MSP430FR4xx and MSP430FR2xx Family

User's Guide

TEXAS INSTRUMENTS

Literature Number: SLAU445D

October 2014–Revised December 2015

Contents

Preface **23**

1 System Resets, Interrupts, and Operating Modes, System Control Module (SYS) **25**

1.1 System Control Module (SYS) Introduction 26
1.2 System Reset and Initialization 26
1.2.1 Device Initial Conditions After System Reset 28
1.3 Interrupts 28
1.3.1 (Non)Maskable Interrupts (NMIs) 29
1.3.2 SNMI Timing 29
1.3.3 Maskable Interrupts 29
1.3.4 Interrupt Processing 30
1.3.5 Interrupt Nesting 31
1.3.6 Interrupt Vectors 31
1.3.7 SYS Interrupt Vector Generators 32
1.4 Operating Modes 32
1.4.1 Low-Power Modes and Clock Requests 35
1.4.2 Entering and Exiting Low-Power Modes LPM0 Through LPM4 36
1.4.3 Low-Power Modes LPM3.5 and LPM4.5 (LPMx.5) 36
1.4.4 Extended Time in Low-Power Modes 38
1.5 Principles for Low-Power Applications 39
1.6 Connection of Unused Pins 39
1.7 Reset Pin ($\overline{\text{RST}}$/NMI) Configuration 39
1.8 Configuring JTAG Pins 40
1.9 Memory Map – Uses and Abilities 40
1.9.1 Memory Map 40
1.9.2 Vacant Memory Space 42
1.9.3 FRAM Write Protection 42
1.9.4 Bootloader (BSL) 42
1.10 JTAG Mailbox (JMB) System 43
1.10.1 JMB Configuration 43
1.10.2 SYSJMBO0 and SYSJMBO1 Outgoing Mailbox 43
1.10.3 SYSJMBI0 and SYSJMBI1 Incoming Mailbox 43
1.10.4 JMB NMI Usage 43
1.11 Device Security 44
1.11.1 JTAG and SBW Lock Mechanism (Electronic Fuse) 44
1.11.2 BSL Security Mechanism 45
1.12 Device-Specific Configurations 45
1.12.1 MSP430FR413x and MSP430FR203x Configurations 45
1.12.2 MSP430FR2433 Configurations 47
1.13 Device Descriptor Table 48
1.13.1 Identifying Device Type 49
1.13.2 TLV Descriptors 50
1.13.3 Calibration Values 50
1.14 SFR Registers 53
1.14.1 SFRIE1 Register (offset = 00h) [reset = 0000h] 54
1.14.2 SFRIFG1 Register (offset = 02h) [reset = 0082h] 55

1.14.3 SFRRPCR Register (offset = 04h) [reset = 001Ch] 56
1.15 SYS Registers 57
1.15.1 SYSCTL Register (offset = 00h) [reset = 0000h] 58
1.15.2 SYSBSLC Register (offset = 02h) [reset = 0000h] 59
1.15.3 SYSJMBC Register (offset = 06h) [reset = 000Ch] 60
1.15.4 SYSJMBI0 Register (offset = 08h) [reset = 0000h] 61
1.15.5 SYSJMBI1 Register (offset = 0Ah) [reset = 0000h] 61
1.15.6 SYSJMBO0 Register (offset = 0Ch) [reset = 0000h] 62
1.15.7 SYSJMBO1 Register (offset = 0Eh) [reset = 0000h] 62
1.15.8 SYSUNIV Register (offset = 1Ah) [reset = 0000h] 63
1.15.9 SYSSNIV Register (offset = 1Ch) [reset = 0000h] 63
1.15.10 SYSRSTIV Register (offset = 1Eh) [reset = 0002h] 64
1.16 System Configuration Registers 65
1.16.1 MSP430FR203x System Configuration Registers 65
1.16.2 MSP430FR413x System Configuration Registers 69
1.16.3 MSP430FR2433 System Configuration Registers 73

2 Power Management Module (PMM) and Supply Voltage Supervisor (SVS) 77
2.1 Power Management Module (PMM) Introduction 78
2.2 PMM Operation 79
2.2.1 V_{CORE} and the Regulator 79
2.2.2 Supply Voltage Supervisor 79
2.2.3 Supply Voltage Supervisor During Power-Up 80
2.2.4 LPM3.5 and LPM4.5 (LPMx.5) 80
2.2.5 Low-Power Reset 80
2.2.6 Brownout Reset (BOR) 81
2.2.7 LPM3.5 Switch 81
2.2.8 Reference Voltage Generation and Output 81
2.2.9 Temperature Sensor 82
2.2.10 $\overline{\text{RST}}$/NMI 82
2.2.11 PMM Interrupts 82
2.2.12 Port I/O Control 82
2.3 PMM Registers 83
2.3.1 PMMCTL0 Register (offset = 00h) [reset = 9640h] 84
2.3.2 PMMCTL1 Register (offset = 02h) [reset = 0000h] 85
2.3.3 PMMCTL2 Register (offset = 04h) [reset = 3200h] 86
2.3.4 PMMIFG Register (offset = 0Ah) [reset = 0000h] 87
2.3.5 PM5CTL0 Register (offset = 10h) [reset = 0011h] 88

3 Clock System (CS) 89
3.1 CS Introduction 90
3.2 CS Operation 92
3.2.1 CS Module Features for Low-Power Applications 92
3.2.2 Internal Very Low-Power Low-Frequency Oscillator (VLO) 92
3.2.3 Internal Trimmed Low-Frequency Reference Oscillator (REFO) 92
3.2.4 XT1 Oscillator 93
3.2.5 Digitally Controlled Oscillator (DCO) 93
3.2.6 Frequency Locked Loop (FLL) 94
3.2.7 DCO Modulator 94
3.2.8 Disabling FLL Hardware and Modulator 95
3.2.9 FLL Unlock Detection 95
3.2.10 FLL Operation From Low-Power Modes 96
3.2.11 Operation From Low-Power Modes, Requested by Peripheral Modules 96
3.2.12 Fail-Safe Operation 97
3.2.13 Synchronization of Clock Signals 99

3.2.14 Module Oscillator (MODOSC) 99
3.3 CS Registers 101
3.3.1 CSCTL0 Register 102
3.3.2 CSCTL1 Register 103
3.3.3 CSCTL2 Register 104
3.3.4 CSCTL3 Register 105
3.3.5 CSCTL4 Register 106
3.3.6 CSCTL5 Register 107
3.3.7 CSCTL6 Register 108
3.3.8 CSCTL7 Register 110
3.3.9 CSCTL8 Register 112
4 CPUX 113
4.1 MSP430X CPU (CPUX) Introduction 114
4.2 Interrupts 116
4.3 CPU Registers 117
4.3.1 Program Counter (PC) 117
4.3.2 Stack Pointer (SP) 117
4.3.3 Status Register (SR) 119
4.3.4 Constant Generator Registers (CG1 and CG2) 120
4.3.5 General-Purpose Registers (R4 to R15) 121
4.4 Addressing Modes 123
4.4.1 Register Mode 124
4.4.2 Indexed Mode 125
4.4.3 Symbolic Mode 129
4.4.4 Absolute Mode 134
4.4.5 Indirect Register Mode 136
4.4.6 Indirect Autoincrement Mode 137
4.4.7 Immediate Mode 138
4.5 MSP430 and MSP430X Instructions 140
4.5.1 MSP430 Instructions 140
4.5.2 MSP430X Extended Instructions 145
4.6 Instruction Set Description 156
4.6.1 Extended Instruction Binary Descriptions 157
4.6.2 MSP430 Instructions 159
4.6.3 Extended Instructions 211
4.6.4 Address Instructions 254
5 FRAM Controller (FRCTL) 269
5.1 FRAM Introduction 270
5.2 FRAM Organization 270
5.3 FRCTL Module Operation 270
5.4 Programming FRAM Devices 271
5.4.1 Programming FRAM Through JTAG or Spy-Bi-Wire 271
5.4.2 Programming FRAM Through the Bootloader (BSL) 271
5.4.3 Programming FRAM Through a Custom Solution 271
5.5 Wait State Control 271
5.5.1 Wait State and Cache Hit 272
5.6 FRAM ECC 272
5.7 FRAM Write Back 272
5.8 FRAM Power Control 272
5.9 FRAM Cache 273
5.10 FRCTL Registers 274
5.10.1 FRCTL0 Register 275
5.10.2 GCCTL0 Register 276

5.10.3 GCCTL1 Register 277

6 Backup Memory (BKMEM) 278

6.1 Backup Memory Introduction 279

6.2 BKMEM Registers 279

7 Digital I/O 280

7.1 Digital I/O Introduction 281

7.2 Digital I/O Operation 282

7.2.1 Input Registers (PxIN) 282

7.2.2 Output Registers (PxOUT) 282

7.2.3 Direction Registers (PxDIR) 282

7.2.4 Pullup or Pulldown Resistor Enable Registers (PxREN) 282

7.2.5 Function Select Registers (PxSEL0, PxSEL1) 283

7.2.6 Port Interrupts 283

7.3 I/O Configuration 285

7.3.1 Configuration After Reset 285

7.3.2 Configuration of Unused Port Pins 286

7.3.3 Configuration for LPMx.5 Low-Power Modes 286

7.4 Digital I/O Registers 288

7.4.1 P1IV Register 301

7.4.2 P2IV Register 301

7.4.3 P3IV Register 302

7.4.4 P4IV Register 302

7.4.5 PxIN Register 303

7.4.6 PxOUT Register 303

7.4.7 PxDIR Register 303

7.4.8 PxREN Register 304

7.4.9 PxSEL0 Register 304

7.4.10 PxSEL1 Register 304

7.4.11 PxSELC Register 305

7.4.12 PxIES Register 305

7.4.13 PxIE Register 305

7.4.14 PxIFG Register 306

8 Capacitive Touch I/O 307

8.1 Capacitive Touch I/O Introduction 308

8.2 Capacitive Touch I/O Operation 309

8.3 CapTouch Registers 310

8.3.1 CAPTIOxCTL Register (offset = 0Eh) [reset = 0000h] 311

9 CapTIvate™ Module 312

9.1 CapTIvate Introduction 313

9.2 CapTIvate Overview 313

9.2.1 Declarations 313

9.2.2 Terms 314

9.2.3 CapTIvate Configurations 316

9.2.4 Operation Modes 317

9.3 CapTIvate Registers 318

9.3.1 CAPIE Register 319

9.3.2 CAPIFG Register 320

9.3.3 CAPIV Register 321

10 CRC Module 322

10.1 Cyclic Redundancy Check (CRC) Module Introduction 323

10.2 CRC Standard and Bit Order 323

10.3 CRC Checksum Generation 324

10.3.1 CRC Implementation 324
10.3.2 Assembler Examples 325
10.4 CRC Registers 327
10.4.1 CRCDI Register 328
10.4.2 CRCDIRB Register 328
10.4.3 CRCINIRES Register 329
10.4.4 CRCRESR Register 329

11 Watchdog Timer (WDT_A) 330
11.1 WDT_A Introduction 331
11.2 WDT_A Operation 333
11.2.1 Watchdog Timer Counter (WDTCNT) 333
11.2.2 Watchdog Mode 333
11.2.3 Interval Timer Mode 333
11.2.4 Watchdog Timer Interrupts 333
11.2.5 Clock Fail-Safe Feature 334
11.2.6 Operation in Low-Power Modes 334
11.3 WDT_A Registers 335
11.3.1 WDTCTL Register 336

12 Timer_A 337
12.1 Timer_A Introduction 338
12.2 Timer_A Operation 340
12.2.1 16-Bit Timer Counter 340
12.2.2 Starting the Timer 340
12.2.3 Timer Mode Control 341
12.2.4 Capture/Compare Blocks 344
12.2.5 Output Unit 346
12.2.6 Timer_A Interrupts 350
12.2.7 Updating Timer_A Configuration 352
12.3 Timer_A Registers 353
12.3.1 TAxCTL Register 354
12.3.2 TAxR Register 355
12.3.3 TAxCCTLn Register 356
12.3.4 TAxCCRn Register 358
12.3.5 TAxIV Register 358
12.3.6 TAxEX0 Register 359

13 Real-Time Clock (RTC) Counter 360
13.1 RTC Counter Introduction 361
13.2 RTC Counter Operation 362
13.2.1 16-Bit Timer Counter 362
13.2.2 Clock Source Select and Divider 362
13.2.3 Modulo Register (RTCMOD) and Shadow Register 362
13.2.4 RTC Counter Interrupt and External Event/Trigger 363
13.3 RTC Counter Registers 364
13.3.1 RTCCTL Register 365
13.3.2 RTCIV Register 366
13.3.3 RTCMOD Register 367
13.3.4 RTCCNT Register 367

14 32-Bit Hardware Multiplier (MPY32) 368
14.1 32-Bit Hardware Multiplier (MPY32) Introduction 369
14.2 MPY32 Operation 371
14.2.1 Operand Registers 372
14.2.2 Result Registers 373

14.2.3 Software Examples 374
14.2.4 Fractional Numbers 375
14.2.5 Putting It All Together 379
14.2.6 Indirect Addressing of Result Registers 381
14.2.7 Using Interrupts 381
14.2.8 Using DMA 382
14.3 MPY32 Registers 383
14.3.1 MPY32CTL0 Register 385

15 LCD_E Controller 386
15.1 LCD_E Introduction 387
15.2 LCD_E Operation 389
15.2.1 LCD Memory 389
15.2.2 Configuration of Port Pin as LCD Output 391
15.2.3 Configuration of LCD Pin as COM or SEG 392
15.2.4 LCD Timing Generation 394
15.2.5 Blanking the LCD 395
15.2.6 LCD Blinking 395
15.2.7 LCD Voltage and Bias Generation 398
15.2.8 LCD Operation Modes 401
15.2.9 LCD Interrupts 403
15.2.10 Static Mode 405
15.2.11 2-Mux Mode 406
15.2.12 3-Mux Mode 407
15.2.13 4-Mux Mode 408
15.2.14 6-Mux Mode 409
15.2.15 8-Mux Mode 410
15.3 LCD_E Registers 412
15.3.1 LCDCTL0 Register 420
15.3.2 LCDCTL1 Register 421
15.3.3 LCDBLKCTL Register 422
15.3.4 LCDMEMCTL Register 423
15.3.5 LCDVCTL Register 424
15.3.6 LCDPCTL0 Register 426
15.3.7 LCDPCTL1 Register 428
15.3.8 LCDPCTL2 Register 430
15.3.9 LCDPCTL3 Register 432
15.3.10 LCDCSSEL0 Register 434
15.3.11 LCDCSSEL1 Register 436
15.3.12 LCDCSSEL2 Register 438
15.3.13 LCDCSSEL3 Register 440
15.3.14 LCDM[index] Register – Static, 2-Mux, 3-Mux, 4-Mux Mode 442
15.3.15 LCDM[index] Register – 5-Mux, 6-Mux, 7-Mux, 8-Mux Mode 444
15.3.16 LCDIV Register 446

16 ADC Module 447
16.1 ADC Introduction 448
16.2 ADC Operation 450
16.2.1 ADC Core 450
16.2.2 ADC Inputs and Multiplexer 450
16.2.3 Voltage Reference Generator 451
16.2.4 Automatic Power Down 451
16.2.5 Sample and Conversion Timing 451
16.2.6 Conversion Result 454
16.2.7 ADC Conversion Modes 454

16.3 ADC Registers 467
16.3.1 ADCCTL0 Register 468
16.3.2 ADCCTL1 Register 470
16.3.3 ADCCTL2 Register 472
16.3.4 ADCMEM0 Register 473
16.3.5 ADCMEM0 Register, 2s-Complement Format 473
16.3.6 ADCMCTL0 Register 474
16.3.7 ADCHI Register 475
16.3.8 ADCHI Register, 2s-Complement Format 475
16.3.9 ADCLO Register 476
16.3.10 ADCLO Register, 2s-Complement Format 476
16.3.11 ADCIE Register 477
16.3.12 ADCIFG Register 478
16.3.13 ADCIV Register 479
16.3.14 MSP430FR413x SYSCFG2 Register (absolute address = 0164h) [reset = 0000h] 480

17 Enhanced Universal Serial Communication Interface (eUSCI) – UART Mode 481
17.1 Enhanced Universal Serial Communication Interface A (eUSCI_A) Overview 482
17.2 eUSCI_A Introduction – UART Mode 482
17.3 eUSCI_A Operation – UART Mode 484
17.3.1 eUSCI_A Initialization and Reset 484
17.3.2 Character Format 484
17.3.3 Asynchronous Communication Format 484
17.3.4 Automatic Baud-Rate Detection 487
17.3.5 IrDA Encoding and Decoding 488
17.3.6 Automatic Error Detection 489
17.3.7 eUSCI_A Receive Enable 490
17.3.8 eUSCI_A Transmit Enable 490
17.3.9 UART Baud-Rate Generation 491
17.3.10 Setting a Baud Rate 493
17.3.11 Transmit Bit Timing - Error calculation 494
17.3.12 Receive Bit Timing – Error Calculation 494
17.3.13 Typical Baud Rates and Errors 495
17.3.14 Using the eUSCI_A Module in UART Mode With Low-Power Modes 497
17.3.15 eUSCI_A Interrupts 497
17.4 eUSCI_A UART Registers 499
17.4.1 UCAxCTLW0 Register 500
17.4.2 UCAxCTLW1 Register 501
17.4.3 UCAxBRW Register 502
17.4.4 UCAxMCTLW Register 502
17.4.5 UCAxSTATW Register 503
17.4.6 UCAxRXBUF Register 504
17.4.7 UCAxTXBUF Register 504
17.4.8 UCAxABCTL Register 505
17.4.9 UCAxIRCTL Register 506
17.4.10 UCAxIE Register 507
17.4.11 UCAxIFG Register 508
17.4.12 UCAxIV Register 509

18 Enhanced Universal Serial Communication Interface (eUSCI) – SPI Mode 510
18.1 Enhanced Universal Serial Communication Interfaces (eUSCI_A, eUSCI_B) Overview 511
18.2 eUSCI Introduction – SPI Mode 511
18.3 eUSCI Operation – SPI Mode 513
18.3.1 eUSCI Initialization and Reset 513
18.3.2 Character Format 514

18.3.3 Master Mode 514
18.3.4 Slave Mode 515
18.3.5 SPI Enable 516
18.3.6 Serial Clock Control 516
18.3.7 Using the SPI Mode With Low-Power Modes 517
18.3.8 SPI Interrupts 517
18.4 eUSCI_A SPI Registers 519
18.4.1 UCAxCTLW0 Register 520
18.4.2 UCAxBRW Register 521
18.4.3 UCAxSTATW Register 522
18.4.4 UCAxRXBUF Register 523
18.4.5 UCAxTXBUF Register 523
18.4.6 UCAxIE Register 524
18.4.7 UCAxIFG Register 524
18.4.8 UCAxIV Register 525
18.5 eUSCI_B SPI Registers 526
18.5.1 UCBxCTLW0 Register 527
18.5.2 UCBxBRW Register 528
18.5.3 UCBxSTATW Register 529
18.5.4 UCBxRXBUF Register 530
18.5.5 UCBxTXBUF Register 530
18.5.6 UCBxIE Register 531
18.5.7 UCBxIFG Register 531
18.5.8 UCBxIV Register 532

19 Enhanced Universal Serial Communication Interface (eUSCI) – I^{2}C Mode 533

19.1 Enhanced Universal Serial Communication Interface B (eUSCI_B) Overview 534
19.2 eUSCI_B Introduction – I^{2}C Mode 534
19.3 eUSCI_B Operation – I^{2}C Mode 535
19.3.1 eUSCI_B Initialization and Reset 536
19.3.2 I^{2}C Serial Data 536
19.3.3 I^{2}C Addressing Modes 537
19.3.4 I^{2}C Quick Setup 538
19.3.5 I^{2}C Module Operating Modes 539
19.3.6 Glitch Filtering 549
19.3.7 I^{2}C Clock Generation and Synchronization 549
19.3.8 Byte Counter 550
19.3.9 Multiple Slave Addresses 551
19.3.10 Using the eUSCI_B Module in I^{2}C Mode With Low-Power Modes 551
19.3.11 eUSCI_B Interrupts in I^{2}C Mode 552
19.4 eUSCI_B I2C Registers 555
19.4.1 UCBxCTLW0 Register 556
19.4.2 UCBxCTLW1 Register 558
19.4.3 UCBxBRW Register 560
19.4.4 UCBxSTATW 560
19.4.5 UCBxTBCNT Register 561
19.4.6 UCBxRXBUF Register 562
19.4.7 UCBxTXBUF 562
19.4.8 UCBxI2COA0 Register 563
19.4.9 UCBxI2COA1 Register 564
19.4.10 UCBxI2COA2 Register 564
19.4.11 UCBxI2COA3 Register 565
19.4.12 UCBxADDRX Register 565
19.4.13 UCBxADDMASK Register 566

19.4.14 UCBxI2CSA Register 566
19.4.15 UCBxIE Register 567
19.4.16 UCBxIFG Register 569
19.4.17 UCBxIV Register 571

20 Embedded Emulation Module (EEM) 572
20.1 Embedded Emulation Module (EEM) Introduction 573
20.2 EEM Building Blocks 575
20.2.1 Triggers 575
20.2.2 Trigger Sequencer 575
20.2.3 State Storage (Internal Trace Buffer) 575
20.2.4 Cycle Counter 575
20.2.5 Clock Control 576
20.3 EEM Configurations 576

Revision History 577

List of Figures

1-1. BOR, POR, and PUC Reset Circuit 27
1-2. Interrupt Priority 29
1-3. Interrupt Processing 30
1-4. Return From Interrupt 31
1-5. Operation Modes 34
1-6. IR Modulation Combinatory Logics 46
1-7. 1.2-V Reference Output on A4 46
1-8. IR Modulation Combinatory Logics 47
1-9. 1.2-V Reference Output on A4 48
1-10. Devices Descriptor Table 49
1-11. SFRIE1 Register 54
1-12. SFRIFG1 Register 55
1-13. SFRRPCR Register 56
1-14. SYSCTL Register 58
1-15. SYSBSLC Register 59
1-16. SYSJMBC Register 60
1-17. SYSJMBI0 Register 61
1-18. SYSJMBI1 Register 61
1-19. SYSJMBO0 Register 62
1-20. SYSJMBO1 Register 62
1-21. SYSUNIV Register 63
1-22. SYSSNIV Register 63
1-23. SYSRSTIV Register 64
1-24. SYSCFG0 Register 66
1-25. SYSCFG1 Register 67
1-26. SYSCFG2 Register 68
1-27. SYSCFG0 Register 70
1-28. SYSCFG1 Register 71
1-29. SYSCFG2 Register 72
1-30. SYSCFG0 Register 74
1-31. SYSCFG1 Register 75
1-32. SYSCFG2 Register 76
2-1. PMM Block Diagram 78
2-2. Voltage Failure and Resulting PMM Actions 79
2-3. PMM Action at Device Power-Up 80
2-4. PMMCTL0 Register 84
2-5. PMMCTL1 Register 85
2-6. PMMCTL2 Register 86
2-7. PMMIFG Register 87
2-8. PM5CTL0 Register 88
3-1. Clock System (CS) Block Diagram 91
3-2. Modulator Patterns 95
3-3. FLL Unlock Detection 96
3-4. Module Request Clock System 97
3-5. Oscillator Fault Logic 98
3-6. Switch MCLK from DCOCLK to XT1CLK 99
3-7. CSCTL0 Register 102

3-8. CSCTL1 Register 103
3-9. CSCTL2 Register 104
3-10. CSCTL3 Register 105
3-11. CSCTL4 Register 106
3-12. CSCTL5 Register 107
3-13. CSCTL6 Register 108
3-14. CSCTL7 Register 110
3-15. CSCTL8 Register 112
4-1. MSP430X CPU Block Diagram 115
4-2. PC Storage on the Stack for Interrupts 116
4-3. Program Counter..... 117
4-4. PC Storage on the Stack for CALLA 117
4-5. Stack Pointer 118
4-6. Stack Usage 118
4-7. PUSHX.A Format on the Stack 118
4-8. PUSH SP, POP SP Sequence..... 118
4-9. SR Bits 119
4-10. Register-Byte and Byte-Register Operation..... 121
4-11. Register-Word Operation 121
4-12. Word-Register Operation 122
4-13. Register – Address-Word Operation 122
4-14. Address-Word – Register Operation 123
4-15. Indexed Mode in Lower 64KB..... 125
4-16. Indexed Mode in Upper Memory 126
4-17. Overflow and Underflow for Indexed Mode 127
4-18. Example for Indexed Mode..... 128
4-19. Symbolic Mode Running in Lower 64KB 130
4-20. Symbolic Mode Running in Upper Memory 131
4-21. Overflow and Underflow for Symbolic Mode 132
4-22. MSP430 Double-Operand Instruction Format..... 140
4-23. MSP430 Single-Operand Instructions..... 141
4-24. Format of Conditional Jump Instructions 142
4-25. Extension Word for Register Modes 145
4-26. Extension Word for Non-Register Modes..... 145
4-27. Example for Extended Register or Register Instruction 146
4-28. Example for Extended Immediate or Indexed Instruction 147
4-29. Extended Format I Instruction Formats 148
4-30. 20-Bit Addresses in Memory 148
4-31. Extended Format II Instruction Format..... 149
4-32. PUSHM and POPM Instruction Format..... 150
4-33. RRCM, RRAM, RRUM, and RLAM Instruction Format 150
4-34. BRA Instruction Format 150
4-35. CALLA Instruction Format 150
4-36. Decrement Overlap..... 176
4-37. Stack After a RET Instruction 195
4-38. Destination Operand—Arithmetic Shift Left 197
4-39. Destination Operand—Carry Left Shift..... 198
4-40. Rotate Right Arithmetically RRA.B and RRA.W 199
4-41. Rotate Right Through Carry RRC.B and RRC.W 200

4-42. Swap Bytes in Memory 207
4-43. Swap Bytes in a Register 207
4-44. Rotate Left Arithmetically—RLAM[.W] and RLAM.A 234
4-45. Destination Operand-Arithmetic Shift Left 235
4-46. Destination Operand-Carry Left Shift 236
4-47. Rotate Right Arithmetically RRAM[.W] and RRAM.A 237
4-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode 239
4-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode 239
4-50. Rotate Right Through Carry RRCM[.W] and RRCM.A 241
4-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode 243
4-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode 243
4-53. Rotate Right Unsigned RRUM[.W] and RRUM.A 244
4-54. Rotate Right Unsigned RRUX(.B,.A) – Register Mode 245
4-55. Swap Bytes SWPBX.A Register Mode 249
4-56. Swap Bytes SWPBX.A In Memory 249
4-57. Swap Bytes SWPBX[.W] Register Mode 250
4-58. Swap Bytes SWPBX[.W] In Memory 250
4-59. Sign Extend SXTX.A 251
4-60. Sign Extend SXTX[.W] 251
5-1. FRAM Controller Block Diagram 270
5-2. FRAM Power Control Diagram 273
5-3. FRCTL0 Register 275
5-4. GCCTL0 Register 276
5-5. GCCTL1 Register 277
7-1. P1IV Register 301
7-2. P2IV Register 301
7-3. P3IV Register 302
7-4. P4IV Register 302
7-5. PxIN Register 303
7-6. PxOUT Register 303
7-7. PxDIR Register 303
7-8. PxREN Register 304
7-9. PxSEL0 Register 304
7-10. PxSEL1 Register 304
7-11. PxSELC Register 305
7-12. PxIES Register 305
7-13. PxIE Register 305
7-14. PxIFG Register 306
8-1. Capacitive Touch I/O Principle 308
8-2. Capacitive Touch I/O Block Diagram 309
8-3. CAPTIOxCTL Register 311
9-1. CapTIvate Declarations 314
9-2. Channel Definition 315
9-3. Element Definition 315
9-4. CapTIvate Configurations 316
9-5. Self-Capacitance Mode 317
9-6. Mutual-Capacitance Mode 317
9-7. CAPIE Register 319
9-8. CAPIFG Register 320

