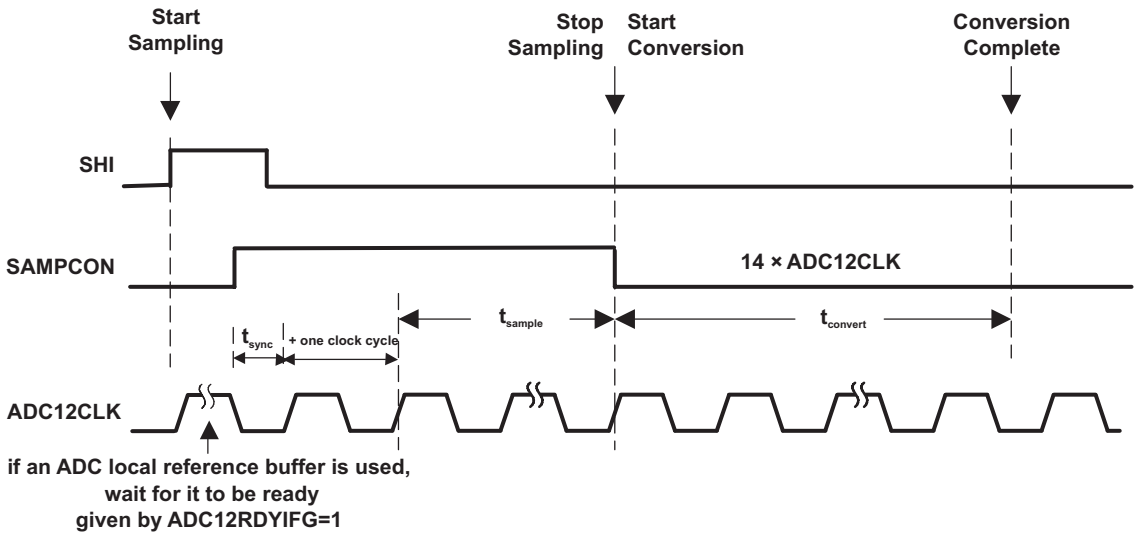


Figure 18-4. Pulse Sample Mode in 12-Bit Mode

18.2.6.3 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 18-5). An internal MUX-on input resistance R_I (see the device-specific data sheet) in series with capacitor C_I (see the device-specific data sheet) 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.

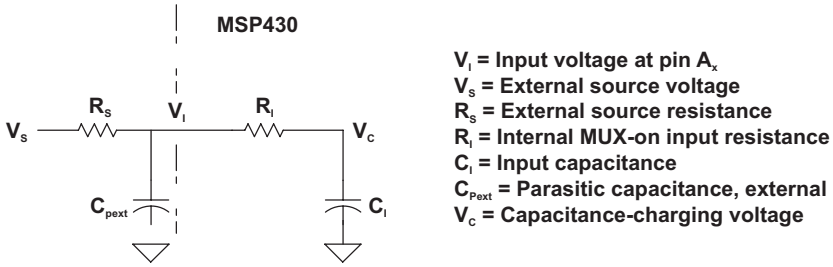


Figure 18-5. Analog Input Equivalent Circuit

The resistance of the source R_S and R_I affect t_{Sample}. Use Equation 15 to calculate the minimum sampling time t_{Sample} for a n-bit conversion, where n equals the bits of resolution.

$$t_{sample} \geq (R_S + R_I) \times \ln(2^{n+2}) \times (C_I + C_{pext}), R_S < 10k\Omega \quad (15)$$

See the device-specific data sheet for R_I and C_I values.

18.2.7 Conversion Memory

32 ADC12MEMx conversion memory registers store the conversion results. Each ADC12MEMx is configured with an associated ADC12MCTLx control register. The ADC12VRSEL bits define the voltage reference, and the ADC12INCHx and ADC12DIF bits select the input channels. The ADC12EOS bit defines the end of sequence when a sequential conversion mode is used. A sequence rolls over from ADC12MEM31 to ADC12MEM0 when the ADC12EOS bit in ADC12MCTL31 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 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 ADC12IFGRx register is set.

There are two formats available to read the conversion result from ADC12MEMx. When ADC12DF = 0, the conversion is right justified and unsigned. For ADC12DF = 0 with ADC12DIF=0 and 8-bit, 10-bit, and 12-bit resolutions, the upper 8, 6, and 4 bits, respectively, of an ADC12MEMx read are always zeros. To convert a ADC12DIF = 1 to binary unsigned, the maximum negative value is added to the conversion. Therefore, 128 is added for 8-bit conversions, 512 is added for 10-bit conversions, and 2048 is added for 12-bit conversions.

When ADC12DF = 1, the conversion result is left justified and twos complement. For 8-bit, 10-bit, and 12-bit resolutions, the lower 8, 6, and 4 bits, respectively, of a ADC12MEMx read are always zeros.

Table 18-1 summarizes the output data formats.

Table 18-1. ADC12_B Conversion Result Formats

Analog Input Voltage Range	ADC12DIF	ADC12DF	ADC12RES	Ideal Conversion Results (With Offset Added When ADC12DIF = 1)	ADC12MEMx Read Value
Vin to V_{R-}: $-V_{REF}$ to $+V_{REF}$	0	0	00	0 to 255	0000h to 00FFh
	0	0	01	0 to 1023	0000h to 03FFh
	0	0	10	0 to 4095	0000h to 0FFFh
	0	1	00	-128 to 127	8000h to 7F00h
	0	1	01	-512 to 511	8000h to 7FC0h
	0	1	10	-2048 to 2047	8000h to 7FF0h
Vin+ to Vin-: $-V_{REF}$ to $+V_{REF}$	1	0	00	-128 to 127 (0 to 255)	0000h to 00FFh
	1	0	01	-512 to 511 (0 to 1023)	0000h to 03FFh
	1	0	10	-2048 to 2047 (0 to 4095)	0000h to 0FFFh
	1	1	00	-128 to 127	8000h to 7F00h
	1	1	01	-512 to 511	8000h to 7FC0h
	1	1	10	-2048 to 2047	8000h to 7FF0h

18.2.8 ADC12_B Conversion Modes

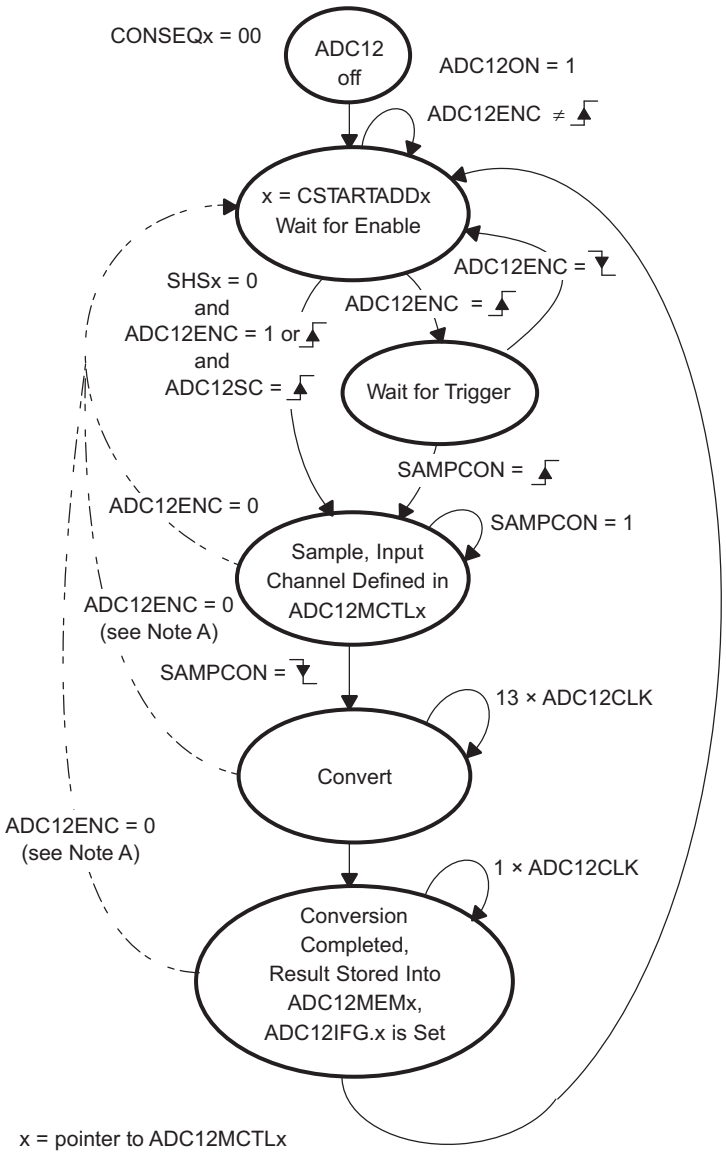
Table 18-2 shows the four operating modes that are selected by the CONSEQx bits. All state diagrams assume a 12-bit resolution setting.

Table 18-2. 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.

18.2.8.1 Single-Channel Single-Conversion Mode

A single channel is sampled and converted once. The ADC result is written to the ADC12MEMx that is defined by the CSTARTADDx bits. Figure 18-6 shows the flow of the single-channel single-conversion mode when RES = 0x2 for 12-bit mode. When ADC12SC triggers a conversion, the ADC12SC bit can trigger successive conversions. When any other trigger source is used, ADC12ENC must be toggled between each conversion.



A Conversion result is unpredictable.

Figure 18-6. Single-Channel Single-Conversion Mode

18.2.8.2 Sequence-of-Channels Mode (Autoscan Mode)

In sequence-of-channels mode, also called autoscan mode, a sequence of channels is sampled and converted once. The ADC results are written to the conversion memories starting with the ADC12MEMx that is defined by the CSTARTADDx bits. The sequence stops after the measurement of the channel with a set ADC12EOS bit. Figure 18-7 shows the sequence-of-channels mode when RES = 0x02 for 12-bit mode. When ADC12SC triggers a sequence, the ADC12SC bit can trigger successive sequences. When any other trigger source is used, ADC12ENC must be toggled between each sequence.

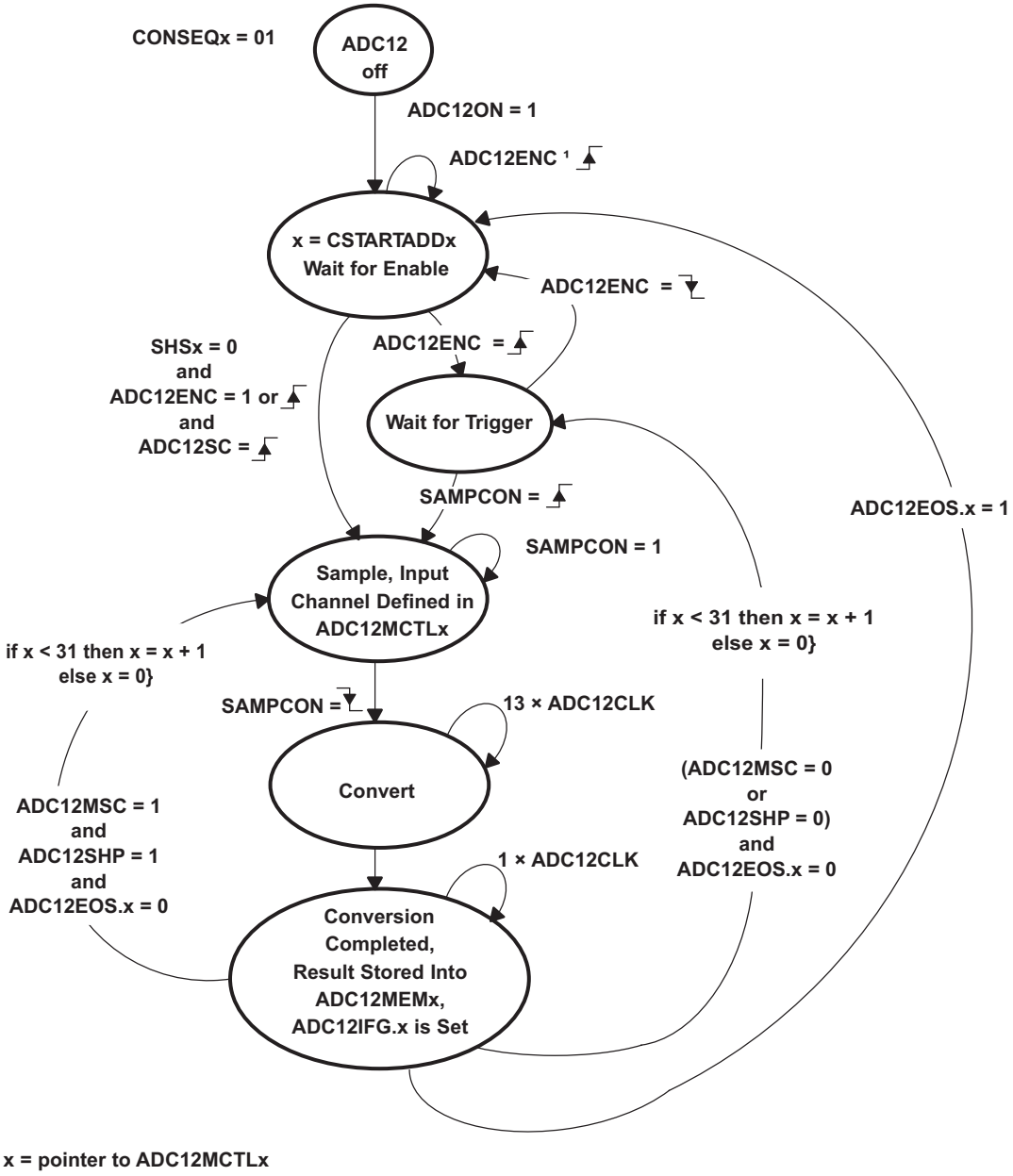


Figure 18-7. Sequence-of-Channels Mode

18.2.8.3 Repeat-Single-Channel Mode

In 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 18-8 shows the repeat-single-channel mode when RES = 0x2 for 12-bit mode.

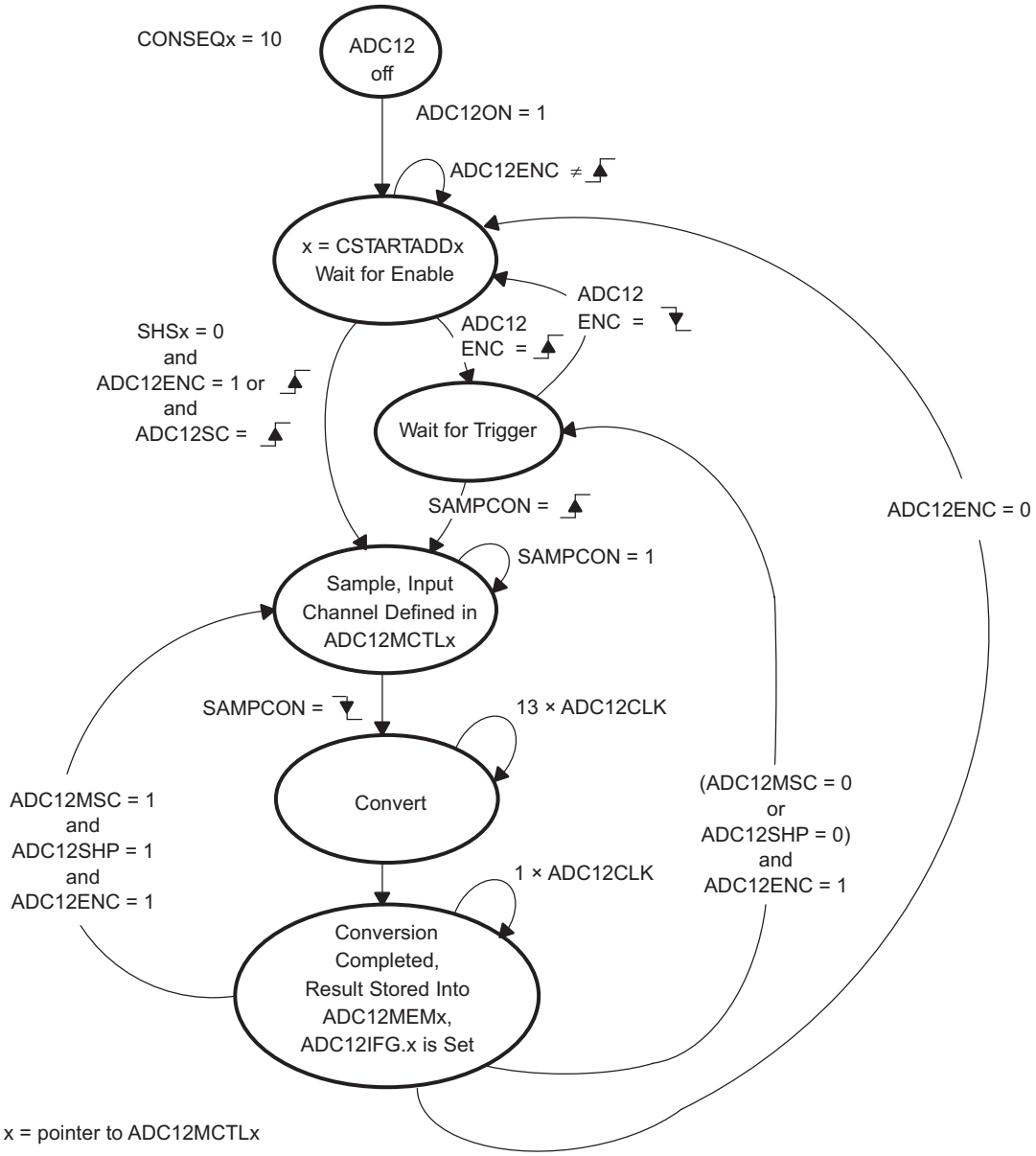


Figure 18-8. Repeat-Single-Channel Mode

18.2.8.4 Repeat-Sequence-of-Channels Mode (Repeated Autoscan Mode)

In repeat-sequence-of-channels mode, a sequence of channels is sampled and converted repeatedly. This mode is also called repeated autoscan mode. The ADC results are written to the conversion memories starting with the ADC12MEMx that is 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 18-9 shows the repeat-sequence-of-channels mode.

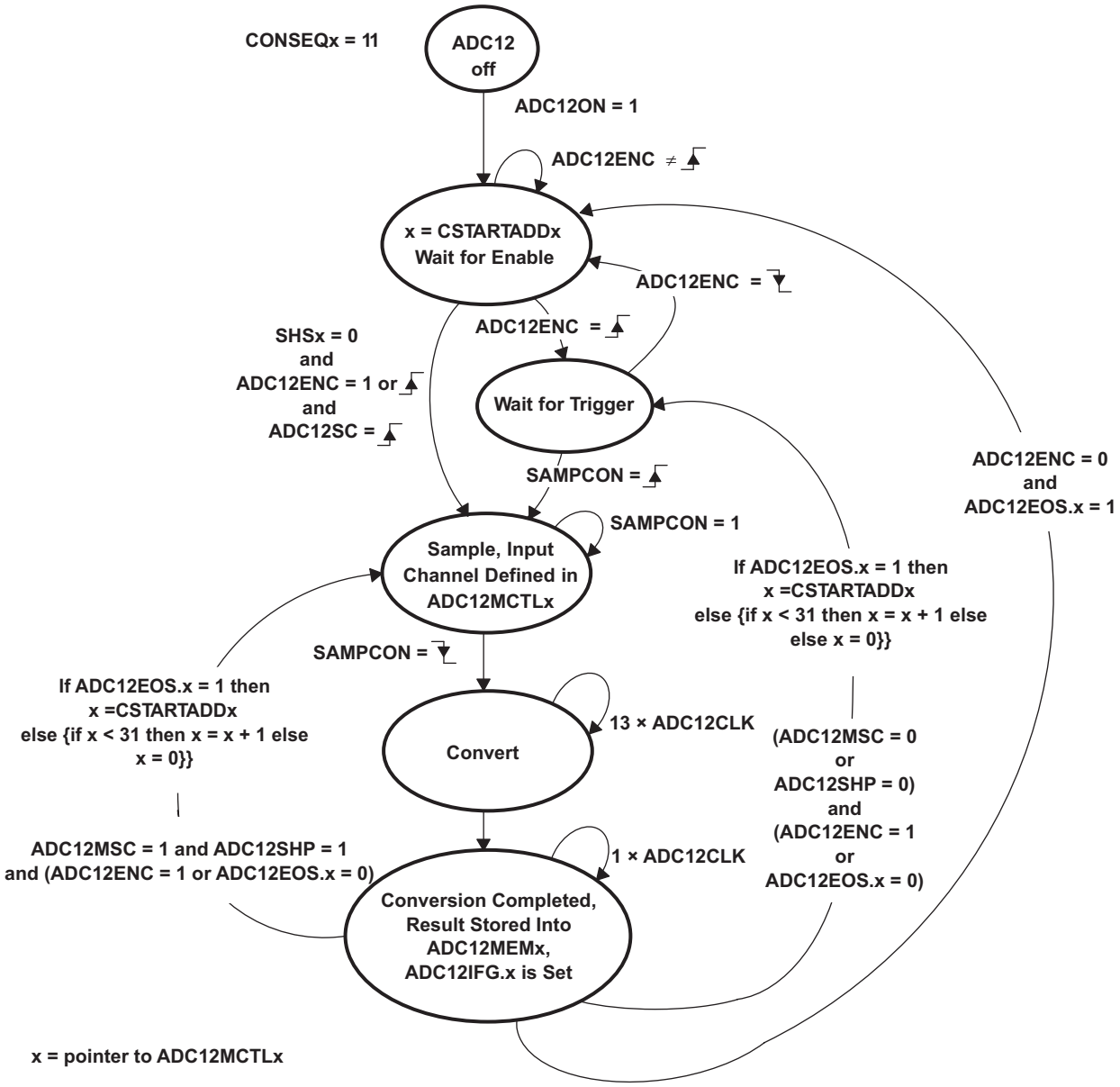


Figure 18-9. Repeat-Sequence-of-Channels Mode

18.2.8.5 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 (if the ADC local reference buffer is used, ADC12VRSEL= 0001, 0011, 0101, 0111, 1001, 1011, 1101, or 1111, there is one clock cycle before the successive conversion is triggered). 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.

18.2.8.6 Stopping Conversions

Stopping ADC12_B activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Reset ADC12ENC in single-channel single-conversion mode to stop a conversion immediately. The results are unreliable. For correct results, poll the busy bit until it is reset before clearing ADC12ENC.
- Reset ADC12ENC during repeat-single-channel operation to stop the converter at the end of the current conversion.
- Reset ADC12ENC during a sequence or repeat-sequence mode to stop the converter at the end of the current conversion.
- Stop any conversion mode 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.

18.2.9 Window Comparator

The window comparator allows to monitor analog signals without any CPU interaction. It is enabled for the desired ADC12MEMx conversion with the ADC12WINC bit in the ADC12MCTLx register. In the following the window comparator interrupts are listed:

- The ADC12LO interrupt flag (ADC12LOIFG) is set if the current result of the ADC12_B conversion is below the low threshold defined in register ADC12LO.
- The ADC12HI interrupt flag (ADC12HIIGH) is set if the current result of the ADC12_B conversion is greater than the high threshold defined in the register ADC12HI.
- The ADC12IN interrupt flag (ADC12INIFG) is set if the current result of the ADC12_B conversion is greater than the low threshold defined in register ADC12LO and less than the high threshold defined in ADC12HI.

These interrupts are generated independently of the conversion mode selected by the user. The update of the window comparator interrupt flags happen after the ADC12IFGx.

The lower and higher threshold in the ADC12LO and ADC12HI registers have to be given in the correct data format. If the binary unsigned data format is selected by ADC12DF = 0, then the thresholds in the registers ADC12LO and ADC12HI must be written as binary unsigned values. If the signed binary (2s complement) data format is selected by ADC12DF = 1, then the thresholds in the registers ADC12LO and ADC12HI must be written as signed binary (2s complement). Altering the ADC12DF register or the ADC12RES register resets the threshold registers.

The interrupt flags are reset by the user software. The ADC12_B sets the interrupt flags each time a new conversion result is available in the ADC12MEMx register if applicable. Interrupt flags are not cleared by hardware. The user software resets the window comparator interrupt flags per the application needs.

18.2.10 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user must enable the temperature sensor input channel by setting the ADC12TCMAP bit equal to 1 in the ADC12CTL3 register. The user must then select the analog input channel ADC12INCHx = 0x1E for the temperature sensor. Any other configuration is done as if an external channel were selected, including reference selection, conversion-memory selection, and so on. The temperature sensor is in the REF module.

A typical temperature sensor transfer function is shown in Figure 18-10. The transfer function shown in Figure 18-10 is only an example. Calibration is required to determine the corresponding voltages for a specific 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 (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 VREF+ 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.

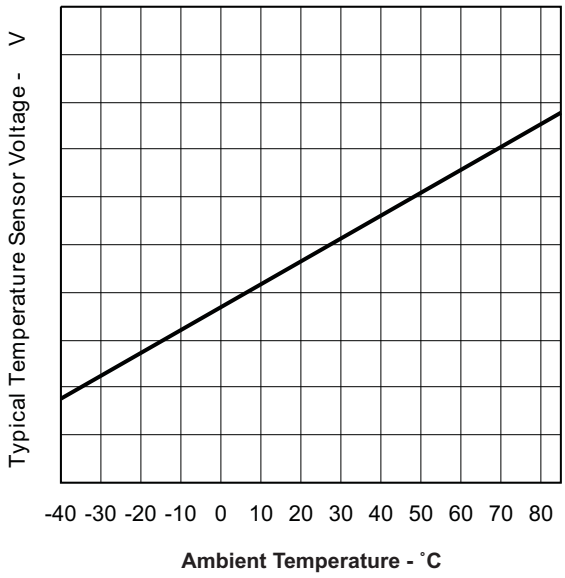


Figure 18-10. Typical Temperature Sensor Transfer Function

18.2.11 ADC12_B 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 ADC 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 18-11 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.

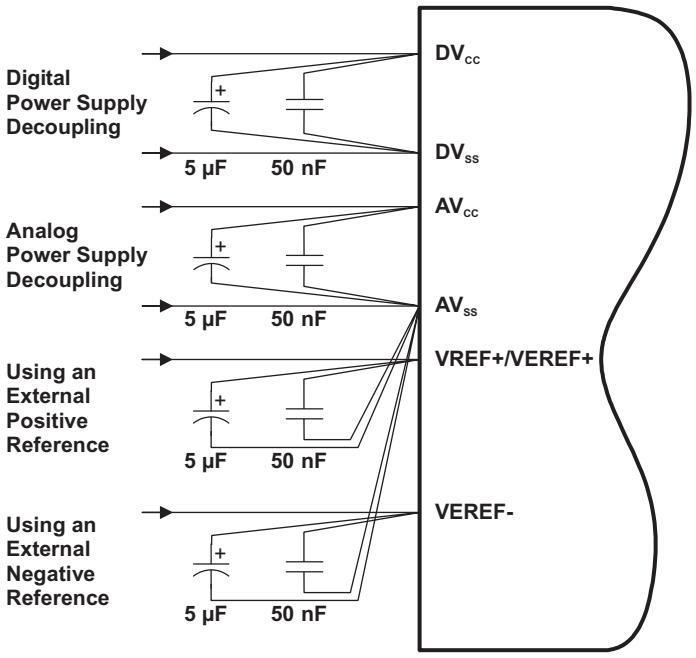


Figure 18-11. ADC12_B Grounding and Noise Considerations

18.2.12 ADC12_B Interrupts

The ADC12_B has 38 interrupt sources:

- ADC12IFG0 to ADC12IFG31
- ADC12OV: ADC12MEMx overflow
- ADC12TOV: ADC12_B conversion time overflow
- ADC12LOIFG, ADC12INIFG, and ADC12HIIFG for ADC12MEMx
- ADC12RDYIFG: ADC12_B local reference buffer ready

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 conversion result written into ADC12MEMx result register also sets the ADC12LOIFG, ADC12INIFG or ADC12HIIFG if applicable. 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. The ADC12RDYIFG is set once the ADC12_B local reference buffer is ready. It can be used during extended sample mode instead of adding the max ADC12_B local reference buffer settle time to the sample signal time.

18.2.12.1 ADC12IV, Interrupt Vector Generator

All ADC12_B interrupt sources are prioritized and combined to source a single interrupt vector. The interrupt vector register ADC12IV is used to determine which ADC12_B interrupt source requested an interrupt.

The highest-priority enabled ADC12_B interrupt generates a number in the ADC12IV register (see Section 18.3.15). This number can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. ADC12_B interrupts that are disabled do not affect the ADC12IV value.

Read access of the ADC12IV register automatically resets the highest pending interrupt condition and flag except the ADC12IFGx flags. ADC12IFGx bits are reset automatically by accessing their associated ADC12MEMx register or may be reset with software.

Write access of the ADC12IV register clears all pending interrupt conditions and flags.

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.

18.2.12.2 ADC12_B 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 software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself, are:

- ADC12IFG0 through ADC12IFG30, ADC12TOV, ADC12OV, ADC12LO, ADC12HI, ADC12IN, ADC12RDY: 16 cycles
- ADC12IFG31: 14 cycles