9-9. CAPIV Register 321
10-1. LFSR Implementation of CRC-CCITT Standard, Bit 0 is the MSB of the Result 323
10-2. Implementation of CRC-CCITT Using the CRCDI and CRCINIRES Registers 325
10-3. CRCDI Register 328
10-4. CRCDIRB Register 328
10-5. CRCINIRES Register 329
10-6. CRCRESR Register 329
11-1. Watchdog Timer Block Diagram 332
11-2. WDTCTL Register 336
12-1. Timer_A Block Diagram 339
12-2. Up Mode 341
12-3. Up Mode Flag Setting 341
12-4. Continuous Mode 342
12-5. Continuous Mode Flag Setting 342
12-6. Continuous Mode Time Intervals 342
12-7. Up/Down Mode 343
12-8. Up/Down Mode Flag Setting 343
12-9. Output Unit in Up/Down Mode 344
12-10. Capture Signal (SCS = 1) 345
12-11. Capture Cycle 345
12-12. Output Example – Timer in Up Mode 347
12-13. Output Example – Timer in Continuous Mode 348
12-14. Output Example – Timer in Up/Down Mode 349
12-15. Capture/Compare TAxCCR0 Interrupt Flag 350
12-16. TAxCTL Register 354
12-17. TAxR Register 355
12-18. TAxCCTLn Register 356
12-19. TAxCCRn Register 358
12-20. TAxIV Register 358
12-21. TAxEX0 Register 359
13-1. RTC Counter Block Diagram 361
13-2. Shadow Register Example 363
13-3. RTCCTL Register 365
13-4. RTCIV Register 366
13-5. RTCMOD Register 367
13-6. RTCCNT Register 367
14-1. MPY32 Block Diagram 370
14-2. Q15 Format Representation 375
14-3. Q14 Format Representation 375
14-4. Saturation Flow Chart 377
14-5. Multiplication Flow Chart 379
14-6. MPY32CTL0 Register 385
15-1. LCD Controller Block Diagram 388
15-2. LCD Memory for Static to 4-Mux Mode – Example for 4-Mux Mode With 240 Segments 390
15-3. LCD Memory for 5-Mux to 8-Mux Mode – Example for 8-Mux Mode With 96 Segments 391
15-4. LCDMx in Static, 2-, 3-, or 4-Mux Mode 393
15-5. LCDMx in 5-, 6-, 7-, or 8-Mux Mode 393
15-6. Example LCDMx and LCDBMx Configuration in Different Blinking Modes 397
15-7. Bias Generation 399

15-8. LCD Operation Mode 1 401
15-9. LCD Operation Mode 2 401
15-10. LCD Operation Mode 3 402
15-11. LCD Operation Mode 4 403
15-12. Example Static Waveforms 405
15-13. Example 2-Mux Waveforms 406
15-14. Example 3-Mux Waveforms 407
15-15. Example 4-Mux Waveforms 408
15-16. Example 6-Mux Waveforms 409
15-17. Example 8-Mux, 1/3 Bias Waveforms (LCDLP = 0) 410
15-18. Example 8-Mux, 1/3 Bias Low-Power Waveforms (LCDLP = 1) 411
15-19. LCDCTL0 Register 420
15-20. LCDCTL1 Register 421
15-21. LCDBLKCTL Register 422
15-22. LCDMEMCTL Register 423
15-23. LCDVCTL Register 424
15-24. LCDPCTL0 Register 426
15-25. LCDPCTL1 Register 428
15-26. LCDPCTL2 Register 430
15-27. LCDPCTL3 Register 432
15-28. LCDCSSEL0 Register 434
15-29. LCDCSSEL1 Register 436
15-30. LCDCSSEL2 Register 438
15-31. LCDCSSEL3 Register 440
15-32. LCDM[*index*] Register 442
15-33. LCDM[*index*] Register 444
15-34. LCDIV Register 446
16-1. ADC Block Diagram 449
16-2. Analog Multiplexer 451
16-3. Extended Sample Mode in 10-bit mode 452
16-4. Extended Sample Mode in 12-bit mode 452
16-5. Pulse Sample Mode in 10-bit mode 453
16-6. Pulse Sample Mode in 12-bit mode 453
16-7. Analog Input Equivalent Circuit 454
16-8. Single-Channel Single-Conversion Mode in 10-bit mode 455
16-9. Single-Channel Single-Conversion Mode in 12-bit mode 456
16-10. Sequence-of-Channels Mode in 10-bit mode 457
16-11. Sequence-of-Channels Mode in 12-bit mode 458
16-12. Repeat-Single-Channel Mode in 10-bit mode 459
16-13. Repeat-Single-Channel Mode in 12-bit mode 460
16-14. Repeat-Sequence-of-Channels Mode in 10-bit mode 461
16-15. Repeat-Sequence-of-Channels Mode in 12-bit mode 462
16-16. Typical Temperature Sensor Transfer Function 464
16-17. ADC Grounding and Noise Considerations 464
16-18. ADCCTL0 Register 468
16-19. ADCCTL1 Register 470
16-20. ADCCTL2 Register 472
16-21. ADCMEM0 Register 473
16-22. ADCMEM0 Register 473

16-23. ADCMCTL0 Register 474
16-24. ADCHI Register 475
16-25. ADCHI Register 475
16-26. ADCLO Register 476
16-27. ADCLO Register 476
16-28. ADCIE Register 477
16-29. ADCIFG Register 478
16-30. ADCIV Register 479
16-31. SYSCFG2 Register 480
17-1. eUSCI_Ax Block Diagram – UART Mode (UCSYNC = 0) 483
17-2. Character Format 484
17-3. Idle-Line Format 485
17-4. Address-Bit Multiprocessor Format 486
17-5. Auto Baud-Rate Detection – Break/Synch Sequence 487
17-6. Auto Baud-Rate Detection – Synch Field 487
17-7. UART vs IrDA Data Format 488
17-8. Glitch Suppression, eUSCI_A Receive Not Started 490
17-9. Glitch Suppression, eUSCI_A Activated 490
17-10. BITCLK Baud-Rate Timing With UCOS16 = 0 491
17-11. Receive Error 495
17-12. UCAxCTLW0 Register 500
17-13. UCAxCTLW1 Register 501
17-14. UCAxBRW Register 502
17-15. UCAxMCTLW Register 502
17-16. UCAxSTATW Register 503
17-17. UCAxRXBUF Register 504
17-18. UCAxTXBUF Register 504
17-19. UCAxABCTL Register 505
17-20. UCAxIRCTL Register 506
17-21. UCAxIE Register 507
17-22. UCAxIFG Register 508
17-23. UCAxIV Register 509
18-1. eUSCI Block Diagram – SPI Mode 512
18-2. eUSCI Master and External Slave (UCSTEM = 0) 514
18-3. eUSCI Slave and External Master 515
18-4. eUSCI SPI Timing With UCMSB = 1 517
18-5. UCAxCTLW0 Register 520
18-6. UCAxBRW Register 521
18-7. UCAxSTATW Register 522
18-8. UCAxRXBUF Register 523
18-9. UCAxTXBUF Register 523
18-10. UCAxIE Register 524
18-11. UCAxIFG Register 524
18-12. UCAxIV Register 525
18-13. UCBxCTLW0 Register 527
18-14. UCBxBRW Register 528
18-15. UCBxSTATW Register 529
18-16. UCBxRXBUF Register 530
18-17. UCBxTXBUF Register 530

18-18. UCBxIE Register 531
18-19. UCBxIFG Register 531
18-20. UCBxIV Register 532
19-1. eUSCI_B Block Diagram – I^{2}C Mode 535
19-2. I^{2}C Bus Connection Diagram 536
19-3. I^{2}C Module Data Transfer 537
19-4. Bit Transfer on I^{2}C Bus 537
19-5. I^{2}C Module 7-Bit Addressing Format 537
19-6. I^{2}C Module 10-Bit Addressing Format 538
19-7. I^{2}C Module Addressing Format With Repeated START Condition 538
19-8. I^{2}C Time-Line Legend 540
19-9. I^{2}C Slave Transmitter Mode 541
19-10. I^{2}C Slave Receiver Mode 542
19-11. I^{2}C Slave 10-Bit Addressing Mode 543
19-12. I^{2}C Master Transmitter Mode 545
19-13. I^{2}C Master Receiver Mode 547
19-14. I^{2}C Master 10-Bit Addressing Mode 548
19-15. Arbitration Procedure Between Two Master Transmitters 548
19-16. Synchronization of Two I^{2}C Clock Generators During Arbitration 549
19-17. UCBxCTLW0 Register 556
19-18. UCBxCTLW1 Register 558
19-19. UCBxBRW Register 560
19-20. UCBxSTATW Register 560
19-21. UCBxTBCNT Register 561
19-22. UCBxRXBUF Register 562
19-23. UCBxTXBUF Register 562
19-24. UCBxI2COA0 Register 563
19-25. UCBxI2COA1 Register 564
19-26. UCBxI2COA2 Register 564
19-27. UCBxI2COA3 Register 565
19-28. UCBxADDRX Register 565
19-29. UCBxADDMASK Register 566
19-30. UCBxI2CSA Register 566
19-31. UCBxIE Register 567
19-32. UCBxIFG Register 569
19-33. UCBxIV Register 571
20-1. Large Implementation of EEM 574

List of Tables

0-1. Register Bit Accessibility and Initial Condition 24
1-1. Interrupt Sources, Flags, and Vectors 31
1-2. Operation Modes 35
1-3. Requested vs Actual LPM 35
1-4. Connection of Unused Pins 39
1-5. BSL and JTAG/SBW Signatures 44
1-6. Tag Values 50
1-7. SFR Base Address 53
1-8. SFR Registers 53
1-9. SFRIE1 Register Description 54
1-10. SFRIFG1 Register Description 55
1-11. SFRRPCR Register Description 56
1-12. SYS Registers 57
1-13. SYSCTL Register Description 58
1-14. SYSBSLC Register Description 59
1-15. SYSJMBC Register Description 60
1-16. SYSJMBI0 Register Description 61
1-17. SYSJMBI1 Register Description 61
1-18. SYSJMBO0 Register Description 62
1-19. SYSJMBO1 Register Description 62
1-20. SYSUNIV Register Description 63
1-21. SYSSNIV Register Description 63
1-22. SYSRSTIV Register Description 64
1-23. MSP430FR203x SYS Configuration Registers 65
1-24. SYSCFG0 Register Description 66
1-25. SYSCFG1 Register Description 67
1-26. SYSCFG2 Register Description 68
1-27. FR413x SYS Configuration Registers 69
1-28. SYSCFG0 Register Description 70
1-29. SYSCFG1 Register Description 71
1-30. SYSCFG2 Register Description 72
1-31. FR2433 SYS Configuration Registers 73
1-32. SYSCFG0 Register Description 74
1-33. SYSCFG1 Register Description 75
1-34. SYSCFG2 Register Description 76
2-1. PMM Registers 83
2-2. PMMCTL0 Register Description 84
2-3. PMMCTL1 Register Description 85
2-4. PMMCTL2 Register Description 86
2-5. PMMIFG Register Description 87
2-6. PM5CTL0 Register Description 88
3-1. Clock Request System and Power Modes 97
3-2. CS Registers 101
3-3. CSCTL0 Register Description 102
3-4. CSCTL1 Register Description 103
3-5. CSCTL2 Register Description 104
3-6. CSCTL3 Register Description 105

3-7. CSCTL4 Register Description 106
3-8. CSCTL5 Register Description 107
3-9. CSCTL6 Register Description 108
3-10. CSCTL7 Register Description 110
3-11. CSCTL8 Register Description 112
4-1. SR Bit Description 119
4-2. Values of Constant Generators CG1, CG2 120
4-3. Source and Destination Addressing 123
4-4. MSP430 Double-Operand Instructions 141
4-5. MSP430 Single-Operand Instructions 141
4-6. Conditional Jump Instructions 142
4-7. Emulated Instructions 142
4-8. Interrupt, Return, and Reset Cycles and Length 143
4-9. MSP430 Format II Instruction Cycles and Length 143
4-10. MSP430 Format I Instructions Cycles and Length 144
4-11. Description of the Extension Word Bits for Register Mode 145
4-12. Description of Extension Word Bits for Non-Register Modes 146
4-13. Extended Double-Operand Instructions 147
4-14. Extended Single-Operand Instructions 149
4-15. Extended Emulated Instructions 151
4-16. Address Instructions, Operate on 20-Bit Register Data 152
4-17. MSP430X Format II Instruction Cycles and Length 153
4-18. MSP430X Format I Instruction Cycles and Length 154
4-19. Address Instruction Cycles and Length 155
4-20. Instruction Map of MSP430X 156
5-1. FRCTL Registers 274
5-2. FRCTL0 Register Description 275
5-3. GCCTL0 Register Description 276
5-4. GCCTL1 Register Description 277
6-1. BKMEM Registers 279
7-1. I/O Configuration 282
7-2. I/O Function Selection for Devices With Only One Selection Bit – PxSEL0 283
7-3. I/O Function Selection for Devices With Two Selection Bits – PxSEL0 and PxSEL1 283
7-4. Digital I/O Registers 288
7-5. P1IV Register Description 301
7-6. P2IV Register Description 301
7-7. P3IV Register Description 302
7-8. P4IV Register Description 302
7-9. PxIN Register Description 303
7-10. PxOUT Register Description 303
7-11. P1DIR Register Description 303
7-12. PxREN Register Description 304
7-13. PxSEL0 Register Description 304
7-14. PxSEL1 Register Description 304
7-15. PxSELC Register Description 305
7-16. PxIES Register Description 305
7-17. PxIE Register Description 305
7-18. PxIFG Register Description 306
8-1. CapTouch Registers 310

8-2. CAPTIOxCTL Register Description 311
9-1. CapTIvate Registers 318
9-2. CAPIE Register Description 319
9-3. CAPIFG Register Description 320
9-4. CAPIV Register Description 321
10-1. CRC Registers 327
10-2. CRCDI Register Description 328
10-3. CRCDIRB Register Description 328
10-4. CRCINIRES Register Description 329
10-5. CRCRESR Register Description 329
11-1. WDT_A Registers 335
11-2. WDTCTL Register Description 336
12-1. Timer Modes 341
12-2. Output Modes 346
12-3. Timer_A Registers 353
12-4. TAxCTL Register Description 354
12-5. TAxR Register Description 355
12-6. TAxCCTLn Register Description 356
12-7. TAxCCRn Register Description 358
12-8. TAxIV Register Description 358
12-9. TAxEX0 Register Description 359
13-1. RTC Counter Registers 364
13-2. RTCCTL Register Description 365
13-3. RTCIV Register Description 366
13-4. RTCMOD Register Description 367
13-5. RTCCNT Register Description 367
14-1. Result Availability (MPYFRAC = 0, MPYSAT = 0) 371
14-2. OP1 Registers 372
14-3. OP2 Registers 372
14-4. SUMEXT and MPYC Contents 373
14-5. Result Availability in Fractional Mode (MPYFRAC = 1, MPYSAT = 0) 376
14-6. Result Availability in Saturation Mode (MPYSAT = 1) 377
14-7. MPY32 Registers 383
14-8. Alternative Registers 384
14-9. MPY32CTL0 Register Description 385
15-1. Differences Between LCD_B, LCD_C, and LCD_E 387
15-2. Divider depending on MUX-Mode 394
15-3. Example for Possible LCD Frequencies 395
15-4. Overview on COM Configuration in Blinking Mode 396
15-5. LCD Voltage and Biasing Characteristics 400
15-6. LCD_E Registers 412
15-7. LCD Memory Registers for Static and 2-Mux to 4-Mux Modes 413
15-8. LCD Blinking Memory Registers for Static and 2-Mux to 4-Mux Modes 415
15-9. LCD Memory Registers for 5-Mux to 8-Mux Modes 417
15-10. LCDCTL0 Register Description 420
15-11. LCDCTL1 Register Description 421
15-12. LCDBLKCTL Register Description 422
15-13. LCDMEMCTL Register Description 423
15-14. LCDVCTL Register Description 424

15-15. LCDPCTL0 Register Description 426
15-16. LCDPCTL1 Register Description 428
15-17. LCDPCTL2 Register Description 430
15-18. LCDPCTL3 Register Description 432
15-19. LCDCSSEL0 Register Description 434
15-20. LCDCSSEL1 Register Description 436
15-21. LCDCSSEL2 Register Description 438
15-22. LCDCSSEL3 Register Description 440
15-23. LCDM[*index*] Register Description.... 442
15-24. LCDM[*index*] Register Description.... 444
15-25. LCDIV Register Description 446
16-1. Conversion Mode Summary.... 454
16-2. ADC Registers 467
16-3. ADCCTL0 Register Description.... 468
16-4. ADCCTL1 Register Description.... 470
16-5. ADCCTL2 Register Description.... 472
16-6. ADCMEM0 Register Description.... 473
16-7. ADCMEM0 Register Description.... 473
16-8. ADCMCTL0 Register Description.... 474
16-9. ADCHI Register Description.... 475
16-10. ADCHI Register Description.... 475
16-11. ADCLO Register Description.... 476
16-12. ADCLO Register Description.... 476
16-13. ADCIE Register Description.... 477
16-14. ADCIFG Register Description.... 478
16-15. ADCIV Register Description.... 479
16-16. SYSCFG2 Register Description.... 480
17-1. Receive Error Conditions 489
17-2. Modulation Pattern Examples.... 491
17-3. BITCLK16 Modulation Pattern 492
17-4. UCBRSx Settings for Fractional Portion of N = f_{BRCLK}/Baud Rate.... 493
17-5. Recommended Settings for Typical Crystals and Baud Rates 496
17-6. UART State Change Interrupt Flags 498
17-7. eUSCI_A UART Registers.... 499
17-8. UCAxCTLW0 Register Description 500
17-9. UCAxCTLW1 Register Description 501
17-10. UCAxBRW Register Description.... 502
17-11. UCAxMCTLW Register Description 502
17-12. UCAxSTATW Register Description.... 503
17-13. UCAxRXBUF Register Description 504
17-14. UCAxTXBUF Register Description 504
17-15. UCAxABCTL Register Description 505
17-16. UCAxIRCTL Register Description 506
17-17. UCAxIE Register Description.... 507
17-18. UCAxIFG Register Description.... 508
17-19. UCAxIV Register Description.... 509
18-1. UCxSTE Operation 513
18-2. eUSCI_A SPI Registers.... 519
18-3. UCAxCTLW0 Register Description 520

18-4. UCAxBRW Register Description 521
18-5. UCAxSTATW Register Description 522
18-6. UCAxRXBUF Register Description 523
18-7. UCAxTXBUF Register Description 523
18-8. UCAxIE Register Description 524
18-9. UCAxIFG Register Description 524
18-10. UCAxIV Register Description 525
18-11. eUSCI_B SPI Registers 526
18-12. UCBxCTLW0 Register Description 527
18-13. UCBxBRW Register Description 528
18-14. UCBxSTATW Register Description 529
18-15. UCBxRXBUF Register Description 530
18-16. UCBxTXBUF Register Description 530
18-17. UCBxIE Register Description 531
18-18. UCBxIFG Register Description 531
18-19. UCBxIV Register Description 532
19-1. Glitch Filter Length Selection Bits 549
19-2. I^{2}C State Change Interrupt Flags 552
19-3. eUSCI_B Registers 555
19-4. UCBxCTLW0 Register Description 556
19-5. UCBxCTLW1 Register Description 558
19-6. UCBxBRW Register Description 560
19-7. UCBxSTATW Register Description 560
19-8. UCBxTBCNT Register Description 561
19-9. UCBxRXBUF Register Description 562
19-10. UCBxTXBUF Register Description 562
19-11. UCBxI2COA0 Register Description 563
19-12. UCBxI2COA1 Register Description 564
19-13. UCBxI2COA2 Register Description 564
19-14. UCBxI2COA3 Register Description 565
19-15. UCBxADDRX Register Description 565
19-16. UCBxADDMASK Register Description 566
19-17. UCBxI2CSA Register Description 566
19-18. UCBxIE Register Description 567
19-19. UCBxIFG Register Description 569
19-20. UCBxIV Register Description 571
20-1. EEM Configurations 576

Read This First

About This Manual

This manual describes the modules and peripherals of the MSP430FR4xx 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	Bootloader; 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	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:

Table 0-1. 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

Chapter 1

SLAU445D–October 2014–Revised December 2015

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 and UNMI) event source selection and management
- User data-exchange mechanism through the JTAG mailbox (JMB)
- Bootloader (BSL) entry mechanism
- Configuration management (device descriptors)
- Providing interrupt vector generators for reset and NMIs
- FRAM write protection
- On-chip module-to-module signaling control

Topic **Page**

1.1 System Control Module (SYS) Introduction **26**
1.2 System Reset and Initialization **26**
1.3 Interrupts **28**
1.4 Operating Modes **32**
1.5 Principles for Low-Power Applications **39**
1.6 Connection of Unused Pins **39**
1.7 Reset Pin ($\overline{RST}$/NMI) Configuration **39**
1.8 Configuring JTAG Pins **40**
1.9 Memory Map – Uses and Abilities **40**
1.10 JTAG Mailbox (JMB) System **43**
1.11 Device Security **44**
1.12 Device-Specific Configurations **45**
1.13 Device Descriptor Table **48**
1.14 SFR Registers **53**
1.15 SYS Registers **57**
1.16 System Configuration Registers **65**

1.1 System Control Module (SYS) Introduction

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

1.2 System Reset and Initialization

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

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

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

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

- BOR signal
- Software POR event

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

- POR signal
- Watchdog timer expiration when in 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 chapter for details)
- Fetch from peripheral area

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

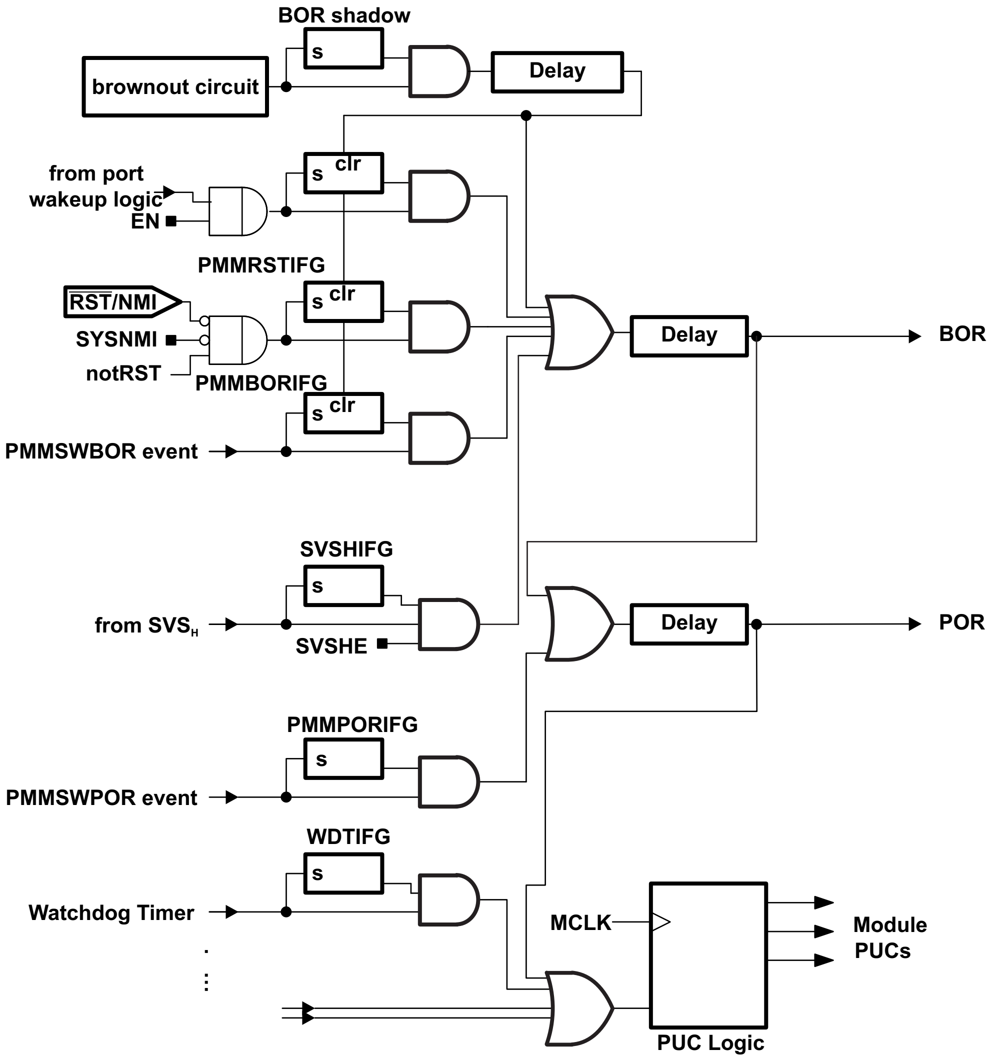


Figure 1-1. BOR, POR, and PUC Reset Circuit

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 on configuring the $\overline{\text{RST}}$/NMI pin.
- I/O pins are set to input mode as described in the Digital I/O chapter.
- Other peripheral modules and registers are initialized as described in their respective chapters in this manual.
- Status register (SR) is reset.
- The watchdog timer powers up active in watchdog mode.
- Program counter (PC) is loaded with the boot code address and boot code execution begins at that address. Upon completion of the boot code, the PC is loaded with the address contained at the SYSRSTIV reset location (0FFFEh).

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

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

NOTE: A device that is unprogrammed or blank is defined as having its reset vector value, 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 which interrupt is acted on when more than one interrupt is pending simultaneously.

There are three types of interrupts:

- System reset
- (Non)maskable
- Maskable

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

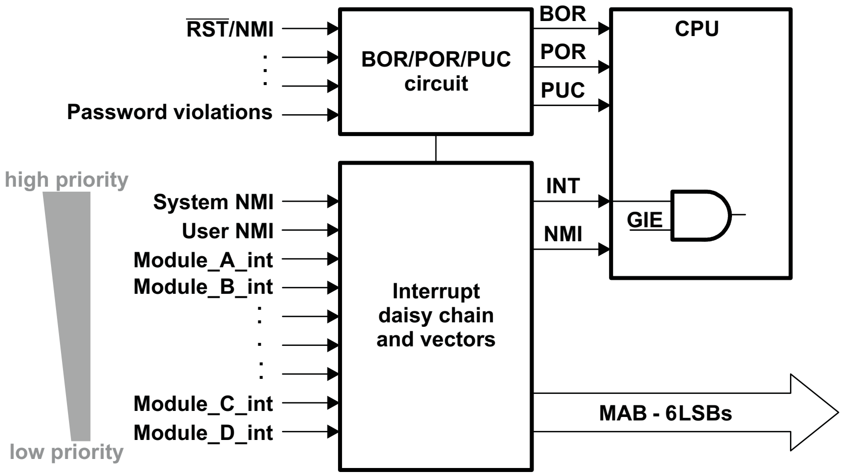


Figure 1-2. Interrupt Priority

1.3.1 (Non)Maskable Interrupts (NMIs)

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

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.

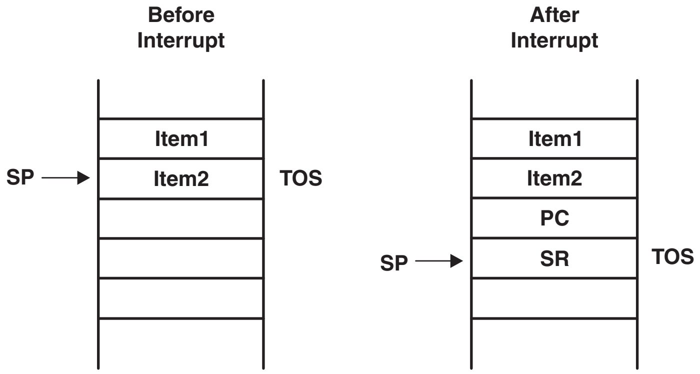


Figure 1-3. Interrupt Processing

NOTE: Enabling 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 shown in Figure 1-4.

1. The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, and the other bits 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.

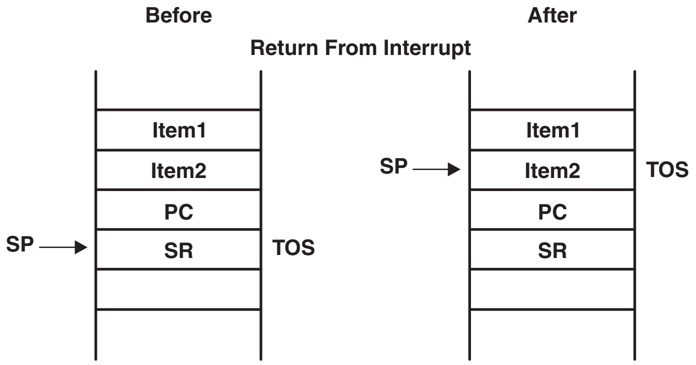


Figure 1-4. Return From Interrupt

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 that are 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	... WDTIFG KEYV	... Reset	... 0FFFEh	... Highest
System NMI: JTAG Mailbox	JMBINIFG, JMBOUTIFG	(Non)maskable	0FFFCh	...
User NMI: NMI, oscillator fault, FRAM memory access violation	... NMIIFG OFIFG	... (Non)maskable (Non)maskable	... 0FFFAh	... ...
Device specific			0FFF8h	...
...			...	...
Watchdog timer	WDTIFG	Maskable	...	...
...			...	...
Device specific			...	...
Reserved		Maskable	...	Lowest

Some interrupt enable bits, interrupt flags, and the control bits for the RST/NMI pin are located in the special function registers (SFRs). The SFRs 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, the RAM can be used 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. When the bit is 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, 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   VMA_ISR      ; Vector 10: VMAIFG
         JMP   JMBI_ISR     ; Vector 12: JMBINIFG
JMBO_ISR:                   ; Vector 14: JMBOUTIFG
        ...                 ; Task_E starts here
        RETI                ; Return
VMA_ISR:                    ; Vector A
       ...                  ; Task_A starts here
        RETI                ; Return
JMBI_ISR:                   ; Vector C
       ...                  ; Task_C starts here
       RETI                 ; Return
```

1.4 Operating Modes

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

The operating modes take into account three different needs:

- Low power
- Speed and data throughput
- Minimizing current consumption of individual peripherals

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. Wake-up 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. Wake-up from LPM4.5 is possible from a power sequence, a $\overline{\text{RST}}$ event, or from specific I/O. Wake-up from LPM3.5 is possible from a power sequence, a $\overline{\text{RST}}$ event, an RTC event, an LF crystal fault, or from specific I/O.

NOTE: The TEST/SBWTCK pin is used for interfacing to the development tools through Spy-Bi-Wire. When the TEST/SBWTCK pin is high, wake-up times from LPM2 (device specific) , 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 (device specific), LPM3, and LPM4 with the device connected to a development tool (for example, MSP-FET430UIF). See the PMM chapter for details.

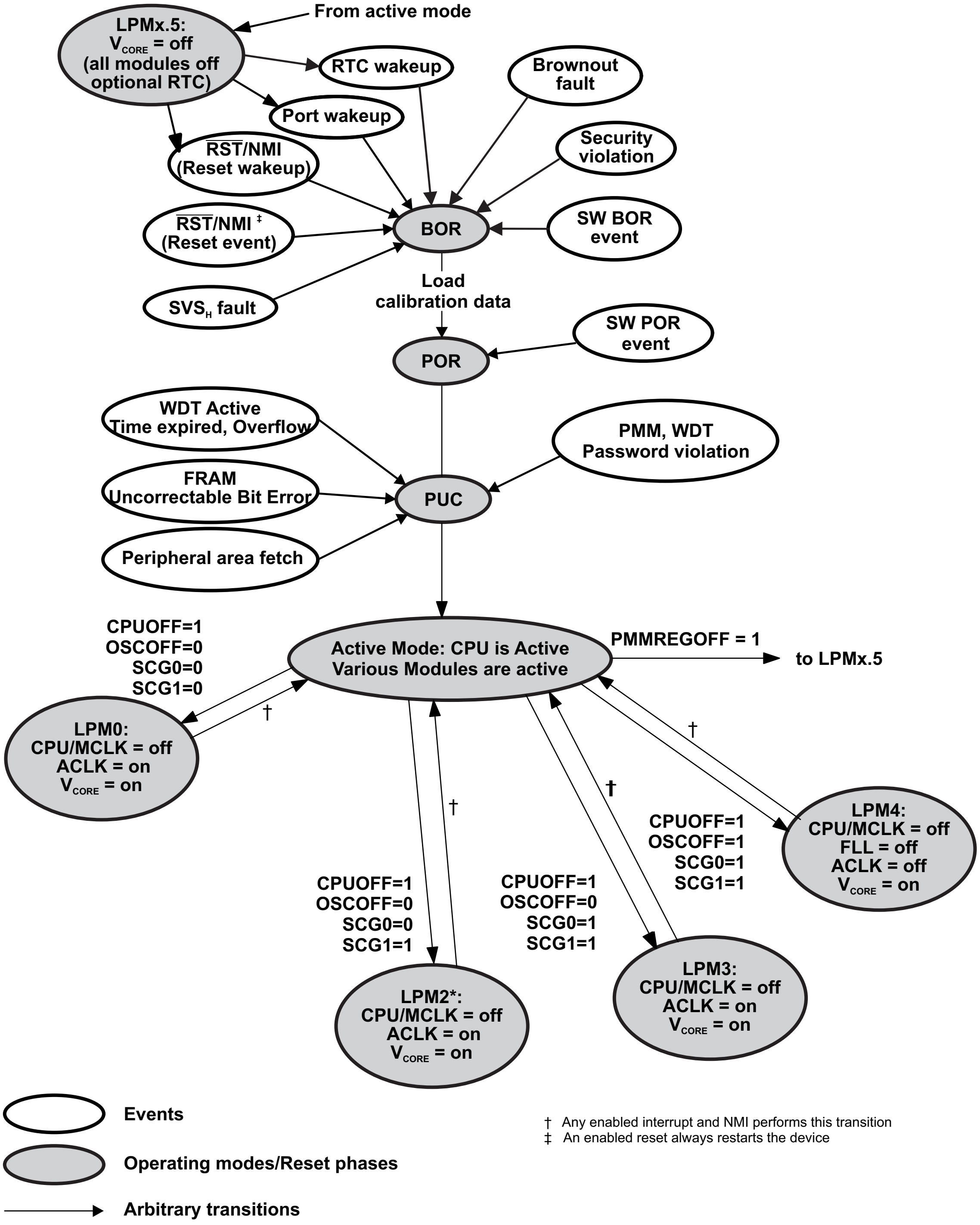


Figure 1-5. Operation Modes

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). FLL is enabled if DCO is enabled.
0	0	0	1	LPM0	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is enabled if DCO is enabled.
1	0	0	1	LPM2 (device specific)	CPU, MCLK, and FLL are disabled. ACLK is active. SMCLK is disabled. FLL is disabled.
1	1	0	1	LPM3	CPU, MCLK, and FLL are disabled. ACLK is active. SMCLK is disabled. FLL 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. RAM retention in backup memory. In this mode, RTC and LCD operation is possible when configured properly. See the RTC and LCD modules 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] LPMx.5 modes are entered by following the correct entry sequence as defined in Section 1.4.2.

[2] The system clocks and the low-power modes can be affected by the clock request system. See the CS chapter for details.

1.4.1 *Low-Power Modes and Clock Requests*

A peripheral module request 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. For details, see Section 3.2.11, *Operation From Low-Power Modes, Requested by Peripherals Modules*.

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

Table 1-3. Requested vs Actual LPM

Requested (SR Bits According to Table 1-2	Actual LPM ...		
	If No Clock Requested	If Only ACLK Requested	If SMCLK Requested
LPM0	LPM0	LPM0	LPM0
LPM2 (device specific)	LPM2	LPM2	LPM0
LPM3	LPM3	LPM3	LPM0
LPM4	LPM4	LPM3	LPM0

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 to return 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. See the device 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 wakeup from LPMx.5.

The wake-up time from LPMx.5 is much longer than the wake-up time from any other power mode (see the device-specific 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, enable any interrupt sources from these modules as wakeup sources, if necessary. See 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) To disable 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 enters LPM3.5 if any module that is connected to the RTC LDO is enabled. The device enters LPM4.5 if none of the modules that are connected to the RTC LDO are enabled.

1.4.3.2 Exit From LPMx.5

The following conditions cause an exit from LPMx.5:

- A wake-up 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 wake-up event from the RTC, if enabled. The corresponding interrupt flag in the RTC is set. The PMMLPM5IFG bit is set.
- A wake-up signal 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 (see the device-specific 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 wakeup 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 (see 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 is serviced as a normal interrupt.

1.4.3.4 Wake up from LPM4.5

Do the following steps after a wakeup 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 is serviced as a normal interrupt.

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

1.4.4 *Extended Time in Low-Power Modes*

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

```
; Enter LPM3 Example with DCO/2 settings (to be updated upon the completion of CS module)
   MOV   #FLLD0+FLLN                          ; Set DCO Output divided by 2
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR      ; Enter LPM3
;  ...                                        ; Program stops
;

;  Interrupt Service Routine
   BIC   #CPUOFF+OSCOFF+SCG1+SCG0,0(SR)       ; Exit LPM3 on RETI
   RETI
```

1.5 Principles for Low-Power Applications

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

Table 1-4 lists the correct termination of unused pins.

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)
$\overline{\text{RST}}$/NMI	DV$_{CC}$ or V$_{CC}$	47-kΩ pullup or internal pullup selected with 10-nF (1.1 nF) pulldown[2]
TDO TDI TMS TCK	Open	The JTAG pins are shared with general-purpose I/O function. If not being used, these should be switched to port function. When used as JTAG pins, these pins should remain open.
TEST	Open	This pin always has an internal pulldown enabled.

[1] For any unused pin with a secondary function that is shared with general-purpose I/O, follow the Px.0 to Px.7 unused pin connection guidelines.

[2] The pulldown capacitor should not exceed 1.1 nF when using devices with Spy-Bi-Wire interface in Spy-Bi-Wire mode with TI tools like FET interfaces or GANG programmers.

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

The reset pin can be configured as a reset function (default) or as an NMI function by the Special Function Register (SFR), SFRRPCR. Setting SYSNMI causes the $\overline{\text{RST}}$/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 $\overline{\text{RST}}$/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 $\overline{\text{RST}}$/NMI pin is unused, it is required to have either the internal pullup selected and enabled or an external resistor connected to the $\overline{\text{RST}}$/NMI pin as shown in Table 1-4.

There is a digital filter that suppresses short pulses on the reset pin to avoid unintended resets of the device. The minimum reset pulse duration is specified in the device-specific data sheet. The filter is active only if the pin is configured in its reset function. It is disabled if the pin is used as external NMI source.

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 4-wire JTAG mode in 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 4-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 4-wire JTAG mode to general-purpose I/O.

1.9 Memory Map – Uses and Abilities

1.9.1 Memory Map

1.9.1.1 MSP430FR4xx Memory Map

This memory map represents the MSP430FR4xx devices. Although the address ranges differ from device to device, overall behavior remains the same.

		Segment erase by user possible	Bank erase by user possible	Mass erase by user possible	Always able to access PMM registers from[1]; Mass erase by user possible	Protectable for read/write accesses	Generates PUC on fetch access	Can generate NMI on read/write/fetch
Address Range	**Name and Usage**	Properties						
00000h-00FFFh	Peripherals with gaps							
00000h-000FFh	Reserved for system extension							
00100h-00FEFh	Peripherals						x	
00FF0h-00FF3h	Descriptor type[2]						x	
00FF4h-00FF7h	Start address of descriptor structure						x	
01800h-019FFh	Information Memory B	x	x	x	x	x		
02000h-027FFh	RAM 2KB							
0C400h-0FFFFh	Program 15KB	x	x[3]	x	x	x		
0FF80h-0FFFFh	Interrupt Vectors							

[1] Access rights are separately programmable for SYS and PMM.

[2] On vacant memory space, the value 03FFFh is driven on the data bus.

[3] Fixed ID for all MSP430 devices. See Section 1.13.1 for further details.

1.9.1.2 MSP430FR2433 Memory Map

This memory map represents the MSP430FR2433 device. Although the address ranges differ from device to device, overall behavior remains the same.

		Segment erase by user possible	Bank erase by user possible	Mass erase by user possible	Always able to access PMM registers from[1]; Mass erase by user possible	Protectable for read/write accesses	Generates PUC on fetch access	Can generate NMI on read/write/fetch
Address Range	**Name and Usage**	Properties						
00000h-00FFFh	Peripherals with gaps							
00000h-000FFh	Reserved for system extension							
00100h-00FEFh	Peripherals						x	
00FF0h-00FF3h	Descriptor type[2]						x	
00FF4h-00FF7h	Start address of descriptor structure						x	
01800h-019FFh	Information Memory B	x	x	x	x	x		
02000h-02FFFh	RAM 4KB							
0C400h-0FFFFh	Program 15KB	x	x[3]	x	x	x		
0FF80h-0FFFFh	Interrupt Vectors							

[1] Access rights are separately programmable for SYS and PMM.

[2] On vacant memory space, the value 03FFFh is driven on the data bus.

[3] Fixed ID for all MSP430 devices. See Section 1.13.1 for further details.

1.9.2 Vacant Memory Space

Vacant memory is nonexistent memory space. Accesses to vacant memory space generate a system (non)maskable interrupt (SNMI), when the interrupt is enabled (VMAIE = 1). Reads from vacant memory result 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, the boot code memory space behaves like vacant memory space and also causes an NMI on access.

1.9.3 FRAM Write Protection

FRAM write protection allows the user to prevent any unwanted write protection to FRAM contents. The SYS module offers two separate write protection.

- User program FRAM protection – always used to store user main program and constant data protected by the PFWP bit in the SYSCFG0 register
- User data FRAM protection – always fixed from 1800h to 19FFh and protected by the DFWP bit in the SYSCFG0 register
- Before FRAM access, the write-protect password must be written togehter with the program or data FRAM protection bit; see the device-specific SYSCFG0 register in Section 1.16 for details.

When write protection is enabled, any write access to the protected FRAM causes an invalid write operation but does not generate an interrupt or reset. TI recommends enabling write protection at the beginning of the user initialization routine. To write date to FRAM, write the data as soon as the write protection is disabled, and then immediately enable write protection again when the write is complete.

> **CAUTION**
>
> To protect the program stored in FRAM from unintended writes, FRAM write protection must be enabled at all times, except when an intentional write operation is performed. The write operation should be completed within as short a time as possible with interrupts disabled to reduce the risk of an unintended write operation.

1.9.4 Bootloader (BSL)

The bootloader (BSL) (formerly known as the bootstrap loader) 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 (FRAM memory), the data memory (RAM), and the peripherals can be modified by the BSL as required. The user can define custom BSL code for FRAM-based devices and protect it against erasure and unintentional or unauthorized access.

On devices without USB, a basic BSL program is provided by TI. This supports the commonly used UART protocol with RS232 interfacing, allowing flexible use of both hardware and software. To use the BSL, a specific BSL entry sequence must be applied to specific device pins. The correct entry sequence causes SYSBSLIND to be set. An added sequence of commands initiates the desired function. A boot-loading session can be exited by continuing operation at a defined user program address or by applying the standard reset sequence.

Access to the device memory by the BSL is protected against misuse by a user-defined password. Devices with USB have a USB based BSL program provided by TI. For more details, see the *MSP430FR4xx and MSP430FR2xx Bootloader (BSL) User's Guide* (SLAU610).

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

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

1.10 JTAG Mailbox (JMB) System

The SYS module provides the capability to exchange user data through the regular JTAG or SBW 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 MSP430 devices of this family and uses only a few or no user application resources. The JTAG interface was chosen because it is available on all MSP430 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)

1.10.1 JMB Configuration

The JMB supports two transfer modes, 16 bit and 32 bit. Set JMBMODE to enable 32-bit transfer mode. Clear JMBMODE to enable 16-bit transfer mode.

1.10.2 SYSJMBO0 and SYSJMBO1 Outgoing Mailbox

Two 16-bit registers are available for outgoing messages to the JTAG/SBW port. SYSJMBO0 is only used when using 16-bit transfer mode (JMBMODE = 0). SYSJMBO1 is used in addition to SYSJMBO0 when using 32-bit transfer mode (JMBMODE = 1). When the application wishes to send a message to the JTAG port, it writes data to SYSJMBO0 for 16-bit mode, or JBOUT0 and JBOUT1 for 32-bit mode.

JMBOUT0FG and JMBOUT1FG are read only flags that indicate the status of SYSJMBO0 and SYSJMBO1, respectively. When JMBOUT0FG is set, SYSJMBO0 has been read by the JTAG port and is ready to receive new data. When JMBOUT0FG is reset, the SYSJMBO0 is not ready to receive new data. JMBOUT1FG behaves similarly.

1.10.3 SYSJMBI0 and SYSJMBI1 Incoming Mailbox

Two 16-bit registers are available for incoming messages from the JTAG port. Only SYSJMBI0 is used in 16-bit transfer mode (JMBMODE = 0). SYSJMBI1 is used in addition to SYSJMBI0 in 32-bit transfer mode (JMBMODE = 1). To send a message to the application, the JTAG port writes data to SYSJMBI0 (for 16-bit mode) or to SYSJMBI0 and SYSJMBI1 (for 32-bit mode).

JMBIN0FG and JMBIN1FG are flags that indicate the status of SYSJMBI0 and SYSJMBI1, respectively. When JMBIN0FG = 1, SYSJMBI0 has data that is available for reading. When JMBIN0FG = 0, no new data is available in SYSJMBI0. JMBIN1FG behaves similarly.

To configure JMBIN0FG and JMBIN1FG to clear automatically, set JMBCLR0OFF = 0 and JMBCLR1OFF = 0, respectively. Otherwise, these flags must be cleared by software.

1.10.4 JMB NMI Usage

To avoid unnecessary polling, the JMB handshake mechanism can be configured to use interrupts. In 16-bit mode, JMBOUTIFG is set when SYSJMBO0 has been read by the JTAG port and is ready to receive data. In 32-bit mode, JMBOUTIFG is set when both SYSJMBO0 and SYSJMBO1 have 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 SYSJMBO0. In 32-bit mode, JMBOUTIFG is cleared automatically when data is written to both SYSJMBO0 and SYSJMBO1. In addition, the JMBOUTIFG can be cleared when reading SYSSNIV. Clearing JMBOUTIE disables the NMI interrupt.

In 16-bit mode, JMBINIFG is set when SYSJMBI0 is available for reading. In 32-bit mode, JMBINIFG is set when both SYSJMBI0 and SYSJMBI1 are available for reading. If JMBOUTIE is set, these events cause a system NMI. In 16-bit mode, JMBINIFG is cleared automatically when SYSJMBI0 is read. In 32-bit mode, JMBINIFG Is cleared automatically when both SYSJMBI0 and SYSJMBI1 are read. In addition, the JMBINIFG can be cleared when reading SYSSNIV. Clearing JMBINIE disables the NMI interrupt.

1.11 Device Security

This section describes options for securing the device to prevent unauthorized access from JTAG/SBW or BSL to the device memory. See Table 1-5 for a summary of security options.

Table 1-5. BSL and JTAG/SBW Signatures

Name	Addresses	Value	Device Security
BSL Password	FFE0h-FFFFh	User defined + Vector table configuration	This password must be provided by the BSL host before the device is accessible by the BSL.
BSL Signature	FF84h-FF87h	5555_5555h	BSL is disabled.
		AAAA_AAAAh	BSL is password protected. Mass erase on wrong BSL password feature disabled.
		Any other value	BSL is password protected. Mass erase on wrong BSL password feature enabled.
JTAG/SBW Signature	FF80h-FF83h	FFFF_FFFFh	JTAG/SBW is unlocked.
		0000_0000h	
		Any other value	JTAG/SBW is locked.

1.11.1 JTAG and SBW Lock Mechanism (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 remains functional, but JTAG commands that give direct access into the device are completely disabled. Locking the device requires the programming of two signatures 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, TI 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.

A device can be locked by writing any value other than 0000h or FFFFh 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 0000h or FFFFh. Some JTAG commands are still possible when 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 1.10.4 for details).

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

1.11.2 BSL Security Mechanism

Two BSL signatures, BSL Signature 1 (memory location FF84h) and BSL Signature 2 (memory location FF86h) reside in FRAM and can be used to control the behavior of the BSL. Writing 5555h to BSL Signature 1 and 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 AAAAh to both BSL Signature 1 and 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 With the Bootloader (BSL) User's Guide* (SLAU319) and the *MSP430FR4xx and MSP430FR2xx Bootloader (BSL) User's Guide* (SLAU610).

1.12 Device-Specific Configurations

The following sections describe the device-specific configurations.

Section 1.12.1	MSP430FR413x and MSP430FR203x Configurations
Section 1.12.2	MSP430FR2433 Configurations

1.12.1 MSP430FR413x and MSP430FR203x Configurations

This section describes the configurations that are specific to MSP430FR413x and MSP430FR203x devices.

1.12.1.1 FRAM Write Protection

The FRAM protection allows users to protect user code and data from accidental write operation. The write operation to main code FRAM and information FRAM are protected by the PFWP and DFWP bits, respectively, in the SYSCFG0 register. After a PUC reset, both bits default to 1 and writes to FRAM are disabled. User code must clear the corresponding bit before write operation. See Section 1.16.2.1 for MSP430FR413x devices and Section 1.16.1.1 for MSP430FR203x devices.

1.12.1.2 Infrared Modulation Function

The SYS module includes IR modulation logic that the device can use to easily generate accurately modulated IR waveforms, such as RC-5 data format, directly on a external output pin. Figure 1-6 shows the detailed of the circuitry implementation. Set the IREN bit in the SYSCFG1 register to enable the logic. If IREN is cleared, this function is bypassed and the external pin defaults to general-purpose I/O.

This function has two different PWM input signals to support either ASK or FSK modulations. In ASK modulation, the first PWM is used for carrier generation and the second generates the envelope. In FSK modulation, the first PWM and the second PWM represent the two different offset frequencies. The IRMSEL bit in SYSCFG1 register specifies the selected mode. Before the modulated data is output to the external pin, the signal can be inverted by setting the IRPSEL bit in SYSCFG1 register for adapting to different external drive circuitry.

The IR modulation function can be used with data generated by either hardware or software. In hardware data generation, the data comes from eUSCI_A and the 8-bit data is automatically serially sent. In software data generation, IRDATA bit in SYSCFG1 register is used to control the logic 0 or 1 to be sent. The IRDSSEL bit in SYSCFG1 registers control the data flow from hardware or firmware.

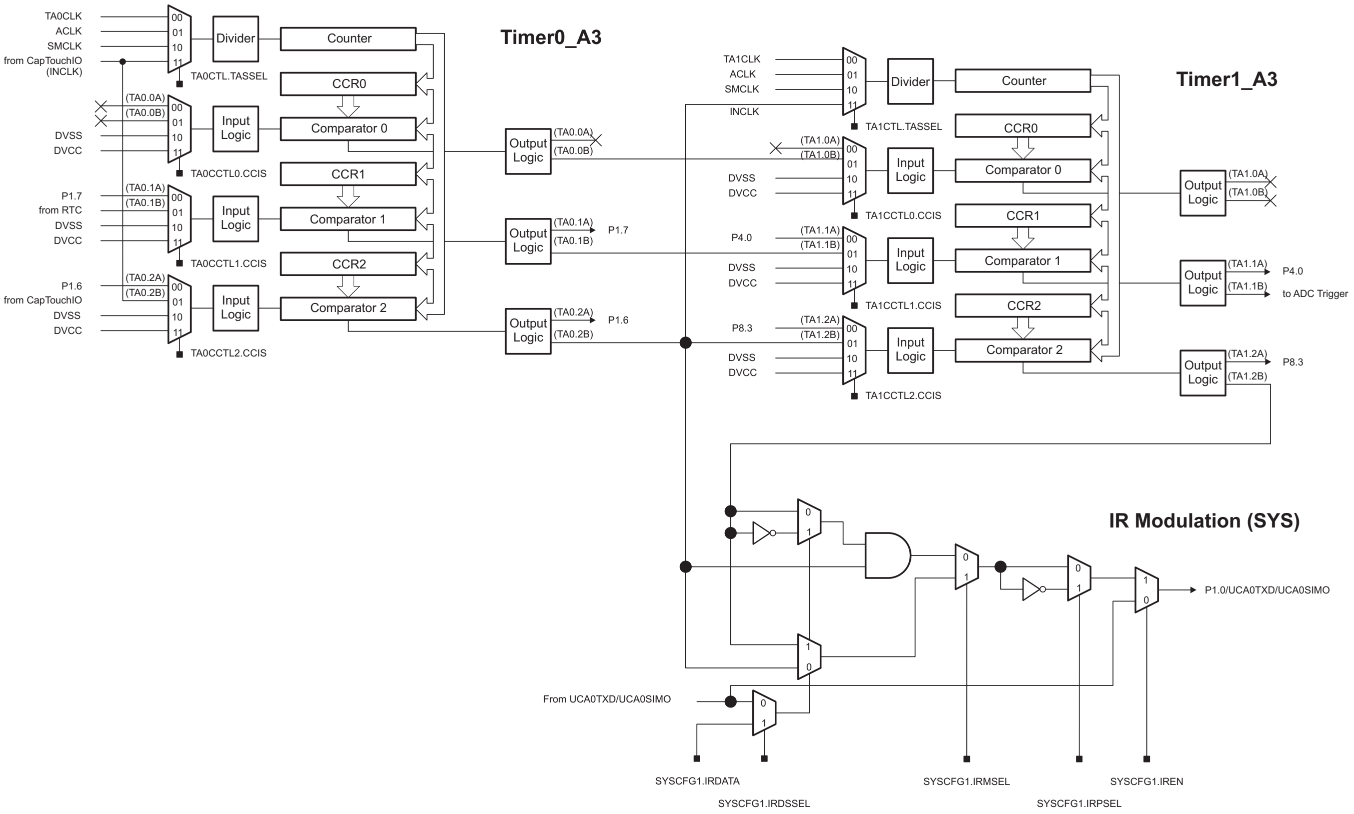


Figure 1-6. IR Modulation Combinatory Logics

1.12.1.3 ADC Pin Enable and 1.2-V Reference Settings

ADC pins are multiplexed with I/O functions. When the ADC channel is used, the I/O function must be disabled to avoid function conflicts over these pins: A0 to A11. Set the ADCPTCLx bit in the SYSCFG2 register to disable the I/O functions. See Section 1.16.2.3 for MSP430FR413x devices and Section 1.16.1.3 for MSP430FR203x devices.

When ADC A4 channel is enabled, the 1.2-V on-chip reference can be output to P1.4 by setting PMM registers. See Figure 1-7 and the PMM Chapter.

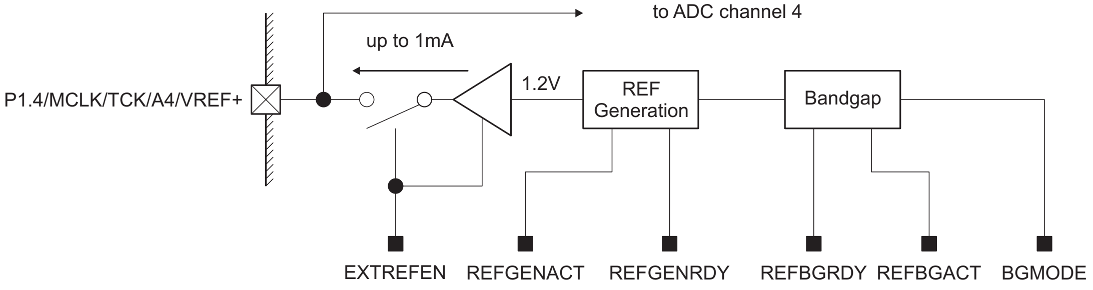


Figure 1-7. 1.2-V Reference Output on A4

1.12.1.4 LCD Power Pin Enable

In MSP430FR413x devices, LCD power pins are multiplexed with I/O functions. When LCD is used, the I/O function must be disabled to avoid function conflicts on these pins: LCDCAP0, LCDCAP1, R13, R23, R33. Set the LCDPCTL bit in SYSCFG2 register to disable the I/O functions and enable the LCD power functions (see Section 1.16.2.3).

1.12.2 MSP430FR2433 Configurations

This section describes the configurations that are specific to MSP430FR2433 device.

1.12.2.1 FRAM Write Protection

The FRAM protection allows users to protect user code and data from accidental write operation. The write operation to main code FRAM and information FRAM are protected by the PFWP and DFWP bits, respectively, in the SYSCFG0 register. After a PUC reset, both bits default to 1 and writes to FRAM are disabled. User code must write correct password and clear the corresponding bit before write operation. See Section 1.16.3.1 for MSP430FR2433 device.

1.12.2.2 Infrared Modulation Function

The SYS module includes IR modulation logic that can easily generate accurately modulated IR waveforms, such as RC-5 data format, directly on a external output pin. Figure 1-8 shows the detailed of the circuitry implementation. Set the IREN bit in the SYSCFG1 register to enable the logic. If IREN is cleared, this function is bypassed, and the external pin defaults to general-purpose I/O.

This function has two different PWM input signals to support either ASK or FSK modulations. In ASK modulation, the first PWM is used for carrier generation and the second generates the envelope. In FSK modulation, the first PWM and the second PWM represent the two different offset frequencies. The IRMSEL bit in SYSCFG1 register specifies the selected mode. Before the modulated data is output to the external pin, the signal can be inverted by setting the IRPSEL bit in SYSCFG1 register for adapting to different external drive circuitry.

The IR modulation function can be used with data generated by either hardware or software. In hardware data generation, the data comes from eUSCI_A or eUSCI_B and the 8-bit data is automatically serially sent. In software data generation, IRDATA bit in SYSCFG1 register is used to control the logic 0 or 1 to be sent. The IRDSSEL bit in SYSCFG1 registers control the data flow from hardware or firmware.

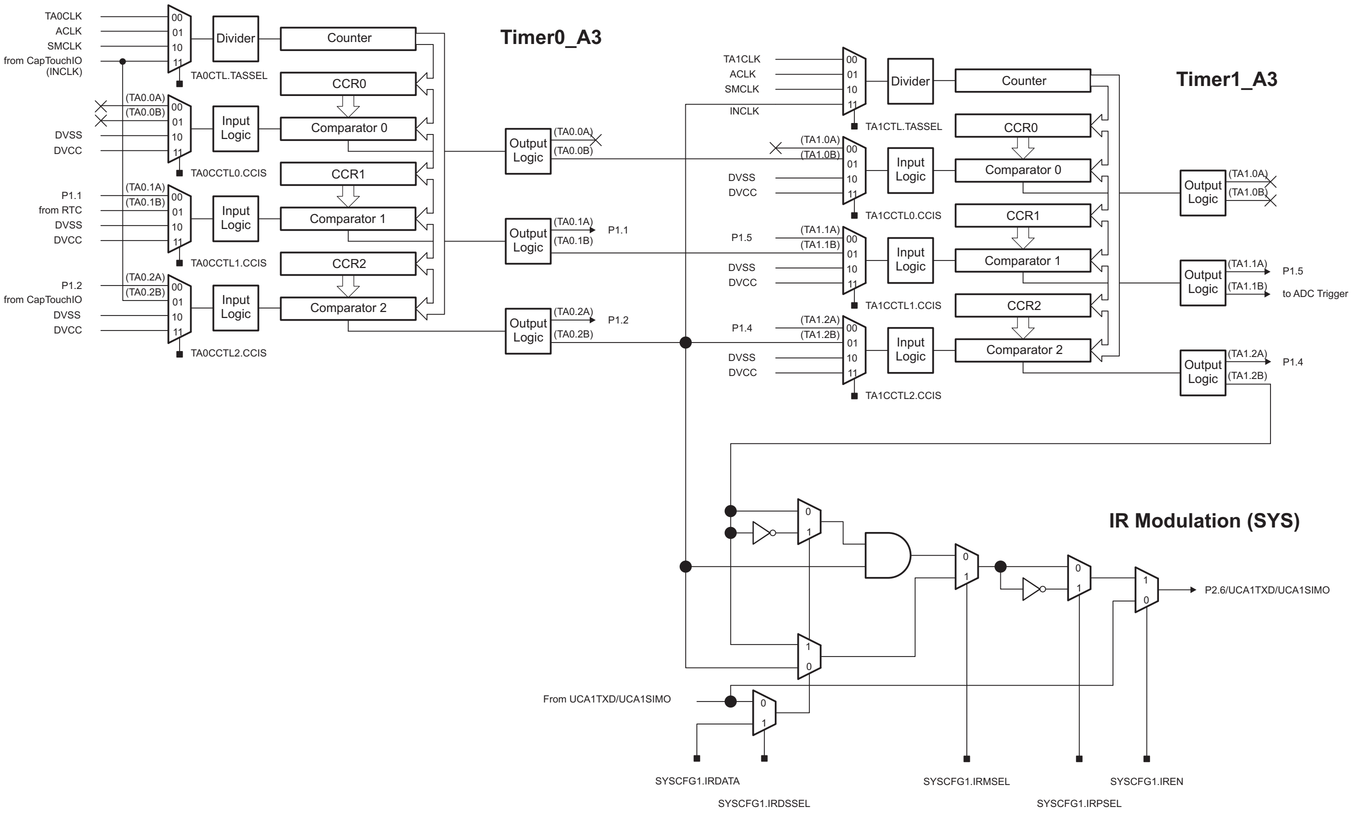


Figure 1-8. IR Modulation Combinatory Logics

1.12.2.3 ADC Pin Enable and 1.2-V Reference Settings

ADC pins are multiplexed with I/O functions. When the ADC channel is used, the I/O function must be disabled to avoid function conflicts over these pins: A0 to A11. Set the ADCPTCLx bit in the SYSCFG2 register to disable the I/O functions. See Section 1.16.3.3 for the MSP430FR2433 device.

When the A4 channel of the ADC is enabled, the 1.2-V on-chip reference can be output to P1.4 by setting PMM registers (see Figure 1-9 and the PMM Chapter).

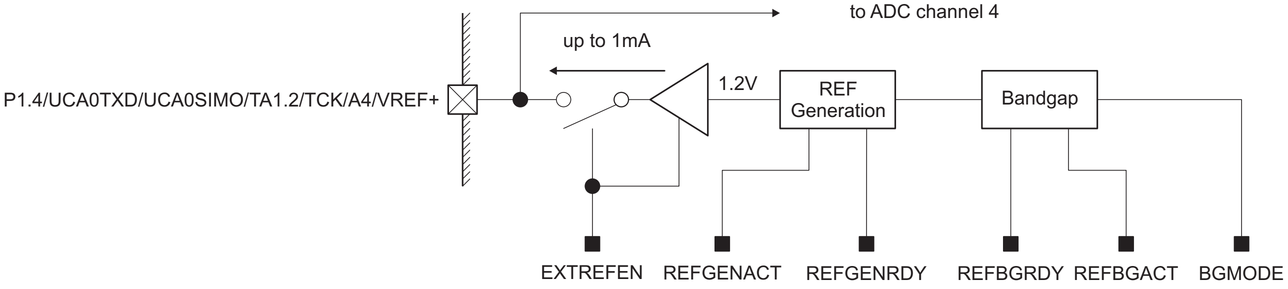


Figure 1-9. 1.2-V Reference Output on A4

1.13 Device Descriptor Table

Each device provides a data structure in memory that allows an unambiguous identification of the device as well as a description of the available modules on a given device. SYS provides this information and can be used by device-adaptive software tools and libraries to clearly identify a particular device and all of its modules and capabilities. The validity of the device descriptor can be verified by cyclic redundancy check (CRC). The CRC checksum covers a device-specific TLV range. See the TLV table in the device-specific data sheet for the definitions. Figure 1-10 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.

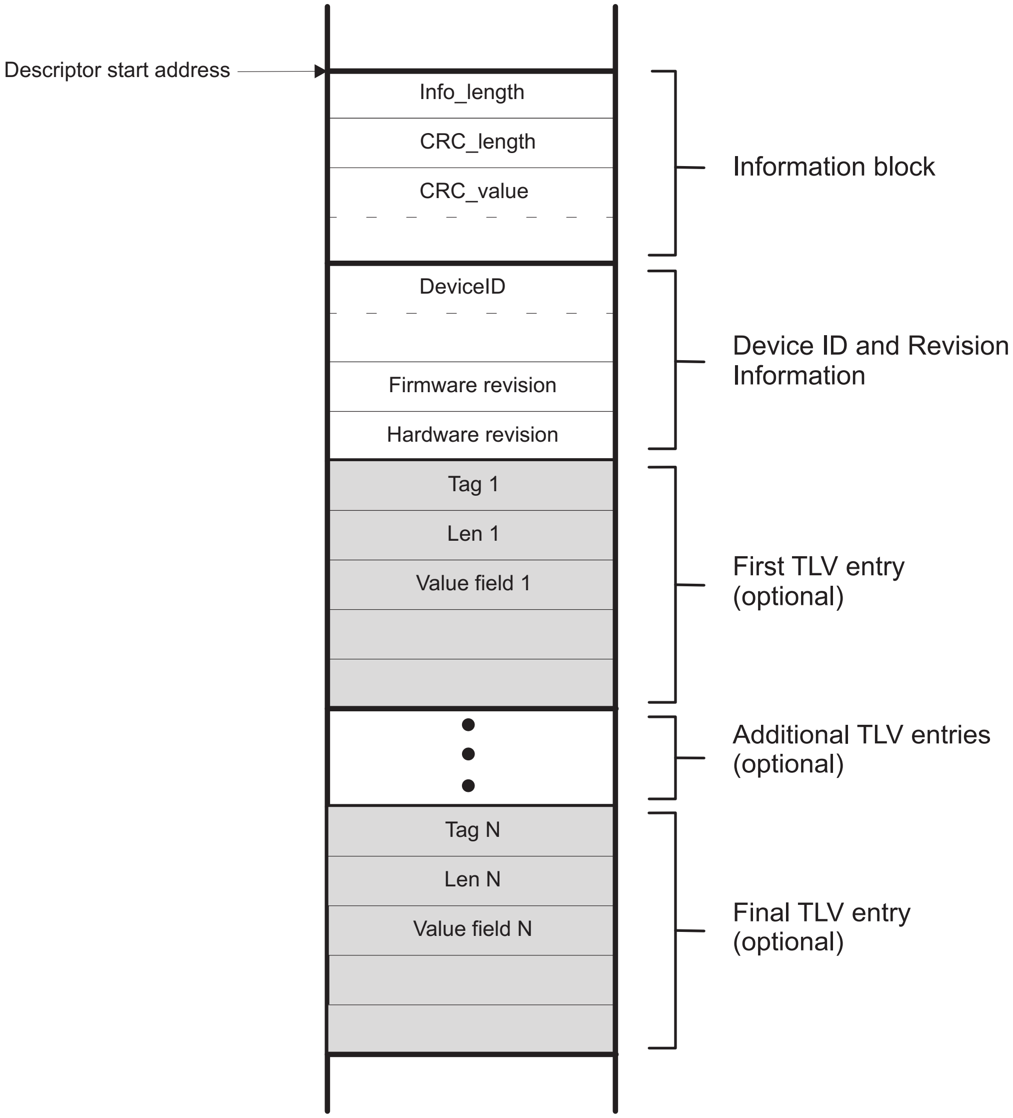


Figure 1-10. Devices Descriptor Table

1.13.1 Identifying Device Type

The value at address location 00FF0h identifies the family branch of the device. All values starting with 80h indicate a hierarchical structure that consists of the information block and a tag-length-value (TLV) structure with the various descriptors. Any value other than 80h at address location 00FF0h indicates that the device is of an older family and contains a flat descriptor beginning at location 0FF0h. The information block, shown in Figure 1-10 contains the device ID, die revisions, firmware revisions, and other manufacturer and tool related information. The descriptors contains information about the available peripherals and their subtypes and addresses and provides the information required to build adaptive hardware drivers for operating systems.

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

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

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

1.13.2 TLV Descriptors

The TLV descriptors follow the information block. Because the information block is always a fixed length, the start location of the TLV descriptors is fixed for a given device family. 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 that identifies the descriptor type. Table 1-6 shows the currently supported tags.

Table 1-6. Tag Values

Short Name	Value	Description
LDTAG	01h	Legacy descriptor
PDTAG	02h	Peripheral discovery descriptor
Reserved	03h	Future use
Reserved	04h	Future use
BLANK	05h	Blank descriptor
Reserved	06h	Future use
ADCCAL	11h	ADC calibration
REFCAL	12h	REF calibration
Reserved	13h-FDh	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, which is written in 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
}
```