The interrupt handler for ADC12IFG31 shows a way to check immediately if a higher-prioritized interrupt occurred during the processing of ADC12IFG31. This saves nine cycles if another ADC12_B 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      ADHI         ; Vector 6: ADC12HIIFG
   JMP      ADLO         ; Vector 8: ADC12LOIFG
   JMP      ADIN         ; Vector A: ADC12INIFG
   JMP      ADM0         ; Vector C: ADC12IFG0
      ...               ; Vectors E-70
   JMP      ADM30        ; Vector 72: ADC12IFG30
      ...
            JMP                         ADRDY                                   ;
Vector 76: ADC12RDYIFG
;
; Handler for ADC12IFG31 starts here. No JMP required.
;
;
ADM31     MOV   &ADC12MEM31,xxx     ; Move result, flag is reset
          ...                       ; Other instruction needed?
          JMP   INT_ADC12           ; Check other int pending
;
; ADC12IFG30-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 ADC12MEMx overflow
          RETI                      ; Return;
ADHI      ...                       ; Handle window comparator high Interrupt
          RETI                      ; Return;
ADLO      ...                       ; Handle window comparator low Interrupt
          RETI                      ; Return;
ADIN      ...                       ; Handle window comparator in window Interrupt
          RETI                      ; Return;
ADRDY      ...                      ; Handle window comparator in window Interrupt
          RETI                      ; Return;
```

18.3 ADC12_B Registers

Table 18-3lists the memory-mapped registers for the ADC12_B. See the device-specific data sheet for the base memory address of these registers. All other register offset addresses not listed in Table 18-3 should be considered as reserved locations, and the register contents should not be modified.

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 18-3. ADC12_B Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	ADC12CTL0	ADC12_B Control 0	Read/write	Word	0000h	Section 18.3.1
00h	ADC12CTL0_L		Read/write	Byte	00h	
01h	ADC12CTL0_H		Read/write	Byte	00h	
02h	ADC12CTL1	ADC12_B Control 1	Read/write	Word	0000h	Section 18.3.2
02h	ADC12CTL1_L		Read/write	Byte	00h	
03h	ADC12CTL1_H		Read/write	Byte	00h	
04h	ADC12CTL2	ADC12_B Control 2	Read/write	Word	0020h	Section 18.3.3
04h	ADC12CTL2_L		Read/write	Byte	20h	
05h	ADC12CTL2_H		Read/write	Byte	00h	
06h	ADC12CTL3	ADC12_B Control 3	Read/write	Byte	0000h	Section 18.3.4
06h	ADC12CTL3_L		Read/write	Byte	00h	
07h	ADC12CTL3_H		Read/write		00h	
08h	ADC12LO	ADC12_B Window Comparator Low Threshold Register	Read/write	Word	0000h	Section 18.3.8
08h	ADC12LO_L		Read/write	Byte	00h	
09h	ADC12LO_H		Read/write	Byte	00h	
0Ah	ADC12HI	ADC12_B Window Comparator High Threshold Register	Read/write	Word	0FFFh	Section 18.3.7
0Ah	ADC12HI_L		Read/write	Byte	FFh	
0Bh	ADC12HI_H		Read/write	Byte	0Fh	
0Ch	ADC12IFGR0	ADC12_B Interrupt Flag 0	Read/write	Word	0000h	Section 18.3.12
0Ch	ADC12IFGR0_L		Read/write	Byte	00h	
0Dh	ADC12IFGR0_H		Read/write	Byte	00h	
0Eh	ADC12IFGR1	ADC12_B Interrupt Flag 1	Read/write	Word	0000h	Section 18.3.13
0Eh	ADC12IFGR1_L		Read/write	Byte	00h	
0Fh	ADC12IFGR1_H		Read/write	Byte	00h	
10h	ADC12IFGR2	ADC12_B Interrupt Flag 2	Read/write	Word	0000h	Section 18.3.14
10h	ADC12IFGR2_L		Read/write	Byte	00h	
11h	ADC12IFGR2_H		Read/write	Byte	00h	
12h	ADC12IER0	ADC12_B Interrupt Enable 0	Read/write	Word	0000h	Section 18.3.9
12h	ADC12IER0_L		Read/write	Byte	00h	
13h	ADC12IER0_H		Read/write	Byte	00h	
14h	ADC12IER1	ADC12_B Interrupt Enable 1	Read/write	Word	0000h	Section 18.3.10
14h	ADC12IER1_L		Read/write	Byte	00h	
15h	ADC12IER1_H		Read/write	Byte	00h	
16h	ADC12IER2	ADC12_B Interrupt Enable 2	Read/write	Word	0000h	Section 18.3.11
16h	ADC12IER2_L		Read/write	Byte	00h	
17h	ADC12IER2_H		Read/write	Byte	00h	

Table 18-3. ADC12_B Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
18h	ADC12IV	ADC12_B Interrupt Vector	Read	Word	0000h	Section 18.3.15
18h	ADC12IV_L		Read	Byte	00h	
19h	ADC12IV_H		Read	Byte	00h	
20h	ADC12MCTL0	ADC12_B Memory Control 0	Read/write	Word	0000h	Section 18.3.6
20h	ADC12MCTL0_L		Read/write	Byte	00h	
21h	ADC12MCTL0_H		Read/write	Byte	00h	
22h	ADC12MCTL1	ADC12_B Memory Control 1	Read/write	Word	0000h	Section 18.3.6
22h	ADC12MCTL1_L		Read/write	Byte	00h	
23h	ADC12MCTL1_H		Read/write	Byte	00h	
24h	ADC12MCTL2	ADC12_B Memory Control 2	Read/write	Word	0000h	Section 18.3.6
24h	ADC12MCTL2_L		Read/write	Byte	00h	
25h	ADC12MCTL2_H		Read/write	Byte	00h	
26h	ADC12MCTL3	ADC12_B Memory Control 3	Read/write	Word	0000h	Section 18.3.6
26h	ADC12MCTL3_L		Read/write	Byte	00h	
27h	ADC12MCTL3_H		Read/write	Byte	00h	
28h	ADC12MCTL4	ADC12_B Memory Control 4	Read/write	Word	0000h	Section 18.3.6
28h	ADC12MCTL4_L		Read/write	Byte	00h	
29h	ADC12MCTL4_H		Read/write	Byte	00h	
2Ah	ADC12MCTL5	ADC12_B Memory Control 5	Read/write	Word	0000h	Section 18.3.6
2Ah	ADC12MCTL5_L		Read/write	Byte	00h	
2Bh	ADC12MCTL5_H		Read/write	Byte	00h	
2Ch	ADC12MCTL6	ADC12_B Memory Control 6	Read/write	Word	0000h	Section 18.3.6
2Ch	ADC12MCTL6_L		Read/write	Byte	00h	
2Dh	ADC12MCTL6_H		Read/write	Byte	00h	
2Eh	ADC12MCTL7	ADC12_B Memory Control 7	Read/write	Word	0000h	Section 18.3.6
2Eh	ADC12MCTL7_L		Read/write	Byte	00h	
2Fh	ADC12MCTL7_H		Read/write	Byte	00h	
30h	ADC12MCTL8	ADC12_B Memory Control 8	Read/write	Word	0000h	Section 18.3.6
30h	ADC12MCTL8_L		Read/write	Byte	00h	
31h	ADC12MCTL8_H		Read/write	Byte	00h	
32h	ADC12MCTL9	ADC12_B Memory Control 9	Read/write	Word	0000h	Section 18.3.6
32h	ADC12MCTL9_L		Read/write	Byte	00h	
33h	ADC12MCTL9_H		Read/write	Byte	00h	
34h	ADC12MCTL10	ADC12_B Memory Control 10	Read/write	Word	0000h	Section 18.3.6
34h	ADC12MCTL10_L		Read/write	Byte	00h	
35h	ADC12MCTL10_H		Read/write	Byte	00h	
36h	ADC12MCTL11	ADC12_B Memory Control 11	Read/write	Word	0000h	Section 18.3.6
36h	ADC12MCTL11_L		Read/write	Byte	00h	
37h	ADC12MCTL11_H		Read/write	Byte	00h	
38h	ADC12MCTL12	ADC12_B Memory Control 12	Read/write	Word	0000h	Section 18.3.6
38h	ADC12MCTL12_L		Read/write	Byte	00h	
39h	ADC12MCTL12_H		Read/write	Byte	00h	
3Ah	ADC12MCTL13	ADC12_B Memory Control 13	Read/write	Word	0000h	Section 18.3.6
3Ah	ADC12MCTL13_L		Read/write	Byte	00h	
3Bh	ADC12MCTL13_H		Read/write	Byte	00h	

Table 18-3. ADC12_B Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
3Ch	ADC12MCTL14	ADC12_B Memory Control 14	Read/write	Word	0000h	Section 18.3.6
3Ch	ADC12MCTL14_L		Read/write	Byte	00h	
3Dh	ADC12MCTL14_H		Read/write	Byte	00h	
3Eh	ADC12MCTL15	ADC12_B Memory Control 15	Read/write	Word	0000h	Section 18.3.6
3Eh	ADC12MCTL15_L		Read/write	Byte	00h	
3Fh	ADC12MCTL15_H		Read/write	Byte	00h	
40h	ADC12MCTL16	ADC12_B Memory Control 16	Read/write	Word	0000h	Section 18.3.6
40h	ADC12MCTL16_L		Read/write	Byte	00h	
41h	ADC12MCTL16_H		Read/write	Byte	00h	
42h	ADC12MCTL17	ADC12_B Memory Control 17	Read/write	Word	0000h	Section 18.3.6
42h	ADC12MCTL17_L		Read/write	Byte	00h	
43h	ADC12MCTL17_H		Read/write	Byte	00h	
44h	ADC12MCTL18	ADC12_B Memory Control 18	Read/write	Word	0000h	Section 18.3.6
44h	ADC12MCTL18_L		Read/write	Byte	00h	
45h	ADC12MCTL18_H		Read/write	Byte	00h	
46h	ADC12MCTL19	ADC12_B Memory Control 19	Read/write	Word	0000h	Section 18.3.6
46h	ADC12MCTL19_L		Read/write	Byte	00h	
47h	ADC12MCTL19_H		Read/write	Byte	00h	
48h	ADC12MCTL20	ADC12_B Memory Control 20	Read/write	Word	0000h	Section 18.3.6
48h	ADC12MCTL20_L		Read/write	Byte	00h	
49h	ADC12MCTL20_H		Read/write	Byte	00h	
4Ah	ADC12MCTL21	ADC12_B Memory Control 21	Read/write	Word	0000h	Section 18.3.6
4Ah	ADC12MCTL21_L		Read/write	Byte	00h	
4Bh	ADC12MCTL21_H		Read/write	Byte	00h	
4Ch	ADC12MCTL22	ADC12_B Memory Control 22	Read/write	Word	0000h	Section 18.3.6
4Ch	ADC12MCTL22_L		Read/write	Byte	00h	
4Dh	ADC12MCTL22_H		Read/write	Byte	00h	
4Eh	ADC12MCTL23	ADC12_B Memory Control 23	Read/write	Word	0000h	Section 18.3.6
4Eh	ADC12MCTL23_L		Read/write	Byte	00h	
4Fh	ADC12MCTL23_H		Read/write	Byte	00h	
50h	ADC12MCTL24	ADC12_B Memory Control 24	Read/write	Word	0000h	Section 18.3.6
50h	ADC12MCTL24_L		Read/write	Byte	00h	
51h	ADC12MCTL24_H		Read/write	Byte	00h	
52h	ADC12MCTL25	ADC12_B Memory Control 25	Read/write	Word	0000h	Section 18.3.6
52h	ADC12MCTL25_L		Read/write	Byte	00h	
53h	ADC12MCTL25_H		Read/write	Byte	00h	
54h	ADC12MCTL26	ADC12_B Memory Control 26	Read/write	Word	0000h	Section 18.3.6
54h	ADC12MCTL26_L		Read/write	Byte	00h	
55h	ADC12MCTL26_H		Read/write	Byte	00h	
56h	ADC12MCTL27	ADC12_B Memory Control 27	Read/write	Word	0000h	Section 18.3.6
56h	ADC12MCTL27_L		Read/write	Byte	00h	
57h	ADC12MCTL27_H		Read/write	Byte	00h	
58h	ADC12MCTL28	ADC12_B Memory Control 28	Read/write	Word	0000h	Section 18.3.6
58h	ADC12MCTL28_L		Read/write	Byte	00h	
59h	ADC12MCTL28_H		Read/write	Byte	00h	

Table 18-3. ADC12_B Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
5Ah	ADC12MCTL29	ADC12_B Memory Control 29	Read/write	Word	0000h	Section 18.3.6
5Ah	ADC12MCTL29_L		Read/write	Byte	00h	
5Bh	ADC12MCTL29_H		Read/write	Byte	00h	
5Ch	ADC12MCTL30	ADC12_B Memory Control 30	Read/write	Word	0000h	Section 18.3.6
5Ch	ADC12MCTL30_L		Read/write	Byte	00h	
5Dh	ADC12MCTL30_H		Read/write	Byte	00h	
5Eh	ADC12MCTL31	ADC12_B Memory Control 31	Read/write	Word	0000h	Section 18.3.6
5Eh	ADC12MCTL31_L		Read/write	Byte	00h	
5Fh	ADC12MCTL31_H		Read/write	Byte	00h	
60h	ADC12MEM0	ADC12_B Memory 0	Read/write	Word	undefined	Section 18.3.5
60h	ADC12MEM0_L		Read/write	Byte	undefined	
61h	ADC12MEM0_H		Read/write	Byte	undefined	
62h	ADC12MEM1	ADC12_B Memory 1	Read/write	Word	undefined	Section 18.3.5
62h	ADC12MEM1_L		Read/write	Byte	undefined	
63h	ADC12MEM1_H		Read/write	Byte	undefined	
64h	ADC12MEM2	ADC12_B Memory 2	Read/write	Word	undefined	Section 18.3.5
64h	ADC12MEM2_L		Read/write	Byte	undefined	
65h	ADC12MEM2_H		Read/write	Byte	undefined	
66h	ADC12MEM3	ADC12_B Memory 3	Read/write	Word	undefined	Section 18.3.5
66h	ADC12MEM3_L		Read/write	Byte	undefined	
67h	ADC12MEM3_H		Read/write	Byte	undefined	
68h	ADC12MEM4	ADC12_B Memory 4	Read/write	Word	undefined	Section 18.3.5
68h	ADC12MEM4_L		Read/write	Byte	undefined	
69h	ADC12MEM4_H		Read/write	Byte	undefined	
6Ah	ADC12MEM5	ADC12_B Memory 5	Read/write	Word	undefined	Section 18.3.5
6Ah	ADC12MEM5_L		Read/write	Byte	undefined	
6Bh	ADC12MEM5_H		Read/write	Byte	undefined	
6Ch	ADC12MEM6	ADC12_B Memory 6	Read/write	Word	undefined	Section 18.3.5
6Ch	ADC12MEM6_L		Read/write	Byte	undefined	
6Dh	ADC12MEM6_H		Read/write	Byte	undefined	
6Eh	ADC12MEM7	ADC12_B Memory 7	Read/write	Word	undefined	Section 18.3.5
6Eh	ADC12MEM7_L		Read/write	Byte	undefined	
6Fh	ADC12MEM7_H		Read/write	Byte	undefined	
70h	ADC12MEM8	ADC12_B Memory 8	Read/write	Word	undefined	Section 18.3.5
70h	ADC12MEM8_L		Read/write	Byte	undefined	
71h	ADC12MEM8_H		Read/write	Byte	undefined	
72h	ADC12MEM9	ADC12_B Memory 9	Read/write	Word	undefined	Section 18.3.5
72h	ADC12MEM9_L		Read/write	Byte	undefined	
73h	ADC12MEM9_H		Read/write	Byte	undefined	
74h	ADC12MEM10	ADC12_B Memory 10	Read/write	Word	undefined	Section 18.3.5
74h	ADC12MEM10_L		Read/write	Byte	undefined	
75h	ADC12MEM10_H		Read/write	Byte	undefined	
76h	ADC12MEM11	ADC12_B Memory 11	Read/write	Word	undefined	Section 18.3.5
76h	ADC12MEM11_L		Read/write	Byte	undefined	
77h	ADC12MEM11_H		Read/write	Byte	undefined	

Table 18-3. ADC12_B Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
78h	ADC12MEM12	ADC12_B Memory 12	Read/write	Word	undefined	Section 18.3.5
78h	ADC12MEM12_L		Read/write	Byte	undefined	
79h	ADC12MEM12_H		Read/write	Byte	undefined	
7Ah	ADC12MEM13	ADC12_B Memory 13	Read/write	Word	undefined	Section 18.3.5
7Ah	ADC12MEM13_L		Read/write	Byte	undefined	
7Bh	ADC12MEM13_H		Read/write	Byte	undefined	
7Ch	ADC12MEM14	ADC12_B Memory 14	Read/write	Word	undefined	Section 18.3.5
7Ch	ADC12MEM14_L		Read/write	Byte	undefined	
7Dh	ADC12MEM14_H		Read/write	Byte	undefined	
7Eh	ADC12MEM15	ADC12_B Memory 15	Read/write	Word	undefined	Section 18.3.5
7Eh	ADC12MEM15_L		Read/write	Byte	undefined	
7Fh	ADC12MEM15_H		Read/write	Byte	undefined	
80h	ADC12MEM16	ADC12_B Memory 16	Read/write	Word	undefined	Section 18.3.5
80h	ADC12MEM16_L		Read/write	Byte	undefined	
81h	ADC12MEM16_H		Read/write	Byte	undefined	
82h	ADC12MEM17	ADC12_B Memory 17	Read/write	Word	undefined	Section 18.3.5
82h	ADC12MEM17_L		Read/write	Byte	undefined	
83h	ADC12MEM17_H		Read/write	Byte	undefined	
84h	ADC12MEM18	ADC12_B Memory 18	Read/write	Word	undefined	Section 18.3.5
84h	ADC12MEM18_L		Read/write	Byte	undefined	
85h	ADC12MEM18_H		Read/write	Byte	undefined	
86h	ADC12MEM19	ADC12_B Memory 19	Read/write	Word	undefined	Section 18.3.5
86h	ADC12MEM19_L		Read/write	Byte	undefined	
87h	ADC12MEM19_H		Read/write	Byte	undefined	
88h	ADC12MEM20	ADC12_B Memory 20	Read/write	Word	undefined	Section 18.3.5
88h	ADC12MEM20_L		Read/write	Byte	undefined	
89h	ADC12MEM20_H		Read/write	Byte	undefined	
8Ah	ADC12MEM21	ADC12_B Memory 21	Read/write	Word	undefined	Section 18.3.5
8Ah	ADC12MEM21_L		Read/write	Byte	undefined	
8Bh	ADC12MEM21_H		Read/write	Byte	undefined	
8Ch	ADC12MEM22	ADC12_B Memory 22	Read/write	Word	undefined	Section 18.3.5
8Ch	ADC12MEM22_L		Read/write	Byte	undefined	
8Dh	ADC12MEM22_H		Read/write	Byte	undefined	
8Eh	ADC12MEM23	ADC12_B Memory 23	Read/write	Word	undefined	Section 18.3.5
8Eh	ADC12MEM23_L		Read/write	Byte	undefined	
8Fh	ADC12MEM23_H		Read/write	Byte	undefined	
90h	ADC12MEM24	ADC12_B Memory 24	Read/write	Word	undefined	Section 18.3.5
90h	ADC12MEM24_L		Read/write	Byte	undefined	
91h	ADC12MEM24_H		Read/write	Byte	undefined	
92h	ADC12MEM25	ADC12_B Memory 25	Read/write	Word	undefined	Section 18.3.5
92h	ADC12MEM25_L		Read/write	Byte	undefined	
93h	ADC12MEM25_H		Read/write	Byte	undefined	
94h	ADC12MEM26	ADC12_B Memory 26	Read/write	Word	undefined	Section 18.3.5
94h	ADC12MEM26_L		Read/write	Byte	undefined	
95h	ADC12MEM26_H		Read/write	Byte	undefined	

Table 18-3. ADC12_B Registers (continued)

Offset	Acronym	Register Name	Type	Access	Reset	Section
96h	ADC12MEM27	ADC12_B Memory 27	Read/write	Word	undefined	Section 18.3.5
96h	ADC12MEM27_L		Read/write	Byte	undefined	
97h	ADC12MEM27_H		Read/write	Byte	undefined	
98h	ADC12MEM28	ADC12_B Memory 28	Read/write	Word	undefined	Section 18.3.5
98h	ADC12MEM28_L		Read/write	Byte	undefined	
99h	ADC12MEM28_H		Read/write	Byte	undefined	
9Ah	ADC12MEM29	ADC12_B Memory 29	Read/write	Word	undefined	Section 18.3.5
9Ah	ADC12MEM29_L		Read/write	Byte	undefined	
9Bh	ADC12MEM29_H		Read/write	Byte	undefined	
9Ch	ADC12MEM30	ADC12_B Memory 30	Read/write	Word	undefined	Section 18.3.5
9Ch	ADC12MEM30_L		Read/write	Byte	undefined	
9Dh	ADC12MEM30_H		Read/write	Byte	undefined	
9Eh	ADC12MEM31	ADC12_B Memory 31	Read/write	Word	undefined	Section 18.3.5
9Eh	ADC12MEM31_L		Read/write	Byte	undefined	
9Fh	ADC12MEM31_H		Read/write	Byte	undefined	

18.3.1 ADC12CTL0 Register (offset = 00h) [reset = 0000h]

ADC12_B Control 0 Register

Figure 18-12. ADC12CTL0 Register

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	Reserved		ADC12ON	Reserved		ADC12ENC	ADC12SC
rw-(0)	r-0	r-0	rw-(0)	r-0	r-0	rw-(0)	rw-(0)

Can be modified only when ADC12ENC = 0.

Table 18-4. ADC12CTL0 Register Description

Bit	Field	Type	Reset	Description
15-12	ADC12SHT1x	RW	0	ADC12_B sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM8 to ADC12MEM23. Can be modified only when ADC12ENC = 0. 0000 = 4 ADC12CLK cycles 0001 = 8 ADC12CLK cycles 0010 = 16 ADC12CLK cycles 0011 = 32 ADC12CLK cycles 0100 = 64 ADC12CLK cycles 0101 = 96 ADC12CLK cycles 0110 = 128 ADC12CLK cycles 0111 = 192 ADC12CLK cycles 1000 = 256 ADC12CLK cycles 1001 = 384 ADC12CLK cycles 1010 = 512 ADC12CLK cycles 1011 = 768 ADC12CLK cycles 1100 = 1024 ADC12CLK cycles 1101 = 1024 ADC12CLK cycles 1110 = 1024 ADC12CLK cycles 1111 = 1024 ADC12CLK cycles
11-8	ADC12SHT0x	RW	0	ADC12_B sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM0 to ADC12MEM7 and ADC12MEM24 to ADC12MEM31. Can be modified only when ADC12ENC = 0. 0000 = 4 ADC12CLK cycles 0001 = 8 ADC12CLK cycles 0010 = 16 ADC12CLK cycles 0011 = 32 ADC12CLK cycles 0100 = 64 ADC12CLK cycles 0101 = 96 ADC12CLK cycles 0110 = 128 ADC12CLK cycles 0111 = 192 ADC12CLK cycles 1000 = 256 ADC12CLK cycles 1001 = 384 ADC12CLK cycles 1010 = 512 ADC12CLK cycles 1011 = 768 ADC12CLK cycles 1100 = 1024 ADC12CLK cycles 1101 = 1024 ADC12CLK cycles 1110 = 1024 ADC12CLK cycles 1111 = 1024 ADC12CLK cycles

Table 18-4. ADC12CTL0 Register Description (continued)

Bit	Field	Type	Reset	Description
7	ADC12MSC	RW	0	ADC12_B multiple sample and conversion. Valid only for sequence or repeated modes. Can be modified only when ADC12ENC = 0. 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.
6-5	Reserved	R	0	Reserved. Always reads as 0.
4	ADC12ON	RW	0	ADC12_B on. Can be modified only when ADC12ENC = 0. 0 = ADC12_B off 1 = ADC12_B on
3-2	Reserved	R	0	Reserved. Always reads as 0.
1	ADC12ENC	RW	0	ADC12_B enable conversion 0 = ADC12_B disabled 1 = ADC12_B enabled
0	ADC12SC	RW	0	ADC12_B 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

18.3.2 ADC12CTL1 Register (offset = 02h) [reset = 0000h]

ADC12_B Control 1 Register

Figure 18-13. ADC12CTL1 Register

15	14	13	12	11	10	9	8
Reserved	ADC12PDIV		ADC12SHSx			ADC12SHP	ADC12ISSH
r-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)

Can be modified only when ADC12ENC = 0.

Table 18-5. ADC12CTL1 Register Description

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14-13	ADC12PDIV	RW	0h	ADC12_B predivider. This bit predivides the selected ADC12_B clock source. 00 = Predivide by 1 01 = Predivide by 4 10 = Predivide by 32 11 = Predivide by 64
12-10	ADC12SHSx	RW	0h	ADC12_B sample-and-hold source select 000 = ADC12SC bit 001 = see the device-specific data sheet for source 010 = see the device-specific data sheet for source 011 = see the device-specific data sheet for source 100 = see the device-specific data sheet for source 101 = see the device-specific data sheet for source 110 = see the device-specific data sheet for source 111 = see the device-specific data sheet for source
9	ADC12SHP	RW	0h	ADC12_B 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.
8	ADC12ISSH	RW	0h	ADC12_B invert signal sample-and-hold 0 = The sample-input signal is not inverted. 1 = The sample-input signal is inverted.
7-5	ADC12DIVx	RW	0h	ADC12_B clock divider 000 = /1 001 = /2 010 = /3 011 = /4 100 = /5 101 = /6 110 = /7 111 = /8
4-3	ADC12SSELx	RW	0h	ADC12_B clock source select 00 = ADC12OSC (MODOSC) 01 = ACLK 10 = MCLK 11 = SMCLK

Table 18-5. ADC12CTL1 Register Description (continued)

Bit	Field	Type	Reset	Description
2-1	ADC12CONSEQx	RW	0h	ADC12_B conversion sequence mode select 00 = Single-channel, single-conversion 01 = Sequence-of-channels 10 = Repeat-single-channel 11 = Repeat-sequence-of-channels
0	ADC12BUSY	R	0h	ADC12_B busy. This bit indicates an active sample or conversion operation. 0 = No operation is active. 1 = A sequence, sample, or conversion is active.

18.3.3 ADC12CTL2 Register (offset = 04h) [reset = 0020h]

ADC12_B Control 2 Register

Figure 18-14. ADC12CTL2 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		ADC12RES		ADC12DF	Reserved		ADC12PWRMD
r0	r0	rw-(1)	rw-(0)	rw-(0)	r0	r0	rw-(0)

Table 18-6. ADC12CTL2 Register Description

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	ADC12RES	RW	0h	ADC12_B resolution. This bit defines the conversion result resolution. 00 = 8 bit (10 clock cycle conversion time) 01 = 10 bit (12 clock cycle conversion time) 10 = 12 bit (14 clock cycle conversion time) 11 = Reserved
3	ADC12DF	RW	0h	ADC12_B data read-back format. Data is always stored in the binary unsigned format. 0 = Binary unsigned. Theoretically for ADC12DIF=0 and 12-bit mode the analog input voltage – VREF results in 0000h, the analog input voltage + VREF results in 0FFFh. 1 = Signed binary (2s complement), left aligned. Theoretically forADC12DIF=0 and 12-bit mode the analog input voltage – VREF results in 8000h, the analog input voltage + VREF results in 7FF0h.
2-1	Reserved	R	0h	Reserved. Always reads as 0.
0	ADC12PWRMD	RW	0h	Enables ADC power mode for sample rates at or below 50 ksps. 0 = Regular power mode where sample rate is not restricted 1 = Low power mode enable, sample rate can not be greater than 50 ksps

18.3.4 ADC12CTL3 Register (offset = 06h) [reset = 0000h]

ADC12_B Control 3 Register

Figure 18-15. ADC12CTL3 Register

15	14	13	12	11	10	9	8
Reserved				ADC12ICH3MAP	ADC12ICH2MAP	ADC12ICH1MAP	ADC12ICH0MAP
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12TCMAP	ADC12BATMAP	Reserved	ADC12CSTARTADDx				
rw-(0)	rw-(0)	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ADC12ENC = 0.

Table 18-7. ADC12CTL3 Register Description

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11	ADC12ICH3MAP	RW	0h	Controls internal channel 3 selection to ADC input channel A26. Can be modified only when ADC12ENC = 0. 0 = ADC input channel internal 3 is not selected 1 = ADC input channel internal 3 is selected for ADC input channel A26
10	ADC12ICH2MAP	RW	0h	Controls internal channel 2 selection to ADC input channel A27. Can be modified only when ADC12ENC = 0. 0 = ADC input channel internal 2 is not selected 1 = ADC input channel internal 2 is selected for ADC input channel A27
9	ADC12ICH1MAP	RW	0h	Controls internal channel 1 selection to ADC input channel A28. Can be modified only when ADC12ENC = 0. 0 = ADC input channel internal 1 is not selected 1 = ADC input channel internal 1 is selected for ADC input channel A28
8	ADC12ICH0MAP	RW	0h	Controls internal channel 0 selection to ADC input channel A29. Can be modified only when ADC12ENC = 0. 0 = ADC input channel internal 0 is not selected 1 = ADC input channel internal 0 is selected for ADC input channel A29
7	ADC12TCMAP	RW	0h	Controls temperature sensor ADC input channel selection. Can be modified only when ADC12ENC = 0. 0 = ADC internal temperature sensor channel is not selected for ADC 1 = ADC internal temperature sensor channel is selected for ADC input channel A30
6	ADC12BATMAP	RW	0h	Controls 1/2 AVCC ADC input channel selection. Can be modified only when ADC12ENC = 0. 0 = ADC internal 1/2 x AVCC channel is not selected for ADC 1 = ADC internal 1/2 x AVCC channel is selected for ADC input channel A31
5	Reserved	R	0h	Reserved. Always reads as 0.
4-0	ADC12CSTARTADDx	RW	0h	ADC12_B conversion start address. These bits select which ADC12_B conversion memory register is used for a single conversion or for the first conversion in a sequence. The value of CSTARTADDx is 0h to 1Fh, corresponding to ADC12MEM0 to ADC12MEM31. Can be modified only when ADC12ENC = 0.

18.3.5 ADC12MEMx Register (x = 0 to 31)

ADC12_B Conversion Memory x Register (x = 0 to 31)

Figure 18-16. ADC12MEMx Register

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

Table 18-8. ADC12MEMx Register Description

Bit	Field	Type	Reset	Description
15-0	Conversion Results	RW	0h	If ADC12DF = 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. If the user writes to the conversion memory registers, the results are corrupted. If ADC12DF = 1: 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. The data is stored in the right-justified format and is converted to the left-justified 2s-complement format during read back. If the user writes to the conversion memory registers, the results are corrupted.

18.3.6 ADC12MCTLx Register (x = 0 to 31)

ADC12_B Conversion Memory Control x Register (x = 0 to 31)

Figure 18-17. ADC12MCTLx Register

15	14	13	12	11	10	9	8
Reserved	ADC12WINC	ADC12DIF	Reserved	ADC12VRSEL			
r0	rw-(0)	rw-(0)	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12EOS	Reserved		ADC12INCHx				
rw-(0)	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ADC12ENC = 0.

Table 18-9. ADC12MCTLx Register Description

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14	ADC12WINC	RW	0h	Comparator window enable. Can be modified only when ADC12ENC = 0. 0 = Comparator window disabled 1 = Comparator window enabled
13	ADC12DIF	RW	0h	Differential mode. Can be modified only when ADC12ENC = 0. 0 = Single-ended mode enabled 1 = Differential mode enabled
12	Reserved	R	0h	Reserved. Always reads as 0.
11-8	ADC12VRSEL	RW	0h	Selects combinations of VR+ and VR- sources as well as the buffer selection and buffer on or off. Note: there is only one buffer so it can be used for either VR+ or VR-, but not both. Can be modified only when ADC12ENC = 0. 0000 = VR+ = AVCC, VR- = AVSS 0001 = VR+ = VREF buffered, VR- = AVSS 0010 = VR+ = VEREF-, VR- = AVSS 0011 = VR+ = VEREF+ buffered, VR- = AVSS 0100 = VR+ = VEREF+, VR- = AVSS 0101 = VR+ = AVCC, VR- = VEREF+ buffered 0110 = VR+ = AVCC, VR- = VEREF+ 0111 = VR+ = VREF buffered, VR- = VEREF+ 1000 = Reserved 1001 = VR+ = AVCC, VR- = VREF buffered 1010 = Reserved 1011 = VR+ = VEREF+, VR- = VREF buffered 1100 = VR+ = AVCC, VR- = VEREF- 1101 = VR+ = VREF buffered, VR- = VEREF- 1110 = VR+ = VEREF+, VR- = VEREF- 1111 = VR+ = VEREF+ buffered, VR- = VEREF-
7	ADC12EOS	RW	0h	End of sequence. Indicates the last conversion in a sequence. Can be modified only when ADC12ENC = 0. 0 = Not end of sequence 1 = End of sequence
6-5	Reserved	R	0h	Reserved. Always reads as 0.

Table 18-9. ADC12MCTLx Register Description (continued)

Bit	Field	Type	Reset	Description
4-0	ADC12INCHx	RW	0h	Input channel select. If even channels are set as differential, then odd channel configuration is ignored. Can be modified only when ADC12ENC = 0. 00000 = If ADC12DIF = 0: A0; If ADC12DIF = 1: Ain+ = A0, Ain- =A1 00001 = If ADC12DIF = 0: A1; If ADC12DIF = 1: Ain+ = A0, Ain- =A1 00010 = If ADC12DIF = 0: A2; If ADC12DIF = 1: Ain+ = A2, Ain- =A3 00011 = If ADC12DIF = 0: A3; If ADC12DIF = 1: Ain+ = A2, Ain- =A3 00100 = If ADC12DIF = 0: A4; If ADC12DIF = 1: Ain+ = A4, Ain- =A5 00101 = If ADC12DIF = 0: A5; If ADC12DIF = 1: Ain+ = A4, Ain- =A5 00110 = If ADC12DIF = 0: A6; If ADC12DIF = 1: Ain+ = A6, Ain- =A7 00111 = If ADC12DIF = 0: A7; If ADC12DIF = 1: Ain+ = A6, Ain- =A7 01000 = If ADC12DIF = 0: A8; If ADC12DIF = 1: Ain+ = A8, Ain- =A9 01001 = If ADC12DIF = 0: A9; If ADC12DIF = 1: Ain+ = A8, Ain- =A9 01010 = If ADC12DIF = 0: A10; If ADC12DIF = 1: Ain+ = A10, Ain- =A11 01011 = If ADC12DIF = 0: A11; If ADC12DIF = 1: Ain+ = A10, Ain- =A11 01100 = If ADC12DIF = 0: A12; If ADC12DIF = 1: Ain+ = A12, Ain- =A13 01101 = If ADC12DIF = 0: A13; If ADC12DIF = 1: Ain+ = A12, Ain- =A13 01110 = If ADC12DIF = 0: A14; If ADC12DIF = 1: Ain+ = A14, Ain- =A15 01111 = If ADC12DIF = 0: A15; If ADC12DIF = 1: Ain+ = A14, Ain- =A15 10000 = If ADC12DIF = 0: A16; If ADC12DIF = 1: Ain+ = A16, Ain- =A17 10001 = If ADC12DIF = 0: A17; If ADC12DIF = 1: Ain+ = A16, Ain- =A17 10010 = If ADC12DIF = 0: A18; If ADC12DIF = 1: Ain+ = A18, Ain- =A19 10011 = If ADC12DIF = 0: A19; If ADC12DIF = 1: Ain+ = A18, Ain- =A19 10100 = If ADC12DIF = 0: A20; If ADC12DIF = 1: Ain+ = A20, Ain- =A21 10101 = If ADC12DIF = 0: A21; If ADC12DIF = 1: Ain+ = A20, Ain- =A21 10110 = If ADC12DIF = 0: A22; If ADC12DIF = 1: Ain+ = A22, Ain- =A23 10111 = If ADC12DIF = 0: A23; If ADC12DIF = 1: Ain+ = A22, Ain- =A23 11000 = If ADC12DIF = 0: A24; If ADC12DIF = 1: Ain+ = A24, Ain- =A25 11001 = If ADC12DIF = 0: A25; If ADC12DIF = 1: Ain+ = A24, Ain- =A25 11010 = If ADC12DIF = 0: A26; If ADC12DIF = 1: Ain+ = A26, Ain- =A27 11011 = If ADC12DIF = 0: A27; If ADC12DIF = 1: Ain+ = A26, Ain- =A27 11100 = If ADC12DIF = 0: A28; If ADC12DIF = 1: Ain+ = A28, Ain- =A29 11101 = If ADC12DIF = 0: A29; If ADC12DIF = 1: Ain+ = A28, Ain- =A29 11110 = If ADC12DIF = 0: A30; If ADC12DIF = 1: Ain+ = A30, Ain- =A31 11111 = If ADC12DIF = 0: A31; If ADC12DIF = 1: Ain+ = A30, Ain- =A31

18.3.7 ADC12HI Register (offset = 0Ah) [reset = 0FFFh]

ADC12_B Window Comparator High Threshold Register

Figure 18-18. ADC12HI Register

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

Table 18-10. ADC12HI Register Description

Bit	Field	Type	Reset	Description
15-0	High Threshold	RW	0FFFh	If ADC12DF = 0: The 12-bit threshold value is right justified when ADC12DF = 0. Bits 15-12 are 0. Bit 11 is the MSB. Bits 11-10 are 0 in 10-bit mode, and bits 11-8 are 0 in 8-bit mode. If ADC12DF = 1: The 12-bit threshold value is left justified when ADC12DF = 1, 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.

18.3.8 ADC12LO Register (offset = 08h) [reset = 0000h]

ADC12_B Window Comparator Low Threshold Register

Figure 18-19. ADC12LO Register

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

Table 18-11. ADC12LO Register Description

Bit	Field	Type	Reset	Description
15-0	Low Threshold	RW	0h	If ADC12DF = 0: The 12-bit threshold value is right justified when ADC12DF = 0. Bits 15-12 are 0. Bit 11 is the MSB. Bits 11-10 are 0 in 10-bit mode, and bits 11-8 are 0 in 8-bit mode. If ADC12DF = 1: The 12-bit threshold value is left justified when ADC12DF = 1, 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.