1.13.3 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.

1.13.3.1 1.5-V Reference Calibration

The calibration data consists a word for reference voltage available (1.5 V). The reference voltages are measured at room temperature. The measured values are normalized by 1.5 V before being stored into the TLV structure:

$$Factor_{gain\_1.5Vref} = \frac{V_{REF+}}{1.5V} \times 2^{15} \tag{2}$$

In this way, a conversion result is corrected by multiplying it with the *Factor*$_{gain\_1.5Vref}$ and dividing the result by 2^{15} as shown for each of the respective reference voltages:

$$ADC_{calibrated} = ADC_{raw} \times Factor_{gain\_1.5Vref} \times \frac{1}{2^{15}} \tag{3}$$

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

- Conversion result: 0x0100 = 256 decimal
- Reference voltage calibration factor (*Factor*$_{gain\_1.5Vref}$) : 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 × 0x0002 = 0x0200
- Multiply the result by *Factor*$_{gain\_1.5Vref}$: 0x200 × 0x7BBB = 0x00F7_7600
- Divide the result by 2^{16}: 0x00F7_7600 / 0x0001_0000 = 0x0000_00F7 = 247 decimal

1.13.3.2 ADC Offset and Gain Calibration

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

$$ADC_{offset\_calibrated} = ADC_{raw} + ADC_{offset} \tag{4}$$

The gain factor of the ADC is calculated by Equation 5:

$$Factor_{gain} = \frac{1}{Gain} \times 2^{15} \tag{5}$$

The conversion result is gain corrected by multiplying it with the *Factor*$_{gain}$ and dividing the result by 2^{15}:

$$ADC_{gain\_calibrated} = ADC_{raw} \times Factor_{gain} \times \frac{1}{2^{15}} \tag{6}$$

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

$$ADC_{calibrated} = ADC_{raw} \times Factor_{gain} \times \frac{1}{2^{15}} + ADC_{offset} \tag{7}$$

1.13.3.3 Temperature Sensor Calibration

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

$$V_{sense} = TC_{sensor} \times Temperature + V_{sensor} \tag{8}$$

The temperature coefficient, *TC*$_{SENSOR}$, in mV/°C, represents the slope of the equation. *V*$_{SENSOR}$, in mV, represents the y-intercept of the equation. *Temp*, in °C, is the temperature of interest.

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

$$Temperature = (ADC_{raw} - ADC_{30^{o}C\_1.5Vref}) \times \left(\frac{55^{o}C}{ADC_{85^{o}C\_1.5Vref} - ADC_{30^{o}C\_1.5Vref}} \right) + 30^{o}C \tag{9}$$

1.13.3.4 DCO Calibration

The DCO calibration is stored for a quick setting to maximum DCO frequency (for example, 16 MHz) at room temperature. Loading this value to the CSCTL0 register significantly reduces the FLL lock time when the MCU reboot or exits from a low-power mode. If a possible frequency overshoot caused by temperature drift is expected after exit from an LPM, TI recommends dividing the DCO frequency before use. For more details, see Section 1.4.4.

1.14 SFR Registers

The SFRs are listed in Table 1-8. The base address for the SFRs is listed in Table 1-7. Many of the bits in the SFRs are described in other chapters throughout this user's guide. These bits are marked with a note and a reference. See the module-specific chapter 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 1-7. SFR Base Address

Module	Base Address
SFR	00100h

Table 1-8. SFR Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	SFRIE1	Interrupt Enable	Read/write	Word	0000h	Section 1.14.1
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 1.14.2
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 1.14.3
04h	SFRRPCR_L		Read/write	Byte	1Ch	
05h	SFRRPCR_H		Read/write	Byte	00h	

1.14.1 SFRIE1 Register (offset = 00h) [reset = 0000h]

Interrupt Enable Register

Figure 1-11. 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
rw-0	rw-0	r0	rw-0	rw-0	r0	rw-0	rw-0

[1] See the CS chapter for details.

Table 1-9. 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 flag 0b = Interrupts disabled 1b = Interrupts enabled
6	JMBINIE	RW	0h	JTAG mailbox input interrupt enable flag 0b = Interrupts disabled 1b = Interrupts enabled
5	Reserved	R	0h	Reserved. Always reads as 0.
4	NMIIE	RW	0h	NMI pin interrupt enable flag 0b = Interrupts disabled 1b = Interrupts enabled
3	VMAIE	RW	0h	Vacant memory access interrupt enable flag 0b = Interrupts disabled 1b = Interrupts enabled
2	Reserved	R	0h	Reserved. Always reads as 0.
1	OFIE	RW	0h	Oscillator fault interrupt enable flag 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

1.14.2 SFRIFG1 Register (offset = 02h) [reset = 0082h]

Interrupt Flag Register

Figure 1-12. 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
rw-(1)	rw-(0)	r0	rw-0	rw-0	r0	rw-(1)	rw-0

[1] See the CS chapter for details.

Table 1-10. 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
0	WDTIFG	RW	0h	Watchdog timer interrupt flag. In watchdog mode, WDTIFG self clears 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

1.14.3 SFRRPCR Register (offset = 04h) [reset = 001Ch]

Reset Pin Control Register

Figure 1-13. 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			SYSFLTE	SYSRSTRE	SYSRSTUP	SYSNMIIES	SYSNMI
r0	r0	r0	rw-1	rw-1	rw-1	rw-0	rw-0

Table 1-11. SFRRPCR Register Description

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Always reads as 0.
4	SYSFLTE	RW	1h	Reset pin filter enable 0b = Digital filter on reset pin is disabled 1b = Digital filter on reset pin is enabled
3	SYSRSTRE	RW	1h	Reset pin resistor enable 0b = Pullup/pulldown resistor at the $\overline{\text{RST}}$/NMI pin is disabled 1b = Pullup/pulldown resistor at the $\overline{\text{RST}}$/NMI pin is enabled
2	SYSRSTUP	RW	1h	Reset resistor pin pullup/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

1.15 SYS Registers

The SYS registers are listed in Table 1-12. A detailed description of each register and its bits is provided in the following sections. 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 1-12. SYS Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	SYSCTL	System Control	Read/write	Word	0000h	Section 1.15.1
00h	SYSCTL_L		Read/write	Byte	00h	
01h	SYSCTL_H		Read/write	Byte	00h	
02h	SYSBSLC	Bootloader Configuration	Read/write	Word	0000h	Section 1.15.2
02h	SYSBSLC_L		Read/write	Byte	00h	
03h	SYSBSLC_H		Read/write	Byte	00h	
06h	SYSJMBC	JTAG Mailbox Control	Read/write	Word	000Ch	Section 1.15.3
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 1.15.4
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 1.15.5
0Ah	SYSJMBI1_L		Read/write	Byte	00h	
0Bh	SYSJMBI1_H		Read/write	Byte	00h	
0Ch	SYSJMBO0	JTAG Mailbox Output 0	Read/write	Word	0000h	Section 1.15.6
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 1.15.7
0Eh	SYSJMBO1_L		Read/write	Byte	00h	
0Fh	SYSJMBO1_H		Read/write	Byte	00h	
1Ah	SYSUNIV	User NMI Vector Generator	Read	Word	0000h	Section 1.15.8
1Ch	SYSSNIV	System NMI Vector Generator	Read	Word	0000h	Section 1.15.9
1Eh	SYSRSTIV	Reset Vector Generator	Read	Word	0002h	Section 1.15.10

1.15.1 SYSCTL Register (offset = 00h) [reset = 0000h]

SYS Control Register

Figure 1-14. 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]	rw-[0]	r0	rw-[0]	r0	rw-[0]

Table 1-13. 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 digital functionality of the JTAG pins and permanently enables the JTAG function. This bit can only be set once. After the bit is set, it remains set until a BOR occurs. 0b = Shared JTAG pins (JTAG mode selectable by JTAG/SBW sequence) 1b = Dedicated JTAG pins (explicit 4-wire JTAG mode selection)
4	SYSBSLIND	RW	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. After the bit is set to 1, it only can be cleared by a BOR. 0b = Access from anywhere in memory 1b = Access only from the protected 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 64KB of FRAM FFFFh 1b = Interrupt vectors generated with end address TOP of RAM

1.15.2 SYSBSLC Register (offset = 02h) [reset = 0000h]

Bootloader Configuration Register

Figure 1-15. SYSBSLC Register

15	14	13	12	11	10	9	8
SYSBSLPE	SYSBSLOFF	Reserved					
rw-[0]	rw-[0]	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved					SYSBSLR	Reserved	
r0	r0	r0	r0	r0	rw-[0]	r0	r0

Table 1-14. SYSBSLC Register Description

Bit	Field	Type	Reset	Description
15	SYSBSLPE	RW	0h	Bootloader memory protection enable. By default, this bit is cleared by hardware with a BOR event (as indicated above); however, the boot code that checks for an available BSL may set this bit in software to protect the BSL. Because devices normally come with a TI BSL preprogrammed and protected, the boot code sets this bit. 0b = Area not protected. Read, program, and erase of BSL memory is possible. 1b = Area protected
14	SYSBSLOFF	RW	0h	Bootloader memory disable 0b = BSL memory is addressed when this area is read. 1b = BSL memory behaves like vacant memory. Reads cause 3FFFh to be read. Fetches cause JMP $ to be executed.
13-3	Reserved	R	0h	Reserved. Always reads as 0.
2	SYSBSLR	RW	0h	RAM assigned to BSL 0b = No RAM assigned to BSL area 1b = Lowest 16 bytes of RAM assigned to BSL
1-0	Reserved	R	0h	Reserved. Always reads as 0.

1.15.3 SYSJMBC Register (offset = 06h) [reset = 000Ch]

JTAG Mailbox Control Register

Figure 1-16. 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	JMBMODE	JMBOUT1FG	JMBOUT0FG	JMBIN1FG	JMBIN0FG
rw-(0)	rw-(0)	r0	rw-0	r-(1)	r-(1)	rw-(0)	rw-(0)

Table 1-15. 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 SYSJMBI1 register 1b = JMBIN1FG cleared by software
6	JMBCLR0OFF	RW	0h	Incoming JTAG Mailbox 0 flag auto-clear disable 0b = JMBIN0FG cleared on read of SYSJMBI0 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 SYSJMBI0, SYSJMBI1, SYSJMBO0, and SYSJMBO1. Before changing this bit, pad and flush out any partial content to avoid data drops. 0b = 16-bit transfers using SYSJMBO0 and SYSJMBI0 only 1b = 32-bit transfers using SYSJMBI0, SYSJMBI1, SYSJMBO0, and SYSJMBO1
3	JMBOUT1FG	RW	1h	Outgoing JTAG Mailbox 1 flag. This bit is cleared automatically when a message is written to the upper byte of SYSJMBO1 or as word access (by the CPU or other source) and is set after the message is read by JTAG. 0b = SYSJMBO1 is not ready to receive new data. 1b = SYSJMBO1 is ready to receive new data.
2	JMBOUT0FG	RW	1h	Outgoing JTAG Mailbox 0 flag. This bit is cleared automatically when a message is written to the upper byte of SYSJMBO0 or as word access (by the CPU or other source) and is set after the message is read by JTAG. 0b = SYSJMBO0 is not ready to receive new data. 1b = SYSJMBO0 is ready to receive new data.
1	JMBIN1FG	RW	0h	Incoming JTAG Mailbox 1 flag. This bit is set when a new message (provided by JTAG) is available in SYSJMBI1. This flag is cleared automatically on read of SYSJMBI1 when JMBCLR1OFF = 0 (auto clear mode). If JMBCLR1OFF = 1, JMBIN1FG must be cleared by software. 0b = SYSJMBI1 has no new data 1b = SYSJMBI1 has new data available
0	JMBIN0FG	RW	0h	Incoming JTAG Mailbox 0 flag. This bit is set when a new message (provided by JTAG) is available in SYSJMBI0. This flag is cleared automatically on read of SYSJMBI0 when JMBCLR0OFF = 0 (auto clear mode). If JMBCLR0OFF = 1, JMBIN0FG must be cleared by software. 0b = SYSJMBI1 has no new data 1b = SYSJMBI1 has new data available

1.15.4 SYSJMBI0 Register (offset = 08h) [reset = 0000h]

JTAG Mailbox Input 0 Register

Figure 1-17. SYSJMBI0 Register

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

Table 1-16. SYSJMBI0 Register Description

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

1.15.5 SYSJMBI1 Register (offset = 0Ah) [reset = 0000h]

JTAG Mailbox Input 1 Register

Figure 1-18. SYSJMBI1 Register

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

Table 1-17. SYSJMBI1 Register Description

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

1.15.6 SYSJMBO0 Register (offset = 0Ch) [reset = 0000h]

JTAG Mailbox Output 0 Register

Figure 1-19. 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 1-18. 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

1.15.7 SYSJMBO1 Register (offset = 0Eh) [reset = 0000h]

JTAG Mailbox Output 1 Register

Figure 1-20. 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 1-19. 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

1.15.8 SYSUNIV Register (offset = 1Ah) [reset = 0000h]

User NMI Vector Register

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

Figure 1-21. 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 1-20. 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.

1.15.9 SYSSNIV Register (offset = 1Ch) [reset = 0000h]

System NMI Vector Register

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

Figure 1-22. 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 1-21. 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.

1.15.10 SYSRSTIV Register (offset = 1Eh) [reset = 0002h]

Reset Interrupt Vector Register

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

Figure 1-23. 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-0	r-0	r-0	r-0	r-1	r0

Table 1-22. SYSRSTIV Register Description

Bit	Field	Type	Reset	Description
15-0	SYSRSTIV	R	0h	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, or PUC). Writing to this register clears all pending reset source flags. See the device-specific data sheet for a list of values.

1.16 System Configuration Registers

The system configuration registers are device specific and are only applicable to the specific device family. Each register starts at a word boundary. Either word or byte data can be written to the SYS configuration registers.

For the MSP430FR203x configuration registers, see Section 1.16.1.

For the MSP430FR413x configuration registers, see Section 1.16.2.

For the MSP430FR2433 configuration registers, see Section 1.16.3.

1.16.1 MSP430FR203x System Configuration Registers

All registers are listed in Table 1-23. A detailed description of each register and its bits is provided in the following sections.