18.3.9 ADC12IER0 Register (offset = 12h) [reset = 0000h]

ADC12_B Interrupt Enable 0 Register

Figure 18-20. ADC12IER0 Register

15	14	13	12	11	10	9	8
ADC12IE15	ADC12IE14	ADC12IE13	ADC12IE12	ADC12IE11	ADC12IE10	ADC12IE9	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)

Table 18-12. ADC12IER0 Register Description

Bit	Field	Type	Reset	Description
15	ADC12IE15	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG15 bit. 0 = Interrupt disabled 1 = Interrupt enabled
14	ADC12IE14	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG14 bit. 0 = Interrupt disabled 1 = Interrupt enabled
13	ADC12IE13	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG13 bit. 0 = Interrupt disabled 1 = Interrupt enabled
12	ADC12IE12	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG12 bit. 0 = Interrupt disabled 1 = Interrupt enabled
11	ADC12IE11	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG11 bit. 0 = Interrupt disabled 1 = Interrupt enabled
10	ADC12IE10	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG10 bit. 0 = Interrupt disabled 1 = Interrupt enabled
9	ADC12IE9	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG9 bit. 0 = Interrupt disabled 1 = Interrupt enabled
8	ADC12IE8	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG8 bit. 0 = Interrupt disabled 1 = Interrupt enabled
7	ADC12IE7	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG7 bit. 0 = Interrupt disabled 1 = Interrupt enabled
6	ADC12IE6	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG6 bit. 0 = Interrupt disabled 1 = Interrupt enabled
5	ADC12IE5	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG5 bit. 0 = Interrupt disabled 1 = Interrupt enabled
4	ADC12IE4	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG4 bit. 0 = Interrupt disabled 1 = Interrupt enabled

Table 18-12. ADC12IER0 Register Description (continued)

Bit	Field	Type	Reset	Description
3	ADC12IE3	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG3 bit. 0 = Interrupt disabled 1 = Interrupt enabled
2	ADC12IE2	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG2 bit. 0 = Interrupt disabled 1 = Interrupt enabled
1	ADC12IE1	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG1 bit. 0 = Interrupt disabled 1 = Interrupt enabled
0	ADC12IE0	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG0 bit. 0 = Interrupt disabled 1 = Interrupt enabled

18.3.10 ADC12IER1 Register (offset = 14h) [reset = 0000h]

ADC12_B Interrupt Enable 1 Register

Figure 18-21. ADC12IER1 Register

15	14	13	12	11	10	9	8
ADC12IE31	ADC12IE30	ADC12IE29	ADC12IE28	ADC12IE27	ADC12IE26	ADC12IE25	ADC12IE24
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12IE23	ADC12IE22	ADC12IE21	ADC12IE20	ADC12IE19	ADC12IE18	ADC12IE17	ADC12IE16
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 18-13. ADC12IER1 Register Description

Bit	Field	Type	Reset	Description
15	ADC12IE31	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG31 bit. 0 = Interrupt disabled 1 = Interrupt enabled
14	ADC12IE30	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG30 bit. 0 = Interrupt disabled 1 = Interrupt enabled
13	ADC12IE29	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG29 bit. 0 = Interrupt disabled 1 = Interrupt enabled
12	ADC12IE28	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG28 bit. 0 = Interrupt disabled 1 = Interrupt enabled
11	ADC12IE27	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG27 bit. 0 = Interrupt disabled 1 = Interrupt enabled
10	ADC12IE26	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG26 bit. 0 = Interrupt disabled 1 = Interrupt enabled
9	ADC12IE25	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG25 bit. 0 = Interrupt disabled 1 = Interrupt enabled
8	ADC12IE24	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG24 bit. 0 = Interrupt disabled 1 = Interrupt enabled
7	ADC12IE23	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG23 bit. 0 = Interrupt disabled 1 = Interrupt enabled
6	ADC12IE22	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG22 bit. 0 = Interrupt disabled 1 = Interrupt enabled
5	ADC12IE21	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG21 bit. 0 = Interrupt disabled 1 = Interrupt enabled
4	ADC12IE20	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG20 bit. 0 = Interrupt disabled 1 = Interrupt enabled

Table 18-13. ADC12IER1 Register Description (continued)

Bit	Field	Type	Reset	Description
3	ADC12IE19	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG19 bit. 0 = Interrupt disabled 1 = Interrupt enabled
2	ADC12IE18	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG18 bit. 0 = Interrupt disabled 1 = Interrupt enabled
1	ADC12IE17	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG17 bit. 0 = Interrupt disabled 1 = Interrupt enabled
0	ADC12IE16	RW	0h	Interrupt enable. Enables or disables the interrupt request for ADC12IFG16 bit. 0 = Interrupt disabled 1 = Interrupt enabled

18.3.11 ADC12IER2 Register (offset = 16h) [reset = 0000h]

ADC12_B Interrupt Enable 2 Register

Figure 18-22. ADC12IER2 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	ADC12RDYIE	ADC12TOVIE	ADC12OVIE	ADC12HIIE	ADC12LOIE	ADC12INIE	Reserved
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0

Table 18-14. ADC12IER2 Register Description

Bit	Field	Type	Reset	Description
15-7	Reserved	R	0h	Reserved. Always reads as 0.
6	ADC12RDYIE	RW	0h	ADC12_B local reference buffer ready interrupt enable. The GIE bit must also be set to enable the interrupt. 0 = Interrupt disabled 1 = Interrupt enabled
5	ADC12TOVIE	RW	0h	ADC12_B conversion-time-overflow interrupt enable. The GIE bit must also be set to enable the interrupt. 0 = Interrupt disabled 1 = Interrupt enabled
4	ADC12OVIE	RW	0h	ADC12MEMx overflow-interrupt enable. The GIE bit must also be set to enable the interrupt. 0 = Interrupt disabled 1 = Interrupt enabled
3	ADC12HIIE	RW	0h	Interrupt enable for the exceeding the upper limit interrupt of the window comparator for ADC12MEMx result register. The GIE bit must also be set to enable the interrupt. 0 = Interrupt disabled 1 = Interrupt enabled
2	ADC12LOIE	RW	0h	Interrupt enable for the falling short of the lower limit interrupt of the window comparator for the ADC12MEMx result register. The GIE bit must also be set to enable the interrupt. 0 = Interrupt disabled 1 = Interrupt enabled
1	ADC12INIE	RW	0h	Interrupt enable for the ADC12MEMx result register being greater than the ADC12LO threshold and below the ADC12HI threshold. The GIE bit must also be set to enable the interrupt. 0 = Interrupt disabled 1 = Interrupt enabled
0	Reserved	R	0h	Reserved. Always reads as 0.

18.3.12 ADC12IFGR0 Register (offset = 0Ch) [reset = 0000h]

ADC12_B Interrupt Flag 0 Register

Figure 18-23. ADC12IFGR0 Register

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)

Table 18-15. ADC12IFGR0 Register Description

Bit	Field	Type	Reset	Description
15	ADC12IFG15	RW	0h	ADC12MEM15 interrupt flag. This bit is set when ADC12MEM15 is loaded with a conversion result. The ADC12IFG15 bit is reset if ADC12MEM15 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
14	ADC12IFG14	RW	0h	ADC12MEM14 interrupt flag. This bit is set when ADC12MEM14 is loaded with a conversion result. The ADC12IFG14 bit is reset if ADC12MEM14 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
13	ADC12IFG13	RW	0h	ADC12MEM13 interrupt flag. This bit is set when ADC12MEM13 is loaded with a conversion result. The ADC12IFG13 bit is reset if ADC12MEM13 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
12	ADC12IFG12	RW	0h	ADC12MEM12 interrupt flag. This bit is set when ADC12MEM12 is loaded with a conversion result. The ADC12IFG12 bit is reset if ADC12MEM12 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
11	ADC12IFG11	RW	0h	ADC12MEM11 interrupt flag. This bit is set when ADC12MEM11 is loaded with a conversion result. The ADC12IFG11 bit is reset if ADC12MEM11 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
10	ADC12IFG10	RW	0h	ADC12MEM10 interrupt flag. This bit is set when ADC12MEM10 is loaded with a conversion result. The ADC12IFG10 bit is reset if ADC12MEM10 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
9	ADC12IFG9	RW	0h	ADC12MEM9 interrupt flag. This bit is set when ADC12MEM9 is loaded with a conversion result. The ADC12IFG9 bit is reset if ADC12MEM9 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
8	ADC12IFG8	RW	0h	ADC12MEM8 interrupt flag. This bit is set when ADC12MEM8 is loaded with a conversion result. The ADC12IFG8 bit is reset if ADC12MEM8 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending

Table 18-15. ADC12IFGR0 Register Description (continued)

Bit	Field	Type	Reset	Description
7	ADC12IFG7	RW	0h	ADC12MEM7 interrupt flag. This bit is set when ADC12MEM7 is loaded with a conversion result. The ADC12IFG7 bit is reset if ADC12MEM7 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
6	ADC12IFG6	RW	0h	ADC12MEM6 interrupt flag. This bit is set when ADC12MEM6 is loaded with a conversion result. The ADC12IFG6 bit is reset if ADC12MEM6 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
5	ADC12IFG5	RW	0h	ADC12MEM5 interrupt flag. This bit is set when ADC12MEM5 is loaded with a conversion result. The ADC12IFG5 bit is reset if ADC12MEM5 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
4	ADC12IFG4	RW	0h	ADC12MEM4 interrupt flag. This bit is set when ADC12MEM4 is loaded with a conversion result. The ADC12IFG4 bit is reset if ADC12MEM4 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
3	ADC12IFG3	RW	0h	ADC12MEM3 interrupt flag. This bit is set when ADC12MEM3 is loaded with a conversion result. The ADC12IFG3 bit is reset if ADC12MEM3 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
2	ADC12IFG2	RW	0h	ADC12MEM2 interrupt flag. This bit is set when ADC12MEM2 is loaded with a conversion result. The ADC12IFG2 bit is reset if ADC12MEM2 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
1	ADC12IFG1	RW	0h	ADC12MEM1 interrupt flag. This bit is set when ADC12MEM1 is loaded with a conversion result. The ADC12IFG1 bit is reset if ADC12MEM1 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
0	ADC12IFG0	RW	0h	ADC12MEM0 interrupt flag. This bit is set when ADC12MEM0 is loaded with a conversion result. The ADC12IFG0 bit is reset if ADC12MEM0 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending

18.3.13 ADC12IFGR1 Register (offset = 0Eh) [reset = 0000h]

ADC12_B Interrupt Flag 1 Register

Figure 18-24. ADC12IFGR1 Register

15	14	13	12	11	10	9	8
ADC12IFG31	ADC12IFG30	ADC12IFG29	ADC12IFG28	ADC12IFG27	ADC12IFG26	ADC12IFG25	ADC12IFG24
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12IFG23	ADC12IFG22	ADC12IFG21	ADC12IFG20	ADC12IFG19	ADC12IFG18	ADC12IFG17	ADC12IFG16
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 18-16. ADC12IFGR1 Register Description

Bit	Field	Type	Reset	Description
15	ADC12IFG31	RW	0h	ADC12MEM31 interrupt flag. This bit is set when ADC12MEM31 is loaded with a conversion result. The ADC12IFG31 bit is reset if ADC12MEM31 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
14	ADC12IFG30	RW	0h	ADC12MEM30 interrupt flag. This bit is set when ADC12MEM30 is loaded with a conversion result. The ADC12IFG30 bit is reset if ADC12MEM30 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
13	ADC12IFG29	RW	0h	ADC12MEM29 interrupt flag. This bit is set when ADC12MEM29 is loaded with a conversion result. The ADC12IFG29 bit is reset if ADC12MEM29 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
12	ADC12IFG28	RW	0h	ADC12MEM28 interrupt flag. This bit is set when ADC12MEM28 is loaded with a conversion result. The ADC12IFG28 bit is reset if ADC12MEM28 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
11	ADC12IFG27	RW	0h	ADC12MEM27 interrupt flag. This bit is set when ADC12MEM27 is loaded with a conversion result. The ADC12IFG27 bit is reset if ADC12MEM27 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
10	ADC12IFG26	RW	0h	ADC12MEM26 interrupt flag. This bit is set when ADC12MEM26 is loaded with a conversion result. The ADC12IFG26 bit is reset if ADC12MEM26 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
9	ADC12IFG25	RW	0h	ADC12MEM25 interrupt flag. This bit is set when ADC12MEM25 is loaded with a conversion result. The ADC12IFG25 bit is reset if ADC12MEM25 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
8	ADC12IFG24	RW	0h	ADC12MEM24 interrupt flag. This bit is set when ADC12MEM24 is loaded with a conversion result. The ADC12IFG24 bit is reset if ADC12MEM24 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending

Table 18-16. ADC12IFGR1 Register Description (continued)

Bit	Field	Type	Reset	Description
7	ADC12IFG23	RW	0h	ADC12MEM23 interrupt flag. This bit is set when ADC12MEM23 is loaded with a conversion result. The ADC12IFG23 bit is reset if ADC12MEM23 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
6	ADC12IFG22	RW	0h	ADC12MEM22 interrupt flag. This bit is set when ADC12MEM22 is loaded with a conversion result. The ADC12IFG22 bit is reset if ADC12MEM22 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
5	ADC12IFG21	RW	0h	ADC12MEM21 interrupt flag. This bit is set when ADC12MEM21 is loaded with a conversion result. The ADC12IFG21 bit is reset if ADC12MEM21 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
4	ADC12IFG20	RW	0h	ADC12MEM20 interrupt flag. This bit is set when ADC12MEM20 is loaded with a conversion result. The ADC12IFG20 bit is reset if ADC12MEM20 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
3	ADC12IFG19	RW	0h	ADC12MEM19 interrupt flag. This bit is set when ADC12MEM19 is loaded with a conversion result. The ADC12IFG19 bit is reset if ADC12MEM19 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
2	ADC12IFG18	RW	0h	ADC12MEM18 interrupt flag. This bit is set when ADC12MEM18 is loaded with a conversion result. The ADC12IFG18 bit is reset if ADC12MEM18 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
1	ADC12IFG17	RW	0h	ADC12MEM17 interrupt flag. This bit is set when ADC12MEM17 is loaded with a conversion result. The ADC12IFG17 bit is reset if ADC12MEM17 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending
0	ADC12IFG16	RW	0h	ADC12MEM16 interrupt flag. This bit is set when ADC12MEM16 is loaded with a conversion result. The ADC12IFG16 bit is reset if ADC12MEM16 is accessed, or it can be reset with software. 0 = No interrupt pending 1 = Interrupt pending

18.3.14 ADC12IFGR2 Register (offset = 10h) [reset = 0000h]

ADC12_B Interrupt Flag 2 Register

Figure 18-25. ADC12IFGR2 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	ADC12RDYIFG	ADC12TOVIFG	ADC12OVIFG	ADC12HIIFG	ADC12LOIFG	ADC12INIFG	Reserved
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0

Table 18-17. ADC12IFGR2 Register Description

Bit	Field	Type	Reset	Description
15-7	Reserved	R	0h	Reserved. Always reads as 0.
6	ADC12RDYIFG	RW	0h	ADC12_B local reference buffer ready interrupt flag 0 = No interrupt pending 1 = Interrupt pending
5	ADC12TOVIFG	RW	0h	ADC12_B conversion-time-overflow interrupt flag. 0 = No interrupt pending 1 = Interrupt pending
4	ADC12OVIFG	RW	0h	ADC12MEMx overflow-interrupt flag. 0 = No interrupt pending 1 = Interrupt pending
3	ADC12HIIFG	RW	0h	Interrupt flag for exceeding the upper limit interrupt of the window comparator for ADC12MEMx result register. 0 = No interrupt pending 1 = Interrupt pending
2	ADC12LOIFG	RW	0h	Interrupt flag for falling short of the lower limit interrupt of the window comparator for the ADC12MEMx result register. 0 = No interrupt pending 1 = Interrupt pending
1	ADC12INIFG	RW	0h	Interrupt flag for the ADC12MEMx result register being greater than the ADC12LO threshold and below the ADC12HI threshold interrupt. 0 = No interrupt pending 1 = Interrupt pending
0	Reserved	R	0h	Reserved. Always reads as 0.

18.3.15 ADC12IV Register (offset = 18h) [reset = 0000h]

ADC12_B Interrupt Vector

Figure 18-26. ADC12IV Register

15	14	13	12	11	10	9	8
ADC12IVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
ADC12IVx							
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0

Table 18-18. ADC12IV Register Description

Bit	Field	Type	Reset	Description
15-0	ADC12IVx	RW	0h	ADC12_B interrupt vector value. Writing to this register clears all pending interrupt flags. 000h = Interrupt Source: No interrupt pending, Interrupt Flag: None 002h = Interrupt Source: ADC12MEMx overflow, Interrupt Flag: ADC12OVIFG, Interrupt Priority: Highest 004h = Interrupt Source: Conversion time overflow, Interrupt Flag: ADC12TOVIFG 006h = Interrupt Source: ADC12 window high interrupt flag, Interrupt Flag: ADC12HIIFG 008h = Interrupt Source: ADC12 window low interrupt flag, Interrupt Flag: ADC12LOIFG 00Ah = Interrupt Source: ADC12 in-window interrupt flag, Interrupt Flag: ADC12INIFG 00Ch = Interrupt Source: ADC12MEM0 interrupt flag, Interrupt Flag: ADC12IFG0 00Eh = Interrupt Source: ADC12MEM1 interrupt flag, Interrupt Flag: ADC12IFG1 010h = Interrupt Source: ADC12MEM2 interrupt flag, Interrupt Flag: ADC12IFG2 012h = Interrupt Source: ADC12MEM3 interrupt flag, Interrupt Flag: ADC12IFG3 014h = Interrupt Source: ADC12MEM4 interrupt flag, Interrupt Flag: ADC12IFG4 016h = Interrupt Source: ADC12MEM5 interrupt flag, Interrupt Flag: ADC12IFG5 018h = Interrupt Source: ADC12MEM6 interrupt flag, Interrupt Flag: ADC12IFG6 01Ah = Interrupt Source: ADC12MEM7 interrupt flag, Interrupt Flag: ADC12IFG7 01Ch = Interrupt Source: ADC12MEM8 interrupt flag, Interrupt Flag: ADC12IFG8 01Eh = Interrupt Source: ADC12MEM9 interrupt flag, Interrupt Flag: ADC12IFG9 020h = Interrupt Source: ADC12MEM10 interrupt flag, Interrupt Flag: ADC12IFG10 022h = Interrupt Source: ADC12MEM11 interrupt flag, Interrupt Flag: ADC12IFG11 024h = Interrupt Source: ADC12MEM12 interrupt flag, Interrupt Flag: ADC12IFG12 026h = Interrupt Source: ADC12MEM13 interrupt flag, Interrupt Flag: ADC12IFG13 028h = Interrupt Source: ADC12MEM14 interrupt flag, Interrupt Flag: ADC12IFG14 02Ah = Interrupt Source: ADC12MEM15 interrupt flag, Interrupt Flag: ADC12IFG15 02Ch = Interrupt Source: ADC12MEM16 interrupt flag, Interrupt Flag: ADC12IFG16 02Eh = Interrupt Source: ADC12MEM17 interrupt flag, Interrupt Flag: ADC12IFG17 030h = Interrupt Source: ADC12MEM18 interrupt flag, Interrupt Flag: ADC12IFG18

Table 18-18. ADC12IV Register Description (continued)

Bit	Field	Type	Reset	Description
				032h = Interrupt Source: ADC12MEM19 interrupt flag, Interrupt Flag: ADC12IFG19 034h = Interrupt Source: ADC12MEM20 interrupt flag, Interrupt Flag: ADC12IFG20 036h = Interrupt Source: ADC12MEM21 interrupt flag, Interrupt Flag: ADC12IFG21 038h = Interrupt Source: ADC12MEM22 interrupt flag, Interrupt Flag: ADC12IFG22 03Ah = Interrupt Source: ADC12MEM23 interrupt flag, Interrupt Flag: ADC12IFG23 03Ch = Interrupt Source: ADC12MEM24 interrupt flag, Interrupt Flag: ADC12IFG24 03Eh = Interrupt Source: ADC12MEM25 interrupt flag, Interrupt Flag: ADC12IFG25 040h = Interrupt Source: ADC12MEM26 interrupt flag, Interrupt Flag: ADC12IFG26 042h = Interrupt Source: ADC12MEM27 interrupt flag, Interrupt Flag: ADC12IFG27 044h = Interrupt Source: ADC12MEM28 interrupt flag, Interrupt Flag: ADC12IFG28 046h = Interrupt Source: ADC12MEM29 interrupt flag, Interrupt Flag: ADC12IFG29 048h = Interrupt Source: ADC12MEM30 interrupt flag, Interrupt Flag: ADC12IFG30 04Ah = Interrupt Source: ADC12MEM31 interrupt flag, Interrupt Flag: ADC12IFG31 04Ch = Interrupt Source: ADC12RDYIFG interrupt flag, Interrupt Flag: ADC12RDYIFG

Comparator E (COMP_E) Module

Comparator_E is an analog voltage comparator. This chapter describes the Comparator_E. Comparator_E covers general comparator functionality for up to 16 channels.

Topic **Page**

19.1 COMP_E Introduction **528**
19.2 COMP_E Operation **529**
19.3 COMP_E Registers **535**

19.1 COMP_E Introduction

The COMP_E module supports precision slope analog-to-digital conversions, supply voltage supervision, and monitoring of external analog signals.

Features of COMP_E 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 and voltage hysteresis generator
- Reference voltage input from shared reference
- Ultralow-power comparator mode
- Interrupt driven measurement system for low-power operation support

The Comparator_E block diagram is shown in Figure 19-1.

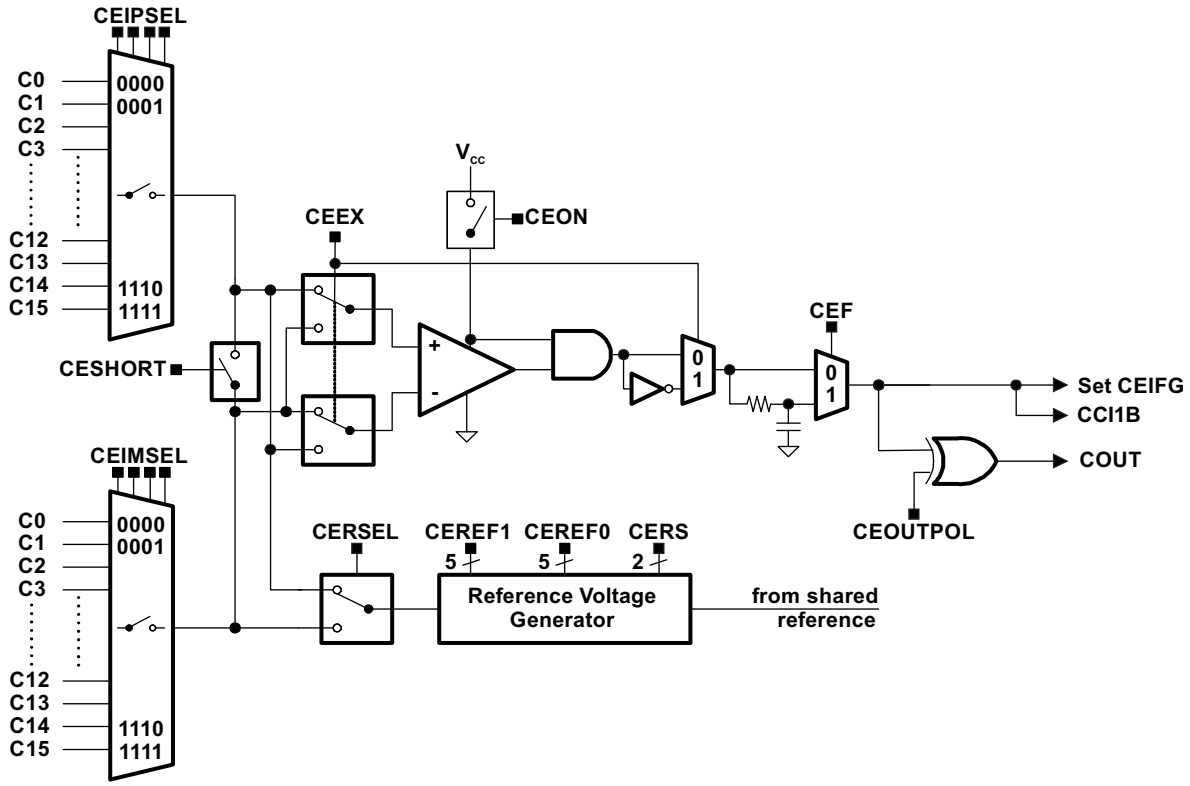


Figure 19-1. Comparator_E Block Diagram

19.2 COMP_E Operation

The COMP_E module is configured by user software. The setup and operation of COMP_E is discussed in the following sections.

19.2.1 Comparator

The comparator compares the analog voltages at the positive (+) and negative (–) input terminals. If the + terminal is more positive than the – terminal, the comparator output CEOUT is high. The comparator can be switched on or off using control bit CEON. The comparator should be switched off when not in use to reduce current consumption. When the comparator is switched off, CEOUT is always low. The bias current of the comparator is programmable.

19.2.2 Analog Input Switches

The analog input switches connect or disconnect the two comparator input terminals to associated port pins using the CEIPSELx and CEIMSELx bits. The comparator terminal inputs can be controlled individually. The CEIPSELx and CEIMSELx bits allow:

- Application of an external signal to the V+ and V– terminals of the comparator
- Application of an external current source (for example, a resistor) to the V+ or V– terminal of the comparator
- 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 CEEX bit controls the input multiplexer, permuting the input signals of the comparator's V+ and V– terminals. Additionally, when the comparator terminals are permuted, the output signal from the comparator is also inverted. This allows the user to determine or compensate for the comparator input offset voltage.

19.2.3 Port Logic

The Px.y pins that are associated with a comparator channel are enabled by the CEIPSELx or CEIMSELx bits to disable the digital components while the terminals are used as comparator inputs. Only one of the comparator input pins is selected as input to the comparator by the input multiplexer at a time.

19.2.4 Input Short Switch

The CESHORT bit shorts the Comparator_E inputs. This can be used to build a simple sample-and-hold for the comparator as shown in Figure 19-2.

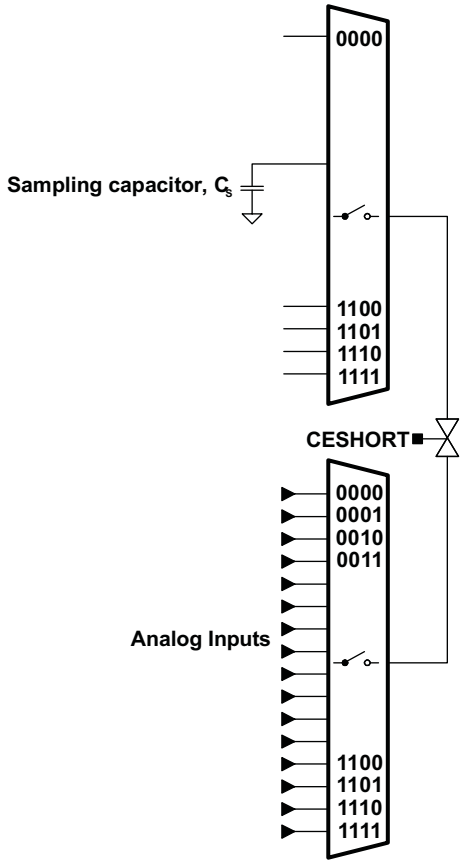


Figure 19-2. Comparator_E Sample-And-Hold

The required sampling time is proportional to the size of the sampling capacitor R_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 TBD 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 signal's 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.

19.2.5 Output Filter

The output of the comparator can be used with or without internal filtering. When control bit CEF is set, the output is filtered with an on-chip RC filter. The delay of the filter can be adjusted in four steps.

All comparator outputs oscillate 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 19-3. The comparator output oscillation reduces the accuracy and resolution of the comparison result. Selecting the output filter can reduce errors associated with comparator oscillation.

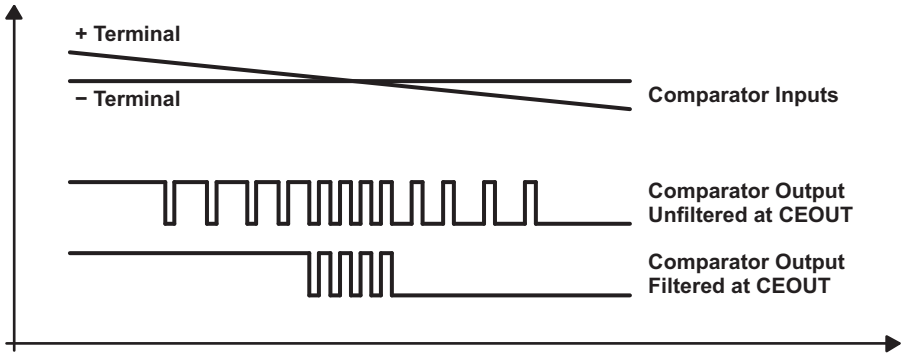


Figure 19-3. RC-Filter Response at the Output of the Comparator

19.2.6 Reference Voltage Generator

The Comparator_E reference block diagram is shown in Figure 19-4.

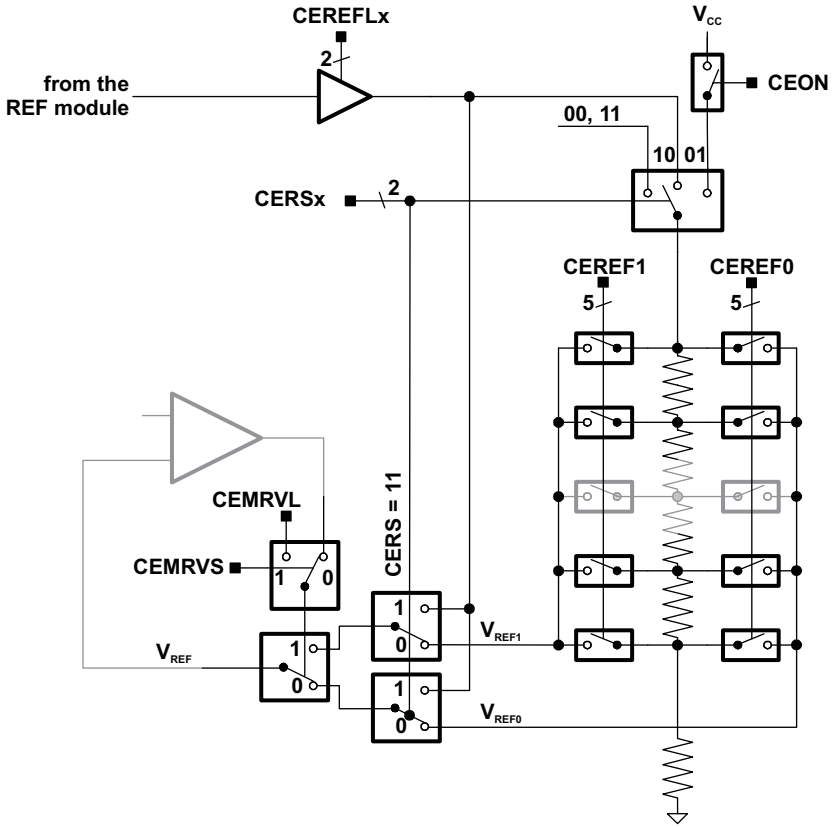


Figure 19-4. Reference Generator Block Diagram

The interrupt flags of the comparator and the comparator output are unchanged while the reference voltage from the shared reference is settling. If CEREFLx is changed from a non-zero value to another non-zero value, the interrupt flags may show unpredictable behavior. It is recommended to set CEREFLx = 00 prior to changing the CEREFLx settings.

The voltage reference generator is used to generate VREF, which can be applied to either comparator input terminal. The CEREF1x (VREF1) and CEREF0x (VREF0) bits control the output of the voltage generator. The CERSEL 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 CEOUT is 1, and Vref0 is used while CEOUT is 0. This allows the generation of a hysteresis without using external components.

19.2.7 Port Disable Register (CEPD)

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 CEPDx bits, when set, disable the corresponding Px.y input buffer as shown in Figure 19-5. When current consumption is critical, any Px.y pin connected to analog signals should be disabled with their associated CEPDx bits.

Selecting an input pin to the comparator multiplexer with the CEIPSEL or CEIMSEL bits automatically disables the input buffer for that pin, regardless of the state of the associated CEPDx bit.

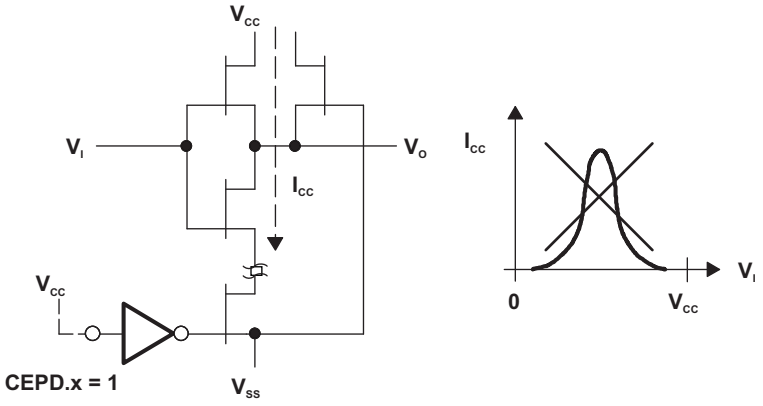


Figure 19-5. Transfer Characteristic and Power Dissipation in a CMOS Inverter and Buffer

19.2.8 Comparator_E Interrupts

One interrupt flag and one interrupt vector are associated with the Comparator_E.

The interrupt flag CEIFG is set on either the rising or falling edge of the comparator output, selected by the CEIES bit. If both the CEIE and the GIE bits are set, then the CEIFG interrupt flag generates an interrupt request.

19.2.9 Comparator_E Used to Measure Resistive Elements

The Comparator_E 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 19-6. A reference resistor Rref is compared to Rmeas.

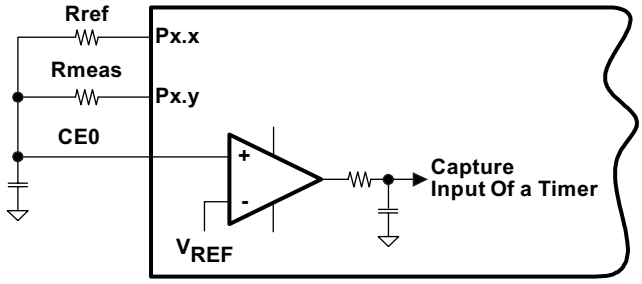


Figure 19-6. Temperature Measurement System

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 CEPDx 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 × V_{CC}.
- The output filter should be used to minimize switching noise.
- CEOUT is used to gate a timer capturing capacitor discharge time.

More than one resistive element can be measured. Additional elements are connected to CE0 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 19-7.

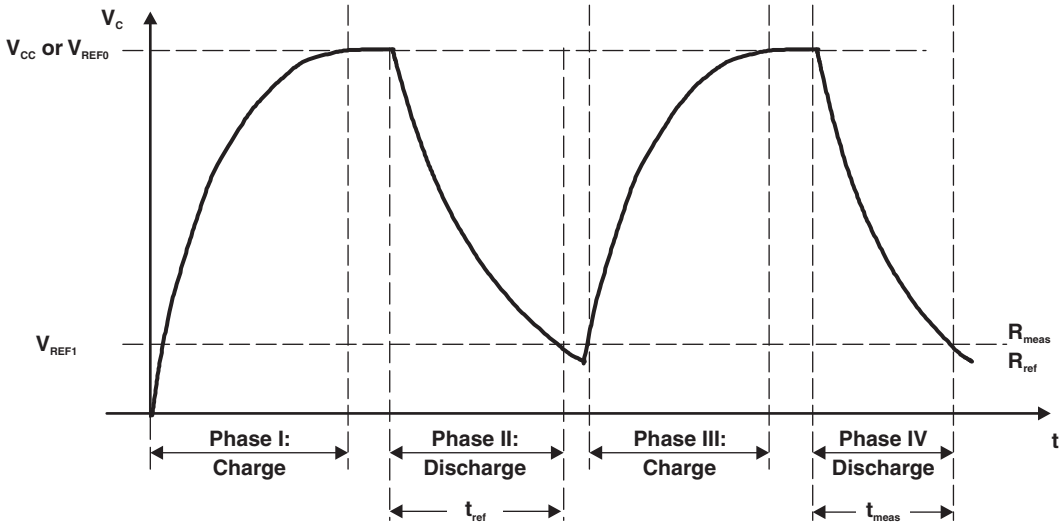


Figure 19-7. Timing for Temperature Measurement Systems

The V_{CC} voltage and the capacitor value should remain constant during the conversion but are not critical, because they cancel in the ratio:

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times \ln \frac{V_{ref1}}{V_{CC}}}{-R_{ref} \times C \times \ln \frac{V_{ref1}}{V_{CC}}}$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}$$

19.3 COMP_E Registers

The Comparator_E registers are listed in Table 19-1. The base address of the Comparator_E module can be found in each device-specific data sheet.

Table 19-1. COMP_E Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	CECTL0	Comparator_E control register 0	Read/write	Word	0000h	Section 19.3.1
02h	CECTL1	Comparator_E control register 1	Read/write	Word	0000h	Section 19.3.2
04h	CECTL2	Comparator_E control register 2	Read/write	Word	0000h	Section 19.3.3
06h	CECTL3	Comparator_E control register 3	Read/write	Word	0000h	Section 19.3.4
0Ch	CEINT	Comparator_E interrupt register	Read/write	Word	0000h	Section 19.3.5
0Eh	CEIV	Comparator_E interrupt vector word	Read	Word	0000h	Section 19.3.6

19.3.1 CECTL0 Register (offset = 00h) [reset = 0000h]

Comparator_E Control Register 0

Figure 19-8. CECTL0 Register

15	14	13	12	11	10	9	8
CEIMEN	Reserved			CEIMSEL			
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
CEIPEN	Reserved			CEIPSEL			
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

Table 19-2. CECTL0 Register Description

Bit	Field	Type	Reset	Description
15	CEIMEN	RW	0h	Channel input enable for the – terminal of the comparator. 0b = Selected analog input channel for V– terminal is disabled. 1b = Selected analog input channel for V– terminal is enabled. The internal reference voltage is disabled for this channel.
14-12	Reserved	R	0h	Reserved. Reads as 0.
11-8	CEIMSEL	RW	0h	Channel input selected for the – terminal of the comparator if CEIMEN is set to 1.
7	CEIPEN	RW	0h	Channel input enable for the V+ terminal of the comparator. 0b = Selected analog input channel for V+ terminal is disabled. 1b = Selected analog input channel for V+ terminal is enabled. The internal reference voltage is disabled for this channel.
6-4	Reserved	R	0h	Reserved. Reads as 0.
3-0	CEIPSEL	RW	0h	Channel input selected for the V+ terminal of the comparator if CEIPEN is set to 1.

19.3.2 CECTL1 Register (offset = 02h) [reset = 0000h]

Comparator_E Control Register 1

Figure 19-9. CECTL1 Register

15	14	13	12	11	10	9	8
Reserved			CEMRVS	CEMRVL	CEON	CEPWRMD	
r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
CEFDLY		CEEX	CESHORT	CEIES	CEF	CEOUTPOL	CEOUT
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

Table 19-3. CECTL1 Register Description

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Reads as 0.
12	CEMRVS	RW	0h	This bit defines if the comparator output selects between VREF0 or VREF1 if CERS = 00, 01, or 10. 0b = Comparator output state selects between VREF0 or VREF1. 1b = CEMRVL selects between VREF0 or VREF1.
11	CEMRVL	RW	0h	This bit is valid of CEMRVS is set to 1. 0b = VREF0 is selected if CERS = 00, 01, or 10 1b = VREF1 is selected if CERS = 00, 01, or 10
10	CEON	RW	0h	On. This bit turns the comparator on. When the comparator is turned off the Comparator_E consumes no power. 0b = Off 1b = On
9-8	CEPWRMD	RW	0h	Power mode 00b = High-speed mode 01b = Normal mode 10b = Ultra-low power mode 11b = Reserved
7-6	CEFDLY	RW	0h	Filter delay. The filter delay can be selected in four steps. See the device-specific data sheet for details. 00b = Typical filter delay of TBD (450) ns 01b = Typical filter delay of TBD (900) ns 10b = Typical filter delay of TBD (1800) ns 11b = Typical filter delay of TBD (3600) ns
5	CEEX	RW	0h	Exchange. This bit permutes the comparator 0 inputs and inverts the comparator 0 output. 0b = Exchange off 1b = Exchange on
4	CESHORT	RW	0h	Input short. This bit shorts the + and – input terminals. 0b = Inputs not shorted 1b = Inputs shorted
3	CEIES	RW	0h	Interrupt edge select for CEIIFG and CEIFG. Changing CEIES might set CEIFG. 0b = Rising edge for CEIFG, falling edge for CEIIFG 1b = Falling edge for CEIFG, rising edge for CEIIFG
2	CEF	RW	0h	Output filter. Available if CEPWRMD = 00 or 01. 0b = Comparator_E output is not filtered 1b = Comparator_E output is filtered
1	CEOUTPOL	RW	0h	Output polarity. This bit defines the CEOUT polarity. 0b = Noninverted 1b = Inverted

Table 19-3. CECTL1 Register Description (continued)

Bit	Field	Type	Reset	Description
0	CEOUT	R	0h	Output value. This bit reflects the value of the Comparator_E output. Writing this bit has no effect on the comparator output.

19.3.3 CECTL2 Register (offset = 04h) [reset = 0000h]

Comparator_E Control Register 2

Figure 19-10. CECTL2 Register

15	14	13	12	11	10	9	8
CEREFACC	CEREFL		CEREF1				
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
CERS		CERSEL	CEREF0				
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 19-4. CECTL2 Register Description

Bit	Field	Type	Reset	Description
15	CEREFACC	RW	0h	Reference accuracy. A reference voltage is requested only if CEREFL > 0. 0b = Static mode 1b = Clocked (low power, low accuracy) mode
14-13	CEREFL	RW	0h	Reference voltage level 00b = Reference amplifier is disabled. No reference voltage is requested. 01b = 1.2 V is selected as shared reference voltage input 10b = 2.0 V is selected as shared reference voltage input 11b = 2.5 V is selected as shared reference voltage input
12-8	CEREF1	RW	0h	Reference resistor tap 1. This register defines the tap of the resistor string while CEOUT = 1.
7-6	CERS	RW	0h	Reference source. This bit define if the reference voltage is derived from VCC or from the precise shared reference. 00b = No current is drawn by the reference circuitry. 01b = VCC applied to the resistor ladder 10b = Shared reference voltage applied to the resistor ladder. 11b = Shared reference voltage supplied to VCCREF. Resistor ladder is off.
5	CERSEL	RW	0h	Reference select. This bit selects to which terminal the VCCREF is applied. When CEEX = 0: 0b = When CEEX = 0: VREF is applied to the V+ terminal; When CEEX = 1: VREF is applied to the V– terminal 1b = When CEEX = 0: VREF is applied to the V– terminal; When CEEX = 1: VREF is applied to the V+ terminal
4-0	CEREF0	RW	0h	Reference resistor tap 0. This register defines the tap of the resistor string while CEOUT = 0.

19.3.4 CECTL3 Register (offset = 06h) [reset = 0000h]

Comparator_E Control Register 3

Figure 19-11. CECTL3 Register

15	14	13	12	11	10	9	8
CEPD15	CEPD14	CEPD13	CEPD12	CEPD11	CEPD10	CEPD9	CEPD8
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
CEPD7	CEPD6	CEPD5	CEPD4	CEPD3	CEPD2	CEPD1	CEPD0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 19-5. CECTL3 Register Description

Bit	Field	Type	Reset	Description
15	CEPD15	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD15 disables the port of the comparator channel 15. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
14	CEPD14	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD14 disables the port of the comparator channel 14. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
13	CEPD13	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD13 disables the port of the comparator channel 13. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
12	CEPD12	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD12 disables the port of the comparator channel 12. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
11	CEPD11	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD11 disables the port of the comparator channel 11. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
10	CEPD10	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD10 disables the port of the comparator channel 10. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
9	CEPD9	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD9 disables the port of the comparator channel 9. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
8	CEPD8	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD8 disables the port of the comparator channel 8. 0b = The input buffer is enabled. 1b = The input buffer is disabled.

Table 19-5. CECTL3 Register Description (continued)

Bit	Field	Type	Reset	Description
7	CEPD7	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD7 disables the port of the comparator channel 7. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
6	CEPD6	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD6 disables the port of the comparator channel 6. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
5	CEPD5	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD5 disables the port of the comparator channel 5. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
4	CEPD4	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD4 disables the port of the comparator channel 4. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
3	CEPD3	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD3 disables the port of the comparator channel 3. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
2	CEPD2	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD2 disables the port of the comparator channel 2. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
1	CEPD1	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD1 disables the port of the comparator channel 1. 0b = The input buffer is enabled. 1b = The input buffer is disabled.
0	CEPD0	RW	0h	Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_E. The bit CEPD0 disables the port of the comparator channel 0. 0b = The input buffer is enabled. 1b = The input buffer is disabled.

19.3.5 CEINT Register (offset = 0Ch) [reset = 0000h]

Comparator_E Interrupt Control Register

Figure 19-12. CEINT Register

15	14	13	12	11	10	9	8
Reserved			CERDYIE	Reserved		CEIIE	CEIE
r-0	r-0	r-0	rw-0	r-0	r-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved			CERDYIFG	Reserved		CEIIFG	CEIFG
r-0	r-0	r-0	rw-0	r-0	r-0	rw-0	rw-0

Table 19-6. CEINT Register Description

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Reads as 0.
12	CERDYIE	RW	0h	Comparator_E ready interrupt enable. 0b = Interrupt is disabled 1b = Interrupt is enabled
11-10	Reserved	R	0h	Reserved. Reads as 0.
9	CEIIE	RW	0h	Comparator_E output interrupt enable inverted polarity 0b = Interrupt is disabled 1b = Interrupt is enabled
8	CEIE	RW	0h	Comparator_E output interrupt enable 0b = Interrupt is disabled 1b = Interrupt is enabled
7-5	Reserved	R	0h	Reserved. Reads as 0.
4	CERDYIFG	RW	0h	Comparator_E ready interrupt flag. This bit is set if the Comparator_E reference sources are settled and the Comparator_E module is operational. This bit has to be cleared by software. 0b = No interrupt pending. 1b = Output interrupt pending.
3-2	Reserved	R	0h	Reserved. Reads as 0.
1	CEIIFG	RW	0h	Comparator_E output inverted interrupt flag. The bit CEIES defines the transition of the output setting this bit. 0b = No interrupt pending. 1b = Output interrupt pending.
0	CEIFG	RW	0h	Comparator_E output interrupt flag. The bit CEIES defines the transition of the output setting this bit. 0b = No interrupt pending. 1b = Output interrupt pending.

19.3.6 CEIV Register (offset = 0Eh) [reset = 0000h]

Comparator_E Interrupt Vector Word Register

Figure 19-13. CEIV Register

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

Table 19-7. CEIV Register Description

Bit	Field	Type	Reset	Description
15-0	CEIV	R	0h	Comparator_E interrupt vector word register. The interrupt vector register reflects only interrupt flags whose interrupt enable bit are set. Reading the CEIV register clears the pending interrupt flag with the highest priority. 00h = No interrupt pending 02h = Interrupt Source: CEOUT interrupt; Interrupt Flag: CEIFG; Interrupt Priority: Highest 04h = Interrupt Source: CEOUT interrupt inverted polarity; Interrupt Flag: CEIIFG 06h = Reserved 08h = Reserved 0Ah = Interrupt Source: Comparator ready interrupt; Interrupt Flag: CERDYIFG; Interrupt Priority: Lowest

Enhanced Universal Serial Communication Interface (eUSCI) – UART Mode

The enhanced universal serial communication interface A (eUSCI_A) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the asynchronous UART mode.

Topic **Page**

20.1 Enhanced Universal Serial Communication Interface A (eUSCI_A) Overview 545
20.2 eUSCI_A Introduction – UART Mode 545
20.3 eUSCI_A Operation – UART Mode 547
20.4 eUSCI_A UART Registers 562

20.1 Enhanced Universal Serial Communication Interface A (eUSCI_A) Overview

The eUSCI_A module supports two serial communication modes:

- UART mode
- SPI mode

20.2 eUSCI_A Introduction – UART Mode

In asynchronous mode, the eUSCI_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-bit 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 (wake up from LPMx.5 is not supported)
- 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, transmit, start bit received, and transmit complete

Figure 20-1 shows the eUSCI_Ax when configured for UART mode.

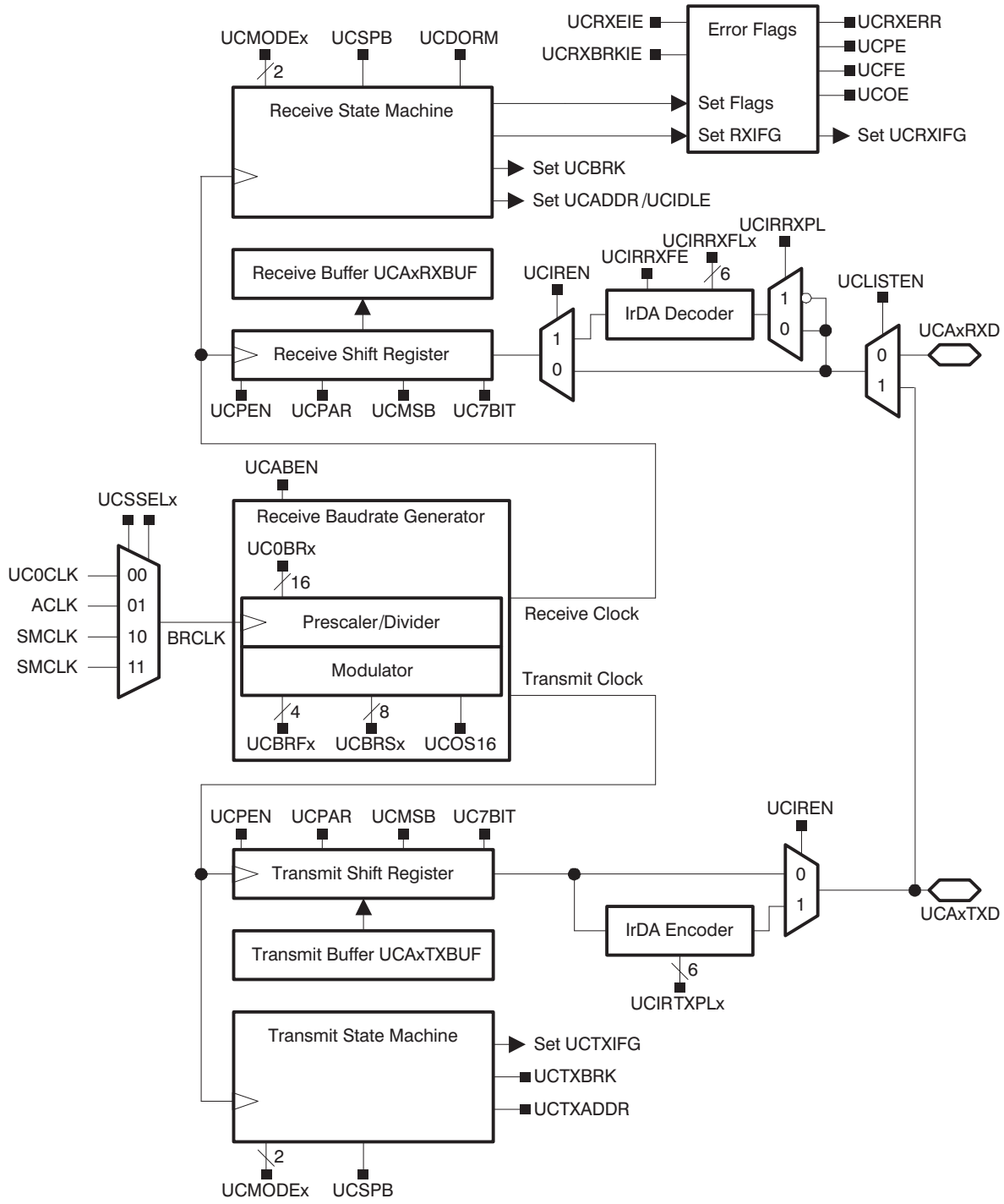


Figure 20-1. eUSCI_Ax Block Diagram – UART Mode (UCSYNC = 0)

20.3 eUSCI_A Operation – UART Mode

In UART mode, the eUSCI_A 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 eUSCI_A. The transmit and receive functions use the same baud-rate frequency.

20.3.1 eUSCI_A Initialization and Reset

The eUSCI_A is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the eUSCI_A in a reset condition. When set, the UCSWRST bit sets the UCTXIFG bit and resets the UCRXIE, UCTXIE, UCRXIFG, UCRXERR, UCBRK, UCPE, UCOE, UCFE, UCSTOE, and UCBTOE bits. Clearing UCSWRST releases the eUSCI_A for operation.

Configuring and reconfiguring the eUSCI_A module should be done when UCSWRST is set to avoid unpredictable behavior.

NOTE: Initializing or reconfiguring the eUSCI_A module

The recommended eUSCI_A initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST,&UCAxCTL1`).
2. Initialize all eUSCI_A registers with UCSWRST = 1 (including UCAxCTL1).
3. Configure ports.
4. Clear UCSWRST via software (`BIC.B #UCSWRST,&UCAxCTL1`).
5. Enable interrupts (optional) via UCRXIE or UCTXIE.

20.3.2 Character Format

The UART character format (see Figure 20-2) 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.

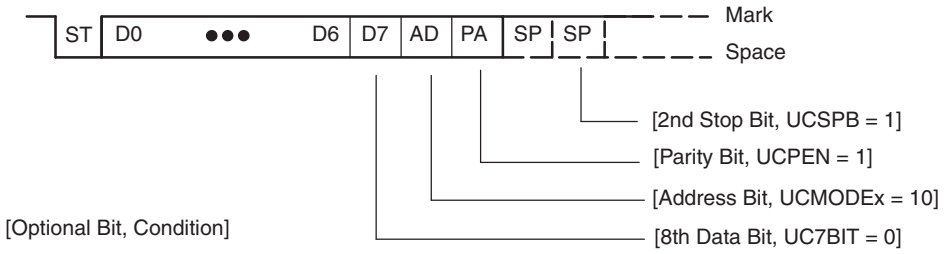


Figure 20-2. Character Format

20.3.3 Asynchronous Communication Format

When two devices communicate asynchronously, no multiprocessor format is required for the protocol. When three or more devices communicate, the eUSCI_A supports the idle-line and address-bit multiprocessor communication formats.

20.3.3.1 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 20-3). 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.

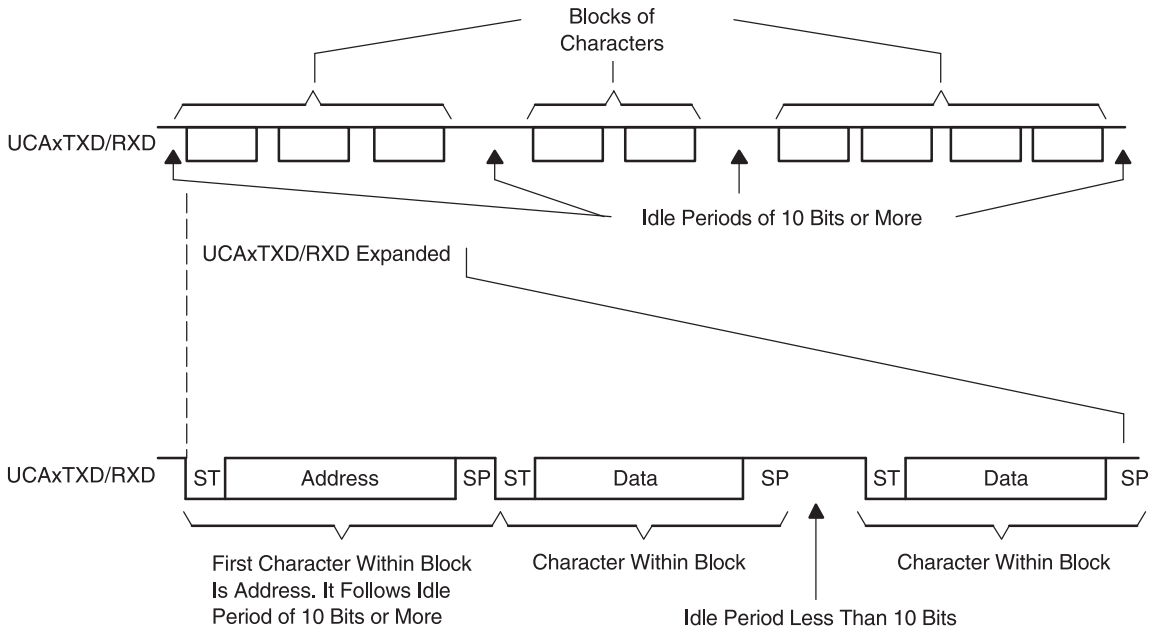


Figure 20-3. Idle-Line Format

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 automatically by the eUSCI_A hardware.

For address transmission in idle-line multiprocessor format, a precise idle period can be generated by the eUSCI_A 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.

20.3.3.1.1 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.

20.3.3.2 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 20-4). The first character in a block of characters carries a set address bit that indicates that the character is an address. The eUSCI_A 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 eUSCI_A 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.

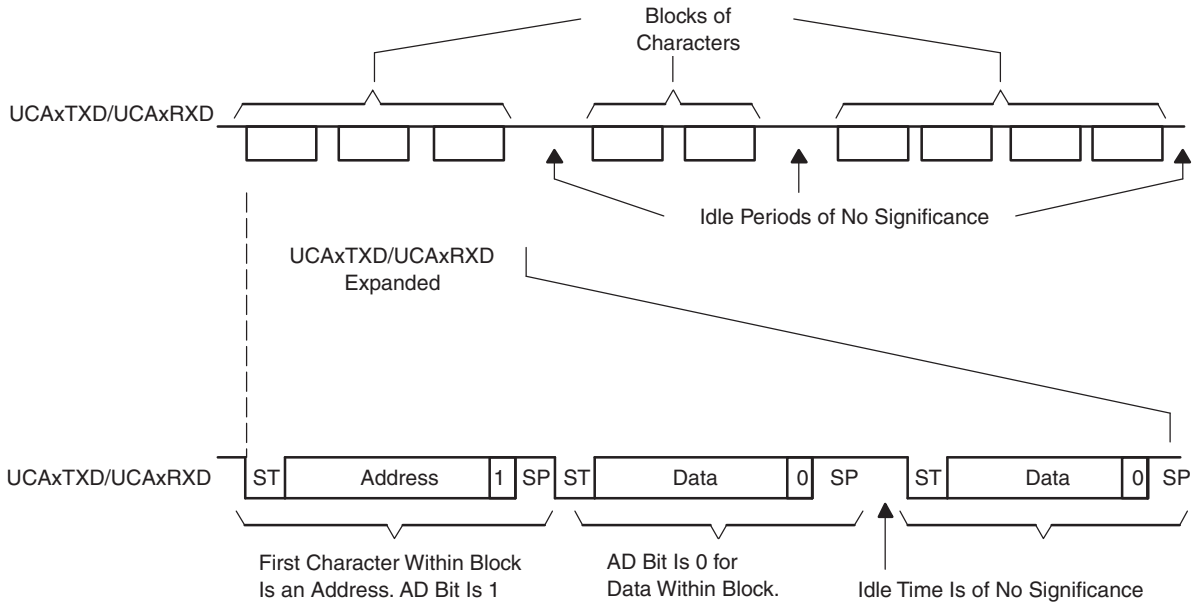


Figure 20-4. Address-Bit Multiprocessor Format

20.3.3.2.1 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.

20.3.4 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 eUSCI_A cannot transmit data while receiving the break/sync field. The synch field follows the break as shown in Figure 20-5.

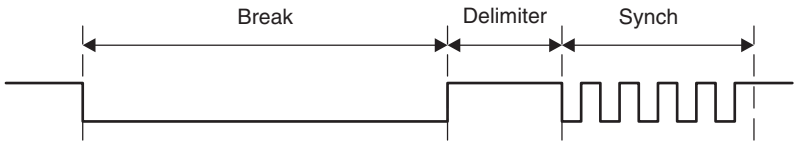


Figure 20-5. Auto Baud-Rate Detection – Break/Synch Sequence

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 20-6). 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 (UCAxBRW and UCAxMCTLW). If the length of the synch field exceeds the measurable time, the synch timeout error flag UCSTOE is set. The result can be read after the receive interrupt flag UCRXIFG is set.

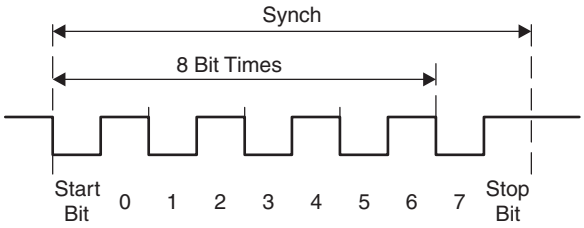


Figure 20-6. Auto Baud-Rate Detection – Synch Field

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 eUSCI_A 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 0FFFFh (2^{16}) counts. This means the minimum baud rate detectable is 244 baud in oversampling mode and 15 baud in low-frequency mode. The highest detectable baudrate is 1 Mbaud.

The automatic baud-rate detection mode can be used in a full-duplex communication system with some restrictions. The eUSCI_A cannot transmit data while receiving the break/sync field and, if a 0h byte with framing error is received, any data transmitted during this time is corrupted. The latter case can be discovered by checking the received data and the UCFE bit.

20.3.4.1 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.

20.3.5 IrDA Encoding and Decoding

When UCIREN is set, the IrDA encoder and decoder are enabled and provide hardware bit shaping for IrDA communication.

20.3.5.1 IrDA Encoding

The encoder sends a pulse for every zero bit in the transmit bit stream coming from the UART (see Figure 20-7). The pulse duration is defined by UCIRTXPLx bits specifying the number of one-half clock periods of the clock selected by UCIRTXCLK.

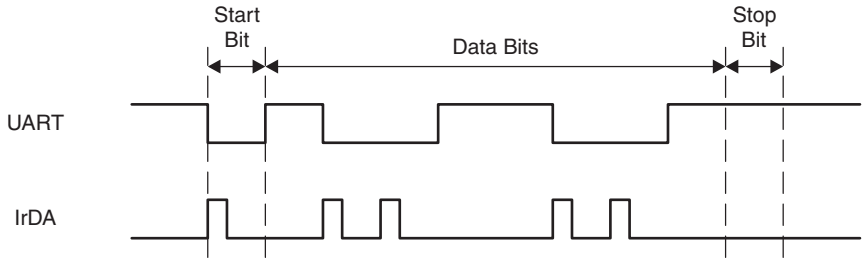


Figure 20-7. UART vs IrDA Data Format

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 be set to a value greater or equal to 5.

20.3.5.2 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.

20.3.6 Automatic Error Detection

Glitch suppression prevents the eUSCI_A from being accidentally started. Any pulse on UCAxRXD shorter than the deglitch time t_t (selected by UCGLITx) 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 eUSCI_A 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 eUSCI_A 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 20-1.

Table 20-1. 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 is received and UCAxRXIFG is set, first read UCAxSTATW 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 UCAxSTATW 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.

20.3.7 eUSCI_A Receive Enable

The eUSCI_A 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, the UART state machine returns to its idle state, and the baud rate generator is turned off.

20.3.7.1 Receive Data Glitch Suppression

Glitch suppression prevents the eUSCI_A from being accidentally started. Any glitch on UCAxRXD shorter than the deglitch time t_t is ignored by the eUSCI_A, and further action is initiated as shown in Figure 20-8 (see the device-specific data sheet for parameters). The deglitch time t_t can be set to four different values using the UCGLITx bits.

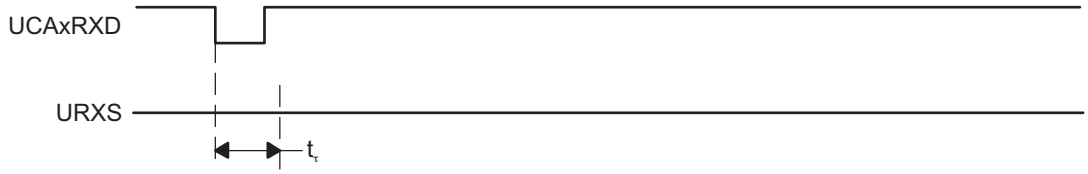


Figure 20-8. Glitch Suppression, eUSCI_A Receive Not Started

When a glitch is longer than t_t, or a valid start bit occurs on UCAxRXD, the eUSCI_A receive operation is started and a majority vote is taken (see Figure 20-9). If the majority vote fails to detect a start bit, the eUSCI_A halts character reception.

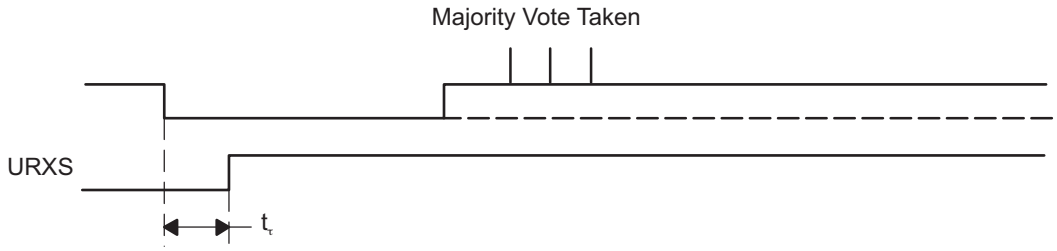


Figure 20-9. Glitch Suppression, eUSCI_A Activated

20.3.8 eUSCI_A Transmit Enable

The eUSCI_A 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.

20.3.9 UART Baud-Rate Generation

The eUSCI_A 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.

A quick setup for finding the correct baudrate settings for the eUSCI_A can be found in Section 20.3.10.

20.3.9.1 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 (for example, 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 eUSCI_A baud rate is one-third the UART source clock frequency BRCLK.

Timing for each bit is shown in Figure 20-10. 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.

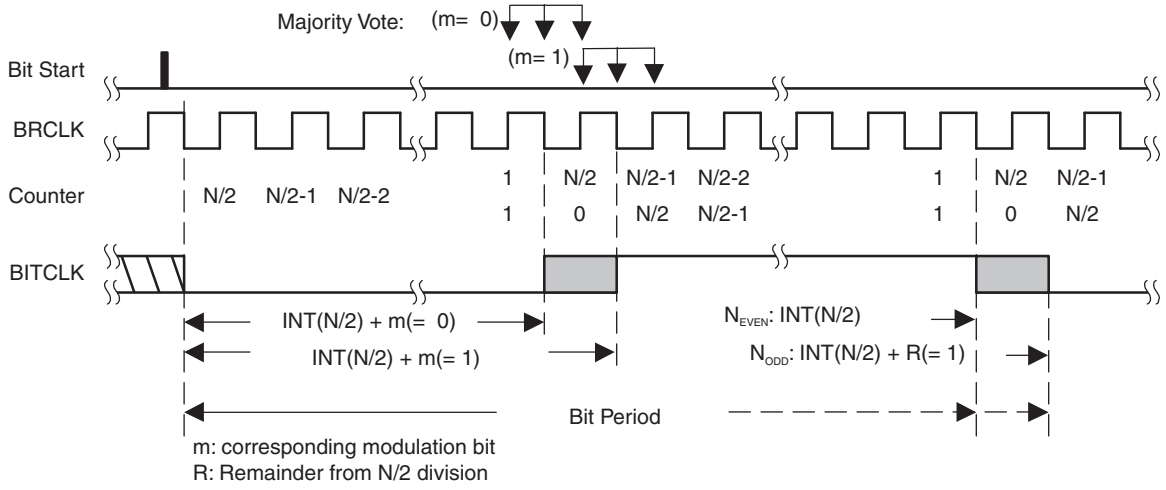


Figure 20-10. BITCLK Baud-Rate Timing With UCOS16 = 0

Modulation is based on the UCBRSx setting as shown in Table 20-2. 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 8 bits but restarts with each new start bit.

Table 20-2. Modulation Pattern Examples

UCBRSx	Bit 0 (Start Bit)	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
0x00	0	0	0	0	0	0	0	0
0x01	0	0	0	0	0	0	0	1
				⋮				
0x35	0	0	1	1	0	1	0	1
0x36	0	0	1	1	0	1	1	0
0x37	0	0	1	1	0	1	1	1
				⋮				
0xff	1	1	1	1	1	1	1	1

The correct setting of UCBRSx can be found as described in Section 20.3.10.

20.3.9.2 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 by 16 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 eUSCI_A baud rate is 1/16 the UART source clock frequency BRCLK.

Modulation for BITCLK16 is based on the UCBRFx setting (see Table 20-3). 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 as previously described.

Table 20-3. BITCLK16 Modulation Pattern

UCBRFx	Number 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

20.3.10 Setting a Baud Rate

For a given BRCLK clock source, the baud rate used determines the required division factor N:

$N = f_{BRCLK}/Baudrate$

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, it is recommended to use the oversampling baud-rate generation mode by setting UCOS16.

NOTE: Baudrate settings quick set up

To calculate the correct the correct settings for the baudrate generation, perform these steps:

1. Calculate $N = f_{BRCLK}/Baudrate$ [if N > 16 continue with step 3, otherwise with step 2]
2. OS16 = 0, UCBRx = INT(N) [continue with step 4]
3. OS16 = 1, UCBRx = INT(N/16), UCBRFx = INT([(N/16) – INT(N/16)] × 16)
4. UCBRSx can be found by looking up the fractional part of N (= N - INT(N)) in table Table 20-4
5. If OS16 = 0 was chosen, a detailed error calculation is recommended to be performed

Table 20-4 can be used as a lookup table for finding the correct UCBRSx modulation pattern for the corresponding fractional part of N. The values there are optimized for transmitting.

Table 20-4. UCBRSx Settings for Fractional Portion of $N = f_{BRCLK}/Baudrate$

Fractional Portion of N	UCBRSx[1]	Fractional Portion of N	UCBRSx[1]
0.0000	0x00	0.5002	0xAA
0.0529	0x01	0.5715	0x6B
0.0715	0x02	0.6003	0xAD
0.0835	0x04	0.6254	0xB5
0.1001	0x08	0.6432	0xB6
0.1252	0x10	0.6667	0xD6
0.1430	0x20	0.7001	0xB7
0.1670	0x11	0.7147	0xBB
0.2147	0x21	0.7503	0xDD
0.2224	0x22	0.7861	0xED
0.2503	0x44	0.8004	0xEE
0.3000	0x25	0.8333	0xBF
0.3335	0x49	0.8464	0xDF
0.3575	0x4A	0.8572	0xEF
0.3753	0x52	0.8751	0xF7
0.4003	0x92	0.9004	0xFB
0.4286	0x53	0.9170	0xFD
0.4378	0x55	0.9288	0xFE

(1) The UCBRSx setting in one row is valid from the fractional portion given in that row until the one in the next row

20.3.10.1 Low-Frequency Baud-Rate Mode Setting

In low-frequency mode, the integer portion of the divisor is realized by the prescaler:

UCBRx = INT(N)

The fractional portion is realized by the modulator with its UCBRSx setting. The recommended way of determining the correct UCBRSx is performing a detailed error calculation as explained in the following sections. However it is also possible to look up the correct settings in table with typical crystals (see Table 20-5).

20.3.10.2 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 = INT([(N/16) – INT(N/16)] × 16)

The second modulation stage setting (UCBRSx) can be found by performing a detailed error calculation or by using Table 20-4 and the fractional part of N = f_{BRCLK}/Baudrate.

20.3.11 Transmit Bit Timing - Error calculation

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.

20.3.11.1 Low-Frequency Baud-Rate Mode Bit Timing

In low-frequency mode, calculation of the length of bit i $T_{bit,TX}[i]$ is 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 of UCBRSx

20.3.11.2 Oversampling Baud-Rate Mode Bit Timing

In oversampling baud-rate mode, calculation of the length of bit i $T_{bit,TX}[i]$ is based on the baud-rate generator UCBRx, UCBRFx and UCBRSx settings:

$$T_{bit,TX}[i] = \frac{1}{f_{BRCLK}} \left((16 * UCBRx) + \sum_{j=0}^{15} m_{UCBRFx}[j] + m_{UCBRSx}[i] \right)$$

Where:

$\leq \sum_{j=0}^{15} m_{UCBRFx}[j]$ = Sum of ones from the corresponding row in Table 20-3

$m_{UCBRSx}[i]$ = Modulation of bit i of UCBRSx

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):

$Error_{TX}[i] = (t_{bit,TX}[i] – t_{bit,ideal,TX}[i]) \times Baudrate \times 100\%$

20.3.12 Receive Bit Timing – Error Calculation

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 eUSCI_A module. Figure 20-11 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.

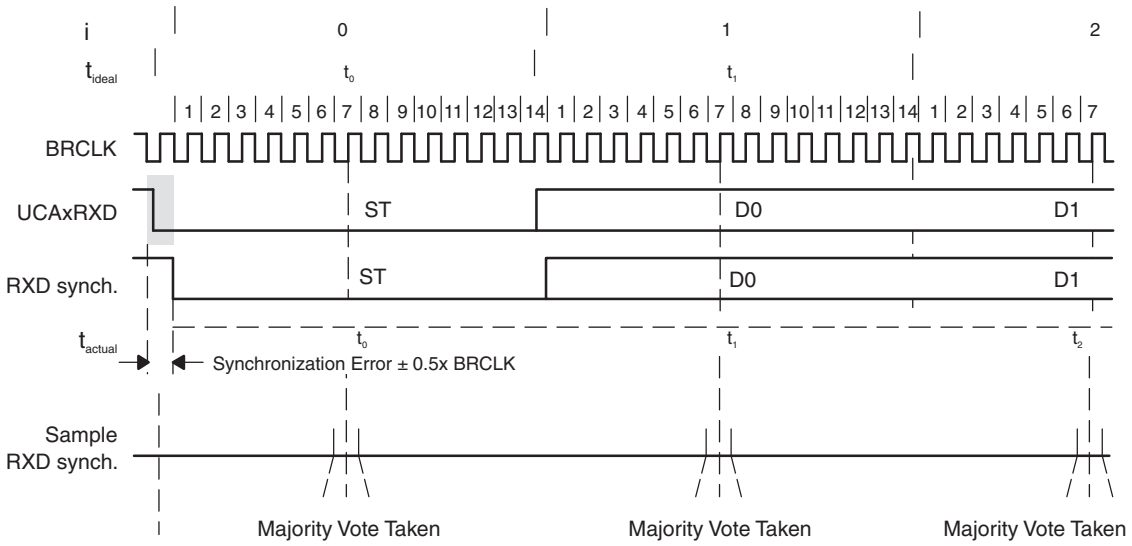


Figure 20-11. Receive Error

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 of UCBRSx

For the oversampling baud-rate mode, the sampling time $t_{bit,RX}[i]$ of bit i is calculated by:

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}} \left((8 * UCBRx) + \sum_{j=0}^{7} m_{UCBRFx}[j] + m_{UCBRSx}[i] \right)$$

Where:

$$T_{bit,RX}[i] = \frac{1}{f_{BRCLK}} \left((16 * UCBRx) + \sum_{j=0}^{15} m_{UCBRFx}[j] + m_{UCBRSx}[i] \right)$$

$\sum_{j=0}^{7 + m_{UCBRSx}[i]} m_{UCBRFx}[j]$ = Sum of ones from columns 0 to (7 + $m_{UCBRSx}[i]$) from the corresponding row in Table 20-3.

$m_{UCBRSx}[i]$ = Modulation of bit i of UCBRSx

This results in an error normalized to one ideal bit time (1/baudrate) according to the following formula:

$Error_{RX}[i] = (t_{bit,RX}[i] – t_{bit,ideal,RX}[i]) \times Baudrate \times 100\%$

20.3.13 Typical Baud Rates and Errors

Standard baud-rate data for UCBRx, UCBRSx, and UCBRFx are listed in Table 20-5 for a 32768-Hz crystal sourcing ACLK and typical SMCLK frequencies. Make sure that the selected BRCLK frequency does not exceed the device specific maximum eUSCI_A 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 20-5. Recommended Settings for Typical Crystals and Baudrates

BRCLK	Baudrate	UCOS16	UCBRx	UCBRFx	UCBRSx	TX error (%)		RX error (%)	
						neg	pos	neg	pos
32768	1200	1	1	11	0x25	-2.29	2.25	-2.56	5.35
32768	2400	0	13	-	0xB6	-3.12	3.91	-5.52	8.84
32768	4800	0	6	-	0xEE	-7.62	8.98	-21	10.25
32768	9600	0	3	-	0x92	-17.19	16.02	-23.24	37.3
1000000	9600	1	6	8	0x20	-0.48	0.64	-1.04	1.04
1000000	19200	1	3	4	0x2	-0.8	0.96	-1.84	1.84
1000000	38400	1	1	10	0x0	0	1.76	0	3.44
1000000	57600	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
1000000	115200	0	8	-	0xD6	-7.36	5.6	-17.04	6.96
1048576	9600	1	6	13	0x22	-0.46	0.42	-0.48	1.23
1048576	19200	1	3	6	0xAD	-0.88	0.83	-2.36	1.18
1048576	38400	1	1	11	0x25	-2.29	2.25	-2.56	5.35
1048576	57600	0	18	-	0x11	-2	3.37	-5.31	5.55
1048576	115200	0	9	-	0x08	-5.37	4.49	-5.93	14.92
4000000	9600	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
4000000	19200	1	13	0	0x84	-0.32	0.32	-0.64	0.48
4000000	38400	1	6	8	0x20	-0.48	0.64	-1.04	1.04
4000000	57600	1	4	5	0x55	-0.8	0.64	-1.12	1.76
4000000	115200	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
4000000	230400	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
4194304	9600	1	27	4	0xFB	-0.11	0.1	-0.33	0
4194304	19200	1	13	10	0x55	-0.21	0.21	-0.55	0.33
4194304	38400	1	6	13	0x22	-0.46	0.42	-0.48	1.23
4194304	57600	1	4	8	0xEE	-0.75	0.74	-2	0.87
4194304	115200	1	2	4	0x92	-1.62	1.37	-3.56	2.06
4194304	230400	0	18	-	0x11	-2	3.37	-5.31	5.55
8000000	9600	1	52	1	0x49	-0.08	0.04	-0.1	0.14
8000000	19200	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
8000000	38400	1	13	0	0x84	-0.32	0.32	-0.64	0.48
8000000	57600	1	8	10	0xF7	-0.32	0.32	-1	0.36
8000000	115200	1	4	5	0x55	-0.8	0.64	-1.12	1.76
8000000	230400	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
8000000	460800	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
8388608	9600	1	54	9	0xEE	-0.06	0.06	-0.11	0.13
8388608	19200	1	27	4	0xFB	-0.11	0.1	-0.33	0
8388608	38400	1	13	10	0x55	-0.21	0.21	-0.55	0.33
8388608	57600	1	9	1	0xB5	-0.31	0.31	-0.53	0.78
8388608	115200	1	4	8	0xEE	-0.75	0.74	-2	0.87
8388608	230400	1	2	4	0x92	-1.62	1.37	-3.56	2.06
8388608	460800	0	18	-	0x11	-2	3.37	-5.31	5.55
12000000	9600	1	78	2	0x0	0	0	0	0.04

Table 20-5. Recommended Settings for Typical Crystals and Baudrates (continued)

BRCLK	Baudrate	UCOS16	UCBRx	UCBRFx	UCBRSx	TX error (%)		RX error (%)	
						neg	pos	neg	pos
12000000	19200	1	39	1	0x0	0	0	0	0.16
12000000	38400	1	19	8	0x65	-0.16	0.16	-0.4	0.24
12000000	57600	1	13	0	0x25	-0.16	0.32	-0.48	0.48
12000000	115200	1	6	8	0x20	-0.48	0.64	-1.04	1.04
12000000	230400	1	3	4	0x2	-0.8	0.96	-1.84	1.84
12000000	460800	1	1	10	0x0	0	1.76	0	3.44
16000000	9600	1	104	2	0xD6	-0.04	0.02	-0.09	0.03
16000000	19200	1	52	1	0x49	-0.08	0.04	-0.1	0.14
16000000	38400	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
16000000	57600	1	17	5	0xDD	-0.16	0.2	-0.3	0.38
16000000	115200	1	8	10	0xF7	-0.32	0.32	-1	0.36
16000000	230400	1	4	5	0x55	-0.8	0.64	-1.12	1.76
16000000	460800	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
16777216	9600	1	109	3	0xB5	-0.03	0.02	-0.05	0.06
16777216	19200	1	54	9	0xEE	-0.06	0.06	-0.11	0.13
16777216	38400	1	27	4	0xFB	-0.11	0.1	-0.33	0
16777216	57600	1	18	3	0x44	-0.16	0.15	-0.2	0.45
16777216	115200	1	9	1	0xB5	-0.31	0.31	-0.53	0.78
16777216	230400	1	4	8	0xEE	-0.75	0.74	-2	0.87
16777216	460800	1	2	4	0x92	-1.62	1.37	-3.56	2.06
20000000	9600	1	130	3	0x25	-0.02	0.03	0	0.07
20000000	19200	1	65	1	0xD6	-0.06	0.03	-0.1	0.1
20000000	38400	1	32	8	0xEE	-0.1	0.13	-0.27	0.14
20000000	57600	1	21	11	0x22	-0.16	0.13	-0.16	0.38
20000000	115200	1	10	13	0xAD	-0.29	0.26	-0.46	0.66
20000000	230400	1	5	6	0xEE	-0.67	0.51	-1.71	0.62
20000000	460800	1	2	11	0x92	-1.38	0.99	-1.84	2.8

20.3.14 Using the eUSCI_A Module in UART Mode With Low-Power Modes

The eUSCI_A module provides automatic clock activation for use with low-power modes. When the eUSCI_A clock source is inactive because the device is in a low-power mode, the eUSCI_A module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the eUSCI_A module returns to its idle condition. After the eUSCI_A module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

20.3.15 eUSCI_A Interrupts

The eUSCI_A has only one interrupt vector that is shared for transmission and for reception.

20.3.15.1 eUSCI_A 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.

20.3.15.2 eUSCI_A 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.

20.3.15.3 eUSCI_A Receive Interrupt Operation

Table 20-6 describes the I^{2}C state change interrupt flags.

Table 20-6. UART State Change Interrupt Flags

Interrupt Flag	Interrupt Condition
UCSTTIFG	START byte received interrupt. This flag is set when the UART module receives a START byte.
UCTXCPTIFG	Transmit complete interrupt. This flag is set, after the complete UART byte in the internal shift register including STOP bit got shifted out and UCAxTXBUF is empty.

20.3.15.4 UCAxIV, Interrupt Vector Generator

The eUSCI_A 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.

Read access of the UCAxIV register automatically resets the highest-pending Interrupt condition and flag. Write access of the UCAxIV register clears all pending Interrupt conditions and flags. If another interrupt flag is set, another interrupt is generated immediately after servicing the initial interrupt.

Example 20-1 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 eUSCI_A0.

Example 20-1. UCAxIV Software Example