Table 1-23. MSP430FR203x SYS Configuration Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
20h	SYSCFG0	System Configuration 0	Read/write	Word	0003h	Section 1.16.1.1
20h	SYSCFG0_L		Read/write	Byte	03h	
21h	SYSCFG0_H		Read/write	Byte	00h	
22h	SYSCFG1	System Configuration 1	Read/write	Word	0000h	Section 1.16.1.2
22h	SYSCFG1_L		Read/write	Byte	00h	
23h	SYSCFG1_H		Read/write	Byte	00h	
24h	SYSCFG2	System Configuration 2	Read/write	Word	0000h	Section 1.16.1.3
24h	SYSCFG2_L		Read/write	Byte	00h	
25h	SYSCFG2_H		Read/write	Byte	00h	

1.16.1.1 MSP430FR203x SYSCFG0 Register (offset = 00h) [reset = 0003h]

System Configuration Register 0

Figure 1-24. SYSCFG0 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						DFWP	PFWP
r0	r0	r0	r0	r0	r0	rw-1	rw-1

Table 1-24. SYSCFG0 Register Description

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved. Always read as 0.
1	DFWP	RW	1h	Data FRAM write protection 0b = Data FRAM write enable 1b = Data FRAM write protected (not writable)
0	PFWP	RW	1h	Program FRAM write protection 0b = Program FRAM write enable 1b = Program FRAM write protected (not writable)

1.16.1.2 MSP430FR203x SYSCFG1 Register (offset = 02h) [reset = 0000h]

System Configuration Register 1

Figure 1-25. SYSCFG1 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			IRDATA	IRDSSEL	IRMSEL	IRPSEL	IREN
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-25. SYSCFG1 Register Description

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Always read as 0.
4	IRDATA	RW	0h	Infrared data 0b = Infrared data logic 0 1b = Infrared data logic 1
3	IRDSSEL	RW	0h	Infrared data source select 0b = From hardware peripherals upon device configuration 1b = From IRDATA bit
2	IRMSEL	RW	0h	Infrared mode select 0b = ASK mode 1b = FSK mode
1	IRPSEL	RW	0h	Infrared polarity select 0b = Normal polarity 1b = Inverted polarity
0	IREN	RW	0h	Infrared enable 0b = Infrared function disabled 1b = Infrared function enabled

1.16.1.3 MSP430FR203x SYSCFG2 Register (offset = 04h) [reset = 0000h]

System Configuration Register 2

Figure 1-26. SYSCFG2 Register

15	14	13	12	11	10	9	8
Reserved						ADCPCTL9	ADCPCTL8
r0	r0	r0	r0	r0	r0	rw-0	rw-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADCPCTL7	ADCPCTL6	ADCPCTL5	ADCPCTL4	ADCPCTL3	ADCPCTL2	ADCPCTL1	ADCPCTL0
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-26. SYSCFG2 Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always read as 0.
9	ADCPCTL9	RW	0h	ADC input A9 pin select 0b = ADC input A9 disabled 1b = ADC input A9 enabled
8	ADCPCTL8	RW	0h	ADC input A8 pin select 0b = ADC input A8 disabled 1b = ADC input A8 enabled
7	ADCPCTL7	RW	0h	ADC input A7 pin select 0b = ADC input A7 disabled 1b = ADC input A7 enabled
6	ADCPCTL6	RW	0h	ADC input A6 pin select 0b = ADC input A6 disabled 1b = ADC input A6 enabled
5	ADCPCTL5	RW	0h	ADC input A5 pin select 0b = ADC input A5 disabled 1b = ADC input A5 enabled
4	ADCPCTL4	RW	0h	ADC input A4 pin select 0b = ADC input A4 disabled 1b = ADC input A4 enabled
3	ADCPCTL3	RW	0h	ADC input A3 pin select 0b = ADC input A3 disabled 1b = ADC input A3 enabled
2	ADCPCTL2	RW	0h	ADC input A2 pin select 0b = ADC input A2 disabled 1b = ADC input A2 enabled
1	ADCPCTL1	RW	0h	ADC input A1 pin select 0b = ADC input A1 disabled 1b = ADC input A1 enabled
0	ADCPCTL0	RW	0h	ADC input A0 pin select 0b = ADC input A0 disabled 1b = ADC input A0 enabled

1.16.2 MSP430FR413x System Configuration Registers

All registers are listed in Table 1-27. A detailed description of each register and its bits is provided in the following sections.

Table 1-27. FR413x SYS Configuration Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
20h	SYSCFG0	System Configuration 0	Read/write	Word	0003h	Section 1.16.2.1
20h	SYSCFG0_L		Read/write	Byte	03h	
21h	SYSCFG0_H		Read/write	Byte	00h	
22h	SYSCFG1	System Configuration 1	Read/write	Word	0000h	Section 1.16.2.2
22h	SYSCFG1_L		Read/write	Byte	00h	
23h	SYSCFG1_H		Read/write	Byte	00h	
24h	SYSCFG2	System Configuration 2	Read/write	Word	0000h	Section 1.16.2.3
24h	SYSCFG2_L		Read/write	Byte	00h	
25h	SYSCFG2_H		Read/write	Byte	00h	

1.16.2.1 MSP430FR413x SYSCFG0 Register (offset = 00h) [reset = 0003h]

System Configuration Register 0

Figure 1-27. SYSCFG0 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						DFWP	PFWP
r0	r0	r0	r0	r0	r0	rw-1	rw-1

Table 1-28. SYSCFG0 Register Description

Bit	Field	Type	Reset	Description
15-2	Reserved	R	0h	Reserved. Always read as 0.
1	DFWP	RW	1h	Data FRAM write protection 0b = Data FRAM write enable 1b = Data FRAM write protected (not writable)
0	PFWP	RW	1h	Program FRAM write protection 0b = Program FRAM write enable 1b = Program FRAM write protected (not writable)

1.16.2.2 MSP430FR413x SYSCFG1 Register (offset = 02h) [reset = 0000h]

System Configuration Register 1

Figure 1-28. SYSCFG1 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			IRDATA	IRDSSEL	IRMSEL	IRPSEL	IREN
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-29. SYSCFG1 Register Description

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Always read as 0.
4	IRDATA	RW	0h	Infrared data 0b = Infrared data logic 0 1b = Infrared data logic 1
3	IRDSSEL	RW	0h	Infrared data source select 0b = From hardware peripherals upon device configuration 1b = From IRDATA bit
2	IRMSEL	RW	0h	Infrared mode select 0b = ASK mode 1b = FSK mode
1	IRPSEL	RW	0h	Infrared polarity select 0b = Normal polarity 1b = Inverted polarity
0	IREN	RW	0h	Infrared enable 0b = Infrared function disabled 1b = Infrared function enabled

1.16.2.3 MSP430FR413x SYSCFG2 Register (offset = 04h) [reset = 0000h]

System Configuration Register 2

Figure 1-29. SYSCFG2 Register

15	14	13	12	11	10	9	8
Reserved			LCDPCTL	Reserved	Reserved	ADCPCTL9	ADCPCTL8
r0	r0	r0	rw-0	r0	r0	rw-0	rw-0

7	6	5	4	3	2	1	0
ADCPCTL7	ADCPCTL6	ADCPCTL5	ADCPCTL4	ADCPCTL3	ADCPCTL2	ADCPCTL1	ADCPCTL0
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-30. SYSCFG2 Register Description

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always read as 0.
12	LCDPCTL	RW	0h	LCD power pin (LCDCAP0, LCDCAP1, R23, R33) control. 0b = LCD power pin disabled 1b = LCD power pin enabled
11-10	Reserved	R	0h	Reserved. Always read as 0.
9	ADCPCTL9	RW	0h	ADC input A9 pin select 0b = ADC input A9 disabled 1b = ADC input A9 enabled
8	ADCPCTL8	RW	0h	ADC input A8 pin select 0b = ADC input A8 disabled 1b = ADC input A8 enabled
7	ADCPCTL7	RW	0h	ADC input A7 pin select 0b = ADC input A7 disabled 1b = ADC input A7 enabled
6	ADCPCTL6	RW	0h	ADC input A6 pin select 0b = ADC input A6 disabled 1b = ADC input A6 enabled
5	ADCPCTL5	RW	0h	ADC input A5 pin select 0b = ADC input A5 disabled 1b = ADC input A5 enabled
4	ADCPCTL4	RW	0h	ADC input A4 pin select 0b = ADC input A4 disabled 1b = ADC input A4 enabled
3	ADCPCTL3	RW	0h	ADC input A3 pin select 0b = ADC input A3 disabled 1b = ADC input A3 enabled
2	ADCPCTL2	RW	0h	ADC input A2 pin select 0b = ADC input A2 disabled 1b = ADC input A2 enabled
1	ADCPCTL1	RW	0h	ADC input A1 pin select 0b = ADC input A1 disabled 1b = ADC input A1 enabled
0	ADCPCTL0	RW	0h	ADC input A0 pin select 0b = ADC input A0 disabled 1b = ADC input A0 enabled

1.16.3 MSP430FR2433 System Configuration Registers

All registers are listed in Table 1-31. A detailed description of each register and its bits is provided in the following sections.

Table 1-31. FR2433 SYS Configuration Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
20h	SYSCFG0	System Configuration 0	Read/write	Word	9603h	Section 1.16.3.1
20h	SYSCFG0_L		Read/write	Byte	96h	
21h	SYSCFG0_H		Read/write	Byte	03h	
22h	SYSCFG1	System Configuration 1	Read/write	Word	0000h	Section 1.16.3.2
22h	SYSCFG1_L		Read/write	Byte	00h	
23h	SYSCFG1_H		Read/write	Byte	00h	
24h	SYSCFG2	System Configuration 2	Read/write	Word	0000h	Section 1.16.3.3
24h	SYSCFG2_L		Read/write	Byte	00h	
25h	SYSCFG2_H		Read/write	Byte	00h	

1.16.3.1 MSP430FR2433 SYSCFG0 Register (offset = 00h) [reset = 9603h]

System Configuration Register 0

Figure 1-30. SYSCFG0 Register

15	14	13	12	11	10	9	8
FRWPPW							
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						DFWP	PFWP
r0	r0	r0	r0	r0	r0	rw-1	rw-1

Table 1-32. SYSCFG0 Register Description

Bit	Field	Type	Reset	Description
15-8	FRWPPW	RW	96h	FRAM protection password[1] FRAM protection password. Write with 0A5h to unlock the FRAM protection registers. Always reads as 096h
7-2	Reserved	R	0h	Reserved. Always read as 0.
1	DFWP	RW	1h	Data FRAM write protection 0b = Data FRAM write enable 1b = Data FRAM write protected (not writable)
0	PFWP	RW	1h	Program FRAM write protection 0b = Program FRAM write enable 1b = Program FRAM write protected (not writable)

[1] The password needs to be operated with FRAM protection bits in a word at the same time.

1.16.3.2 MSP430FR2433 SYSCFG1 Register (offset = 02h) [reset = 0000h]

System Configuration Register 1

Figure 1-31. SYSCFG1 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		Reserved	IRDATA	IRDSSEL	IRMSEL	IRPSEL	IREN
rw-0	rw-0	r0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-33. SYSCFG1 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always read as 0.
7-6	Reserved	RW	0h	Reserved.
5	Reserved	R	0h	Reserved. Always read as 0.
4	IRDATA	RW	0h	Infrared data 0b = Infrared data logic 0 1b = Infrared data logic 1
3	IRDSSEL	RW	0h	Infrared data source select 0b = From hardware peripherals upon device configuration 1b = From IRDATA bit
2	IRMSEL	RW	0h	Infrared mode select 0b = ASK mode 1b = FSK mode
1	IRPSEL	RW	0h	Infrared polarity select 0b = Normal polarity 1b = Inverted polarity
0	IREN	RW	0h	Infrared enable 0b = Infrared function disabled 1b = Infrared function enabled

1.16.3.3 MSP430FR2433 SYSCFG2 Register (offset = 04h) [reset = 0000h]

System Configuration Register 2

Figure 1-32. SYSCFG2 Register

15	14	13	12	11	10	9	8
Reserved							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADCPCTL7	ADCPCTL6	ADCPCTL5	ADCPCTL4	ADCPCTL3	ADCPCTL2	ADCPCTL1	ADCPCTL0
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-34. SYSCFG2 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	RW	0h	Reserved.
7	ADCPCTL7	RW	0h	ADC input A7 pin select 0b = ADC input A7 disabled 1b = ADC input A7 enabled
6	ADCPCTL6	RW	0h	ADC input A6 pin select 0b = ADC input A6 disabled 1b = ADC input A6 enabled
5	ADCPCTL5	RW	0h	ADC input A5 pin select 0b = ADC input A5 disabled 1b = ADC input A5 enabled
4	ADCPCTL4	RW	0h	ADC input A4 pin select 0b = ADC input A4 disabled 1b = ADC input A4 enabled
3	ADCPCTL3	RW	0h	ADC input A3 pin select 0b = ADC input A3 disabled 1b = ADC input A3 enabled
2	ADCPCTL2	RW	0h	ADC input A2 pin select 0b = ADC input A2 disabled 1b = ADC input A2 enabled
1	ADCPCTL1	RW	0h	ADC input A1 pin select 0b = ADC input A1 disabled 1b = ADC input A1 enabled
0	ADCPCTL0	RW	0h	ADC input A0 pin select 0b = ADC input A0 disabled 1b = ADC input A0 enabled

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).

Topic **Page**

2.1 Power Management Module (PMM) Introduction 78
2.2 PMM Operation 79
2.3 PMM Registers 83

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
- Reference voltage output on external pin

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, to 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.

The block diagram of the PMM is shown in Figure 2-1.

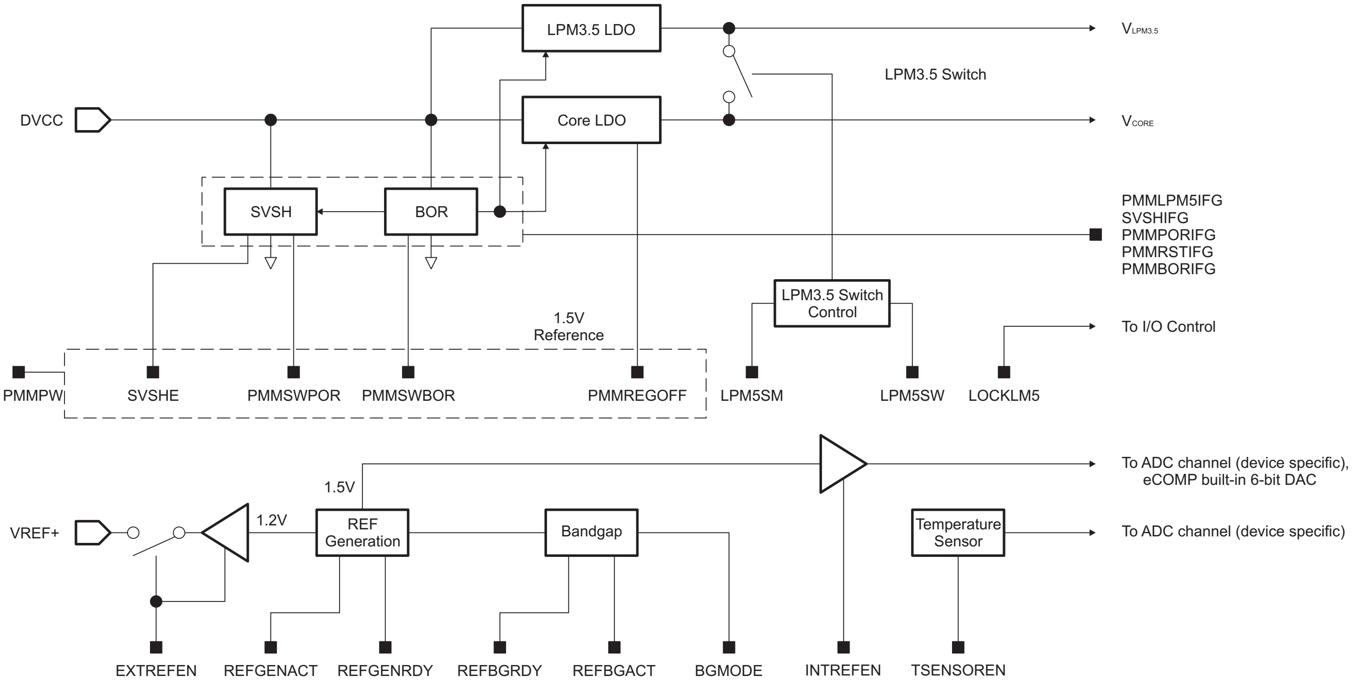


Figure 2-1. PMM Block Diagram

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 or SBW is active

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

2.2.2 Supply Voltage Supervisor

The high-side supervisor (SVSH) oversees DV_{CC}. It is active in all power modes by default. In LPM3, LPM4, LPM3.5, and LPM4.5, it can be disabled by setting SVSHE = 0.

2.2.2.1 SVS Thresholds

As Figure 2-2 shows, there is hysteresis built into the supervision thresholds, so that which threshold is in force depends on whether the voltage rail is rising or falling.

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.

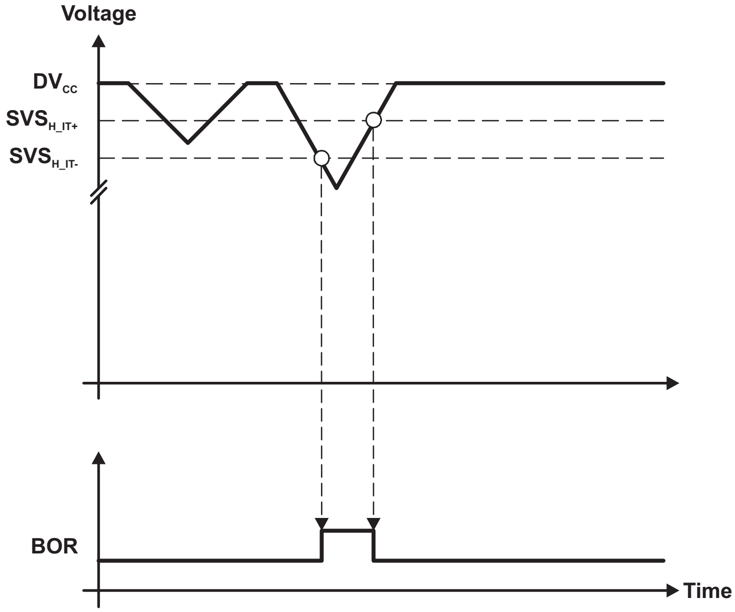


Figure 2-2. Voltage Failure and Resulting PMM Actions

2.2.3 Supply Voltage Supervisor During 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.

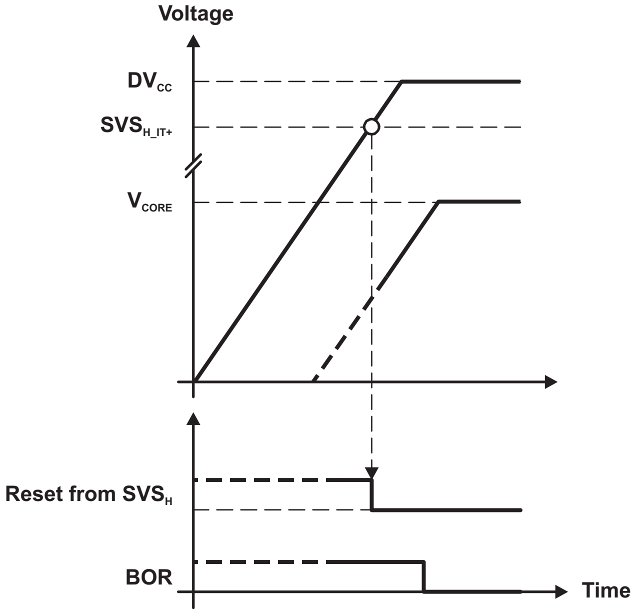


Figure 2-3. PMM Action at Device Power-Up

2.2.4 LPM3.5 and LPM4.5 (LPMx.5)

LPM3.5 and LPM4.5 are low-power modes in which the core voltage regulator of the PMM is completely disabled to provide 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, as a result, the contents of the registers and RAM are lost. Any essential values should be stored to FRAM prior to entering LPMx.5. See the SYS module for complete description and uses of LPMx.5.

LPM3.5 and LPM4.5 can be configured with SVS enabled (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 also results in approximately 10 times faster start-up time than with disabled SVS.

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 after the supply drops below the SVS power-down level. By default, as soon as 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 state results in a current consumption of approximately 50 µA to 100 µA (typical).

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 enters 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 is required to define the I/O states.)

2.2.6 Brownout Reset (BOR)

The primary function of the 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 and for proper reset of the system.

In an application, it may be desired to cause a BOR in 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 in 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 LPM3.5 Switch

The LPM3.5 switch supplies the LPM3.5 power domain with main LDO output, which allows the peripherals to consume more current and operate at high frequency. When the device enters LPM3.5, all peripherals in LPM3.5 domains are isolated from the core domain and fully supplied by the LPM3.5 LDO. The LPM3.5 switch can be either manually or automatically controlled. The LPM3.5 switch mode can be set by LPM5SM in the PM5CTL0 register.

In the automatic control mode (the LPM5SM bit is clear), the LPM3.5 switch is disconnected when the device enters LPM3.5 mode. Upon exiting from LPM3.5 to AM, the device automatically turns on the LPM3.5 switch and those peripherals that were supplied by LPM3.5 are directly powered by the main LDO. Before the power switching completes, do not read or write those peripherals' registers with high frequency. The LPM5SW bit in the PM5CTL0 register reports the status of the LPM3.5 switch and allows software to check the connection of the LPM3.5 switch before high-frequency operation. In this mode, any write to the LPM5SW bit does not work.

In the manual control mode (the LPM5SM bit is set), the LPM3.5 switch is specified by the LPM5SW bit in the PM5CTL0 register. When LPM5SW is set, the LPM3.5 switch is connected. When LPM5SW is clear, the LPM3.5 switch is disconnected. It is recommended to turn off the switch to avoid unnecessary leakage before the device enters LPM3.5. When the device recovers back from LPM3.5 mode, the switch should be turned on to offer sufficient current for high-frequency operation.

The LPM5SW defaults to logic 1, which means that the LPM3.5 switch is always connected after a BOR, POR, or PUC reset.

2.2.8 Reference Voltage Generation and Output

The PMM module has a high-accuracy bandgap for various voltage references on the chip. The bandgap is automatically turned on and off depending on the operating mode. The REFBGRDY bit in the PMMCTL2 register reports the readiness of the bandgap. When REFBGRDY is set, the bandgap reference is ready for use.

Two voltage references are generated for internal (1.5 V) and external (1.2 V) use, respectively. The voltage generator is automatically controlled by the device in response to the voltage reference request (either internal or external). The REFGENACT and REFGENRDY bits represent the status of the generator status if the output works properly at the specified voltage.

The internal reference voltage (1.5 V) is internally connected to an ADC channel (refer to the data sheet for device-specific configuration). The INTREFEN bit in PMMCTL2 controls whether or not the 1.5-V voltage is injected into the specified ADC channel.

The external reference voltage (1.2 V) is connected a given external ADC channel (refer to the data sheet for device-specific configuration). If this ADC channel is multiplexed with other functionality, the 1.2-V output function only works when the ADC is selected as the function on this pin. The EXTREFEN bit in PMMCTL2 controls if the 1.2-V voltage is available to the specified external ADC channel. The external reference voltage supports up to 1-mA drive capability.

2.2.9 Temperature Sensor

The PMM contains a built-in temperature sensor that software can use to monitor the die temperature for fault protection in high temperature environments. The temperature sensor is internally connected to an ADC channel. The connection is device specific and can be found in the ADC section in the data sheet. The TSENSOREN bit in the PMMCTL2 register must be set to turn on the sensor before it is used. The temperature of 25°C is trimmed in manufacture. Therefore, any temperature to be measured can be calculated by Equation 10.

$$T = 0.00355 \times (V_T - V_{25^oC}) + 25^oC \tag{10}$$

2.2.10 $\overline{\text{RST}}$/NMI

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

2.2.11 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.

2.2.12 Port I/O Control

The PMM ensures that I/O pins cannot behave in uncontrolled fashion during an undervoltage event. During these times, outputs are disabled, including both the normal drive and the weak pullup and pulldown functions. If the CPU is functioning normally before an undervoltage event occurs, any pin configured as an input has its PxIN register value latched when 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.

2.3 PMM Registers

Table 2-1 shows the PMM registers and their address offsets. The base address of the PMM module can be found in the device-specific data sheet.

The password defined in the PMMCTL0 register controls access to all PMM registers except PM5CTL0. PM5CTL0 can be accessed without the password. After the correct password is written, write access is enabled (this includes byte access to the PMMCTL0 lower byte). Write access is disabled by writing a wrong password in byte mode to the PMMCTL0 upper byte. Word access to PMMCTL0 with a wrong password causes a PUC. 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	
04h	PMMCTL2	PMM control register 2	Read/write	Word	3200h	Section 2.3.3
04h	PMMCTL2_L		Read/write	Byte	00h	
05h	PMMCTL2_H		Read/write	Byte	33h	
0Ah	PMMIFG	PMM interrupt flag register	Read/write	Word	0000h	Section 2.3.4
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	0011h	Section 2.3.5
10h	PM5CTL0_L		Read/write	Byte	11h	
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
Reserved	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. Write with 0A5h to unlock the PMM registers.
7	Reserved	RW	0h	Reserved. Must be written with 0.
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 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. Set this bit to 1 to trigger 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. Set this bit to 1 to trigger 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 = 0000h]

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 PMMCTL2 Register (offset = 04h) [reset = 3200h]

Power Management Module Control Register 2

Figure 2-6. PMMCTL2 Register

15	14	13	12	11	10	9	8
Reserved	Reserved	REFBGRDY	REFGENRDY	BGMODE	Reserved	REFBGACT	REFGENACT
r0	r0	r-(1)	r-(1)	r-(0)	r0	r-(1)	r-(0)
7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	TSENSOREN	Reserved	EXTREFEN	INTREFEN
r0	r0	r0	r0	rw-(0)	r0	rw-(0)	rw-(0)

Table 2-4. PMMCTL2 Register Description

Bit	Field	Type	Reset	Description
15-14	Reserved	R	0h	Reserved. Always reads as 0
13	REFBGRDY	R	1h	Buffered bandgap voltage ready status. 0b = Buffered bandgap voltage is not ready to be used 1b = Buffered bandgap voltage is ready to be used
12	REFGENRDY	R	1h	Variable reference voltage ready status. 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. Ready only. 0b = Static mode (higher precision) 1b = Sampled mode (lower power consumption)
10	Reserved	R	0h	Reserved. Always reads as 0
9	REFBGACT	R	1h	Reference bandgap active. Ready 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
7-4	Reserved	R	0h	Reserved. Always reads as 0
3	TSENSOREN	RW	0h	Temperature sensor enable 0b = Disable temperature sensor 1b = Enable temperature sensor
2	Reserved	R	0h	Reserved. Always reads as 0
1	EXTREFEN	RW	0h	External reference output enable 0b = Disable external reference output 1b = Enable internal reference output
0	INTREFEN	RW	0h	Internal reference enable 0b = Disable internal reference 1b = Enable internal reference

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

Power Management Module Interrupt Flag Register

Figure 2-7. 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 2-5. 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 reset. 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. The SVSHIFG interrupt flag is only set if the SVSH is the reset source; that is, 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. 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.

2.3.5 PM5CTL0 Register (offset = 10h) [reset = 0011h]

Power Mode 5 Control Register 0

Figure 2-8. 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		LPM5SM	LPM5SW	Reserved			LOCKLPM5
r0	r0	rw-[0]	rw-[1]	r0	r0	r0	rw-[1]

Table 2-6. PM5CTL0 Register Description

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5	LPM5SM	RW	0h	Specifies the operation mode of the LPM3.5 switch. 0b = Automatic mode. The LPM3.5 switch is fully handled by the circuitry during mode switch. 1b = Manual mode. The LPM3.5 switch is specified by LPM5SW bit setting in software.
4	LPM5SW	RW	1h	Reports or sets the LPM3.5 switch connection, based on the switch mode set by LPM5SM. When LPM5SW = 1, the $V_{LPM3.5}$ domain can accept full-speed read and write operation by the CPU MCLK. If the switch is disconnected (LPM5SW = 0), all peripherals within this domain can accept clock operation no faster than 40 kHz. In automatic mode (LPM5SM = 0), this bit represents the switch connection between V_{core} and $V_{LPM3.5}$. Any write to this bit has no effect. In manual mode (LPM5SM = 1), this bit is read/write by software. When this bit is set, the switch connection between V_{core} and $V_{LPM3.5}$ is connected. Otherwise, the switch is disconnected. 0b = LPMx.5 switch disconnected 1b = LPMx.5 switch connected
3-1	Reserved	R	0h	Reserved. Always reads as 0.
0	LOCKLPM5	RW	1h	Lock I/O pin and other LPMx.5 relevant (for example, RTC) configurations upon entry to or exit from LPMx.5. After the LOCKLPM5 bit is set, it can be cleared only by software or by a power cycle. This bit is reset by a power cycle; that is, if SVSH (if enabled) or brownout triggered a reset. 0b = LPMx.5 configuration is not locked and defaults to its reset condition. 1b = LPMx.5 configuration remains locked. Pin state is held during LPMx.5 entry and exit.