```
#pragma vector = USCI_A0_VECTOR __interrupt void USCI_A0_ISR(void) {
    switch(__even_in_range(UCA0IV,18)) {
        case 0x00:      // Vector 0: No interrupts
                 break;
        case 0x02: ...  // Vector 2: UCRXIFG
                 break;
        case 0x04: ...  // Vector 4: UCTXIFG
                 break;
        case 0x06: ...  // Vector 6: UCSTTIFG
                 break;
        case 0x08: ...  // Vector 8: UCTXCPTIFG
                 break;
        default: break;
    }
}
```

20.4 eUSCI_A UART Registers

The eUSCI_A registers applicable in UART mode and their address offsets are listed in Table 20-7. The base address can be found in the device-specific data sheet.

Table 20-7. eUSCI_A UART Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCAxCTLW0	eUSCI_Ax Control Word 0	Read/write	Word	0001h	Section 20.4.1
01h	UCAxCTL0(1)	eUSCI_Ax Control 0	Read/write	Byte	00h	
00h	UCAxCTL1	eUSCI_Ax Control 1	Read/write	Byte	01h	
02h	UCAxCTLW1	eUSCI_Ax Control Word 1	Read/write	Word	0003h	Section 20.4.2
06h	UCAxBRW	eUSCI_Ax Baud Rate Control Word	Read/write	Word	0000h	Section 20.4.3
06h	UCAxBR0(1)	eUSCI_Ax Baud Rate Control 0	Read/write	Byte	00h	
07h	UCAxBR1	eUSCI_Ax Baud Rate Control 1	Read/write	Byte	00h	
08h	UCAxMCTLW	eUSCI_Ax Modulation Control Word	Read/write	Word	00h	Section 20.4.4
0Ah	UCAxSTATW	eUSCI_Ax Status	Read/write	Word	00h	Section 20.4.5
0Ch	UCAxRXBUF	eUSCI_Ax Receive Buffer	Read/write	Word	00h	Section 20.4.6
0Eh	UCAxTXBUF	eUSCI_Ax Transmit Buffer	Read/write	Word	00h	Section 20.4.7
10h	UCAxABCTL	eUSCI_Ax Auto Baud Rate Control	Read/write	Word	00h	Section 20.4.8
12h	UCAxIRCTL	eUSCI_Ax IrDA Control	Read/write	Word	0000h	Section 20.4.9
12h	UCAxIRTCTL	eUSCI_Ax IrDA Transmit Control	Read/write	Byte	00h	
13h	UCAxIRRCTL	eUSCI_Ax IrDA Receive Control	Read/write	Byte	00h	
1Ah	UCAxIE	eUSCI_Ax Interrupt Enable	Read/write	Word	00h	Section 20.4.10
1Ch	UCAxIFG	eUSCI_Ax Interrupt Flag	Read/write	Word	02h	Section 20.4.11
1Eh	UCAxIV	eUSCI_Ax Interrupt Vector	Read	Word	0000h	Section 20.4.12

(1) It is recommended to access these registers using 16-bit access. If 8-bit access is used, the corresponding bit names must be followed by "_H".

20.4.1 UCAxCTLW0 Register

eUSCI_Ax Control Word Register 0

Figure 20-12. UCAxCTLW0 Register

15	14	13	12	11	10	9	8
UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
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

Modify only when UCSWRST = 1

Table 20-8. UCAxCTLW0 Register Description

Bit	Field	Type	Reset	Description
15	UCPEN	RW	0h	Parity enable 0b = Parity disabled 1b = Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.
14	UCPAR	RW	0h	Parity select. UCPAR is not used when parity is disabled. 0b = Odd parity 1b = Even parity
13	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
12	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
11	UCSPB	RW	0h	Stop bit select. Number of stop bits. 0b = One stop bit 1b = Two stop bits
10-9	UCMODEx	RW	0h	eUSCI_A mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0. 00b = UART mode 01b = Idle-line multiprocessor mode 10b = Address-bit multiprocessor mode 11b = UART mode with automatic baud-rate detection
8	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode
7-6	UCSSELx	RW	0h	eUSCI_A clock source select. These bits select the BRCLK source clock. 00b = UCLK 01b = ACLK 10b = SMCLK 11b = SMCLK
5	UCRXEIE	RW	0h	Receive erroneous-character interrupt enable 0b = Erroneous characters rejected and UCRXIFG is not set. 1b = Erroneous characters received set UCRXIFG.
4	UCBRKIE	RW	0h	Receive break character interrupt enable 0b = Received break characters do not set UCRXIFG. 1b = Received break characters set UCRXIFG.

Table 20-8. UCAxCTLW0 Register Description (continued)

Bit	Field	Type	Reset	Description
3	UCDORM	RW	0h	Dormant. Puts eUSCI_A into sleep mode. 0b = Not dormant. All received characters set UCRXIFG. 1b = 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.
2	UCTXADDR	RW	0h	Transmit address. Next frame to be transmitted is marked as address, depending on the selected multiprocessor mode. 0b = Next frame transmitted is data. 1b = Next frame transmitted is an address.
1	UCTXBRK	RW	0h	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. 0b = Next frame transmitted is not a break. 1b = Next frame transmitted is a break or a break/synch.
0	UCSWRST	RW	1h	Software reset enable 0b = Disabled. eUSCI_A reset released for operation. 1b = Enabled. eUSCI_A logic held in reset state.

20.4.2 UCAxCTLW1 Register

eUSCI_Ax Control Word Register 1

Figure 20-13. UCAxCTLW1 Register

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						UCGLITx	
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-1

Table 20-9. UCAxCTLW1 Register Description

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved
1-0	UCGLITx	RW	3h	Deglitch time 00b = Approximately 2 ns 01b = Approximately 50 ns 10b = Approximately 100 ns 11b = Approximately 200 ns

20.4.3 UCAxBRW Register

eUSCI_Ax Baud Rate Control Word Register

Figure 20-14. UCAxBRW Register

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

Modify only when UCSWRST = 1

Table 20-10. UCAxBRW Register Description

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Clock prescaler setting of the Baud rate generator

20.4.4 UCAxMCTLW Register

eUSCI_Ax Modulation Control Word Register

Figure 20-15. UCAxMCTLW Register

15	14	13	12	11	10	9	8
UCBRSx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
UCBRFx				Reserved			UCOS16
rw-0	rw-0	rw-0	rw-0	r0	r0	r0	rw-0

Modify only when UCSWRST = 1

Table 20-11. UCAxMCTLW Register Description

Bit	Field	Type	Reset	Description
15-8	UCBRSx	RW	0h	Second modulation stage select. These bits hold a free modulation pattern for BITCLK.
7-4	UCBRFx	RW	0h	First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. The "Oversampling Baud-Rate Generation" section shows the modulation pattern.
3-1	Reserved	R	0h	Reserved
0	UCOS16	RW	0h	Oversampling mode enabled 0b = Disabled 1b = Enabled

20.4.5 UCAxSTATW Register

eUSCI_Ax Status Register

Figure 20-16. UCAxSTATW 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
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

Modify only when UCSWRST = 1

Table 20-12. UCAxSTATW Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7	UCLISTEN	RW	0h	Listen enable. The UCLISTEN bit selects loopback mode. 0b = Disabled 1b = Enabled. UCAxTXD is internally fed back to the receiver.
6	UCFE	RW	0h	Framing error flag. UCFE is cleared when UCAxRXBUF is read. 0b = No error 1b = Character received with low stop bit
5	UCOE	RW	0h	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. 0b = No error 1b = Overrun error occurred.
4	UCPE	RW	0h	Parity error flag. When UCPEN = 0, UCPE is read as 0. UCPE is cleared when UCAxRXBUF is read. 0b = No error 1b = Character received with parity error
3	UCBRK	RW	0h	Break detect flag. UCBRK is cleared when UCAxRXBUF is read. 0b = No break condition 1b = Break condition occurred.
2	UCRXERR	RW	0h	Receive error flag. This bit indicates a character was received with one or more errors. When UCRXERR = 1, on or more error flags, UCFE, UCPE, or UCOE is also set. UCRXERR is cleared when UCAxRXBUF is read. 0b = No receive errors detected 1b = Receive error detected
1	UCADDR UCIDLE	RW	0h	UCADDR: Address received in address-bit multiprocessor mode. UCADDR is cleared when UCAxRXBUF is read. UCIDLE: Idle line detected in idle-line multiprocessor mode. UCIDLE is cleared when UCAxRXBUF is read. 0b = UCADDR: Received character is data. UCIDLE: No idle line detected 1b = UCADDR: Received character is an address. UCIDLE: Idle line detected
0	UCBUSY	R	0h	eUSCI_A busy. This bit indicates if a transmit or receive operation is in progress. 0b = eUSCI_A inactive 1b = eUSCI_A transmitting or receiving

20.4.6 UCAxRXBUF Register

eUSCI_Ax Receive Buffer Register

Figure 20-17. UCAxRXBUF Register

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
UCRXBUFx							
r	r	r	r	r	r	r	r

Table 20-13. UCAxRXBUF Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCRXBUFx	R	0h	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.

20.4.7 UCAxTXBUF Register

eUSCI_Ax Transmit Buffer Register

Figure 20-18. UCAxTXBUF Register

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
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

Table 20-14. UCAxTXBUF Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCTXBUFx	RW	0h	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.

20.4.8 UCAxABCTL Register

eUSCI_Ax Auto Baud Rate Control Register

Figure 20-19. UCAxABCTL Register

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		UCDELIMx		UCSTOE	UCBTOE	Reserved	UCABDEN
r-0	r-0	rw-0	rw-0	rw-0	rw-0	r-0	rw-0

Modify only when UCSWRST = 1

Table 20-15. UCAxABCTL Register Description

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved
5-4	UCDELIMx	RW	0h	Break/synch delimiter length 00b = 1 bit time 01b = 2 bit times 10b = 3 bit times 11b = 4 bit times
3	UCSTOE	RW	0h	Synch field time out error 0b = No error 1b = Length of synch field exceeded measurable time.
2	UCBTOE	RW	0h	Break time out error 0b = No error 1b = Length of break field exceeded 22 bit times.
1	Reserved	R	0h	Reserved
0	UCABDEN	RW	0h	Automatic baud-rate detect enable 0b = Baud-rate detection disabled. Length of break and synch field is not measured. 1b = Baud-rate detection enabled. Length of break and synch field is measured and baud-rate settings are changed accordingly.

20.4.9 UCAxIRCTL Register

eUSCI_Ax IrDA Control Word Register

Figure 20-20. UCAxIRCTL Register

15	14	13	12	11	10	9	8
UCIRRXFLx						UCIRRXPL	UCIRRXFE
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
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

Modify only when UCSWRST = 1

Table 20-16. UCAxIRCTL Register Description

Bit	Field	Type	Reset	Description
15-10	UCIRRXFLx	RW	0h	Receive filter length. The minimum pulse length for receive is given by: t(MIN) = (UCIRRXFLx + 4) / [2 × f(IRTXCLK)]
9	UCIRRXPL	RW	0h	IrDA receive input UCAxRXD polarity 0b = IrDA transceiver delivers a high pulse when a light pulse is seen. 1b = IrDA transceiver delivers a low pulse when a light pulse is seen.
8	UCIRRXFE	RW	0h	IrDA receive filter enabled 0b = Receive filter disabled 1b = Receive filter enabled
7-2	UCIRTXPLx	RW	0h	Transmit pulse length. Pulse length t(PULSE) = (UCIRTXPLx + 1) / [2 × f(IRTXCLK)]
1	UCIRTXCLK	RW	0h	IrDA transmit pulse clock select 0b = BRCLK 1b = BITCLK16 when UCOS16 = 1. Otherwise, BRCLK.
0	UCIREN	RW	0h	IrDA encoder/decoder enable 0b = IrDA encoder/decoder disabled 1b = IrDA encoder/decoder enabled

20.4.10 UCAxIE Register

eUSCI_Ax Interrupt Enable Register

Figure 20-21. UCAxIE Register

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				UCTXCPTIE	UCSTTIE	UCTXIE	UCRXIE
r-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

Table 20-17. UCAxIE Register Description

Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved
3	UCTXCPTIE	RW	0h	Transmit complete interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
2	UCSTTIE	RW	0h	Start bit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
1	UCTXIE	RW	0h	Transmit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
0	UCRXIE	RW	0h	Receive interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

20.4.11 UCAxIFG Register

eUSCI_Ax Interrupt Flag Register

Figure 20-22. UCAxIFG Register

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				UCTXCPTIFG	UCSTTIFG	UCTXIFG	UCRXIFG
r-0	r-0	r-0	r-0	rw-0	rw-0	rw-1	rw-0

Table 20-18. UCAxIFG Register Description

Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved
3	UCTXCPTIFG	RW	0h	Transmit ready interrupt enable. UCTXRDYIFG is set when the entire byte in the internal shift register got shifted out and UCAxTXBUF is empty. 0b = No interrupt pending 1b = Interrupt pending
2	UCSTTIFG	RW	0h	Start bit interrupt flag. UCSTTIFG is set after a Start bit was received 0b = No interrupt pending 1b = Interrupt pending
1	UCTXIFG	RW	1h	Transmit interrupt flag. UCTXIFG is set when UCAxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	RW	0h	Receive interrupt flag. UCRXIFG is set when UCAxRXBUF has received a complete character. 0b = No interrupt pending 1b = Interrupt pending

20.4.12 UCAxIV Register

eUSCI_Ax Interrupt Vector Register

Figure 20-23. UCAxIV Register

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

Table 20-19. UCAxIV Register Description

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	eUSCI_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Receive buffer full; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest 04h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG 06h = Interrupt Source: Start bit received; Interrupt Flag: UCSTTIFG 08h = Interrupt Source: Transmit complete; Interrupt Flag: UCTXCPTIFG; Interrupt Priority: Lowest

Enhanced Universal Serial Communication Interface (eUSCI) – SPI Mode

The enhanced universal serial communication interfaces, eUSCI_A and eUSCI_B, support multiple serial communication modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface (SPI) mode.

Topic **Page**

21.1 Enhanced Universal Serial Communication Interfaces (eUSCI_A, eUSCI_B) Overview **574**
21.2 eUSCI Introduction – SPI Mode **574**
21.3 eUSCI Operation – SPI Mode **576**
21.4 eUSCI_A SPI Registers **582**
21.5 eUSCI_B SPI Registers **592**

21.1 Enhanced Universal Serial Communication Interfaces (eUSCI_A, eUSCI_B) Overview

Both the eUSCI_A and the eUSCI_B support serial communication in SPI mode.

21.2 eUSCI Introduction – SPI Mode

In synchronous mode, the eUSCI 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 21-1 shows the eUSCI when configured for SPI mode.

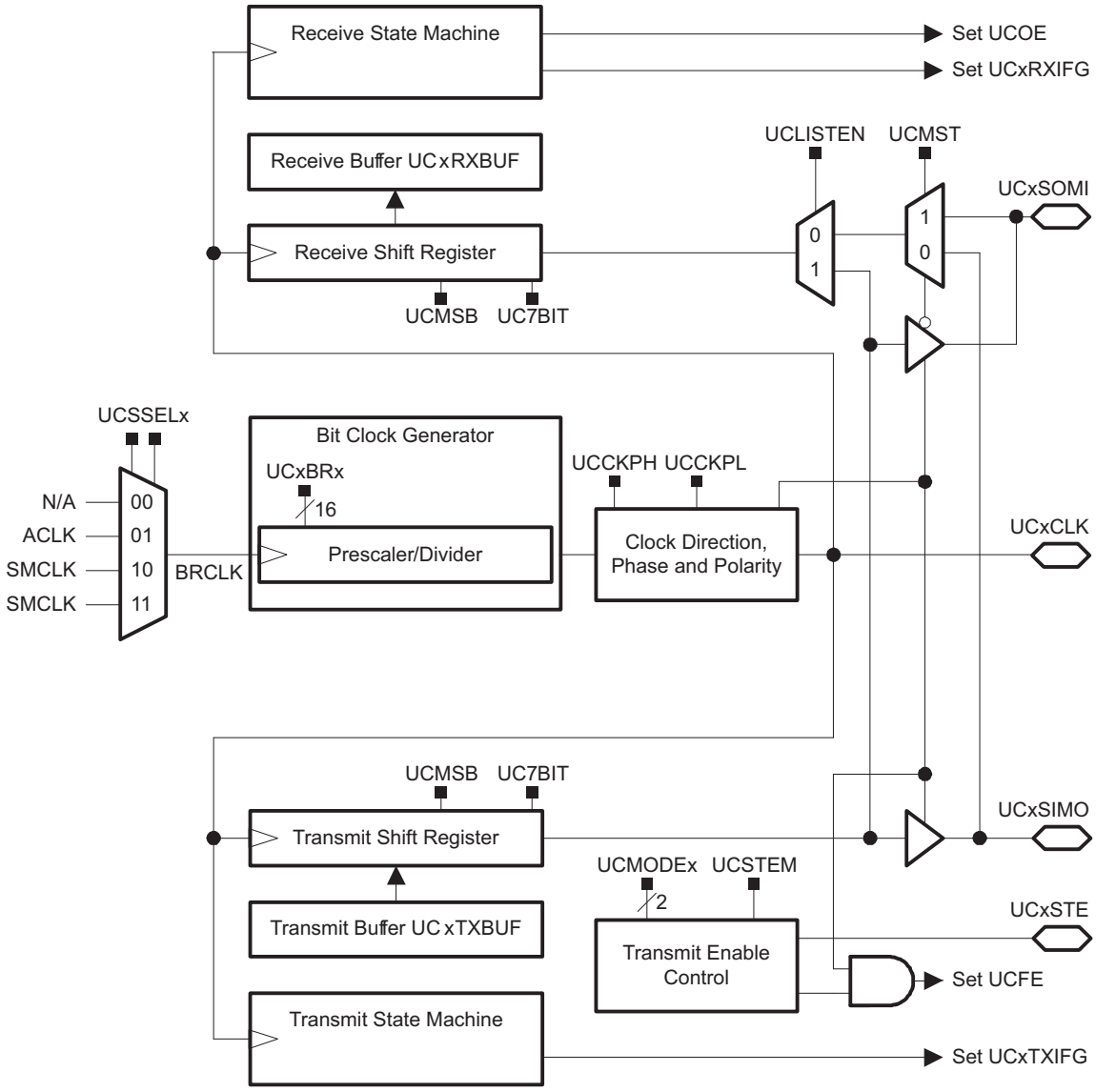


Figure 21-1. eUSCI Block Diagram – SPI Mode

21.3 eUSCI 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 controlled by the master, UCxSTE, is provided to enable a device to receive and transmit data.

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 – eUSCI 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 21-1 describes the UCxSTE operation.

Table 21-1. 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

21.3.1 eUSCI Initialization and Reset

The eUSCI is reset by a PUC or by the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the eUSCI 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 eUSCI for operation.

Configuring and reconfiguring the eUSCI module should be done when UCSWRST is set to avoid unpredictable behavior.

NOTE: Initializing or reconfiguring the eUSCI module

The recommended eUSCI initialization/reconfiguration process is:

1. Set UCSWRST.

   ```
   BIS.B #UCSWRST,&UCxCTL1
   ```
2. Initialize all eUSCI registers with UCSWRST = 1 (including UCxCTL1).
3. Configure ports.
4. Clear UCSWRST via software.

   ```
   BIC.B #UCSWRST,&UCxCTL1
   ```
5. Enable interrupts (optional) via UCRXIE or UCTXIE.

21.3.2 Character Format

The eUSCI 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.

21.3.3 Master Mode

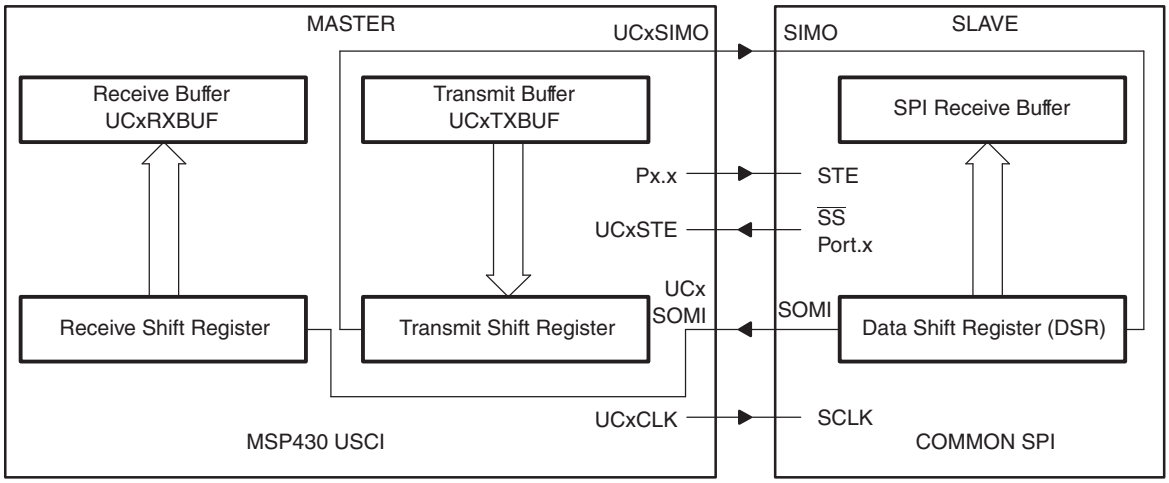


Figure 21-2. eUSCI Master and External Slave (UCSTEM = 0)

Figure 21-2 shows the eUSCI as a master in both 3-pin and 4-pin configurations. The eUSCI 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 eUSCI in master mode, data must be written to UCxTXBUF, because receive and transmit operations operate concurrently.

There two different options for configuring the eUSCI as a 4-pin master, which are described in the next sections:

- The fourth pin is used as input to prevent conflicts with other masters (UCSTEM = 0).
- The fourth pin is used as output to generate a slave enable signal (UCSTEM = 1).

The bit UCSTEM is used to select the corresponding mode.

21.3.3.1 4-Pin SPI Master Mode (UCSTEM = 0)

In 4-pin master mode with UCSTEM = 0, UCxSTE is a digital input that can be used to prevent conflicts with another master and controls the master as described in Table 21-1. When UCxSTE is in the master-inactive state and UCSTEM = 0:

- 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.

21.3.3.2 4-Pin SPI Master Mode (UCSTEM = 1)

If UCSTEM = 1 in 4-pin master mode, UCxSTE is a digital output. In this mode the slave enable signal for a single slave is automatically generated on UCxSTE. The corresponding behavior can be seen in Figure 21-4.

If multiple slaves are desired, this feature is not applicable and the software needs to use general purpose I/O pins instead to generate STE signals for each slave individually.

21.3.4 Slave Mode

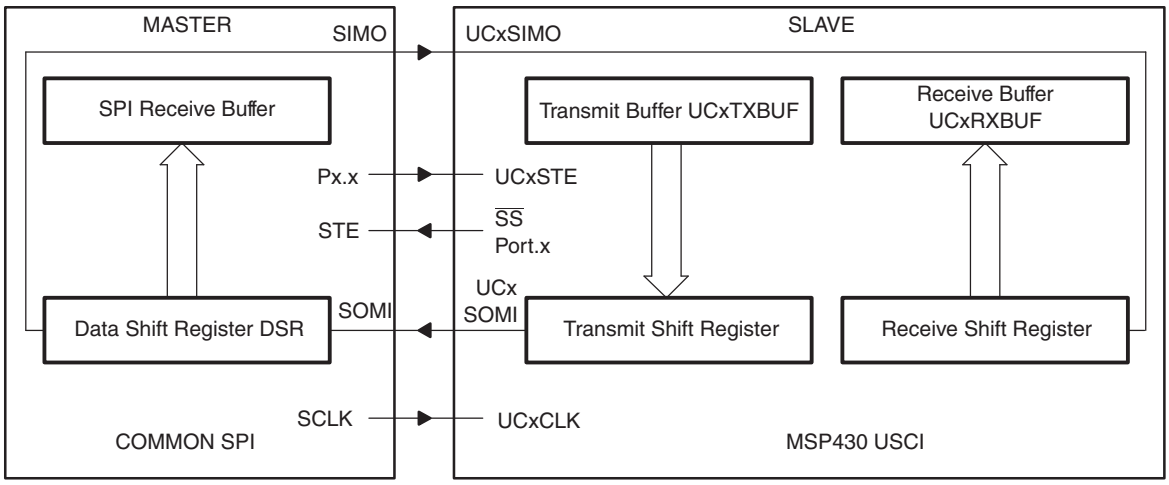


Figure 21-3. eUSCI Slave and External Master

Figure 21-3 shows the eUSCI 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.

21.3.4.1 4-Pin SPI Slave Mode

In 4-pin slave mode, UCxSTE is a digital input used by the slave to enable the transmit and receive operations and is driven 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.

21.3.5 SPI Enable

When the eUSCI 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 eUSCI immediately and any active transfer is terminated.

21.3.5.1 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.

21.3.5.2 Receive Enable

The SPI receives data when a transmission is active. Receive and transmit operations operate concurrently.

21.3.6 Serial Clock Control

UCxCLK is provided by the master on the SPI bus. When UCMST = 1, the bit clock is provided by the eUSCI 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 eUSCI 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 UCxxBRW is the division factor of the eUSCI clock source, BRCLK. With UCBRx = 0 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 eUSCI_A. The UCAxCLK/UCBxCLK frequency is given by:

$f_{BitClock} = f_{BRCLK}/(UCBRx+1)$.

Odd UCBRx settings result in even divisions and, thus, generate a bit clock with a 50/50 duty cycle.

Even UCBRx settings result in odd divisions. In this case, the high phase of the bit clock is one BRCLK cycle longer than the low phase.

When UCBRx = 0, no division is applied to BRCLK ,and the bit clock equals BRCLK.

21.3.6.1 Serial Clock Polarity and Phase

The polarity and phase of UCxCLK are independently configured via the UCCKPL and UCCKPH control bits of the eUSCI. Timing for each case is shown in Figure 21-4.

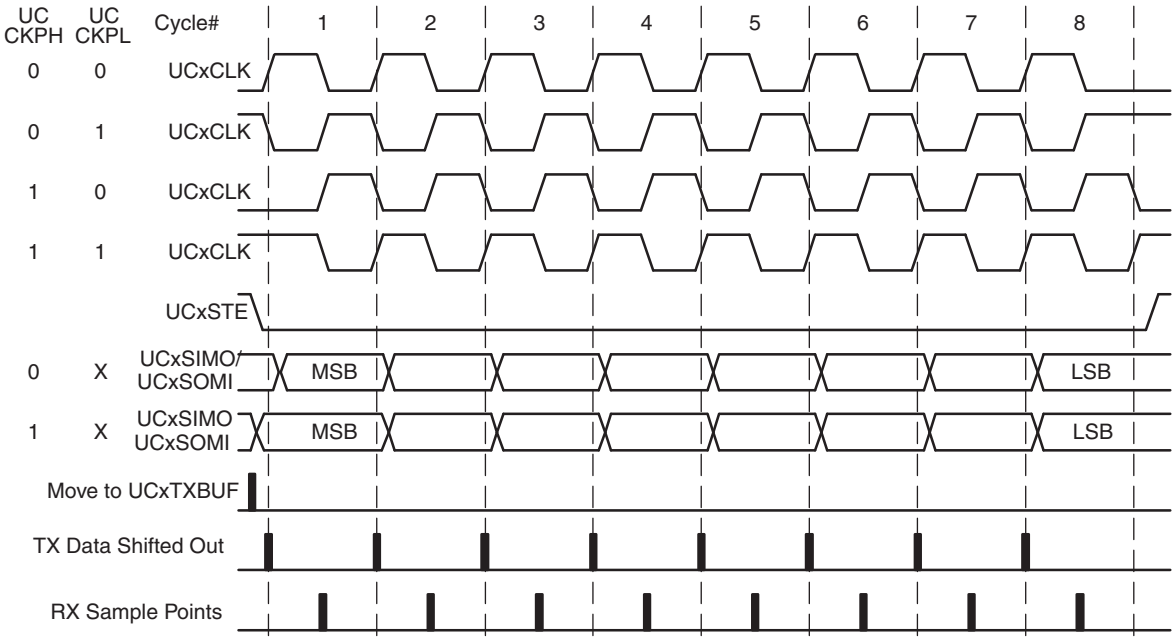


Figure 21-4. eUSCI SPI Timing With UCMSB = 1

21.3.7 Using the SPI Mode With Low-Power Modes

The eUSCI module provides automatic clock activation for use with low-power modes. When the eUSCI clock source is inactive because the device is in a low-power mode, the eUSCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the eUSCI module returns to its idle condition. After the eUSCI 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 eUSCI 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.

21.3.8 SPI Interrupts

The eUSCI has only one interrupt vector that is shared for transmission and for reception. eUSCI_Ax and eUSCI_Bx do not share the same interrupt vector.

21.3.8.1 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.

21.3.8.2 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.

21.3.8.3 UCxIV, Interrupt Vector Generator

The eUSCI 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.

21.3.8.3.1 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 eUSCI_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
```

21.4 eUSCI_A SPI Registers

The eUSCI_A registers applicable in SPI mode and their address offsets are listed in Table 21-2. The base addresses can be found in the device-specific data sheet.

Table 21-2. eUSCI_A SPI Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCAxCTLW0	eUSCI_Ax Control Word 0	Read/write	Word	0001h	Section 21.4.1
00h	UCAxCTL1	eUSCI_Ax Control 1	Read/write	Byte	01h	
01h	UCAxCTL0	eUSCI_Ax Control 0	Read/write	Byte	00h	
06h	UCAxBRW	eUSCI_Ax Bit Rate Control Word	Read/write	Word	0000h	Section 21.4.2
06h	UCAxBR0	eUSCI_Ax Bit Rate Control 0	Read/write	Byte	00h	
07h	UCAxBR1	eUSCI_Ax Bit Rate Control 1	Read/write	Byte	00h	
0Ah	UCAxSTATW	eUSCI_Ax Status	Read/write	Word	00h	Section 21.4.3
0Ch	UCAxRXBUF	eUSCI_Ax Receive Buffer	Read/write	Word	00h	Section 21.4.4
0Eh	UCAxTXBUF	eUSCI_Ax Transmit Buffer	Read/write	Word	00h	Section 21.4.5
1Ah	UCAxIE	eUSCI_Ax Interrupt Enable	Read/write	Word	00h	Section 21.4.6
1Ch	UCAxIFG	eUSCI_Ax Interrupt Flag	Read/write	Word	02h	Section 21.4.7
1Eh	UCAxIV	eUSCI_Ax Interrupt Vector	Read	Word	0000h	Section 21.4.8

21.4.1 UCAxCTLW0 Register

eUSCI_Ax Control Register 0

Figure 21-5. UCAxCTLW0 Register

15	14	13	12	11	10	9	8
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
UCSSELx		Reserved				UCSTEM	UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

Modify only when UCSWRST = 1.

Table 21-3. UCAxCTLW0 Register Description

Bit	Field	Type	Reset	Description
15	UCCKPH	RW	0h	Clock phase select 0b = Data is changed on the first UCLK edge and captured on the following edge. 1b = Data is captured on the first UCLK edge and changed on the following edge.
14	UCCKPL	RW	0h	Clock polarity select 0b = The inactive state is low. 1b = The inactive state is high.
13	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
12	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
11	UCMST	RW	0h	Master mode select 0b = Slave mode 1b = Master mode
10-9	UCMODEx	RW	0h	eUSCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00b = 3-pin SPI 01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11b = Reserved
8	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode
7-6	UCSSELx	RW	0h	eUSCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode. 00b = Reserved 01b = ACLK 10b = SMCLK 11b = SMCLK
5-2	Reserved	R	0h	Reserved
1	UCSTEM	RW	0h	STE mode select in master mode. This byte is ignored in slave or 3-wire mode. 0b = STE pin is used to prevent conflicts with other masters 1b = STE pin is used to generate the enable signal for a 4-wire slave

Table 21-3. UCAxCTLW0 Register Description (continued)

Bit	Field	Type	Reset	Description
0	UCSWRST	RW	1h	Software reset enable 0b = Disabled. eUSCI reset released for operation. 1b = Enabled. eUSCI logic held in reset state.

21.4.2 UCAxBRW Register

eUSCI_Ax Bit Rate Control Register 1

Figure 21-6. UCAxBRW Register

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

Modify only when UCSWRST = 1.

Table 21-4. UCAxBRW Register Description

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Bit clock prescaler setting. $f_{BitClock} = f_{BRCLK}/(UCBRx+1)$

21.4.3 UCAxSTATW Register

eUSCI_Ax Status Register

Figure 21-7. UCAxSTATW 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
UCLISTEN	UCFE	UCOE	Reserved				UCBUSY
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

Modify only when UCSWRST = 1.

Table 21-5. UCAxSTATW Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7	UCLISTEN	RW	0h	Listen enable. The UCLISTEN bit selects loopback mode. 0b = Disabled 1b = Enabled. The transmitter output is internally fed back to the receiver.
6	UCFE	RW	0h	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. 0b = No error 1b = Bus conflict occurred
5	UCOE	RW	0h	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. 0b = No error 1b = Overrun error occurred
4-1	Reserved	RW	0h	Reserved
0	UCBUSY	R	0h	eUSCI busy. This bit indicates if a transmit or receive operation is in progress. 0b = eUSCI inactive 1b = eUSCI transmitting or receiving

21.4.4 UCAxRXBUF Register

eUSCI_Ax Receive Buffer Register

Figure 21-8. UCAxRXBUF 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
UCRXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

Table 21-6. UCAxRXBUF Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCRXBUFx	R	0h	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.

21.4.5 UCAxTXBUF Register

eUSCI_Ax Transmit Buffer Register

Figure 21-9. UCAxTXBUF 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
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

Table 21-7. UCAxTXBUF Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCTXBUFx	RW	0h	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.

21.4.6 UCAxIE Register

eUSCI_Ax Interrupt Enable Register

Figure 21-10. UCAxIE 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							
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Table 21-8. UCAxIE Register Description

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved
1	UCTXIE	RW	0h	Transmit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
0	UCRXIE	RW	0h	Receive interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

21.4.7 UCAxIFG Register

eUSCI_Ax Interrupt Flag Register

Figure 21-11. UCAxIFG 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						UCTXIFG	UCRXIFG
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-0

Table 21-9. UCAxIFG Register Description

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved
1	UCTXIFG	RW	1h	Transmit interrupt flag. UCTXIFG is set when UCxxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	RW	0h	Receive interrupt flag. UCRXIFG is set when UCxxRXBUF has received a complete character. 0b = No interrupt pending 1b = Interrupt pending

21.4.8 UCAxIV Register

eUSCI_Ax Interrupt Vector Register

Figure 21-12. UCAxIV Register

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

Table 21-10. UCAxIV Register Description

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	eUSCI interrupt vector value 000h = No interrupt pending 002h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest 004h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG; Interrupt Priority: Lowest

21.5 eUSCI_B SPI Registers

The eUSCI_B registers applicable in SPI mode and their address offsets are listed in Table 21-11. The base addresses can be found in the device-specific data sheet.

Table 21-11. eUSCI_B SPI Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCBxCTLW0	eUSCI_Bx Control Word 0	Read/write	Word	01C1h	Section 21.5.1
00h	UCBxCTL1	eUSCI_Bx Control 1	Read/write	Byte	C1h	
01h	UCBxCTL0	eUSCI_Bx Control 0	Read/write	Byte	01h	
06h	UCBxBRW	eUSCI_Bx Bit Rate Control Word	Read/write	Word	0000h	Section 21.5.2
06h	UCBxBR0	eUSCI_Bx Bit Rate Control 0	Read/write	Byte	00h	
07h	UCBxBR1	eUSCI_Bx Bit Rate Control 1	Read/write	Byte	00h	
08h	UCBxSTATW	eUSCI_Bx Status	Read/write	Word	00h	Section 21.5.3
0Ch	UCBxRXBUF	eUSCI_Bx Receive Buffer	Read/write	Word	00h	Section 21.5.4
0Eh	UCBxTXBUF	eUSCI_Bx Transmit Buffer	Read/write	Word	00h	Section 21.5.5
2Ah	UCBxIE	eUSCI_Bx Interrupt Enable	Read/write	Word	00h	Section 21.5.6
2Ch	UCBxIFG	eUSCI_Bx Interrupt Flag	Read/write	Word	02h	Section 21.5.7
2Eh	UCBxIV	eUSCI_Bx Interrupt Vector	Read	Word	0000h	Section 21.5.8

21.5.1 UCBxCTLW0 Register

eUSCI_Bx Control Register 0

Figure 21-13. UCBxCTLW0 Register

15	14	13	12	11	10	9	8
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1
7	6	5	4	3	2	1	0
UCSSELx		Reserved				UCSTEM	UCSWRST
rw-1	rw-1	r0	rw-0	rw-0	rw-0	rw-0	rw-1

Modify only when UCSWRST = 1.

Table 21-12. UCBxCTLW0 Register Description

Bit	Field	Type	Reset	Description
15	UCCKPH	RW	0h	Clock phase select 0b = Data is changed on the first UCLK edge and captured on the following edge. 1b = Data is captured on the first UCLK edge and changed on the following edge.
14	UCCKPL	RW	0h	Clock polarity select 0b = The inactive state is low. 1b = The inactive state is high.
13	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
12	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
11	UCMST	RW	0h	Master mode select 0b = Slave mode 1b = Master mode
10-9	UCMODEx	RW	0h	eUSCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00b = 3-pin SPI 01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11b = I2C mode
8	UCSYNC	RW	1h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode
7-6	UCSSELx	RW	3h	eUSCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode. 00b = Reserved 01b = ACLK 10b = SMCLK 11b = SMCLK
5-2	Reserved	R	0h	Reserved
1	UCSTEM	RW	0h	STE mode select in master mode. This byte is ignored in slave or 3-wire mode. 0b = STE pin is used to prevent conflicts with other masters 1b = STE pin is used to generate the enable signal for a 4-wire slave

Table 21-12. UCBxCTLW0 Register Description (continued)

Bit	Field	Type	Reset	Description
0	UCSWRST	RW	1h	Software reset enable 0b = Disabled. eUSCI reset released for operation. 1b = Enabled. eUSCI logic held in reset state.

21.5.2 UCBxBRW Register

eUSCI_Bx Bit Rate Control Register 1

Figure 21-14. UCBxBRW Register

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

Modify only when UCSWRST = 1.

Table 21-13. UCBxBRW Register Description

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Bit clock prescaler setting. $f_{BitClock} = f_{BRCLK}/(UCBRx+1)$

21.5.3 UCBxSTATW Register

eUSCI_Bx Status Register

Figure 21-15. UCBxSTATW 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
UCLISTEN	UCFE	UCOE	Reserved				UCBUSY
rw-0	rw-0	rw-0	r0	r0	r0	r0	r-0

Modify only when UCSWRST = 1.

Table 21-14. UCBxSTATW Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7	UCLISTEN	RW	0h	Listen enable. The UCLISTEN bit selects loopback mode. 0b = Disabled 1b = Enabled. The transmitter output is internally fed back to the receiver.
6	UCFE	RW	0h	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. 0b = No error 1b = Bus conflict occurred
5	UCOE	RW	0h	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. 0b = No error 1b = Overrun error occurred
4-1	Reserved	R	0h	Reserved
0	UCBUSY	R	0h	eUSCI busy. This bit indicates if a transmit or receive operation is in progress. 0b = eUSCI inactive 1b = eUSCI transmitting or receiving

21.5.4 UCBxRXBUF Register

eUSCI_Bx Receive Buffer Register

Figure 21-16. UCBxRXBUF 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
UCRXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

Table 21-15. UCBxRXBUF Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCRXBUFx	R	0h	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.

21.5.5 UCBxTXBUF Register

eUSCI_Bx Transmit Buffer Register

Figure 21-17. UCBxTXBUF 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
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

Table 21-16. UCBxTXBUF Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCTXBUFx	RW	0h	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.

21.5.6 UCBxIE Register

eUSCI_Bx Interrupt Enable Register

Figure 21-18. UCBxIE 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							
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Table 21-17. UCBxIE Register Description

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved
1	UCTXIE	RW	0h	Transmit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
0	UCRXIE	RW	0h	Receive interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

21.5.7 UCBxIFG Register

eUSCI_Bx Interrupt Flag Register

Figure 21-19. UCBxIFG 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						UCTXIFG	UCRXIFG
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-0

Table 21-18. UCBxIFG Register Description

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved
1	UCTXIFG	RW	1h	Transmit interrupt flag. UCTXIFG is set when UCxxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	RW	0h	Receive interrupt flag. UCRXIFG is set when UCxxRXBUF has received a complete character. 0b = No interrupt pending 1b = Interrupt pending

21.5.8 UCBxIV Register

eUSCI_Bx Interrupt Vector Register

Figure 21-20. UCBxIV Register

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

Table 21-19. UCBxIV Register Description

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	eUSCI interrupt vector value 0000h = No interrupt pending 0002h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest 0004h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG; Interrupt Priority: Lowest

Enhanced Universal Serial Communication Interface (eUSCI) – I^2C Mode

The enhanced universal serial communication interface B (eUSCI_B) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the I^2C mode.

Topic **Page**

22.1 Enhanced Universal Serial Communication Interface B (eUSCI_B) Overview 600
22.2 eUSCI_B Introduction – I^2C Mode 600
22.3 eUSCI_B Operation – I^2C Mode 601
22.4 eUSCI_B I2C Registers 621

22.1 Enhanced Universal Serial Communication Interface B (eUSCI_B) Overview

The eUSCI_B module supports two serial communication modes:

- I^2C mode
- SPI mode

If more than one eUSCI_B module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two eUSCI_B modules, they are named eUSCI0_B and eUSCI1_B.

22.2 eUSCI_B Introduction – I^2C Mode

In I^2C mode, the eUSCI_B module provides an interface between the device and I^2C-compatible devices connected by the two-wire I^2C serial bus. External components attached to the I^2C bus serially transmit or receive serial data to or from the eUSCI_B module through the 2-wire I^2C interface.

The eUSCI_B I^2C mode features include:

- 7-bit and 10-bit device addressing modes
- General call
- START, RESTART, STOP
- Multi-master transmitter or receiver mode
- Slave receiver or 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
- 8-bit byte counter with interrupt capability and automatic STOP assertion
- Up to four hardware slave addresses, each having its own interrupt and DMA trigger
- Mask register for slave address and address received interrupt
- Clock low timeout interrupt to avoid bus stalls
- Slave operation in LPM4
- Slave receiver START detection for auto wake-up from LPMx modes (not LPM3.5 and LPM4.5)

Figure 22-1 shows the eUSCI_B when configured in I^2C mode.

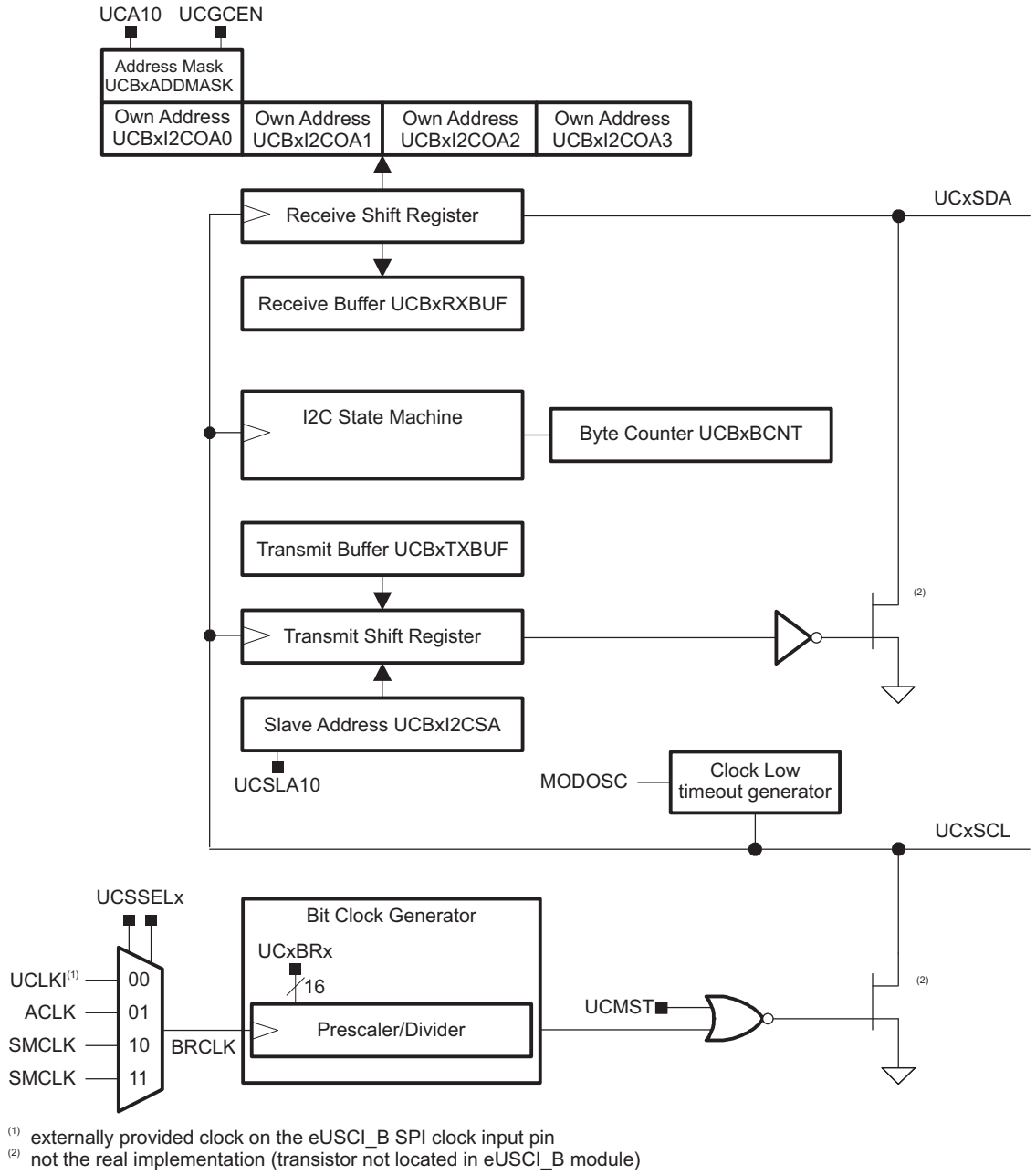


Figure 22-1. eUSCI_B Block Diagram – I²C Mode

22.3 eUSCI_B Operation – I²C Mode

The I²C mode supports any slave or master I²C-compatible device. Figure 22-2 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.

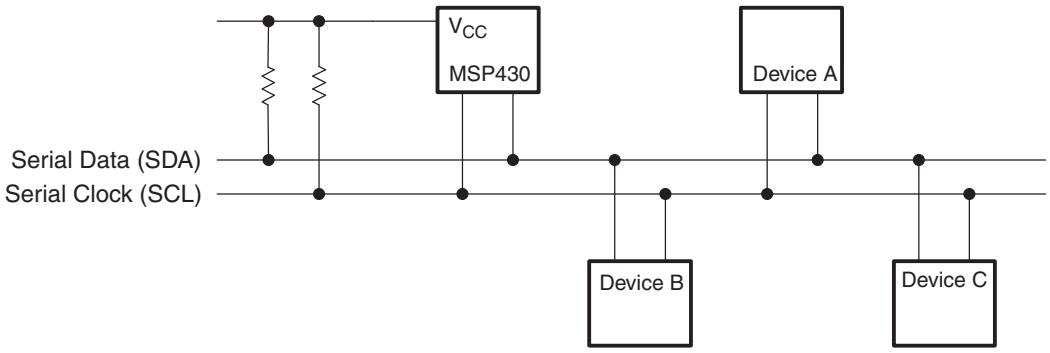


Figure 22-2. I²C Bus Connection Diagram

NOTE: SDA and SCL levels

The SDA and SCL pins must not be pulled up above the device V_{CC} level.

22.3.1 eUSCI_B Initialization and Reset

The eUSCI_B is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the eUSCI_B 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 eUSCI_B for operation.

Configuring and reconfiguring the eUSCI_B 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.
- UCBxSTAT, bits 15-9 and 6-4 are cleared.
- Registers UCBxIE and UCBxIFG are cleared.
- All other bits and registers remain unchanged.

NOTE: Initializing or re-configuring the eUSCI_B module

The recommended eUSCI_B initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST,&UCxCTL1`).
2. Initialize all eUSCI_B registers with UCSWRST = 1 (including UCxCTL1).
3. Configure ports.
4. Clear UCSWRST via software (`BIC.B #UCSWRST,&UCxCTL1`).
5. Enable interrupts (optional).

22.3.2 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 22-3.

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.

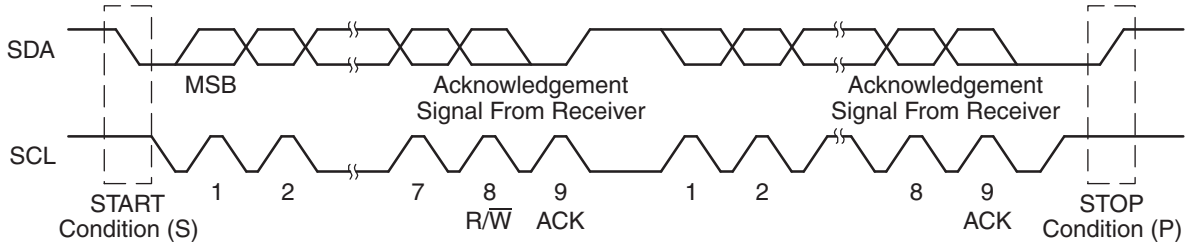


Figure 22-3. I²C Module Data Transfer

START and STOP conditions are generated by the master and are shown in Figure 22-3. 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 22-4). The high and low state of SDA can change only when SCL is low, otherwise START or STOP conditions are generated.

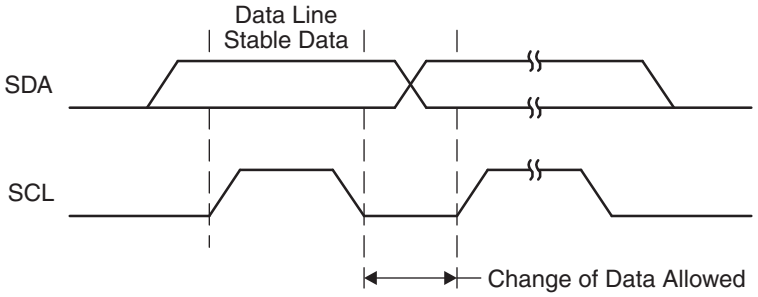


Figure 22-4. Bit Transfer on I²C Bus

22.3.3 I²C Addressing Modes

The I²C mode supports 7-bit and 10-bit addressing modes.

22.3.3.1 7-Bit Addressing

In the 7-bit addressing format (see Figure 22-5), 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.

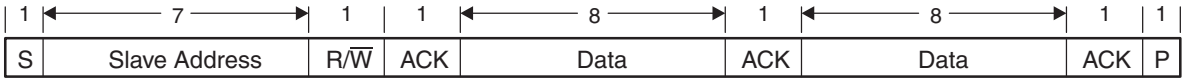


Figure 22-5. I²C Module 7-Bit Addressing Format

22.3.3.2 10-Bit Addressing

In the 10-bit addressing format (see Figure 22-6), 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 eUSCI_B module.

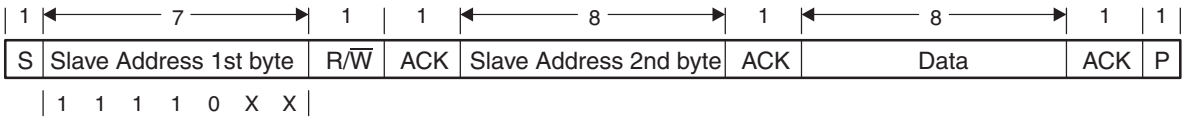


Figure 22-6. I²C Module 10-Bit Addressing Format

22.3.3.3 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 22-7.

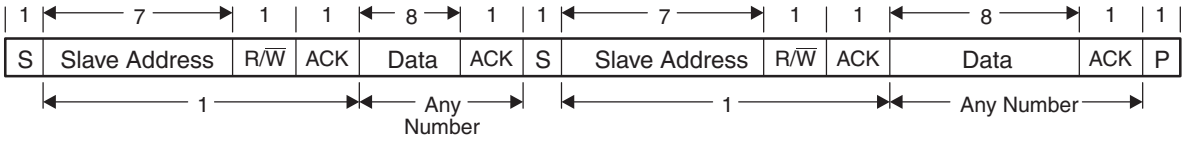


Figure 22-7. I²C Module Addressing Format With Repeated START Condition

22.3.4 I²C Quick Setup

This section gives a quick introduction into the operation of the eUSCI_B in I2C mode. The basic steps to start communication are described and shown as a software example. More detailed information about the possible configurations and details can be found in Section 22.3.5.

The latest code examples can be found on the MSP430 web under "Code Examples".

To set up the eUSCI_B as a master transmitter that transmits to a slave with the address 0x12h, only a few steps are needed (see Example 22-1).

Example 22-1. Master TX With 7-Bit Address

```
UCBxCTL1 |= UCSWRST;           // put eUSCI_B in reset state
UCBxCTLW0 |= UCMODE_3 + UCMST; // I2C master mode
UCBxBRW = 0x0008;              // baudrate = SMCLK / 8
UCBxCTLW1 = UCASTP_2;          // autom. STOP assertion
UCBxTBCNT = 0x07;              // TX 7 bytes of data
UCBxI2CSA = 0x0012;            // address slave is 12hex
P2SEL |= 0x03;                 // configure I2C pins (device specific)
UCBxCTL1 &= ^UCSWRST;          // eUSCI_B in operational state
UCBxIE |= UCTXIE;              // enable TX-interrupt
GIE;                           // general interrupt enable
...
// inside the eUSCI_B TX interrupt service routine
UCBxTXBUF = 0x77;              // fill TX buffer
```

As shown in the code example, all configurations must be done while UCSWRST is set. To select the I²C operation of the eUSCI_B, UCMODE must be set accordingly. The baudrate of the transmission is set by writing the correct divider in the UCBxBRW register. The default clock selected is SMCLK. How many bytes are transmitted in one frame is controlled by the byte counter threshold register UCBxTBCNT together with the UCASTPx bits.

The slave address to send to is specified in the UCBxI2CSA register. Finally, the ports must be configured. This step is device dependent; see the data sheet for the pins that must be used.

Each byte that is to be transmitted must be written to the UCBxTXBUF inside the interrupt service routine. The recommended structure of the interrupt service routine can be found in Example 22-3.

Example 22-2 shows the steps needed to set up the eUSCI_B as a slave with the address 0x12h that is able to receive and transmit data to the master.

Example 22-2. Slave RX With 7-Bit Address

```
UCBxCTL1 |= UCSWRST;         // eUSCI_B in reset state
UCBxCTLW0 |= UCMODE_3;       // I2C slave mode
UCBxI2COA0 = 0x0012;         // own address is 12hex
P2SEL |= 0x03;               // configure I2C pins (device specific)
UCBxCTL1 &= ^UCSWRST;        // eUSCI_B in operational state
UCBxIE |= UCTXIE + UCRXIE;   // enable TX&RX-interrupt
GIE;                         // general interrupt enable
...
// inside the eUSCI_B TX interrupt service routine
UCBxTXBUF = 0x77;            // send 077h
...
// inside the eUSCI_B RX interrupt service routine
data = UCBxRXBUF;            // data is the internal variable
```

As shown in Example 22-2, all configurations must be done while UCSWRST is set. For the slave, I²C operation is selected by setting UCMODE. The slave address is specified in the UCBxI2COA0 register. To enable the interrupts for receive and transmit requests, the according bits in UCBxIE and, at the end, GIE need to be set. Finally the ports must be configured. This step is device dependent; see the data sheet for the pins that are used.

The RX interrupt service routine is called for every byte received by a master device. The TX interrupt service routine is executed each time the master requests a byte. The recommended structure of the interrupt service routine can be found in Example 22-3.

22.3.5 I²C Module Operating Modes

In I²C mode, the eUSCI_B 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 22-8 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 eUSCI_B module, either as master or slave, is shown by rectangles that are taller than the others.

Actions taken by the eUSCI_B module are shown in grey rectangles with an arrow indicating where in 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.

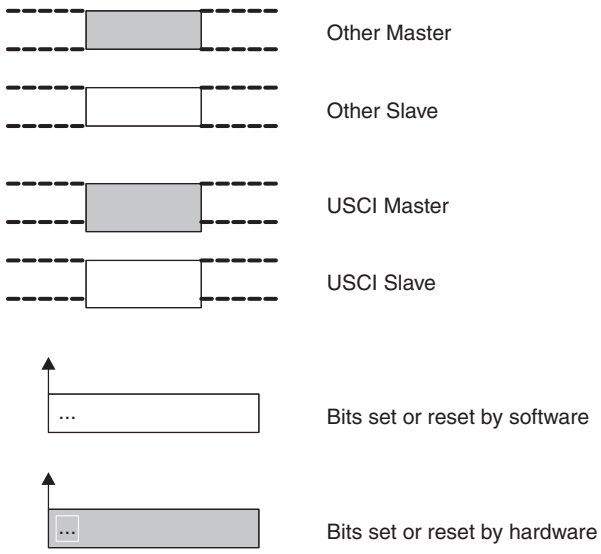


Figure 22-8. I²C Time-Line Legend

22.3.5.1 Slave Mode

The eUSCI_B module is configured as an I²C slave by selecting the I²C mode with UCMODEx = 11 and UCSYNC = 1 and clearing the UCMST bit.

Initially, the eUSCI_B module must be configured in receiver mode by clearing the UCTR bit to receive the I²C address. Afterwards, transmit and receive operations are controlled automatically, depending on the R/$\overline{W}$ bit received together with the slave address.