Clock System (CS)

The Clock System (CS) module provides the various clocks used on MCU. This chapter describes the operation of the CS module, which is implemented in all devices.

Topic **Page**

3.1 CS Introduction 90
3.2 CS Operation 92
3.3 CS Registers 101

3.1 CS Introduction

The CS module supports low system cost and low power consumption. This module supports four internal and two external clock sources, by which users can optimize the clock configuration for different design goals. Not all clock sources present in one device. For a detailed description of the configuration for any given device, see the device-specific data sheet. All clock sources can be fully selected by software. External clock sources can use either crystal or ceramic oscillators or resonators.

The CS module includes up to six clock sources:

- XT1CLK: High-frequency or low-frequency oscillator that can be used with a high-frequency ceramic or crystal oscillator or a low-frequency 32768-Hz crystal. XT1CLK can be used as a clock reference into the FLL. Some devices only support the low-frequency oscillator for XT1CLK. Refer to the device-specific data sheet for more details.
- VLOCLK: Internal very-low-power low-frequency oscillator with 10-kHz typical frequency
- REFOCLK: Internal trimmed low-frequency oscillator with 32768-Hz typical frequency. Can be used as a clock reference into the FLL.
- DCOCLK: Internal digitally controlled oscillator (DCO) that can be stabilized by the FLL.
- MODCLK: Internal high-frequency oscillator with 5-MHz typical frequency.

Three clock signals are available from the CS module:

- ACLK: Auxiliary clock. ACLK can be used for peripherals low-frequency operation. This clock is software selectable as XT1CLK or REFOCLK. The selected clock source must always be approximately 32 kHz, no more than 40 kHz (typical). ACLK is software selectable by individual peripheral modules.
- MCLK: Master clock. MCLK is the main clock source of CPU, CRC, and some other digital peripherals directly operated by the CPU or its clock. This clock is software selectable as REFOCLK, DCOCLK, XT1CLK, or VLOCLK. When available, the selected clock source can be predivided by 1, 2, 4, 8, 16, 32, 64, or 128.
- SMCLK: Subsystem master clock. SMCLK is the clock for the peripherals that can work independently from CPU operation. This clock always derives from MCLK. When available, SMCLK can be predivided by 1, 2, 4, or 8. SMCLK is software selectable by individual peripheral modules.

Figure 3-1 shows the block diagram of the CS module.

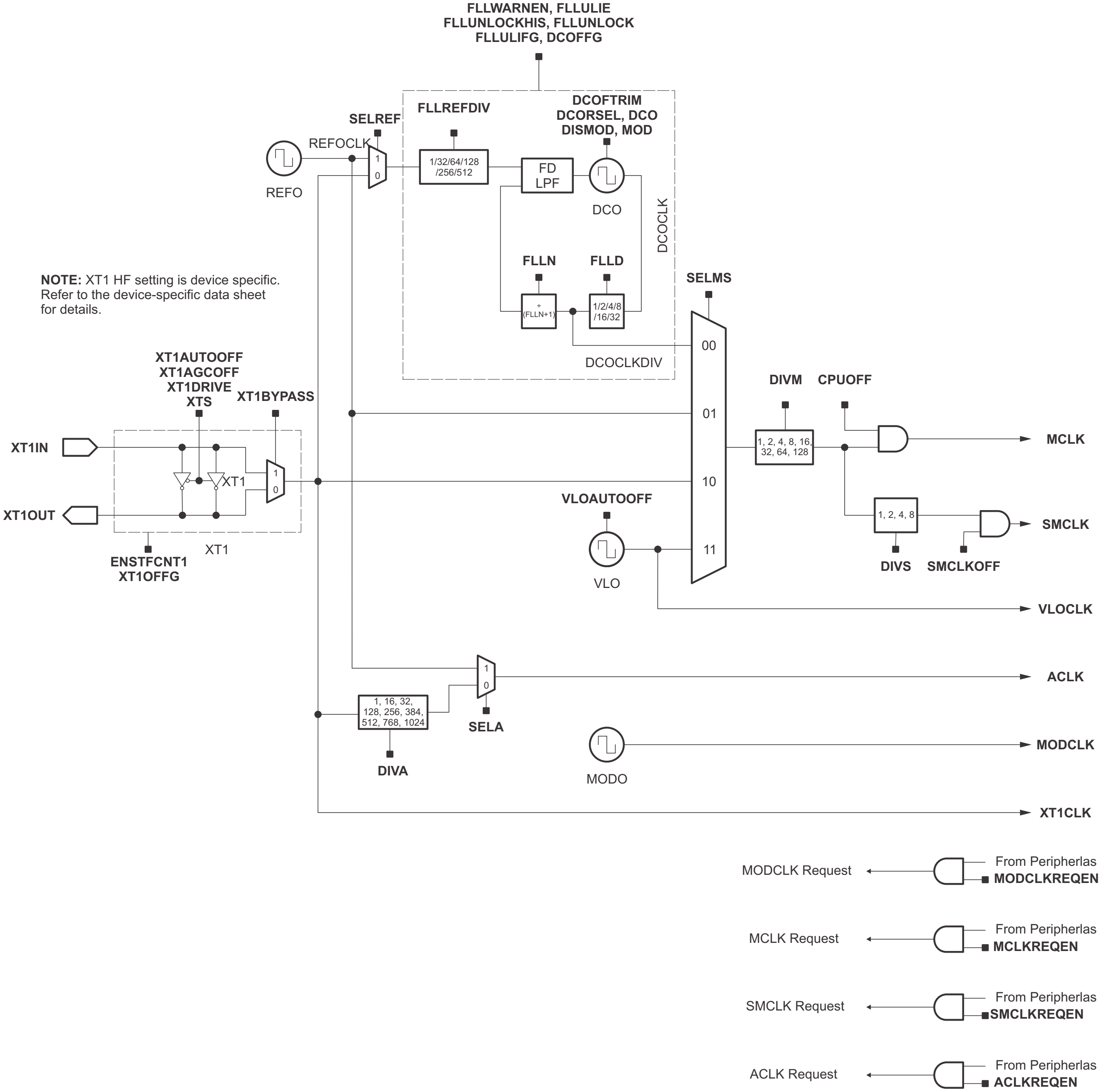


Figure 3-1. Clock System (CS) Block Diagram

3.2 CS Operation

After a PUC, the CS module default configuration is:

- MCLK and SMCLK use DCOCLKDIV, which is locked by the FLL and referenced by REFO if XT1 is not available.
- ACLK uses REFO.
- XT1 external crystal oscillator is selected as the XT1CLK clock source. XT1IN and XT1OUT pins are set to general-purpose I/Os and XT1 remains disabled until the I/O ports are configured for XT1 operation.

After PUC, DCO locked by FLL operation with XT1CLK is selected by default. The FLL stabilizes MCLK and SMCLK to 1 MHz and $f_{DCOCLKDIV}$ = 1 MHz

An external 32768-Hz crystal can be used as the FLL reference. By default, the crystal pins (XT1IN, XT1OUT) are shared with general-purpose I/Os. To enable XT1, the PSEL bits associated with the crystal pins must be set to use the external 32768-Hz crystal as the clock source. After the crystal starts up and settles, the FLL reference clock is automatically switched to XT1CLK when XT1OFFG, DCOFFG, and OFIFG are clear.

A default monitor is engaged with XT1 oscillation. If XT1 is used but does not work properly, fault protection logic forces REFO as the FLL reference clock.

The status register control bits (SCG0, SCG1, OSCOFF, and CPUOFF) configure the MSP430 operating modes and enable or disable portions of the CS module. Registers CSCTL0 through CSCTL8 configure the CS module.

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

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.

MCLK can be sourced from any of the available clock sources (DCOCLK, REFOCLK, XT1CLK, or VLOCLK). SMCLK is derived from MCLK and always uses the same clock source as MCLK.

ACLK can be source from either REFO or XT1CLK.

3.2.2 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 low-power clock source for applications that do not require an accurate time base.

VLOCLK is active in the following conditions:

- VLO is selected as the source of MCLK and SMCLK (SELMS = {3}), and MCLK or SMCLK is active.
- The VLOAUTOOFF bit is cleared and the MCU is in AM through LPM4.
- At least one peripheral requests VLO as clock source.

3.2.3 Internal Trimmed Low-Frequency Reference Oscillator (REFO)

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

REFO is enabled under any of the following conditions:

- REFO is a source for MCLK and SMCLK (SELMS = {1}) and MCLK or SMCLK is active.
- REFO is a source for ACLK (SELA = {1}) and ACLK is active.
- REFO is a source for FLLREFCLK (SELREF = {1}) and DCO is active.

3.2.4 XT1 Oscillator

The XT1 oscillator supports low-current consumption using a 32768-Hz watch crystal in low-frequency (LF) mode. A watch crystal connects to XIN and XOUT and requires external loading capacitors on both terminals. These capacitors should be sized according to the crystal or resonator specifications.

The drive settings of XT1 can be increased with the XT1DRIVE bits. At power up, the XT1 starts with the highest drive settings for fast reliable startup. After startup, user software can reduce the drive strength to reduce power consumption.

In some devices, the XT1 oscillator supports high-frequency crystals or resonators when in high-frequency (HF) mode (XTS = 1). The high-frequency crystal or resonator connects to XT1IN and XT1OUT and requires external capacitors on both terminals. These capacitors should be sized according to the crystal or resonator specifications.

The XT1 pins are shared with general-purpose I/O ports. At power up, the default operation is general-purpose I/O ports. XT1 remains disabled until the ports shared with XT1 are configured for XT1 operation. The configuration of the shared I/O is determined by the Px.SEL bit associated with the XT1IN pin and the XT1BYPASS bit. Setting the Px.SEL bit causes the XT1IN and XT1OUT ports to be configured for XT1 operation.

If XT1BYPASS is also set, XT1 is configured for bypass mode of operation, and the oscillator associated with XT1 is powered down. In bypass mode of operation, XT1IN can accept an external clock input signal and XT1OUT is configured as a general-purpose I/O. The Px.SEL bit associated with XT1OUT is a don't care.

If the Px.SEL bit associated with XT1IN is cleared, both XT1IN and XT1OUT ports are configured as general-purpose I/Os, and XT1 is disabled.

XT1 is enabled under any of the following conditions:

- XT1 is a source for MCLK and SMCLK (SELMS = {2}) and MCLK or SMCLK is active.
- XT1 is a source for ACLK (SELA = {0} and ACLK is active.
- XT1 is a source for FLLREFCLK (SELREF = {0}) and DCO is active.
- XT1AUTOOFF is clear and the MCU is in AM through LPM4.
- At least one peripheral requests XT1 as clock source.

NOTE: XT1 in HF mode configuration

ACLK is the auxiliary clock. ACLK must be approximately 32 kHz and no faster than 40 kHz (typical). There is a divider (DIVA) if ACLK sources from XT1 in HF mode. The divider setting depends on the external high-frequency oscillator value.

This divider is always bypassed if ACLK sources from XT1 in LF mode.

3.2.5 Digitally Controlled Oscillator (DCO)

The DCO is an integrated digitally controlled oscillator. The DCO frequency can be adjusted by software using the DCORSEL, DCO, and MOD bits. Optionally, the DCO frequency can be stabilized by the FLL to a multiple frequency of FLLREFCLK ÷ n. The FLL accepts different reference sources selected by the SELREF bits. Reference sources include XT1CLK and REFOCLK. The value of n is defined by the FLLREFDIV bits (n = 1, 32, 64, 128, 256, or 512). When XT1 only supports a 32-kHz clock, FLLREFDIV is always read and written as 0 (n = 1). The default is n = 1. There may be scenarios in which FLL operation is not required or desired, and therefore no FLLREFCLK is necessary.

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

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

$f_{DCOCLK} = 2^{FLLD} \times (FLLN + 1) \times (f_{FLLREFCLK} \div n)$

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

3.2.5.1 Adjusting DCO Frequency

By default, FLL operation is enabled. FLL operation can be disabled by setting SCG0 or SCG1. When the FLL is disabled, the DCO continues to operate at the current settings defined in CSCTL0 and CSCTL1. The DCO frequency can be adjusted manually if desired. Otherwise, the DCO frequency is stabilized by the FLL operation.

After a PUC, DCORSEL = {1} and DCO = {0}. MCLK and SMCLK are sourced from DCOCLKDIV. Because the CPU executes code from MCLK, which is sourced from the fast-starting DCO, code execution begins from PUC in less than 5 µs.

The frequency of DCOCLK is set by the following functions:

- The three DCORSEL bits select one of eight nominal frequency ranges for the DCO. These ranges are defined for each individual device in the device-specific data sheet.
- The nine DCO bits divide the DCO range selected by the DCORSEL bits into 512 frequency steps, separated by approximately 0.1%.
- The five MOD bits switch between the frequency selected by the DCO bits and the next-higher frequency set by {DCO + 1} (see Section 3.2.7). When DCO = {511}, the MOD bits have no effect, because the DCO is already at the highest setting for the selected DCORSEL range.

3.2.6 Frequency Locked Loop (FLL)

The FLL continuously counts up or down a frequency integrator. The output of the frequency integrator that drives the DCO can be read in CSCTL0 (bits MOD and DCO).

Nine of the integrator bits (CSCTL0 bits 8 to 0) set the DCO frequency tap. 512 taps are implemented for the DCO, and each is approximately 0.1% higher than the previous. The modulator mixes two adjacent DCO frequencies to produce fractional taps.

For a given DCO bias range setting, time must be allowed for the DCO to settle on the proper tap for normal operation. The value n is defined by the FLLREFDIV bits (n = 1, 32, 64, 128, 256, or 512). When XT1 only supports a 32-kHz clock, FLLREFDIV is always read and written as 0 (n = 1). For a typical 32768-Hz clock source, FLLREFDIV should always be set to 0 (that is, n = 1).

3.2.7 DCO Modulator

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

The modulator mixing formula is:

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

Figure 3-2 shows the modulator operation.

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

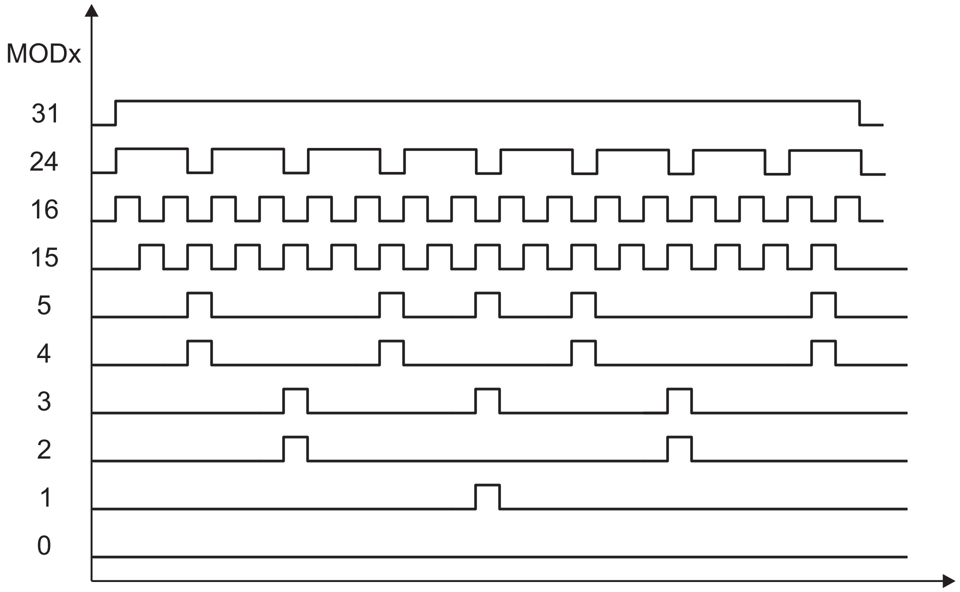


Figure 3-2. Modulator Patterns

3.2.8 Disabling FLL Hardware and Modulator

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

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

NOTE: DCO operation without FLL

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

3.2.9 FLL Unlock Detection

The FLL unlock detection function can generate PUC reset or an interrupt, when the divided DCO output fails to lock the reference clock.

When the FLL is enabled, the FLLUNLOCK bits reflect the DCO status if it is locked, too slow, too fast, or out of DCO range. When FLL recovers as locked, the FLLUNLOCK bit will be cleared and the FLLUNLOCKHIS bits will automatically log previous unlock status.

To reconfigure the DCO frequency or FLL reference clock, it is recommended to clear CSCTL0 first. This ensures that the DCO starts ramping up from the lowest frequency to avoid a frequency above specification due to temperature or supply voltage drift over time. This operation must be followed by waiting at least two MCLK cycles before the FLL is enabled. After the wait cycles, poll the FLLUNLOCK bits to determine if the FLL is locked in the target frequency range. If CSCTL0 register is not cleared in the reconfiguration, seven REFCLK cycles are required before polling FLLUNLOCK bits. Then, poll FLLUNLOCK to make sure that the FLL locked.

The recommended process to reconfigure the FLL is:

1. Disable the FLL (BIS.W #SCG0, SR)
2. Switch the FLL reference clock if required
3. Clear the CSCTL0 register (CLR.B CSCTL0)

4. Reconfigure the DCO or FLL to the target range
5. Wait at least two MCLK cycles to allow the DCO and FLL to settle
6. Enable the FLL (BIC.W #SCG0, SR)
7. Poll the FLLUNLOCK bits until the FLL is locked

If the FLLULPUC bit is set (FLLULPUC = 1), when DCO runs too fast (FLLUNLOCK = 10b), the FLLULIFG bit flag being set causes a PUC reset.

If FLLWARNEN bit is set, when FLLUNLOCKHIS changes to unlock, the OFIFG flag is set.

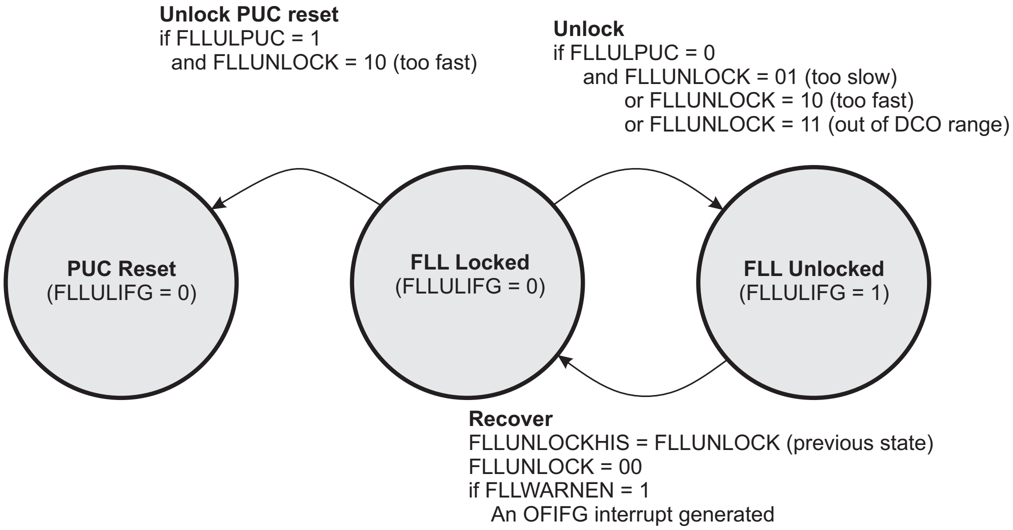

Unlock PUC reset
if FLLULPUC = 1
and FLLUNLOCK = 10 (too fast)
Unlock
if FLLULPUC = 0
and FLLUNLOCK = 01 (too slow)
or FLLUNLOCK = 10 (too fast)
or FLLUNLOCK = 11 (out of DCO range)
PUC Reset
(FLLULIFG = 0)
FLL Locked
(FLLULIFG = 0)
FLL Unlocked
(FLLULIFG = 1)
Recover
FLLUNLOCKHIS = FLLUNLOCK (previous state)
FLLUNLOCK = 00
if FLLWARNEN = 1
An OFIFG interrupt generated

Figure 3-3. FLL Unlock Detection

3.2.10 FLL Operation From Low-Power Modes

An interrupt service request clears SCG1, CPUOFF, and OSCOFF if set, but does not clear SCG0. This means that for FLL operation from within an interrupt service routine entered from LPM1, LPM3, or LPM4, the FLL remains disabled and the DCO operates at the previous setting as defined in CSCTL0 and CSCTL1. SCG0 can be cleared by user software if FLL operation is required.

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

A peripheral module requests its clock sources automatically from the CS module if required for its proper operation, regardless of the current mode of operation (see Figure 3-4).

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 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 LPM settings.

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

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

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

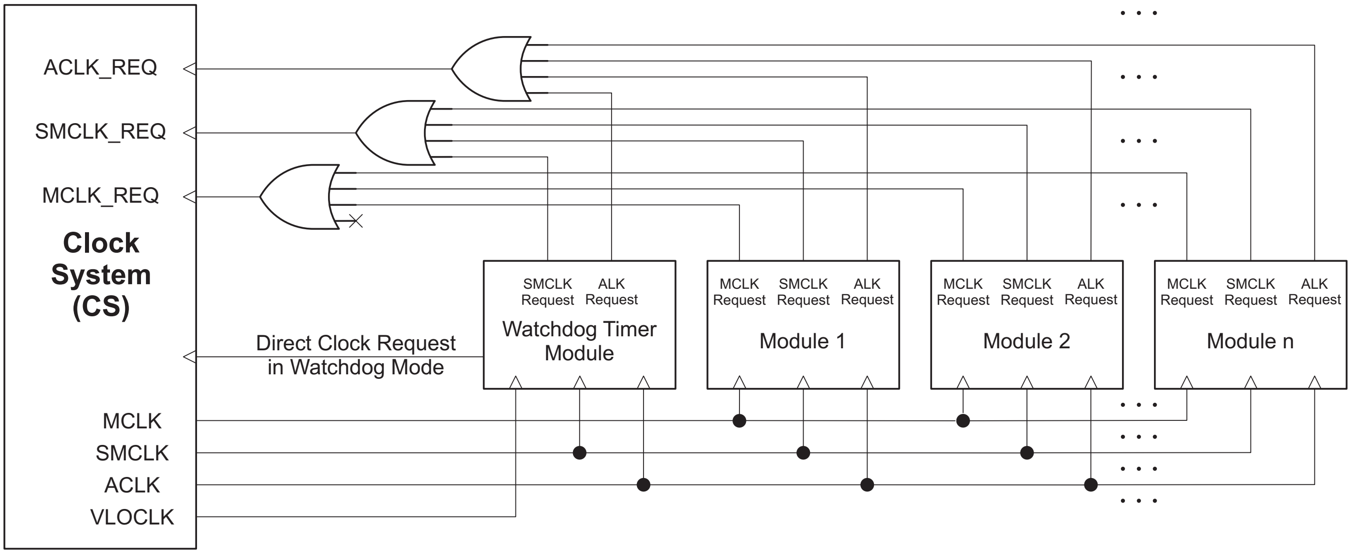


Figure 3-4. Module Request Clock System

By default, the clock request logic is enabled. The clock request logic can be disabled by clearing ACLKREQEN, MCLKREQEN, or SMCLKREQEN, for each respective system clock. When ACLKREQEN or MCLKREQEN bits are set, or active, the clock is available to the system and prevents entry into a low-power mode until all modules requesting the clock are disabled. When ACLKREQEN or MCLKREQEN bits are cleared, or disabled, the clock is always halted as defined by the low-power modes. The SMCLKREQEN logic behaves similarly, but it is also influenced by the SMCLKOFF bit in the CSCTL5 register. Table 3-1 shows the relationship between the system clocks and the low-power modes in conjunction with the clock request logic.

Table 3-1. Clock Request System and Power Modes

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

3.2.11.1 LPM3.5 and LPM4.5 Clock Request Handling

After clearing ACLK request enable signal (ACLKREQEN = 0), the device is able to enter LPMx.5. Refer to the PMM chapter for details on the requirements to enter LPMx.5.

3.2.12 Fail-Safe Operation

The CS module incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault for XT1 and DCO as shown in Figure 3-5. The available fault conditions are:

- High-frequency or low-frequency oscillator fault (XT1OFFG) for XT1
- DCO fault flag (DCOFFG) for the DCO

The crystal oscillator fault bit XT1OFFG is set if the corresponding crystal oscillator is turned on and not operating properly. Once set, the fault bits remain set until software resets them, even if the fault condition no longer exists. If software clears the fault bits and the fault condition still exists, the fault bits are automatically set again; otherwise, they remain cleared.

When using XT1 operation in LF mode as the reference source into the FLL (SELREF = {0}), a crystal fault automatically causes the FLL reference source, FLLREFCLK, to be sourced by the REFO. XT1OFFG is set. When using XT1 operation in HF mode as the reference source into the FLL, a crystal fault causes no FLLREFCLK signal to be generated and the FLL continues to count down to zero in an attempt to lock FLLREFCLK ÷ n and DCOCLK ÷ [2^{FLLD} × (FLLN + 1)]. The DCO tap moves to the lowest position (DCO bits are cleared) and the DCOFFG is set. DCOFFG is also set if the N-multiplier value is set too high for the selected DCO frequency range, resulting in the DCO tap moving to the highest position (CSCTL0.8 to CSCTL0.0 are set). The DCOFFG remains set until cleared by the user. If the user clears the DCOFFG and the fault condition remains, it is automatically set, otherwise it remains cleared. XT1HFOFFG is set.

The OFIFG oscillator-fault interrupt flag is set and latched at POR or when any oscillator fault (XT1OFFG or DCOFFG) is detected. When OFIFG is set and OFIE is set, the OFIFG requests an NMI. When the interrupt is granted, the OFIE is not reset automatically as it is in previous MSP430 families. It is no longer required to reset the OFIE. NMI entry/exit circuitry removes this requirement. The OFIFG flag must be cleared by software. The source of the fault can be identified by checking the individual fault bits.

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

If SMCLK sources from XT1 in LF mode, an oscillator fault causes SMCLK to be automatically switched to the REFO for its clock source (REFOCLK). If SMCLK sources from XT1 in HF mode, an oscillator fault causes SMCLK to be automatically switched to the DCO for its clock source (DCOCLKDIV). This fault switch does not change the SELMS bit settings. This condition must be handled by user software.

If ACLK sources from XT1 in LF or HF mode, an oscillator fault causes ACLK to be automatically switched to the REFO for its clock source (REFOCLK). This does not change the SELA bit settings. This condition must be handled by user software.