The eUSCI_B slave address is programmed with the UCBxI2COA0 register. Support for multiple slave addresses is explained in Section 22.3.9. 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 eUSCI_B module receives the transmitted address and compares it against its own address stored in UCBxI2COA0. The UCSTTIFG flag is set when address received matches the eUSCI_B slave address.

22.3.5.1.1 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 eUSCI_B module is automatically configured as a transmitter and UCTR and UCTXIFG0 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 and the data is transmitted. As soon as the data is transferred into the shift register, the UCTXIFG0 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 followed by a STOP condition, the UCSTPIFG flag is set. If the NACK is followed by a repeated START condition, the eUSCI_B I²C state machine returns to its address-reception state.

Figure 22-9 shows the slave transmitter operation.

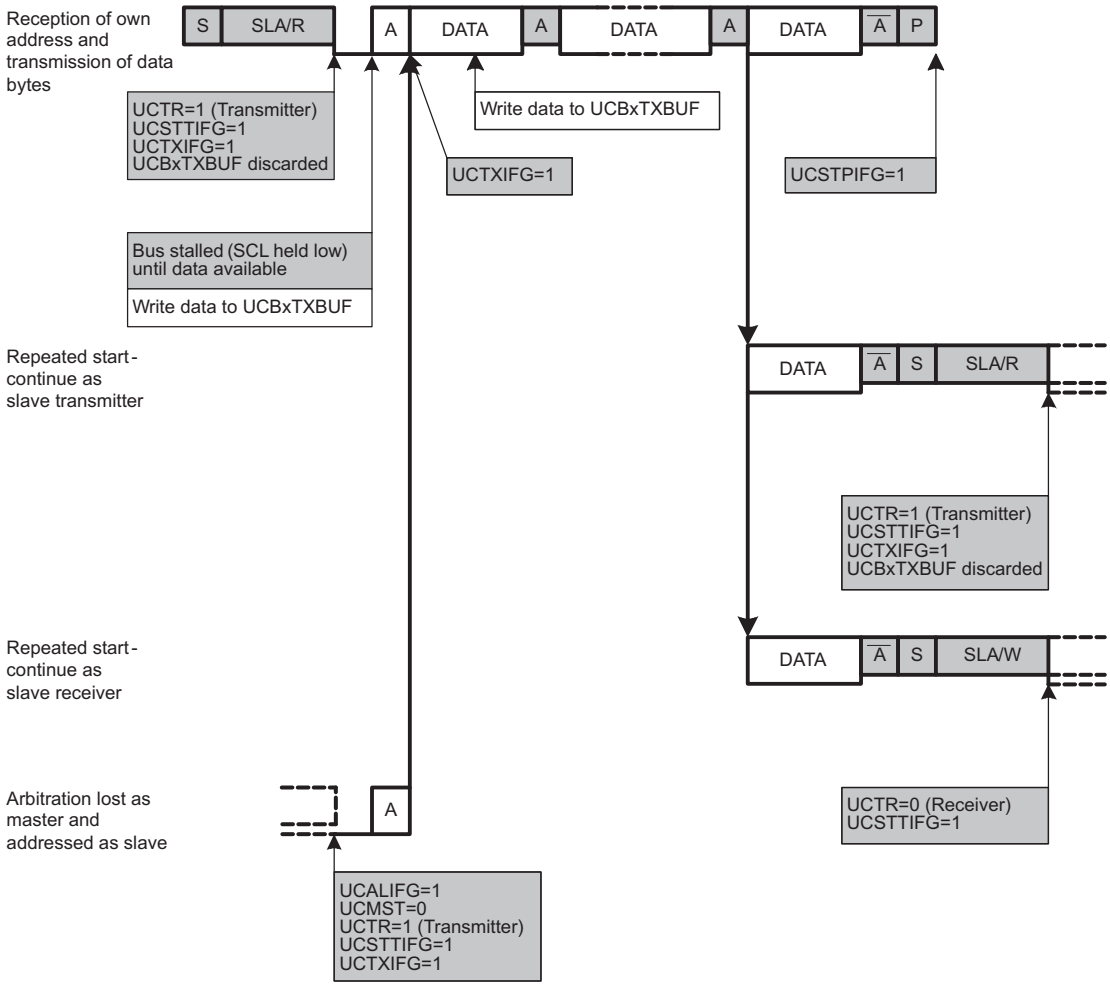


Figure 22-9. I²C Slave Transmitter Mode

22.3.5.1.2 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 receives data from the master, the eUSCI_B module is automatically configured as a receiver and UCTR is cleared. After the first data byte is received, the receive interrupt flag UCRXIFG0 is set. The eUSCI_B 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 eUSCI_B I²C state machine returns to its address-reception state.

Figure 22-10 shows the I²C slave receiver operation.

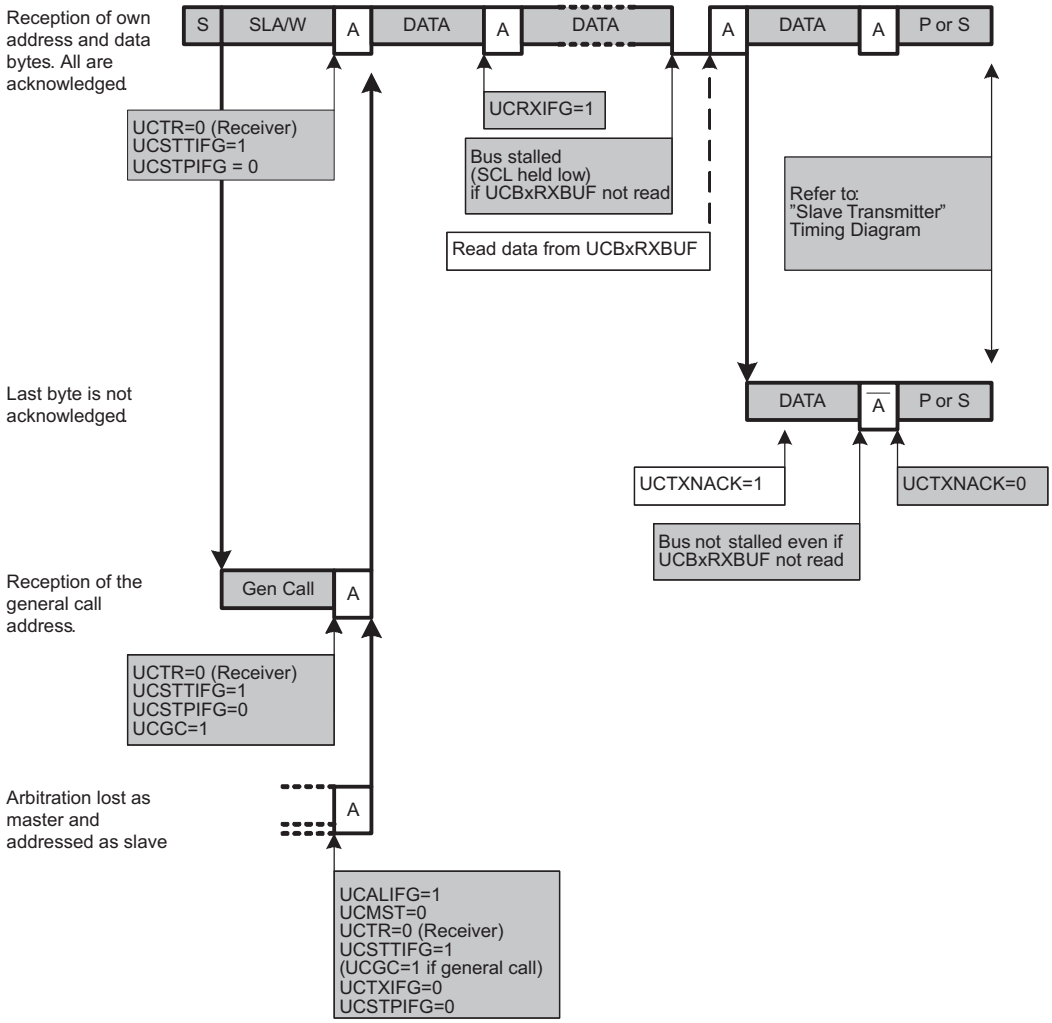


Figure 22-10. I²C Slave Receiver Mode

22.3.5.1.3 I²C Slave 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCA10 = 1 and is as shown in Figure 22-11. In 10-bit addressing mode, the slave is in receive mode after the full address is received. The eUSCI_B 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 eUSCI_B modules switches to transmitter mode with UCTR = 1.

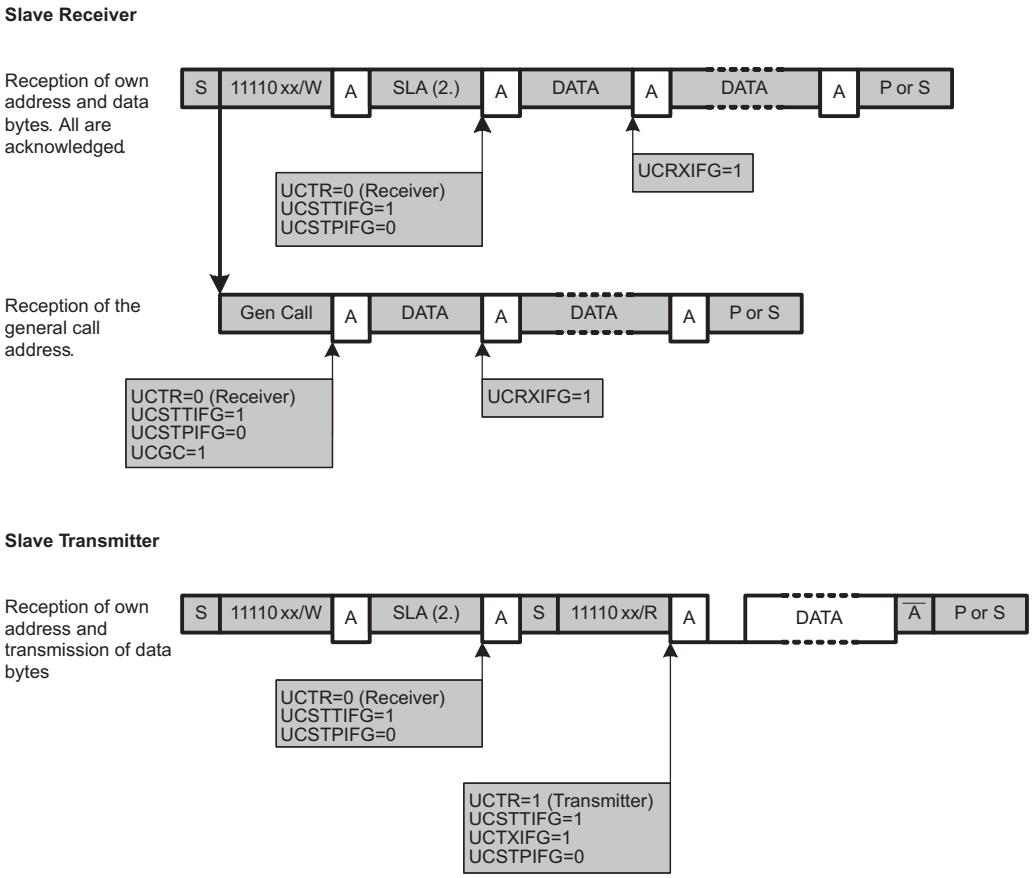


Figure 22-11. I²C Slave 10-Bit Addressing Mode

22.3.5.2 Master Mode

The eUSCI_B 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 UCBxI2COA0 register. Support for multiple slave addresses is explained in Section 22.3.9. When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the eUSCI_B module responds to a general call.

NOTE: Addresses and multi-master systems

In master mode with own-address detection enabled (UCOAEN = 1)—especially in multi-master systems—it is not allowed to specify the same address in the own address and slave address register (UCBxI2CSA = UCBxI2COAx). This would mean that the eUSCI_B addresses itself.

The user software must ensure that this situation does not occur. There is no hardware detection for this case, and the consequence is unpredictable behavior of the eUSCI_B.

22.3.5.2.1 I²C 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 eUSCI_B module waits until the bus is available, then generates the START condition, and transmits the slave address. The UCTXIFG0 bit is set when the START condition is generated and the first data to be transmitted can be written into UCBxTXBUF. **The UCTXSTT flag is cleared as soon as the complete address is sent.**

The data written into UCBxTXBUF is transmitted if arbitration is not lost during transmission of the slave address. UCTXIFG0 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:

- No automatic STOP is generated
- The UCTXSTP bit is not set
- The 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 address or while the eUSCI_B module waits for data to be written into UCBxTXBUF, a STOP condition is generated, even if no data was transmitted to the slave. **In this case, the UCSTPIFG is set.** When transmitting a single byte of data, the UCTXSTP bit must be set while the byte is being transmitted or any time 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, UCTXIFG0 is set, indicating data transmission has begun, and the UCTXSTP bit may be set. When UCASTPx = 10 is set, the byte counter is used for STOP generation and the user does not need to set the UCTXSTP. **This is recommended when transmitting only one byte.**

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 or UCTXSTP is also discarded.

Figure 22-12 shows the I²C master transmitter operation.

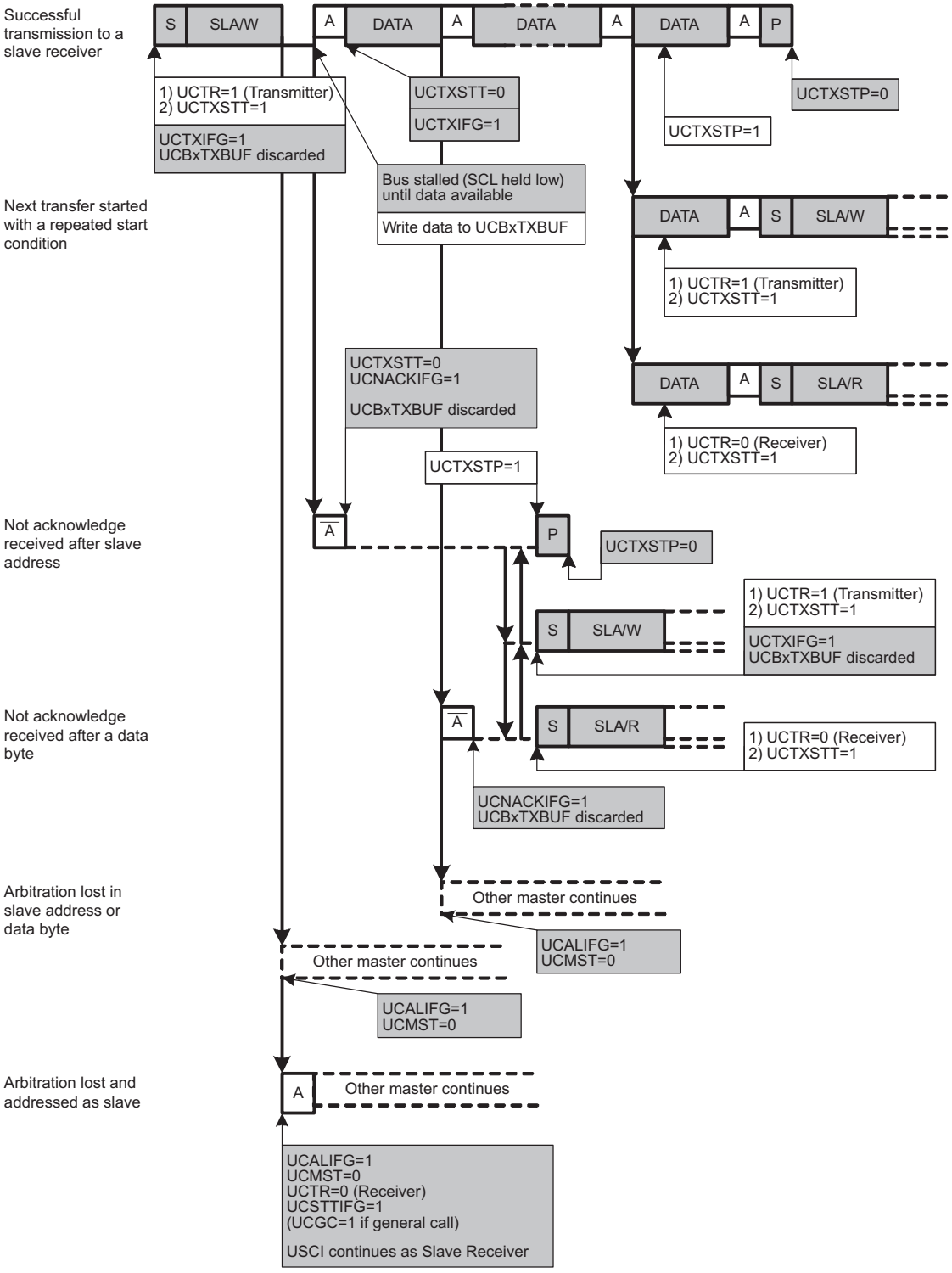


Figure 22-12. I²C Master Transmitter Mode

22.3.5.2.2 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 eUSCI_B module checks if the bus is available, generates the START condition, and transmits the slave address. The UCTXSTT flag is cleared as soon as the complete address is sent.

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:

- No automatic STOP is generated
- The UCTXSTP bit is not set
- The UCTXSTT bit is not set

If a STOP condition was generated by the eUSCI_B module, the UCSTPIFG is 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.

A STOP condition is either generated by the automatic STOP generation or by setting the UCTXSTP bit. The next byte received from the slave is followed by a NACK and a STOP condition. This NACK occurs immediately if the eUSCI_B module is currently waiting for UCBxRXBUF to be read.

If a RESTART is sent, UCTR may be set or cleared to configure transmitter or receiver, and a different slave address may be written into UCBxI2CSA if desired.

Figure 22-13 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.

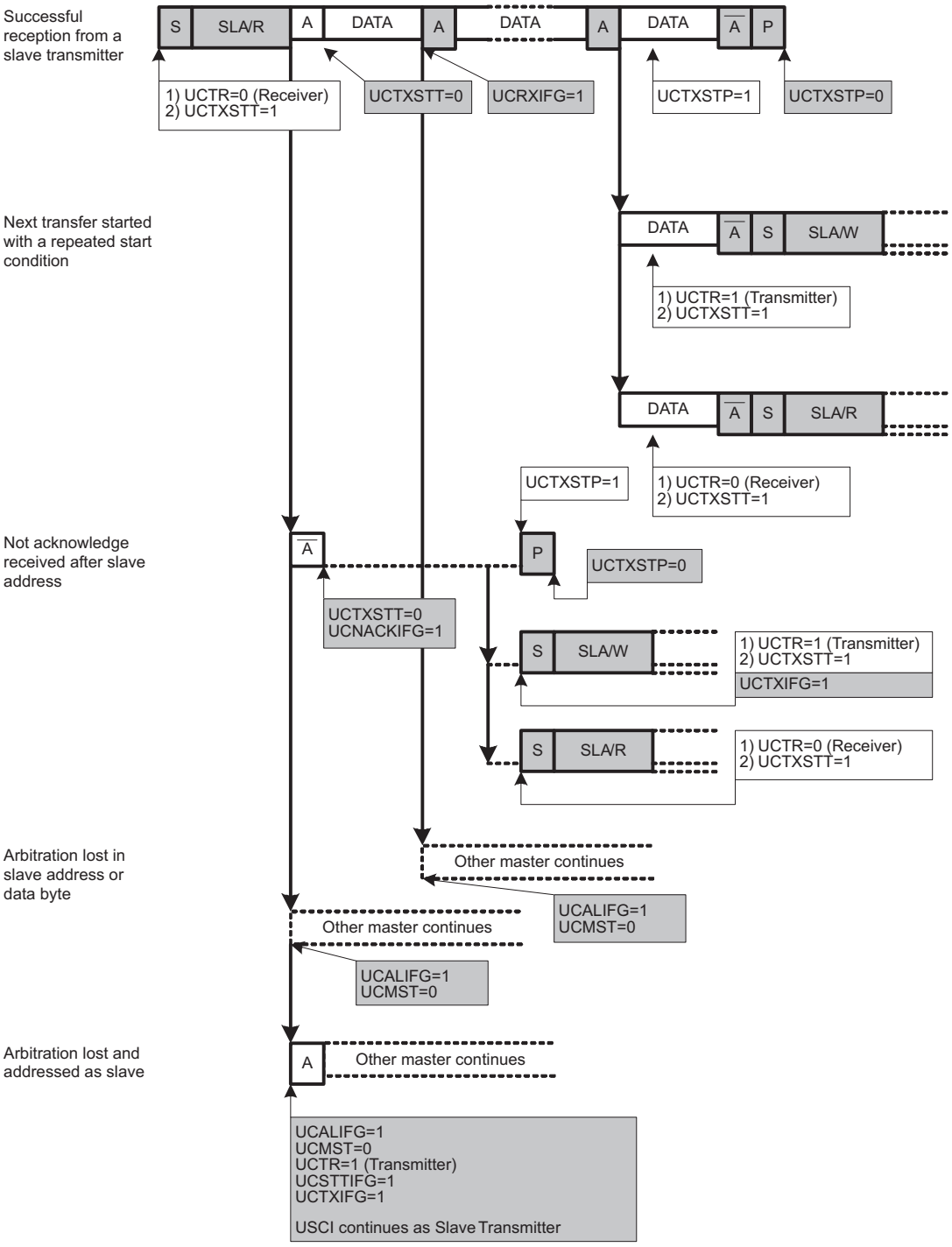


Figure 22-13. I²C Master Receiver Mode

22.3.5.2.3 I²C Master 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCSLA10 = 1 and is shown in Figure 22-14.

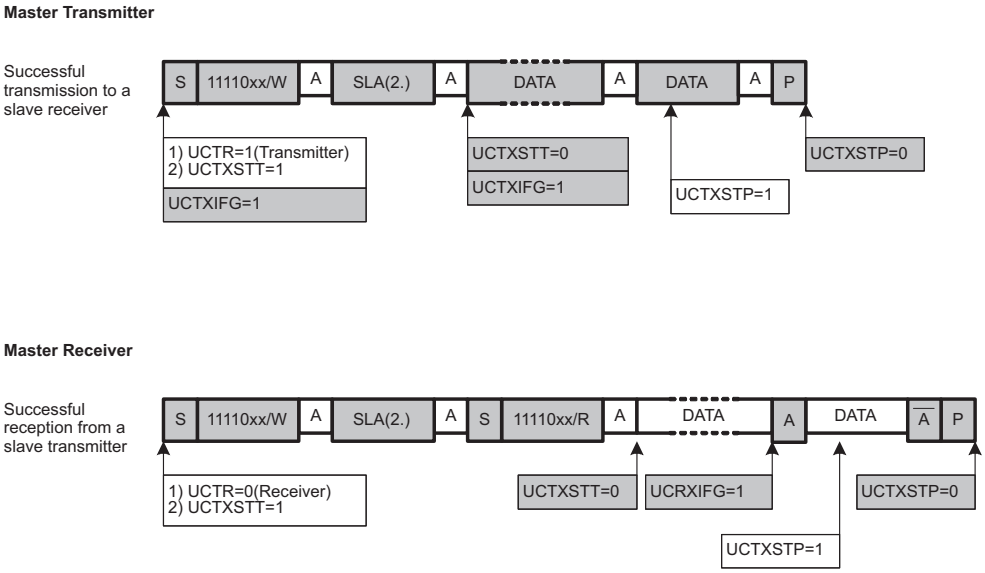


Figure 22-14. I²C Master 10-Bit Addressing Mode

22.3.5.3 Arbitration

If two or more master transmitters simultaneously start a transmission on the bus, an arbitration procedure is invoked. Figure 22-15 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.

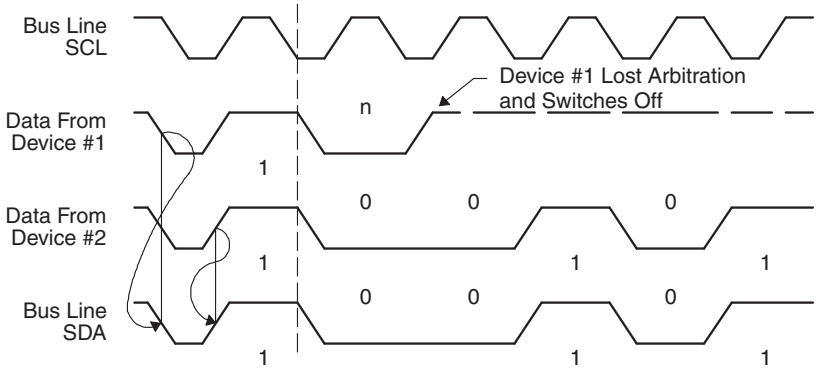


Figure 22-15. Arbitration Procedure Between Two Master Transmitters

There is an undefined condition if the arbitration procedure is still in progress when one master sends a repeated START or a STOP condition while the other master is still sending data. In other words, the following combinations result in an undefined condition:

- Master 1 sends a repeated START condition and master 2 sends a data bit.
- Master 1 sends a STOP condition and master 2 sends a data bit.
- Master 1 sends a repeated START condition and master 2 sends a STOP condition.

22.3.6 Glitch Filtering

According to the I²C standard, both the SDA and the SCL line need to be glitch filtered. The eUSCI_B module provides the UCGLITx bits to configure the length of this glitch filter:

Table 22-1. Glitch Filter Length Selection Bits

UCGLITx	Corresponding Glitch Filter Length on SDA and SCL	According to I²C Standard
00	Pulses of max 50-ns length are filtered	yes
01	Pulses of max 25-ns length are filtered.	no
10	Pulses of max 12.5-ns length are filtered.	no
11	Pulses of max 6.25-ns length are filtered.	no

22.3.7 I²C Clock Generation and Synchronization

The I²C clock SCL is provided by the master on the I²C bus. When the eUSCI_B is in master mode, BITCLK is provided by the eUSCI_B 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 eUSCI_B 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 eUSCI_B 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 22-16 shows the clock synchronization. This allows a slow slave to slow down a fast master.

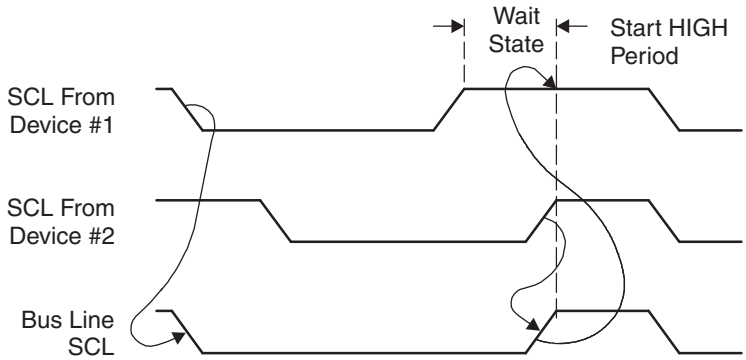


Figure 22-16. Synchronization of Two I²C Clock Generators During Arbitration

22.3.7.1 Clock Stretching

The eUSCI_B 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 eUSCI_B module already released SCL due to the following conditions:

- eUSCI_B is acting as master and a connected slave drives SCL low.

- eUSCI_B is acting as master and another master drives SCL low during arbitration.

The UCSCLLOW bit is also active if the eUSCI_B 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 be 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.

22.3.7.2 Avoiding Clock Stretching

Even though clock stretching is part of the I2C specification, there are applications in which clock stretching should be avoided.

The clock is stretched by the eUSCI_B under the following conditions:

- The internal shift register is expecting data, but the TXIFG is still pending
- The internal shift register is full, but the RXIFG is still pending
- The arbitration lost interrupt is pending
- UCSWACK is selected and UCBxI2COA0 did cause a match

To avoid clock stretching, all of these situations for clock stretch either need to be avoided or the corresponding interrupt flags need to be processed before the actual clock stretch can occur.

Using the DMA (on devices that contain a DMA) is the most secure way to avoid clock stretching. If no DMA is available, the software must ensure that the corresponding interrupts are serviced in time before the clock is stretched.

In slave transmitter mode, the TXIFG is set only after the reception of the direction bit; therefore, there is only a short amount of time for the software to write the TXBUF before a clock stretch occurs. This situation can be remedied by using the early Transmit Interrupt (see Section 22.3.11.2).

22.3.7.3 Clock Low Timeout

The UCCLTOIFG interrupt allows the software to react if the clock is low longer than a defined time. It is possible to detect the situation, when a clock is stretched by a master or slave for a too long time. The user can then, for example, reset the eUSCI_B module by using the UCSWRST bit.

The clock low timeout feature is enabled using the UCCLTO bits. It is possible to select one of three predefined times for the clock low timeout. If the clock has been low longer than the time defined with the UCCLTO bits and the eUSCI_B was actively receiving or transmitting, the UCCLTOIFG is set and an interrupt request is generated if UCCLTOIE and GIE are set as well. The UCCLTOIFG is set only once, even if the clock is stretched a multiple of the time defined in UCCLTO.

22.3.8 Byte Counter

The eUSCI_B module supports hardware counting of the bytes received or transmitted. The counter is automatically active and counts up for each byte seen on the bus in both master and slave mode.

The byte counter is incremented at the second bit position of each byte independently of the following ACK or NACK. A START or RESTART condition resets the counter value to zero. Address bytes do not increment the counter. The byte counter is also incremented at the second bit position, if an arbitration lost occurs during the first bit of data.

22.3.8.1 Byte Counter Interrupt

If UCASTPx = 01 or 10 the UCBCNTIFG is set when the byte counter threshold value UCBxTBCNT is reached in both master- and slave-mode. Writing zero to UCBxTBCNT does not generate an interrupt.

Because the UCBCNTIFG has a lower interrupt priority than the UCBTXIFG and UCBRXIFG, it is recommended to only use it for protocol control together with the DMA handling the received and transmitted bytes. Otherwise the application must have enough processor bandwidth to ensure that the UCBCNT interrupt routine is executed in time to generate for example a RESTART.

22.3.8.2 Automatic STOP Generation

When the eUSCI_B module is configured as a master, the byte counter can be used for automatic STOP generation by setting the UCASTPx = 10. Before starting the transmission using UCTXSTT, the byte counter threshold UCBxTBCNT must be set to the number of bytes that are to be transmitted or received. After the number of bytes that are configured in UCBxTBCNT have been transmitted, the eUSCI_B automatically generates a STOP condition.

UCBxTBCNT cannot be used if the user wants to transmit the slave address only without any data. In this case, it is recommended to set UCTXSTT and UCTXSTP at the same time.

22.3.9 Multiple Slave Addresses

The eUSCI_B module supports two different ways of implementing multiple slave addresses at the same time:

- Hardware support for up to 4 different slave addresses, each with its own interrupt flag and DMA trigger
- Software support for up to 2^{10} different slave addresses all sharing one interrupt

22.3.9.1 Multiple Slave Address Registers

The registers UCBxI2COA0, UCBxI2COA1, UCBxI2COA2, and UCBxI2COA3 contain four slave addresses. Up to four address registers are compared against a received 7- or 10-bit address. Each slave address must be activated by setting the UCAOEN bit in the corresponding UCBxI2COAx register. Register UCBxI2COA3 has the highest priority if the address received on the bus matches more than one of the slave address registers. The priority decreases with the index number of the address register, so that UCBxI2COA0 in combination with the address mask has the lowest priority.

When one of the slave registers matches the 7- or 10-bit address seen on the bus, the address is acknowledged. In the following the corresponding receive- or transmit-interrupt flag (UCTXIFGx or UCRXIFGx) to the received address is updated. The state change interrupt flags are independent of the address comparison result. They are updated according to the bus condition.

22.3.9.2 Address Mask Register

The address mask register can be used when the eUSCI_B is configured in slave or in multiple-master mode. To activate this feature, at least one bit of the address mask in register UCBxADDMASK must be cleared.

If the received address matches the own address in UCBxI2COA0 on all bit positions that are not masked by UCBxADDMASK, the eUSCI_B module considers the received address as its own address. If UCSWACK = 0, the module sends an acknowledge automatically. If UCSWACK = 1, the user software must evaluate the received address in register UCBxADDRX after the UCSTTIFG is set. To acknowledge the received address, the software must set UCTXACK to 1.

The eUSCI_B module also automatically acknowledges a slave address that is seen on the bus if the address matches any of the enabled slave addresses defined in UCBxI2COA1 to UCBxI2COA3.

NOTE: UCSWACK and slave-transmitter

If the user selects manual acknowledge of slave addresses, TXIFG is set if the slave is addressed as a transmitter. If the software decides not to acknowledge the address, TXIFG0 must be reset.

22.3.10 Using the eUSCI_B Module in I²C Mode With Low-Power Modes

The eUSCI_B module provides automatic clock activation for use with low-power modes. When the eUSCI_B clock source is inactive because the device is in a low-power mode, the eUSCI_B module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the eUSCI_B module returns to its idle condition. After the eUSCI_B module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

In I²C slave mode, no internal clock source is required because the clock is provided by the external master. It is possible to operate the eUSCI_B in I²C 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.

22.3.11 eUSCI_B Interrupts in I²C Mode

The eUSCI_B has only one interrupt vector that is shared for transmission, reception, and the state change.

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 UCTXIFGx and UCRXIFGx flags on devices with a DMA controller. It is possible to react on each slave address with an individual DMA channel.

All interrupt flags are not cleared automatically, but they need to be cleared together by user interactions (for example, reading the UCRXBUF clears UCRXIFGx). If the user wants to use an interrupt flag he needs to ensure that the flag has the correct state before the corresponding interrupt is enabled.

22.3.11.1 I²C Transmit Interrupt Operation

The UCTXIFG0 interrupt flag is set whenever the transmitter is able to accept a new byte. When operating as a slave with multiple slave addresses, the UCTXIFGx flags are set corresponding to which address was received before. If, for example, the slave address specified in register UCBxI2COA3 did match the address seen on the bus, the UCTXIFG3 indicates that the UCBxTXBUF is ready to accept a new byte.

When operating in master mode with automatic STOP generation (UCASTPx = 10), the UCTXIFG0 is set as many times as defined in UCBxTBCNT.

An interrupt request is generated if UCTXIEx and GIE are also set. UCTXIFGx is automatically reset if a write to UCBxTXBUF occurs or if the UCALIFG is cleared. UCTXIFGx is set when:

- Master mode: UCTXSTT was set by the user
- Slave mode: own address was received(UCETXINT = 0) or START was received (UCETXINT = 1)

UCTXIEx is reset after a PUC or when UCSWRST = 1.

22.3.11.2 Early I²C Transmit Interrupt

Setting the UCETXINT causes UCTXIFG0 to be sent out automatically when a START condition is sent and the eUSCI_B is configured as slave. In this case, it is not allowed to enable the other slave addresses UCBxI2COA1-UCBxI2COA3. This allows the software more time to handle the UCTXIFG0 compared to the normal situation, when UCTXIFG0 is sent out after the slave address match was detected. Situations where the UCTXIFG0 was set and afterward no slave address match occurred need to be handled in software. The use of the byte counter is recommended to handle this.

22.3.11.3 I²C Receive Interrupt Operation

The UCRXIFG0 interrupt flag is set when a character is received and loaded into UCBxRXBUF. When operating as a slave with multiple slave addresses, the UCRXIFGx flag is set corresponding to which address was received before.

An interrupt request is generated if UCRXIEx and GIE are also set. UCRXIFGx and UCRXIEx are reset after a PUC signal or when UCSWRST = 1. UCRXIFGx is automatically reset when UCxRXBUF is read.

22.3.11.4 I²C State Change Interrupt Operation

Table 22-2 describes the I²C state change interrupt flags.

Table 22-2. I²C State Change Interrupt Flags

Interrupt Flag	Interrupt Condition
UCALIFG	Arbitration lost interrupt. Arbitration can be lost when two or more transmitters start a transmission simultaneously, or when the eUSCI_B 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 used in master mode only.
UCCLTOIFG	Clock low timeout. This interrupt flag is set, if the clock is held low longer than defined by the UCCLTO bits.
UCBIT9IFG	This interrupt flag is generated each time the eUSCI_B is transferring the nineth clock cycle of a byte of data. This gives the user the ability to follow the I²C communication in software if wanted. UCBIT9IFG is not set for address information.
UCBCNTIFG	Byte counter interrupt. This flag is set when the byte counter value reaches the value defined in UCBxTBCNT and UCASTPx = 01 or 10. This bit allows to organize following communications, especially if a RESTART will be issued.
UCSTTIFG	START condition detected interrupt. This flag is set when the I²C module detects a START condition together with its own address(1). UCSTTIFG is used in slave mode only.
UCSTPIFG	STOP condition detected interrupt. This flag is set when the I²C module detects a STOP condition on the bus. UCSTPIFG is used in slave and master mode.

(1) The address evaluation includes the address mask register if it is used.

22.3.11.5 UCBxIV, Interrupt Vector Generator

The eUSCI_B 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.

Read access 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.

Write access of the UCBxIV register clears all pending Interrupt conditions and flags.

Example 22-3 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 eUSCI0_B.

Example 22-3. UCBxIV Software Example

```
#pragma vector = USCI_B0_VECTOR __interrupt void USCI_B0_ISR(void) {
    switch(__even_in_range(UCB0IV,0x1e))    {
        case 0x00:     // Vector 0: No interrupts
                   break;
        case 0x02: ... // Vector 2: ALIFG
                   break;
        case 0x04: ... // Vector 4: NACKIFG
                   break;
        case 0x06: ... // Vector 6: STTIFG
                   break;
        case 0x08: ... // Vector 8: STPIFG
                   break;
        case 0x0a: ... // Vector 10: RXIFG3
                   break;
        case 0x0c: ... // Vector 12: TXIFG3
                   break;
        case 0x0e: ... // Vector 14: RXIFG2
                   break;
        case 0x10: ... // Vector 16: TXIFG2
                   break;
        case 0x12: ... // Vector 18: RXIFG1
                   break;
        case 0x14: ... // Vector 20: TXIFG1
                   break;
        case 0x16: ... // Vector 22: RXIFG0
                   break;
        case 0x18: ... // Vector 24: TXIFG0
                   break;
        case 0x1a: ... // Vector 26: BCNTIFG
                   break;
        case 0x1c: ... // Vector 28: clock low timeout
                   break;
        case 0x1e: ... // Vector 30: 9th bit
                   break;
        default:   break;
    }
}
```

22.4 eUSCI_B I2C Registers

The eUSCI_B registers applicable in I2C mode and their address offsets are listed in Table 22-3. The base address can be found in the device-specific data sheet.

Table 22-3. eUSCI_B Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCBxCTLW0	eUSCI_Bx Control Word 0	Read/write	Word	01C1h	Section 22.4.1
00h	UCBxCTL1	eUSCI_Bx Control 1	Read/write	Byte	C1h	
01h	UCBxCTL0	eUSCI_Bx Control 0	Read/write	Byte	01h	
02h	UCBxCTLW1	eUSCI_Bx Control Word 1	Read/write	Word	0000h	Section 22.4.2
06h	UCBxBRW	eUSCI_Bx Bit Rate Control Word	Read/write	Word	0000h	Section 22.4.3
06h	UCBxBR0	eUSCI_Bx Bit Rate Control 0	Read/write	Byte	00h	
07h	UCBxBR1	eUSCI_Bx Bit Rate Control 1	Read/write	Byte	00h	
08h	UCBxSTATW	eUSCI_Bx Status Word	Read	Word	0000h	Section 22.4.4
08h	UCBxSTAT	eUSCI_Bx Status	Read	Byte	00h	
09h	UCBxBCNT	eUSCI_Bx Byte Counter Register	Read	Byte	00h	
0Ah	UCBxTBCNT	eUSCI_Bx Byte Counter Threshold Register	Read/Write	Word	00h	Section 22.4.5
0Ch	UCBxRXBUF	eUSCI_Bx Receive Buffer	Read/write	Word	00h	Section 22.4.6
0Eh	UCBxTXBUF	eUSCI_Bx Transmit Buffer	Read/write	Word	00h	Section 22.4.7
14h	UCBxI2COA0	eUSCI_Bx I2C Own Address 0	Read/write	Word	0000h	Section 22.4.8
16h	UCBxI2COA1	eUSCI_Bx I2C Own Address 1	Read/write	Word	0000h	Section 22.4.9
18h	UCBxI2COA2	eUSCI_Bx I2C Own Address 2	Read/write	Word	0000h	Section 22.4.10
1Ah	UCBxI2COA3	eUSCI_Bx I2C Own Address 3	Read/write	Word	0000h	Section 22.4.11
1Ch	UCBxADDRX	eUSCI_Bx Received Address Register	Read	Word		Section 22.4.12
1Eh	UCBxADDMASK	eUSCI_Bx Address Mask Register	Read/write	Word	03FFh	Section 22.4.13
20h	UCBxI2CSA	eUSCI_Bx I2C Slave Address	Read/write	Word	0000h	Section 22.4.14
2Ah	UCBxIE	eUSCI_Bx Interrupt Enable	Read/write	Word	0000h	Section 22.4.15
2Ch	UCBxIFG	eUSCI_Bx Interrupt Flag	Read/write	Word	0002h	Section 22.4.16
2Eh	UCBxIV	eUSCI_Bx Interrupt Vector	Read	Word	0000h	Section 22.4.17

22.4.1 UCBxCTLW0 Register

eUSCI_Bx Control Word Register 0

Figure 22-17. UCBxCTLW0 Register

15	14	13	12	11	10	9	8
UCA10	UCSLA10	UCMM	Reserved	UCMST	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0	r1
7	6	5	4	3	2	1	0
UCSSELx		UCTXACK	UCTR	UCTXNACK	UCTXSTP	UCTXSTT	UCSWRST
rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

Modify only when UCSWRST = 1.

Table 22-4. UCBxCTLW0 Register Description

Bit	Field	Type	Reset	Description
15	UCA10	RW	0h	Own addressing mode select. Modify only when UCSWRST = 1. 0b = Own address is a 7-bit address. 1b = Own address is a 10-bit address.
14	UCSLA10	RW	0h	Slave addressing mode select 0b = Address slave with 7-bit address 1b = Address slave with 10-bit address
13	UCMM	RW	0h	Multi-master environment select. Modify only when UCSWRST = 1. 0b = Single master environment. There is no other master in the system. The address compare unit is disabled. 1b = Multi-master environment
12	Reserved	R	0h	Reserved
11	UCMST	RW	0h	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. 0b = Slave mode 1b = Master mode
10-9	UCMODEx	RW	0h	eUSCI_B mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. Modify only when UCSWRST = 1. 00b = 3-pin SPI 01b = 4-pin SPI (master or slave enabled if STE = 1) 10b = 4-pin SPI (master or slave enabled if STE = 0) 11b = I2C mode
8	UCSYNC	RW	1h	Synchronous mode enable. For eUSCI_B always read and write as 1.
7-6	UCSSELx	RW	3h	eUSCI_B clock source select. These bits select the BRCLK source clock. These bits are ignored in slave mode. Modify only when UCSWRST = 1. 00b = UCLKI 01b = ACLK 10b = SMCLK 11b = SMCLK
5	UCTXACK	RW	0h	Transmit ACK condition in slave mode with enabled address mask register. After the UCSTTIFG has been set, the user needs to set or reset the UCTXACK flag to continue with the I2C protocol. The clock is stretched until the UCBxCTL1 register has been written. This bit is cleared automatically after the ACK has been send. 0b = Do not acknowledge the slave address 1b = Acknowledge the slave address

Table 22-4. UCBxCTLW0 Register Description (continued)

Bit	Field	Type	Reset	Description
4	UCTR	RW	0h	Transmitter/receiver 0b = Receiver 1b = Transmitter
3	UCTXNACK	RW	0h	Transmit a NACK. UCTXNACK is automatically cleared after a NACK is transmitted. Only for slave receiver mode. 0b = Acknowledge normally 1b = Generate NACK
2	UCTXSTP	RW	0h	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. This bit is a don't care, if automatic UCASTPx is different from 01 or 10. 0b = No STOP generated 1b = Generate STOP
1	UCTXSTT	RW	0h	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. 0b = Do not generate START condition 1b = Generate START condition
0	UCSWRST	RW	1h	Software reset enable. Modify only when UCSWRST = 1. 0b = Disabled. eUSCI_B released for operation. 1b = Enabled. eUSCI_B logic held in reset state.

22.4.2 UCBxCTLW1 Register

eUSCI_Bx Control Word Register 1

Figure 22-18. UCBxCTLW1 Register

15	14	13	12	11	10	9	8
Reserved							UCETXINT
r0	r0	r0	r0	r0	r0	r0	rw-0
7	6	5	4	3	2	1	0
UCCLTO		UCSTPNACK	UCSWACK	UCASTPx		UCGLITx	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Modify only when UCSWRST = 1.

Table 22-5. UCBxCTLW1 Register Description

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0h	Reserved
8	UCETXINT	RW	0h	Early UCTXIFG0. Only in slave mode. When this bit is set, the slave addresses defined in UCxI2COA1 to UCxI2COA3 must be disabled. Modify only when UCSWRST = 1. 0b = UCTXIFGx is set after an address match with UCxI2COAx and the direction bit indicating slave transmit 1b = UCTXIFG0 is set for each START condition
7-6	UCCLTO	RW	0h	Clock low timeout select. Modify only when UCSWRST = 1. 00b = Disable clock low timeout counter 01b = 135 000 MODCLK cycles (approximately 28 ms) 10b = 150 000 MODCLK cycles (approximately 31 ms) 11b = 165 000 MODCLK cycles (approximately 34 ms)
5	UCSTPNACK	RW	0h	The UCSTPNACK bit allows to make the eUSCI_B master acknowledge the last byte in master receiver mode as well. This is not conform to the I2C specification and should only be used for slaves, which automatically release the SDA after a fixed packet length. Modify only when UCSWRST = 1. 0b = Send a non-acknowledge before the STOP condition as a master receiver (conform to I2C standard) 1b = All bytes are acknowledged by the eUSCI_B when configured as master receiver
4	UCSWACK	RW	0h	Using this bit it is possible to select, whether the eUSCI_B module triggers the sending of the ACK of the address or if it is controlled by software. 0b = The address acknowledge of the slave is controlled by the eUSCI_B module 1b = The user needs to trigger the sending of the address ACK by issuing UCTXACK
3-2	UCASTPx	RW	0h	Automatic STOP condition generation. In slave mode only UCBCNTIFG is available. Modify only when UCSWRST = 1. 00b = No automatic STOP generation. The STOP condition is generated after the user sets the UCTXSTP bit. The value in UCBxTBCNT is a don't care. 01b = UCBCNTIFG is set with the byte counter reaches the threshold defined in UCBxTBCNT 10b = A STOP condition is generated automatically after the byte counter value reached UCBxTBCNT. UCBCNTIFG is set with the byte counter reaching the threshold. 11b = Reserved

Table 22-5. UCBxCTLW1 Register Description (continued)

Bit	Field	Type	Reset	Description
1-0	UCGLITx	RW	0h	Deglitch time 00b = 50 ns 01b = 25 ns 10b = 12.5 ns 11b = 6.25 ns

22.4.3 UCBxBRW Register

eUSCI_Bx Bit Rate Control Word Register

Figure 22-19. UCBxBRW Register

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

Modify only when UCSWRST = 1.

Table 22-6. UCBxBRW Register Description

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Bit clock prescaler. Modify only when UCSWRST = 1.

22.4.4 UCBxSTATW

eUSCI_Bx Status Word Register

Figure 22-20. UCBxSTATW Register

15	14	13	12	11	10	9	8
UCBCNTx							
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	UCSCLLOW	UCGC	UCBBUSY	Reserved			
r0	r-0	r-0	r-0	r-0	r0	r0	r0

Table 22-7. UCBxSTATW Register Description

Bit	Field	Type	Reset	Description
15-8	UCBCNTx	R	0h	Hardware byte counter value. Reading this register returns the number of bytes received or transmitted on the I2C-Bus since the last START or RESTART. There is no synchronization of this register done. When reading UCBxBCNT during the first bit position, a faulty readback can occur.
7	Reserved	R	0h	Reserved
6	UCSCLLOW	R	0h	SCL low 0b = SCL is not held low 1b = SCL is held low
5	UCGC	R	0h	General call address received. UCGC is automatically cleared when a START condition is received. 0b = No general call address received 1b = General call address received
4	UCBBUSY	R	0h	Bus busy 0b = Bus inactive 1b = Bus busy
3-0	Reserved	R	0h	Reserved

22.4.5 UCBxTBCNT Register

eUSCI_Bx Byte Counter Threshold Register

Figure 22-21. UCBxTBCNT 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
UCTBCNTx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Modify only when UCSWRST = 1.

Table 22-8. UCBxTBCNT Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCTBCNTx	RW	0h	The byte counter threshold value is used to set the number of I2C data bytes after which the automatic STOP or the UCSTPIFG should occur. This value is evaluated only if UCASTPx is different from 00. Modify only when UCSWRST = 1.

22.4.6 UCBxRXBUF Register

eUSCI_Bx Receive Buffer Register

Figure 22-22. UCBxRXBUF 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
UCRXBUFx							
r	r	r	r	r	r	r	r

Table 22-9. UCBxRXBUF Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCRXBUFx	R	0h	The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCBxRXBUF resets the UCRXIFGx flags.

22.4.7 UCBxTXBUF

eUSCI_Bx Transmit Buffer Register

Figure 22-23. UCBxTXBUF 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
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

Table 22-10. UCBxTXBUF Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved
7-0	UCTXBUFx	RW	0h	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 the UCTXIFGx flags.

22.4.8 UCBxI2COA0 Register

eUSCI_Bx I2C Own Address 0 Register

Figure 22-24. UCBxI2COA0 Register

15	14	13	12	11	10	9	8
UCGCEN	Reserved				UCOAEN	I2COA0	
rw-0	r0	r0	r0	r0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
I2COA0							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Modify only when UCSWRST = 1.

Table 22-11. UCBxI2COA0 Register Description

Bit	Field	Type	Reset	Description
15	UCGCEN	RW	0h	General call response enable. This bit is only available in UCBxI2COA0. Modify only when UCSWRST = 1. 0b = Do not respond to a general call 1b = Respond to a general call
14-11	Reserved	R	0h	Reserved
10	UCOAEN	RW	0h	Own Address enable register. With this register it can be selected if the I2C slave-address related to this register UCBxI2COA0 is evaluated or not. Modify only when UCSWRST = 1. 0b = The slave address defined in I2COA0 is disabled 1b = The slave address defined in I2COA0 is enabled
9-0	I2COAx	RW	0h	I2C own address. The I2COA0 bits contain the local address of the eUSCIx_B 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. Modify only when UCSWRST = 1.

22.4.9 UCBxI2COA1 Register

eUSCI_Bx I2C Own Address 1 Register

Figure 22-25. UCBxI2COA1 Register

15	14	13	12	11	10	9	8
Reserved					UCOAEN	I2COA1	
rw-0	r0	r0	r0	r0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
I2COA1							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Modify only when UCSWRST = 1.

Table 22-12. UCBxI2COA1 Register Description

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved
10	UCOAEN	RW	0h	Own Address enable register. With this register it can be selected if the I2C slave-address related to this register UCBxI2COA1 is evaluated or not. Modify only when UCSWRST = 1. 0b = The slave address defined in I2COA1 is disabled 1b = The slave address defined in I2COA1 is enabled
9-0	I2COA1	RW	0h	I2C own address. The I2COAx bits contain the local address of the eUSCIx_B 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. Modify only when UCSWRST = 1.

22.4.10 UCBxI2COA2 Register

eUSCI_Bx I2C Own Address 2 Register

Figure 22-26. UCBxI2COA2 Register

15	14	13	12	11	10	9	8
Reserved					UCOAEN	I2COA2	
rw-0	r0	r0	r0	r0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
I2COA2							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Modify only when UCSWRST = 1.

Table 22-13. UCBxI2COA2 Register Description

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved
10	UCOAEN	RW	0h	Own Address enable register. With this register it can be selected if the I2C slave-address related to this register UCBxI2COA2 is evaluated or not. Modify only when UCSWRST = 1. 0b = The slave address defined in I2COA2 is disabled 1b = The slave address defined in I2COA2 is enabled
9-0	I2COA2	RW	0h	I2C own address. The I2COAx bits contain the local address of the eUSCIx_B 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. Modify only when UCSWRST = 1.

22.4.11 UCBxI2COA3 Register

eUSCI_Bx I2C Own Address 3 Register

Figure 22-27. UCBxI2COA3 Register

15	14	13	12	11	10	9	8
Reserved					UCOAEN	I2COA3	
rw-0	r0	r0	r0	r0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
I2COA3							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Modify only when UCSWRST = 1.

Table 22-14. UCBxI2COA3 Register Description

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved
10	UCOAEN	RW	0h	Own Address enable register. With this register it can be selected if the I2C slave-address related to this register UCBxI2COA3 is evaluated or not. Modify only when UCSWRST = 1. 0b = The slave address defined in I2COA3 is disabled 1b = The slave address defined in I2COA3 is enabled
9-0	I2COA3	RW	0h	I2C own address. The I2COA3 bits contain the local address of the eUSCIx_B 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. Modify only when UCSWRST = 1.

22.4.12 UCBxADDRX Register

eUSCI_Bx I2C Received Address Register

Figure 22-28. UCBxADDRX Register

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

Table 22-15. UCBxADDRX Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved
9-0	ADDRXx	R	0h	Received Address Register. This register contains the last received slave address on the bus. Using this register and the address mask register it is possible to react on more than one slave address using one eUSCI_B module.

22.4.13 UCBxADDMASK Register

eUSCI_Bx I2C Address Mask Register

Figure 22-29. UCBxADDMASK Register

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

Modify only when UCSWRST = 1.

Table 22-16. UCBxADDMASK Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved
9-0	ADDMASKx	RW	3FFh	Address Mask Register. By clearing the corresponding bit of the own address, this bit is a don't care when comparing the address on the bus to the own address. Using this method, it is possible to react on more than one slave address. When all bits of ADDMASKx are set, the address mask feature is deactivated. Modify only when UCSWRST = 1.

22.4.14 UCBxI2CSA Register

eUSCI_Bx I2C Slave Address Register

Figure 22-30. UCBxI2CSA Register

15	14	13	12	11	10	9	8
Reserved						I2CSAx	
r-0	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

Table 22-17. UCBxI2CSA Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved
9-0	I2CSAx	RW	0h	I2C slave address. The I2CSAx bits contain the slave address of the external device to be addressed by the eUSCIx_B 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.

22.4.15 UCBxIE Register

eUSCI_Bx I2C Interrupt Enable Register

Figure 22-31. UCBxIE Register

15	14	13	12	11	10	9	8
Reserved	UCBIT9IE	UCTXIE3	UCRXIE3	UCTXIE2	UCRXIE2	UCTXIE1	UCRXIE1
r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
UCCLTOIE	UCBCNTIE	UCNACKIE	UCALIE	UCSTPIE	UCSTTIE	UCTXIE0	UCRXIE0
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 22-18. UCBxIE Register Description

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved
14	UCBIT9IE	RW	0h	Bit position 9 interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
13	UCTXIE3	RW	0h	Transmit interrupt enable 3 0b = Interrupt disabled 1b = Interrupt enabled
12	UCRXIE3	RW	0h	Receive interrupt enable 3 0b = Interrupt disabled 1b = Interrupt enabled
11	UCTXIE2	RW	0h	Transmit interrupt enable 2 0b = Interrupt disabled 1b = Interrupt enabled
10	UCRXIE2	RW	0h	Receive interrupt enable 2 0b = Interrupt disabled 1b = Interrupt enabled
9	UCTXIE1	RW	0h	Transmit interrupt enable 1 0b = Interrupt disabled 1b = Interrupt enabled
8	UCRXIE1	RW	0h	Receive interrupt enable 1 0b = Interrupt disabled 1b = Interrupt enabled
7	UCCLTOIE	RW	0h	Clock low timeout interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled
6	UCBCNTIE	RW	0h	Byte counter interrupt enable. 0b = Interrupt disabled 1b = Interrupt enabled
5	UCNACKIE	RW	0h	Not-acknowledge interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
4	UCALIE	RW	0h	Arbitration lost interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
3	UCSTPIE	RW	0h	STOP condition interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

Table 22-18. UCBxIE Register Description (continued)

Bit	Field	Type	Reset	Description
2	UCSTTIE	RW	0h	START condition interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
1	UCTXIE0	RW	0h	Transmit interrupt enable 0 0b = Interrupt disabled 1b = Interrupt enabled
0	UCRXIE0	RW	0h	Receive interrupt enable 0 0b = Interrupt disabled 1b = Interrupt enabled

22.4.16 UCBxIFG Register

eUSCI_Bx I2C Interrupt Flag Register

Figure 22-32. UCBxIFG Register

15	14	13	12	11	10	9	8
Reserved	UCBIT9IFG	UCTXIFG3	UCRXIFG3	UCTXIFG2	UCRXIFG2	UCTXIFG1	UCRXIFG1
r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
UCCLTOIFG	UCBCNTIFG	UCNACKIFG	UCALIFG	UCSTPIFG	UCSTTIFG	UCTXIFG0	UCRXIFG0
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1	rw-0

Table 22-19. UCBxIFG Register Description

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved
14	UCBIT9IFG	RW	0h	Bit position 9 interrupt flag 0b = No interrupt pending 1b = Interrupt pending
13	UCTXIFG3	RW	1h	eUSCI_B transmit interrupt flag 3. UCTXIFG3 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA3 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
12	UCRXIFG3	RW	0h	Receive interrupt flag 2. UCRXIFG2 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA2 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
11	UCTXIFG2	RW	0h	eUSCI_B transmit interrupt flag 2. UCTXIFG2 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA2 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
10	UCRXIFG2	RW	0h	Receive interrupt flag 2. UCRXIFG2 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA2 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
9	UCTXIFG1	RW	1h	eUSCI_B transmit interrupt flag 1. UCTXIFG1 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA1 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
8	UCRXIFG1	RW	0h	Receive interrupt flag 1. UCRXIFG1 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA1 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
7	UCCLTOIFG	RW	0h	Clock low timeout interrupt flag 0b = No interrupt pending 1b = Interrupt pending
6	UCBCNTIFG	RW	0h	Byte counter interrupt flag. When using this interrupt the user needs to ensure enough processing bandwidth (see the Byte Counter Interrupt section). 0b = No interrupt pending 1b = Interrupt pending

Table 22-19. UCBxIFG Register Description (continued)

Bit	Field	Type	Reset	Description
5	UCNACKIFG	RW	0h	Not-acknowledge received interrupt flag. This flag only is updated when operating in master mode. 0b = No interrupt pending 1b = Interrupt pending
4	UCALIFG	RW	0h	Arbitration lost interrupt flag 0b = No interrupt pending 1b = Interrupt pending
3	UCSTPIFG	RW	0h	STOP condition interrupt flag 0b = No interrupt pending 1b = Interrupt pending
2	UCSTTIFG	RW	0h	START condition interrupt flag 0b = No interrupt pending 1b = Interrupt pending
1	UCTXIFG0	RW	0h	eUSCI_B transmit interrupt flag 0. UCTXIFG0 is set when UCBxTXBUF is empty in master mode or in slave mode, if the slave address defined in UCBxI2COA0 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG0	RW	0h	eUSCI_B receive interrupt flag 0. UCRXIFG0 is set when UCBxRXBUF has received a complete character in master mode or in slave mode, if the slave address defined in UCBxI2COA0 was on the bus in the same frame. 0b = No interrupt pending 1b = Interrupt pending

22.4.17 UCBxIV Register

eUSCI_Bx Interrupt Vector Register

Figure 22-33. UCBxIV Register

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

Table 22-20. UCBxIV Register Description

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	eUSCI_B interrupt vector value. It generates an value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending interrupt flags. 00h = No interrupt pending 02h = Interrupt Source: Arbitration lost; Interrupt Flag: UCALIFG; Interrupt Priority: Highest 04h = Interrupt Source: Not acknowledgment; Interrupt Flag: UCNACKIFG 06h = Interrupt Source: Start condition received; Interrupt Flag: UCSTTIFG 08h = Interrupt Source: Stop condition received; Interrupt Flag: UCSTPIFG 0Ah = Interrupt Source: Slave 3 Data received; Interrupt Flag: UCRXIFG3 0Ch = Interrupt Source: Slave 3 Transmit buffer empty; Interrupt Flag: UCTXIFG3 0Eh = Interrupt Source: Slave 2 Data received; Interrupt Flag: UCRXIFG2 10h = Interrupt Source: Slave 2 Transmit buffer empty; Interrupt Flag: UCTXIFG2 12h = Interrupt Source: Slave 1 Data received; Interrupt Flag: UCRXIFG1 14h = Interrupt Source: Slave 1 Transmit buffer empty; Interrupt Flag: UCTXIFG1 16h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG0 18h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG0 1Ah = Interrupt Source: Byte counter zero; Interrupt Flag: UCBCNTIFG 1Ch = Interrupt Source: Clock low timeout; Interrupt Flag: UCCLTOIFG 1Eh = Interrupt Source: Nineth bit position; Interrupt Flag: UCBIT9IFG; Priority: Lowest

Embedded Emulation Module (EEM)

This chapter describes the embedded emulation module (EEM) that is implemented in all devices.

Topic **Page**

23.1 Embedded Emulation Module (EEM) Introduction 639
23.2 EEM Building Blocks 641
23.3 EEM Configurations 642

23.1 Embedded Emulation Module (EEM) Introduction

Every MSP430 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 or breakpoints on memory address bus (MAB) or memory data bus (MDB)
- Up to two (device-dependent) hardware triggers or 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 or 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 EEM implementation.

For more details on how the features of the EEM can be used together with the IAR Embedded Workbench™ debugger or with Code Composer Studio (CCS), see *Advanced Debugging Using the Enhanced Emulation Module* (SLAA393) at www.msp430.com. Most other debuggers supporting the MSP430 devices have the same or a similar feature set. For details, see the user's guide of the applicable debugger.

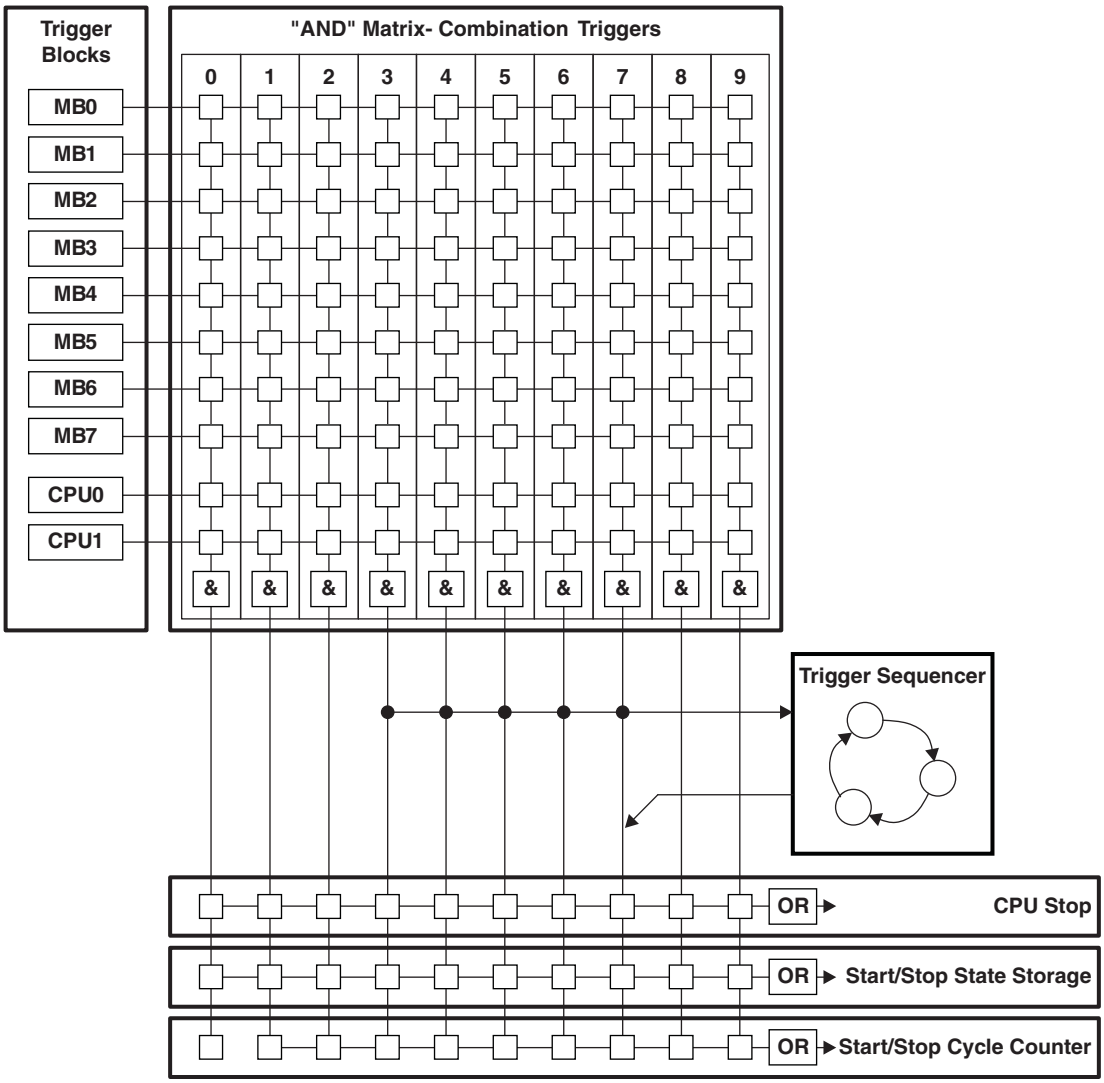


Figure 23-1. Large Implementation of EEM

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 (that is, 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 (for example, 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. The implemented configuration is device dependent, and details can be found in the device-specific data sheet and these documents:

Advanced Debugging Using the Enhanced Emulation Module (EEM) With CCS Version 4 (SLAA393)

IAR Embedded Workbench Version 3+ for MSP430 User's Guide (SLAU138)

Code Composer Studio v4.2 for MSP430 User's Guide (SLAU157)

Table 23-1. 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 or stop)

In general, the following features can be found on any device:

- At least two MAB or 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

Revision History

Changes from revision A (January 2013) to revision B (September 2013)

Location	Comments
Section 1.4.1	Added section.
Section 1.4.3	Added section.
Section 1.14.3.1	Updated equations for 1.2-V reference.
Section 2.2.4	Changed description and added note about watchdog.
Table 2-1	Corrected PMMCTL0 and PMMCTL1 reset values.
Table 2-5	Changed LOCKLPM5 description.
Section 3.2.3	Changed list of conditions when HFXT is enabled.
Section 4.3.4.1	Corrected last code example.
Section 5.5	Removed the option for automatic wait state control.
Table 5-1	Changed FRCTL0_L reset value.
Section 5.10.1	Changed bit 3 of FRCTL0 register to Reserved.
Section 6.2.2	Updated description. Clarified access to the IPE segment by code, JTAG, and DMA.
Section 6.3	Changed note about Discontinuity instructions at segment boundaries.
Section 8.2.6	Changed access type for PxIV registers to "word or byte". Changed description of clearing PxIV flags on read.
Table 8-3, Figure 8-6	Changed reset values of PxOUT registers to undefined.
Section 10.2.7	Changed step 6 of procedure.
Section 10.2.11, Section 10.2.11.1.1	Added "The DMA triggers must be configured as level-sensitive triggers."
Section 10.2.11.1.2 through Section 10.2.11.4.2	Added description of using DMA without CPU interaction.
Section 11.2	Added section.
Section 12.2.5	Changed description of fail-safe features.
Section 15.2.1	Added "It is possible to switch between BCD and hexadecimal format while the RTC is counting."
Table 15-1	Corrected RTCCTL1 reset value.
Table 17-1, Section 17.3.1	Corrected REFCTL0 reset value.
Table 17-2	Changed REFTCOFF description.
Section 19.2.2	Removed option to route internal reference to external pins.
Section 20.2	Added "(wake up from LPMx.5 is not supported)".
Table 20-5	Corrected register names in column headings.
Section 21.3.6	Changed description.
Table 21-3	Changed UCMODEx enum 11b to Reserved.
Table 21-4	Added formula to UCBRx description.
Section 22.3.8	Changed "The byte counter is also incremented at the second byte position" to "second bit position".
Section 22.3.9.2	Changed description.
Example 22-3	Fix vector numbers in sample code.

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from original release (October 2012) to revision A (January 2013)

Location	Comments
Section 1.3.4, Section 4.6.2.19, Section 4.6.2.20	Added note "Enable and Disable Interrupt"
Table 4-1	Added to descriptions of CPUOFF and OSCOFF.
Section 5.10.1	Fixed typo in description of FRCTLPW.

Changes from original release (October 2012) to revision A (January 2013) (continued)

Location	Comments
Section 8.3	Changed section.
Section 13.2.1.1	Added INCLK as a clock source option.
Section 13.3.1	Changed TASSEL bit option 11b to INCLK.
Section 14.2.1.2	Added INCLK as a clock source option.
Section 14.3.1	Changed TBSSEL bit option 11b to INCLK.
Section 6.6.2	Fixed typo in MPUSEG2IFG description
Section 15.2.4	Added note "Changing RT0IP or RT1IP"
Section 15.2.6	Added to description regarding interrupt flags.
Section 20.4.5	Changed descriptions of UCFE, UCPE, and UCBRK.
Section 20.4.11	Changed UCTXIFG reset to correct value.
Table 21-11	Corrected register offsets.
Section 21.3.3.1	Updated UCxSTE description.

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265