NOTE: DCO active during oscillator fault

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

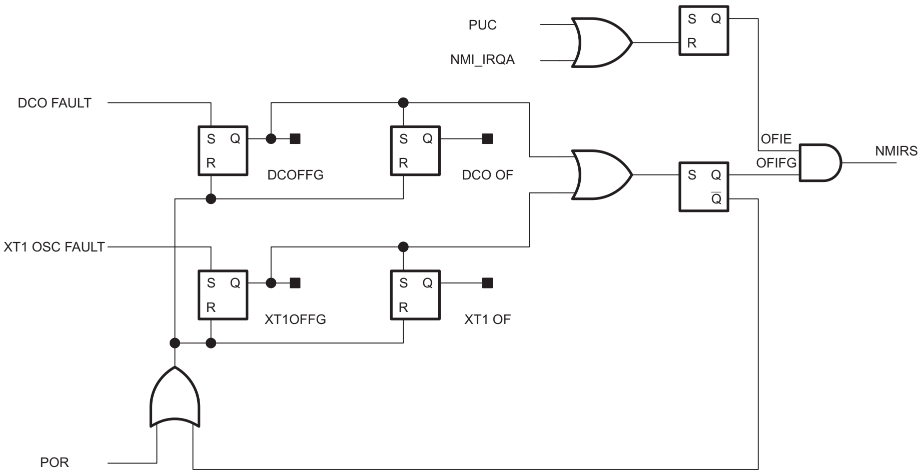


Figure 3-5. Oscillator Fault Logic

NOTE: Fault conditions

DCO_Fault: DCOFFG is set if DCO bits in CSCTL0 register value equals {0} or {511} and DCO is unlocked. DCO_Fault is ignored when FLL is disabled. It is suggested to clear DCOFFG before FLL disabled.

XT1_OscFault: This signal is set after the XT1 oscillator has stopped operation and is cleared after operation resumes. The fault condition causes XT1OFFG to be set and remain set. If the user clears XT1OFFG and the fault condition still exists, XT1OFFG remains set.

Fault logic

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

Fault logic counters

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

In XT1 LF mode, the maximum count is 8192. In XT1 HF mode, the maximum count is 1024. In bypass modes, regardless of LF or HF settings, the maximum count is 8192.

3.2.13 Synchronization of Clock Signals

When switching MCLK or SMCLK from one clock source to the another, the switch is synchronized as shown in Figure 3-6 to avoid critical race conditions.

- 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.

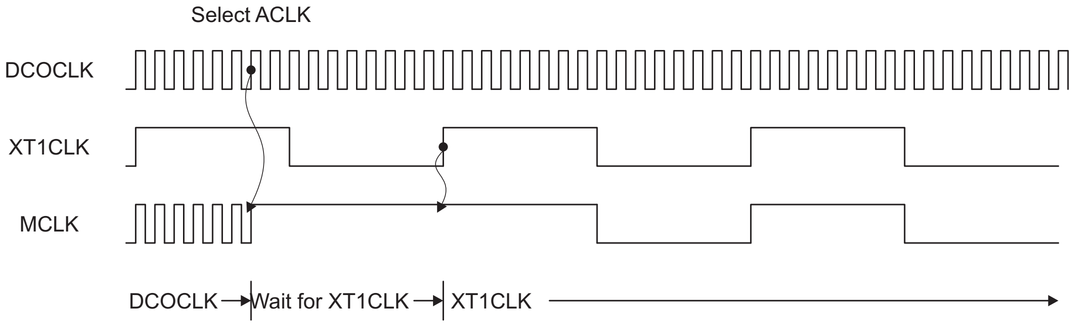


Figure 3-6. Switch MCLK from DCOCLK to XT1CLK

3.2.14 Module Oscillator (MODOSC)

The CS module also supports an internal oscillator, MODOSC, that is used by ADC and, optionally, by other modules in the system. The MODOSC sources MODCLK.

3.2.14.1 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 use unconditional requests; for example, the ADC.

The ADC may optionally use MODOSC as a clock source for its conversion clock. The user chooses the MODOSC as the conversion clock source. During a conversion, the ADC module issues an unconditional request for the MODOSC clock source. Upon doing so, the MODOSC source is enabled, if not already enabled by a previous request from another module.

3.3 CS Registers

Table 3-2 lists the CS registers with offsets. See the device-specific data sheet for the base address.

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

Table 3-2. CS Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	CSCTL0	Clock System Control Register 0	Read/write	Word	0000h	Section 3.3.1
02h	CSCTL1	Clock System Control Register 1	Read/write	Word	0033h	Section 3.3.2
04h	CSCTL2	Clock System Control Register 2	Read/write	Word	101Fh	Section 3.3.3
06h	CSCTL3	Clock System Control Register 3	Read/write	Word	0000h	Section 3.3.4
08h	CSCTL4	Clock System Control Register 4	Read/write	Word	0100h	Section 3.3.5
0Ah	CSCTL5	Clock System Control Register 5	Read/write	Word	1000h	Section 3.3.6
0Ch	CSCTL6	Clock System Control Register 6	Read/write	Word	08C1h	Section 3.3.7
0Eh	CSCTL7	Clock System Control Register 7	Read/write	Word	0740h	Section 3.3.8
10h	CSCTL8	Clock System Control Register 8	Read/write	Word	0007h	Section 3.3.9

3.3.1 CSCTL0 Register

Clock System Control Register 0

Figure 3-7. CSCTL0 Register

15	14	13	12	11	10	9	8
Reserved		MOD					DCO
r0	r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

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

Table 3-3. CSCTL0 Register Description

Bit	Field	Type	Reset	Description
15-14	Reserved	R	0h	Reserved. Always reads as 0.
13-9	MOD	RW	0h	Modulation bit counter. These bits select the modulation pattern. All MOD bits are modified automatically during FLL operation. The DCO register value is incremented when the modulation bit counter rolls over from 31 to 0. If the modulation bit counter decrements from 0 to the maximum count, the DCO register value is also decreased.
8-0	DCO	RW	0h	DCO tap selection. These bits select the DCO tap and are modified automatically during FLL operation.

3.3.2 CSCTL1 Register

Clock System Control Register 1

Figure 3-8. 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**
DCOFTRIMEN	DCOFTRIM			DCORSEL			DISMOD
rw-[0]	rw-0	rw-1	rw-1	rw-0	rw-0	rw-1	rw-1

Table 3-4. CSCTL1 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7	DCOFTRIMEN	RW	0h	DCO frequency trim enable. When this bit is set, DCOFTRIM value is selected to set DCO frequency. Otherwise, DCOFTRIM value is bypassed and DCO applies default settings in manufacture. 0b = Disable frequency trim 1b = Enable frequency trim
6-4	DCOFTRIM	RW	3h	DCO frequency trim. These bits trims the DCO frequency. By default, it is chip-specific trimmed. These bits can also be trimmed by user code.
3-1	DCORSEL	RW	1h	DCO range select 000b = 1 MHz 001b = 2 MHz (Default) 010b = 4 MHz 011b = 8 MHz 100b = 12 MHz 101b = 16 MHz 110b = Reserved 111b = Reserved
0	DISMOD	RW	1h	Modulation. This bit enables or disables the modulation. 0b = Modulation enabled 1b = Modulation disabled

3.3.3 CSCTL2 Register

Clock System Control Register 2

Figure 3-9. CSCTL2 Register

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

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

Table 3-5. CSCTL2 Register Description

Bit	Field	Type	Reset	Description
15	Reserved	R	0h	Reserved. Always reads as 0.
14-12	FLLD	RW	1h	FLL loop divider. These bits divide f_{DCOCLK} in the FLL feedback loop. This results in an additional multiplier for the multiplier bits. See also multiplier bits. 000b = f_{DCOCLK} ÷ 1 001b = f_{DCOCLK} ÷ 2 (Default) 010b = f_{DCOCLK} ÷ 4 011b = f_{DCOCLK} ÷ 8 100b = f_{DCOCLK} ÷ 16 101b = f_{DCOCLK} ÷ 32 110b = Reserved for future use 111b = Reserved for future use
11-10	Reserved	R	0h	Reserved. Always reads as 0.
9-0	FLLN	RW	1Fh	Multiplier bits. These bits set the multiplier value N of the DCO. N must be greater than 0. Writing zero to FLLN causes N to be set to 1.

3.3.4 CSCTL3 Register

Clock System Control Register 3

Figure 3-10. CSCTL3 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		SELREF		Reserved	FLLREFDIV[1]		
r0	r0	rw-0	rw-0	r0	rw-0	rw-0	rw-0

[1] These bits are always read and written as 000b, when XT1 only supports 32 kHz.

Table 3-6. CSCTL3 Register Description

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	SELREF	RW	0h	FLL reference select. These bits select the FLL reference clock source. 00b = XT1CLK 01b = REFOCLK 10b = Reserved for future use 11b = Reserved for future use.
3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	FLLREFDIV	RW	0h	FLL reference divider. These bits define the divide factor for f(FLLREFCLK). If XT1 supports high frequency input higher than 32 kHz, the divided frequency is used as the FLL reference frequency. 000b = $f_{FLLREFCLK}$ ÷ 1 001b = $f_{FLLREFCLK}$ ÷ 32 010b = $f_{FLLREFCLK}$ ÷ 64 011b = $f_{FLLREFCLK}$ ÷ 128 100b = $f_{FLLREFCLK}$ ÷ 256 101b = $f_{FLLREFCLK}$ ÷ 512 110b = Reserved for future use 111b = Reserved for future use If XT1 only supports 32-kHz clock, FLLREFDIV always reads and should be written as zero: 000b = $f_{FLLREFCLK}$ ÷ 1

3.3.5 CSCTL4 Register

Clock System Control Register 4

Figure 3-11. CSCTL4 Register

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

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

Table 3-7. CSCTL4 Register Description

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0h	Reserved. Always reads as 0.
8	SELA	RW	1h	Selects the ACLK source. 0b = XT1CLK with divider (must be no more than 40 kHz) 1b = REFO (internal 32-kHz clock source)
7-3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	SELMS	RW	0h	Selects the MCLK and SMCLK source. 000b = DCOCLKDIV 001b = REFOCLK 010b = XT1CLK 011b = VLOCLK 1xxb = Reserved for future use

3.3.6 CSCTL5 Register

Clock System Control Register 5

Figure 3-12. CSCTL5 Register

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

Table 3-8. CSCTL5 Register Description

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12	VLOAUTOOFF	RW	1h	VLO automatic off enable. This bit turns off VLO, if VLO is not used. 0b = VLO always on 1b = VLO automatically turned off if not used(default)
11-9	Reserved	R	0h	Reserved. Always reads as 0.
8	SMCLKOFF	R/W	0h	SMCLK off. This bit turns off SMCLK clock. 0b = SMCLK on 1b = SMCLK off
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	DIVS	RW	0h	SMCLK source divider. SMCLK directly derives from MCLK. SMCLK frequency is the combination of DIVM and DIVS out of selected clock source. 00b = ÷ 1 01b = ÷ 2 10b = ÷ 4 11b = ÷ 8
3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	DIVM	RW	0h	MCLK source divider 000b = ÷ 1 001b = ÷ 2 010b = ÷ 4 011b = ÷ 8 100b = ÷ 16 101b = ÷ 32 110b = ÷ 64 111b = ÷ 128

3.3.7 CSCTL6 Register

Clock System Control Register 6

Figure 3-13. CSCTL6 Register

15	14	13	12	11	10	9	8
Reserved				DIVA			
r0	r0	r0	r0	rw-1	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
XT1DRIVE		XTS[1]	XT1BYPASS	XT1HFFREQ		XT1AGCOFF	XT1AUTOOFF
rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

[1] This bit is read-only as 0 if the device does not support XT1 HF mode. See the device-specific data sheet for configuration information.

Table 3-9. CSCTL6 Register Description

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11-8	DIVA	RW	8h	ACLK source divider.[1][2] 0000b = ÷1 0001b = ÷16 0010b = ÷32 0011b = ÷64 0100b = ÷128 0101b = ÷256 0110b = ÷384 0111b = ÷512 1000b = ÷768 1001b= ÷1024 1111b to 1010b = Reserved
7-6	XT1DRIVE	RW	3h	The XT1 oscillator current can be adjusted to its drive needs. Initially, it starts with the highest supply current for reliable and quick startup. If needed, user software can reduce the drive strength. The configuration of these bits is retained during LPM3.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPM3.5 before clearing LOCKLPM5 is required. 00b = Lowest drive strength and current consumption 01b = Lower drive strength and current consumption 10b = Higher drive strength and current consumption 11b = Highest drive strength and current consumption
5	XTS	RW[3]	0h[4]	XT1 mode select. 0b = Low-frequency mode 1b = High-frequency mode
4	XT1BYPASS	RW	0h	XT1 bypass select. 0b = XT1 source internally 1b = XT1 sources externally from pin
3-2	XT1HFFREQ	RW	0h	The XT1 high-frequency selection. These bits must be set to appropriate frequency for crystal or bypass modes of operation. [1] 00b = 1 MHz to 4 MHz 01b = Above 4 MHz to 6 MHz 10b = Above 6 MHz to 16 MHz 11b = Above 16 MHz to 24 MHz

[1] These bits are valid only in XT1 HF mode. The divider setting depends on the external high-frequency oscillator value, because ACLK is fixed to no more than 40 kHz (typical). See the device-specific data sheet for details.

[2] This divider is always bypassed if ACLK sources from XT1 in LF mode.

[3] The bits are read-only if XT1 HF mode is not supported in the device. See the device-specific data sheet for configuration information.

[4] The bits are read-only as 0 if XT1 HF mode is not supported in the device. See the device-specific data sheet for configuration information.

Table 3-9. CSCTL6 Register Description (continued)

Bit	Field	Type	Reset	Description
1	XT1AGCOFF	RW	0h	Automatic gain control (AGC) disable. 0b = AGC on 1b = AGC off
0	XT1AUTOOFF	RW	1h	XT1 automatic off enable. This bit allows XT1 turned turns off when it is not used. 0b = XT1 is on if XT1 is selected by the port selection and XT1 is not in bypass mode of operation. 1b = XT1 is off if it is not used as a source for ACLK, MCLK, or SMCLK or is not used as a reference source required for FLL operation.

3.3.8 CSCTL7 Register

Clock System Control Register 7

Figure 3-14. CSCTL7 Register

15	14	13	12	11	10	9	8
Reserved		FLLWARNEN	FLLULPUC	FLLUNLOCKHIS		FLLUNLOCK	
r0	r0	rw-0	rw-(0)	rw-(0)	rw-(1)	r-1	r-1

7	6	5	4	3	2	1	0
Reserved	ENSTFCNT1	Reserved	FLLULIFG	Reserved		XT1OFFG	DCOFFG
r-0	rw-(1)	r0	rw-(0)	r0	r0	rw-(0)	rw-(0)

Table 3-10. CSCTL7 Register Description

Bit	Field	Type	Reset	Description
15-14	Reserved	R	0h	Reserved. Always reads as 0.
13	FLLWARNEN	RW	0h	Warning enable. If this bit is set, an interrupt is generated based on the FLLUNLOCKHIS bits. If FLLUNLOCKHIS is not equal to 00, an OFIFG is generated. 0b = FLLUNLOCKHIS status cannot set OFIFG. 1b = FLLUNLOCKHIS status can set OFIFG.
12	FLLULPUC	RW	0h	FLL unlock PUC enable. If the FLLULPUC bit is set, a reset (PUC) is triggered if FLLULIFG is set. FLLULIFG indicates when FLLUNLOCK bits equal 10 (too fast). FLLULPUC is automatically cleared upon servicing the event. If FLLULPUC is cleared (0), no PUC can be triggered by FLLULIFG.
11-10	FLLUNLOCKHIS	RW	1h	Unlock history bits. These bits indicate the FLL unlock condition history. As soon as any unlock condition happens, the respective bits are set and remain set until cleared by software by writing 0 to it or by a POR. 00b = FLL is locked. No unlock situation has been detected since the last reset of these bits. 01b = DCOCLK has been too slow since the bits were cleared. 10b = DCOCLK has been too fast since the bits were cleared. 11b = DCOCLK has been both too fast and too slow since the bits were cleared.
9-8	FLLUNLOCK	R	3h	Unlock. These bits indicate the current FLL unlock condition. These bits are both set as long as the DCOFFG flag is set. 00b = FLL is locked. No unlock condition currently active. 01b = DCOCLK is currently too slow. 10b = DCOCLK is currently too fast. 11b = DCOERROR. DCO out of range.
7	Reserved	R	0h	Reserved. Always reads as 0.
6	ENSTFCNT1	RW	1h	Enable start counter for XT1. 0b = Startup fault counter disabled. Counter is cleared. 1b = Startup fault counter enabled.
5	Reserved	R	0h	Reserved. Always reads as 0.
4	FLLULIFG	RW	0h	FLL unlock interrupt flag. This flag is set when FLLUNLOCK bits equal 10b (DCO too fast). If FLLULPUC is also set, a PUC is triggered when FLLULIFG is set. 0b = FLLUNLOCK bits not equal to 10b 1b = FLLUNLOCK bits equal to 10b
3-2	Reserved	R	0h	Reserved. Always reads as 0.
1	XT1OFFG	RW	0h	XT1 oscillator fault flag. If this bit is set, the OFIFG flag is also set. XT1OFFG is set if a XT1 fault condition exists. XT1OFFG can be cleared by software. If the XT1 fault condition still remains, XT1OFFG is set. 0b = No fault condition occurred 1b = XT1 fault. An XT1 fault occurred

Table 3-10. CSCTL7 Register Description (continued)

Bit	Field	Type	Reset	Description
0	DCOFFG	RW	0h	DCO fault flag. If this bit is set, the OFIFG flag is also set. The DCOFFG bit is set if DCO = {0} or DCO = {511}. DCOFFG can be cleared by software. If the DCO fault condition still remains, DCOFFG is set. As long as DCOFFG is set, FLLUNLOCK shows the DCOERROR condition. 0b = No fault condition occurred 1b = DCO fault. A DCO fault occurred

3.3.9 CSCTL8 Register

Clock System Control Register 8

Figure 3-15. CSCTL8 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				MODOSCREQEN	SMCLKREQEN	MCLKREQEN	ACLKREQEN
r0	r0	r0	r0	rw-(0)	rw-(1)	rw-(1)	rw-(1)

Table 3-11. CSCTL8 Register Description

Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved. Always reads as 0.
3	MODOSCREQEN	RW	0h	MODOSC clock request enable. Setting this enables conditional module requests for MODOSC. 0b = MODOSC conditional requests are disabled. 1b = MODOSC 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 114
4.2 Interrupts 116
4.3 CPU Registers 117
4.4 Addressing Modes 123
4.5 MSP430 and MSP430X Instructions 140
4.6 Instruction Set Description 156

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.

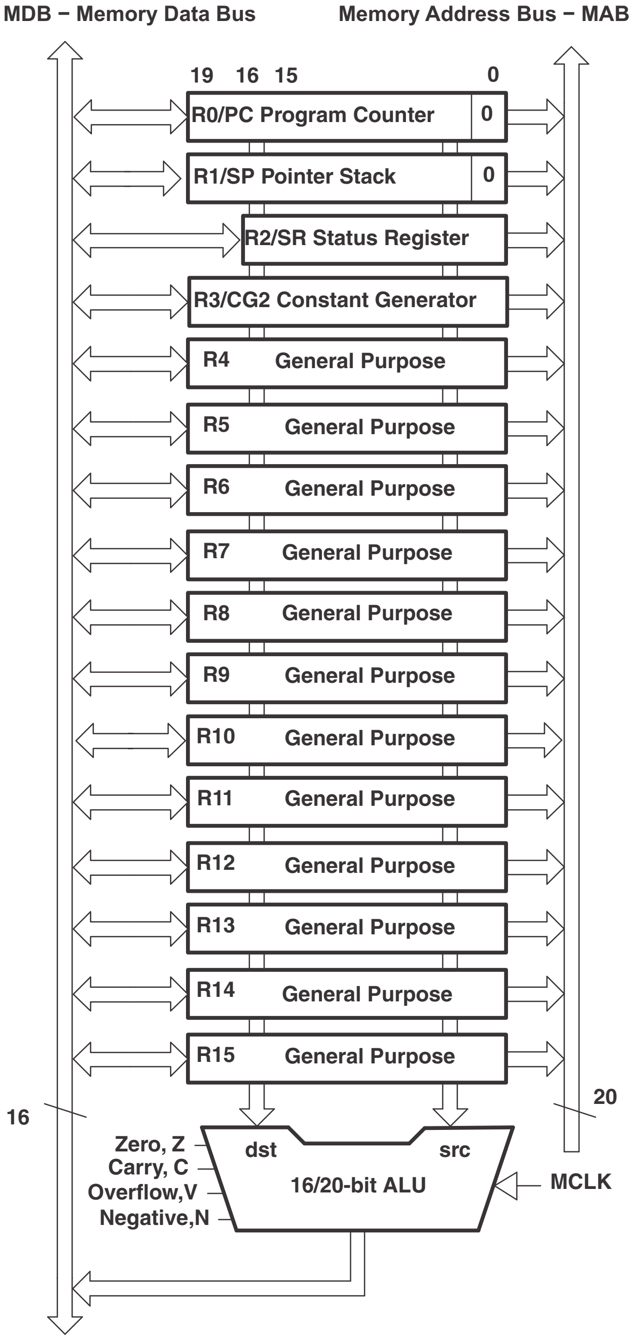


Figure 4-1. MSP430X CPU Block Diagram

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 64KB memory. This means all interrupt handlers must start in the lower 64KB 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.

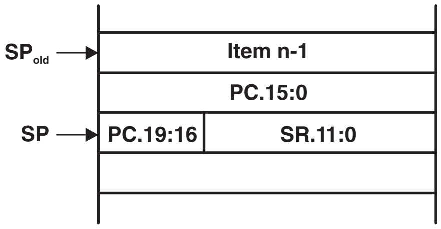


Figure 4-2. PC Storage on the Stack for Interrupts

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.

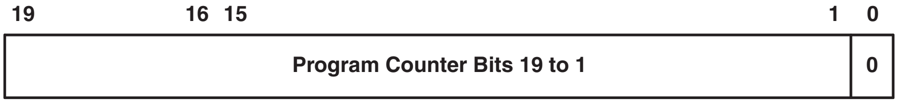


Figure 4-3. Program Counter

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

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

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

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

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

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

The BR and CALL instructions reset the upper four PC bits to 0. Only addresses in the lower 64KB address range can be reached with the BR or CALL instruction. When branching or calling, addresses beyond the lower 64KB 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.

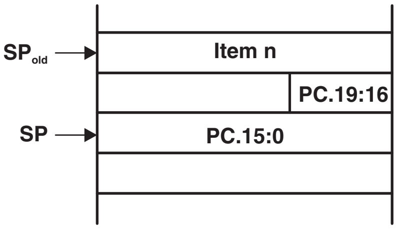


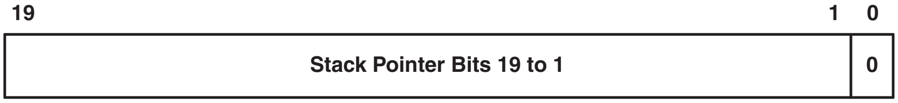
Figure 4-4. PC Storage on the Stack for CALLA

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

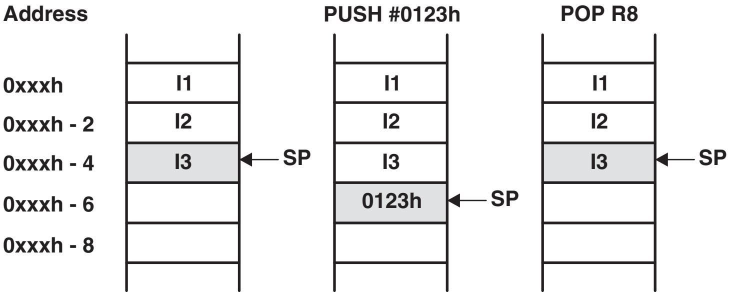


Figure 4-6. Stack Usage

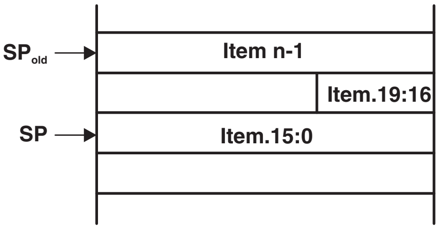
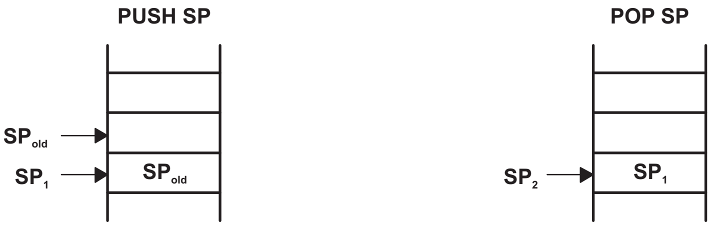


Figure 4-7. PUSHX.A Format on the Stack

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.

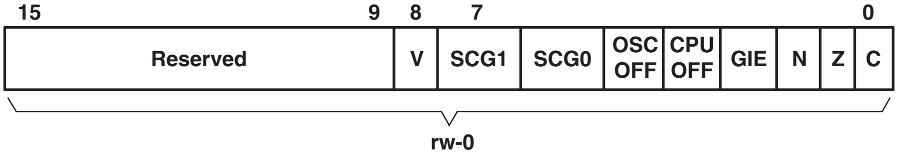


Figure 4-9. SR Bits

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. When this bit is set, it turns off the LFXT1 crystal oscillator when LFXT1CLK is not used for MCLK or SMCLK.	
CPUOFF	CPU off. When this bit is set, it turns off the CPU.	
SCG1	The bits CPUOFF, OSCOFF, SCG0 and SCG1 request the system to enter a low-power mode	
SCG0		
OSCOFF		
CPUOFF		
GIE	General interrupt enable. When this bit is set, it enables maskable interrupts. When it is 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.

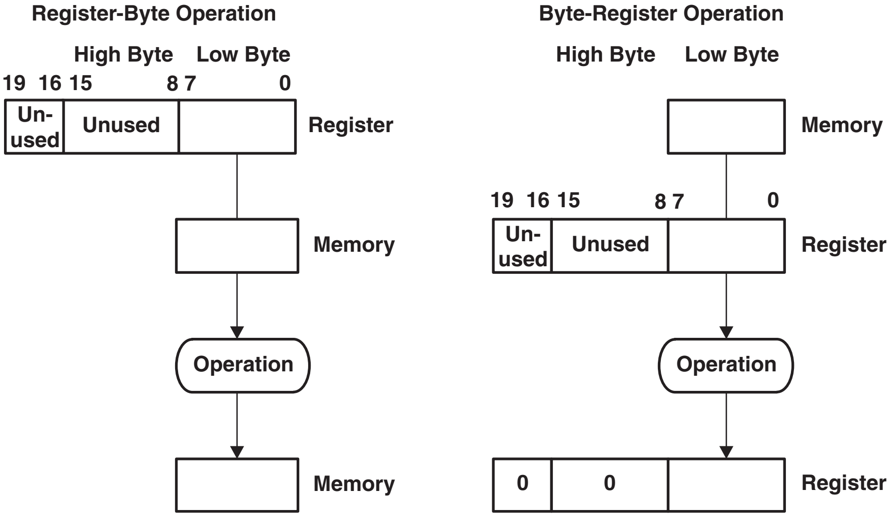


Figure 4-10. Register-Byte and Byte-Register Operation

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.

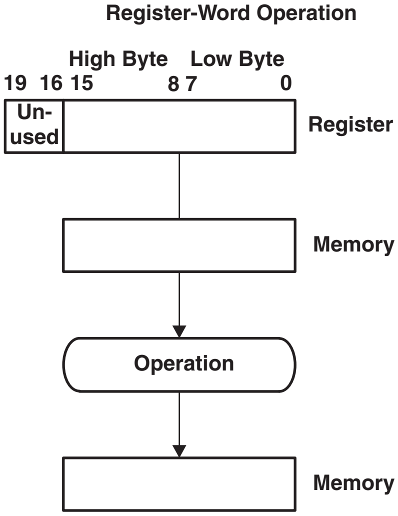


Figure 4-11. Register-Word Operation

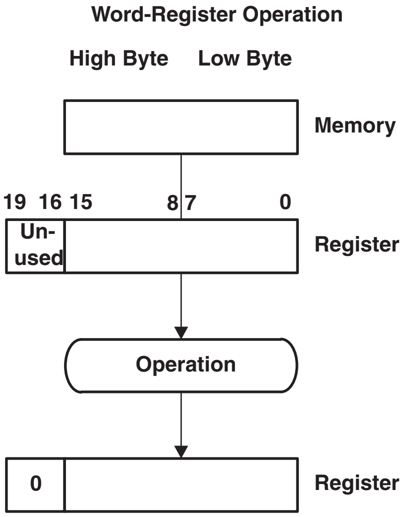


Figure 4-12. Word-Register Operation

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.

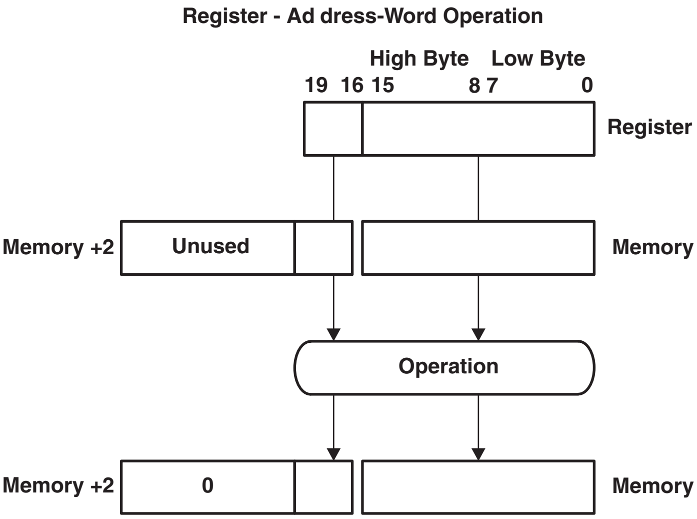


Figure 4-13. Register – Address-Word Operation

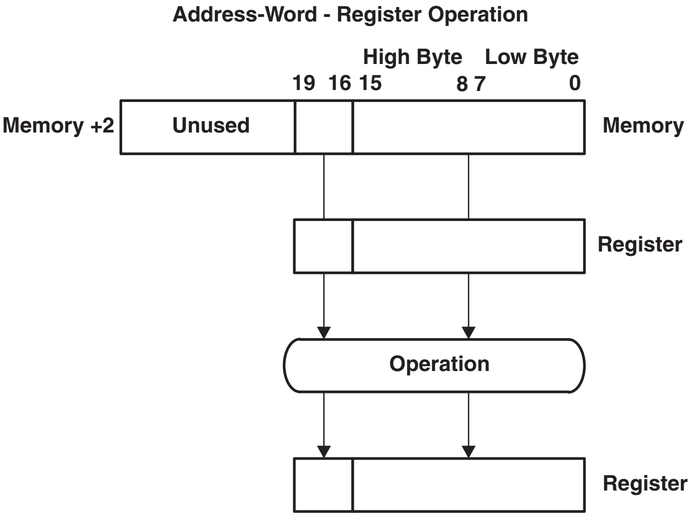


Figure 4-14. Address-Word – Register Operation

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.

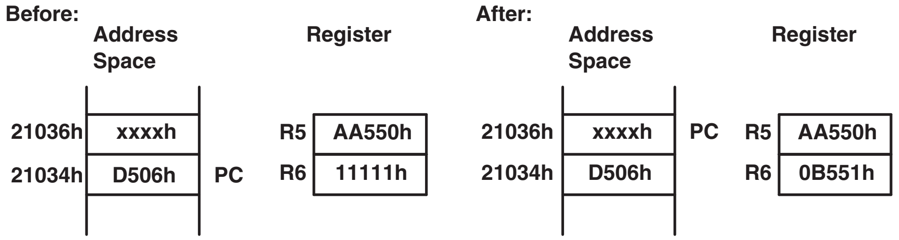
Word operation: Word operation reads the 16 LSBs of the source register Rsrc and writes the result to the 16 LSBs of the destination register Rdst. The bits Rdst.19:16 are cleared. The register Rsrc is not modified.

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.

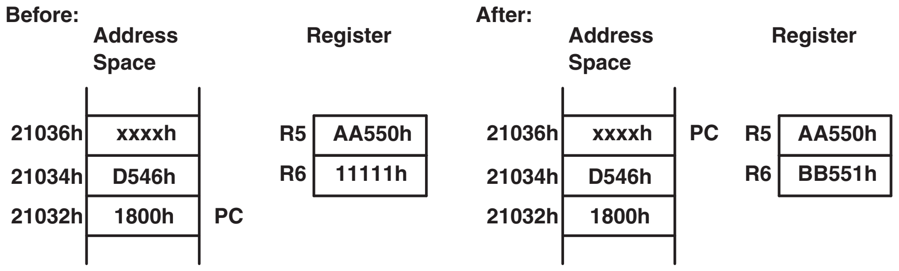


A550h.or.1111h = B551h

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

4.4.2.1 Indexed Mode in Lower 64KB Memory

If the CPU register Rn points to an address in the lower 64KB 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 64KB and does not overflow or underflow out of the lower 64KB 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.

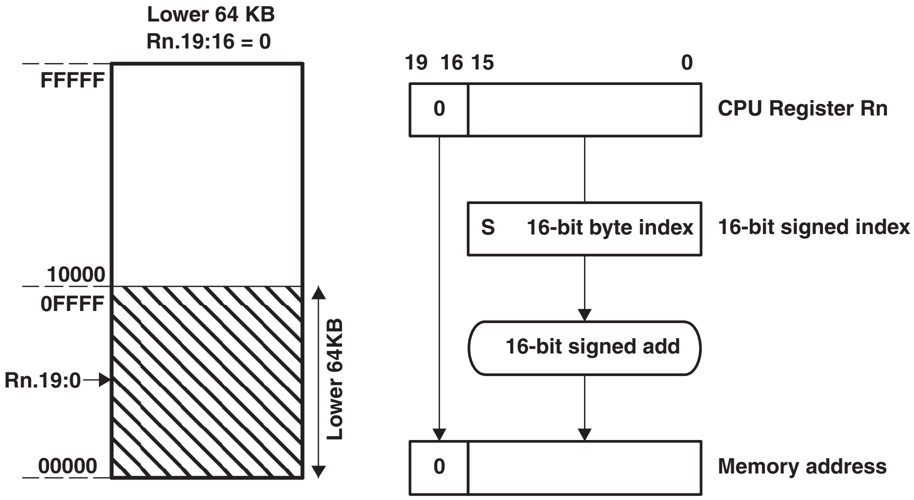
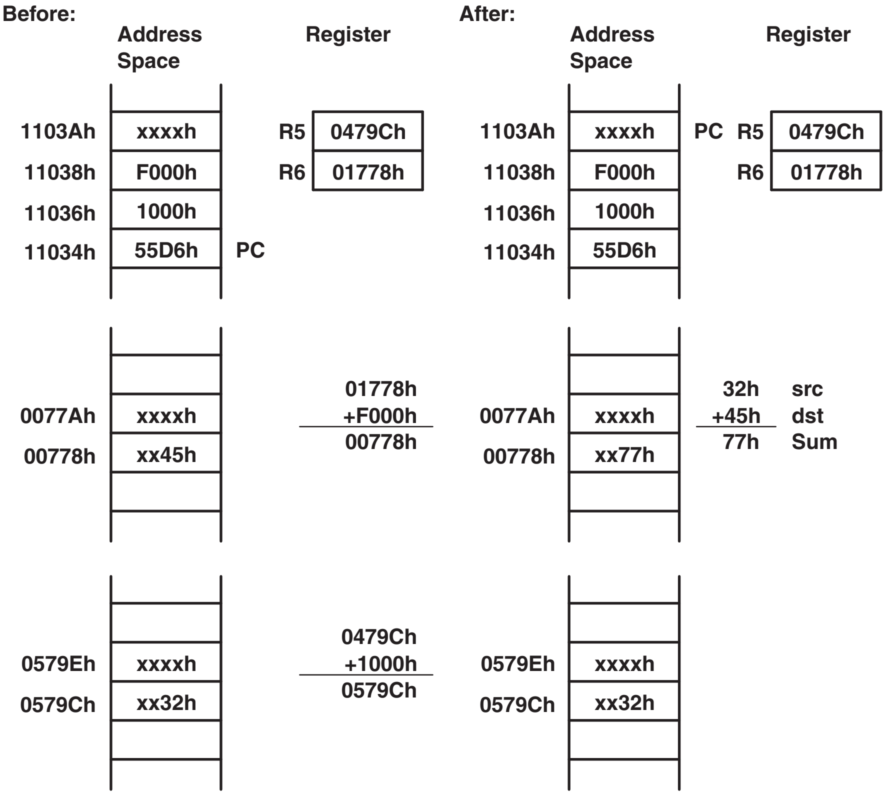


Figure 4-15. Indexed Mode in Lower 64KB

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 64KB 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 64KB 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 ±32KB, 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 64KB memory space (see Figure 4-16 and Figure 4-17).

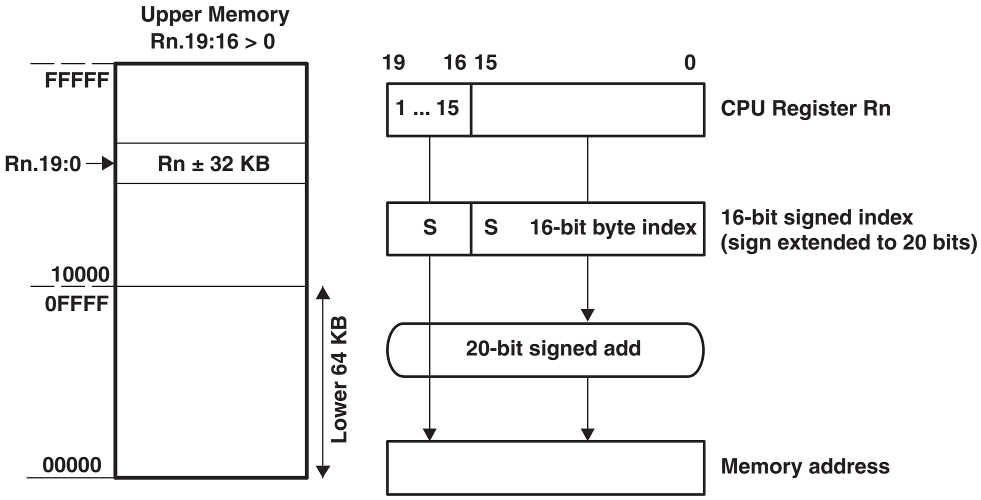


Figure 4-16. Indexed Mode in Upper Memory

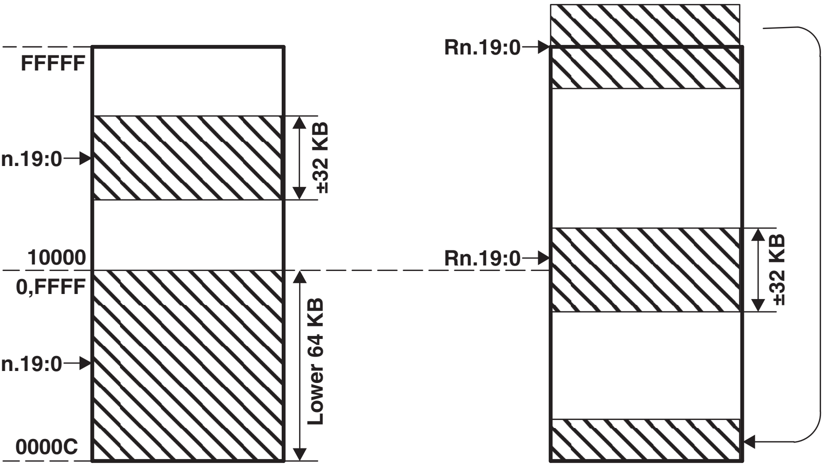


Figure 4-17. Overflow and Underflow for Indexed Mode

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.

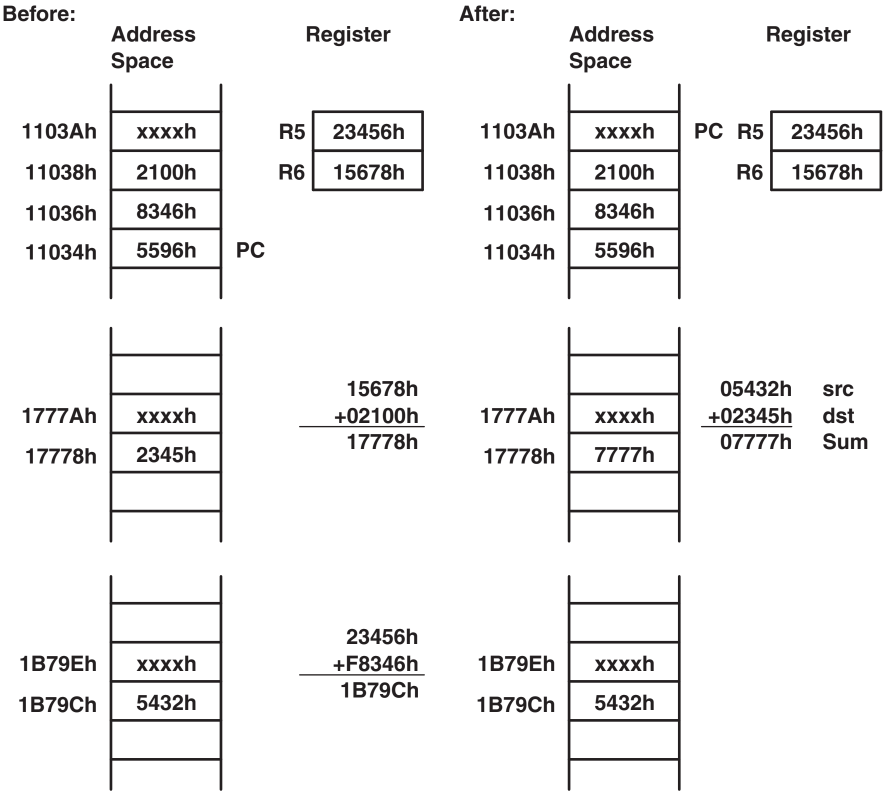


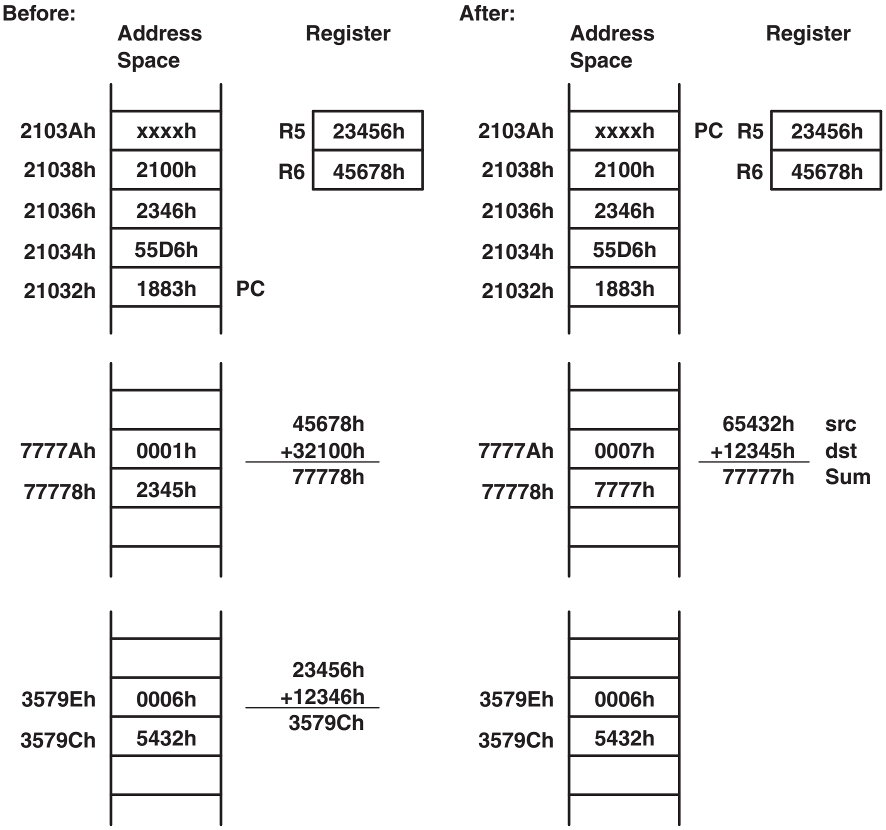
Figure 4-18. Example for Indexed Mode

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

4.4.3.1 Symbolic Mode in Lower 64KB

If the PC points to an address in the lower 64KB 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 64KB and does not overflow or underflow out of the lower 64KB 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.

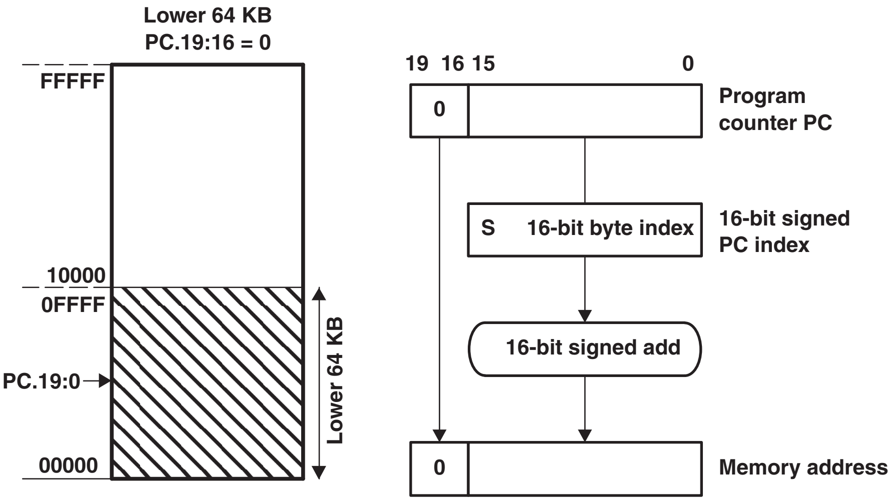
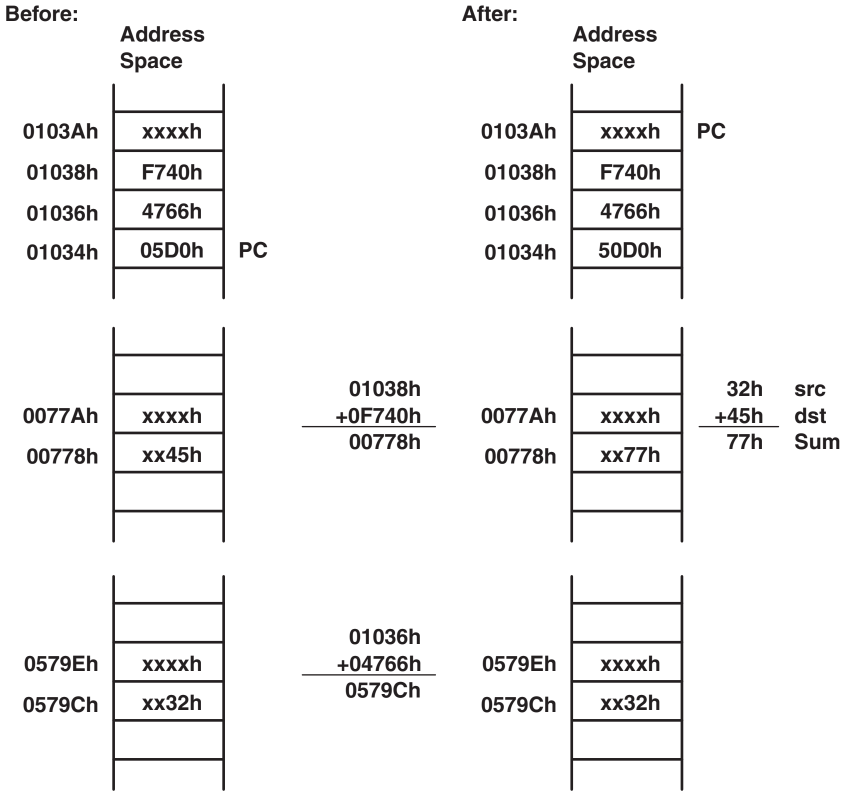


Figure 4-19. Symbolic Mode Running in Lower 64KB

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 64KB.
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 64KB 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 ± 32KB, 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 64KB memory space as shown in Figure 4-20 and Figure 4-21.

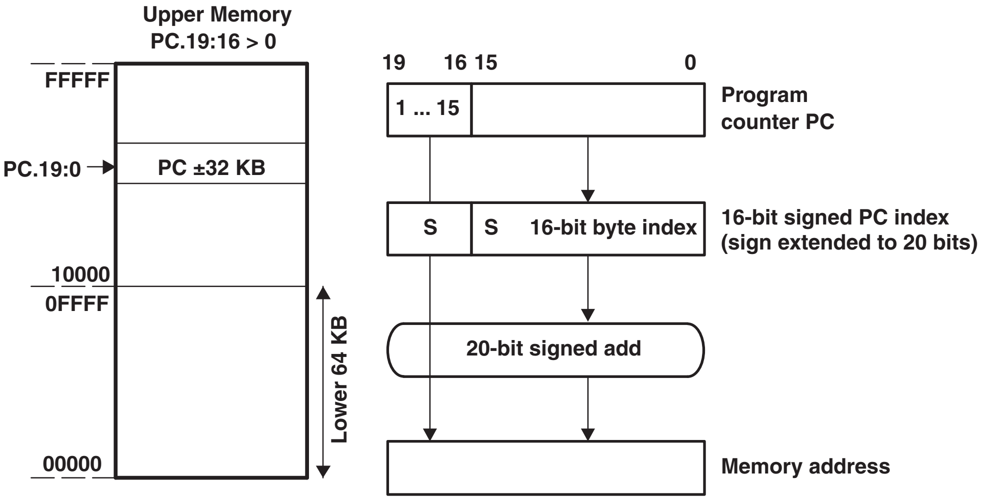


Figure 4-20. Symbolic Mode Running in Upper Memory

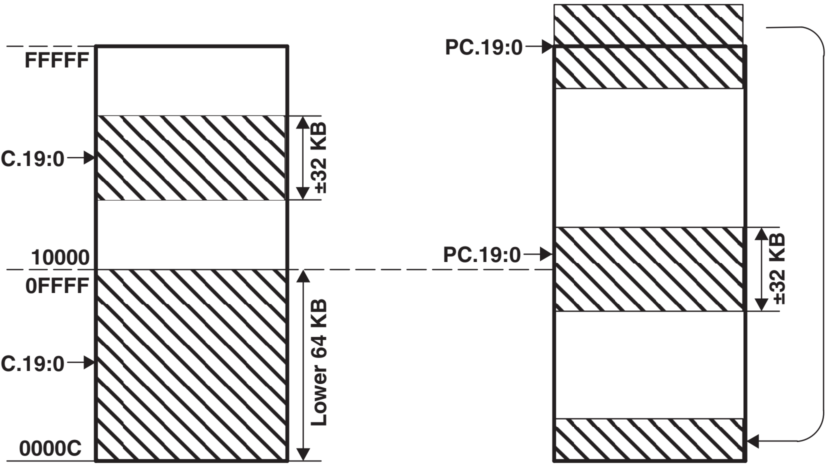


Figure 4-21. Overflow and Underflow for Symbolic Mode

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.

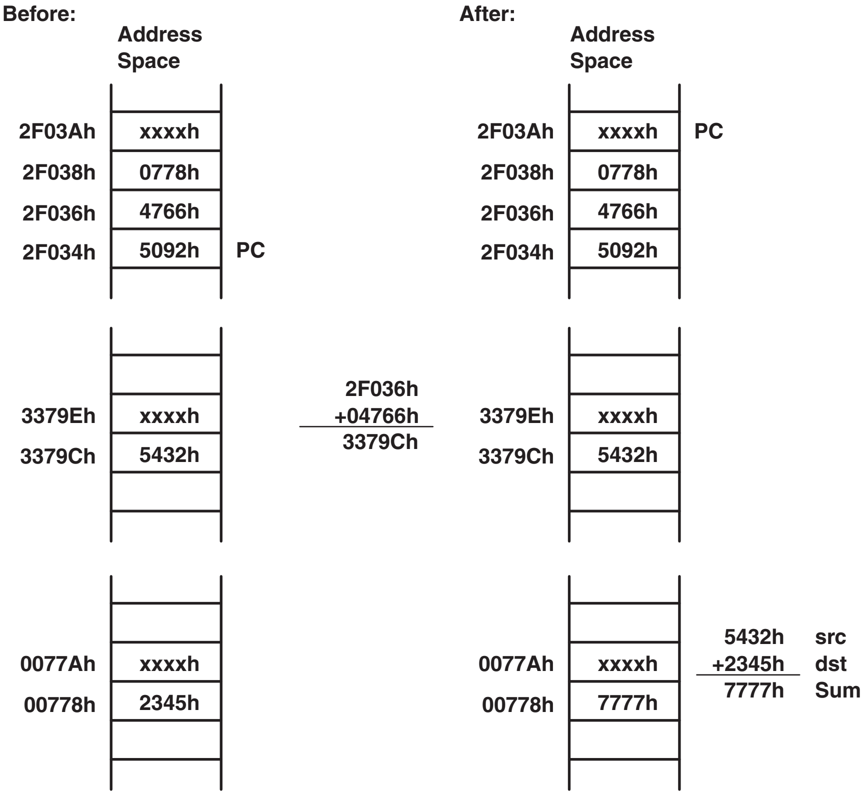
Comment: Valid for source and destination. The assembler calculates the PC index and inserts it

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.

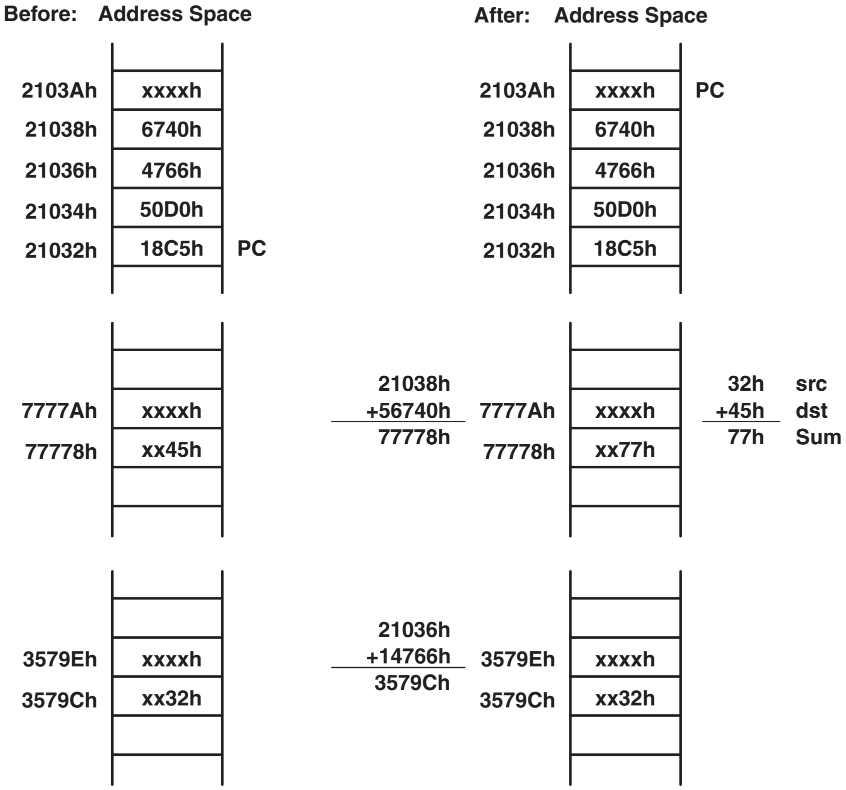
Comment: Valid for source and destination. The assembler calculates the register index and inserts it.

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

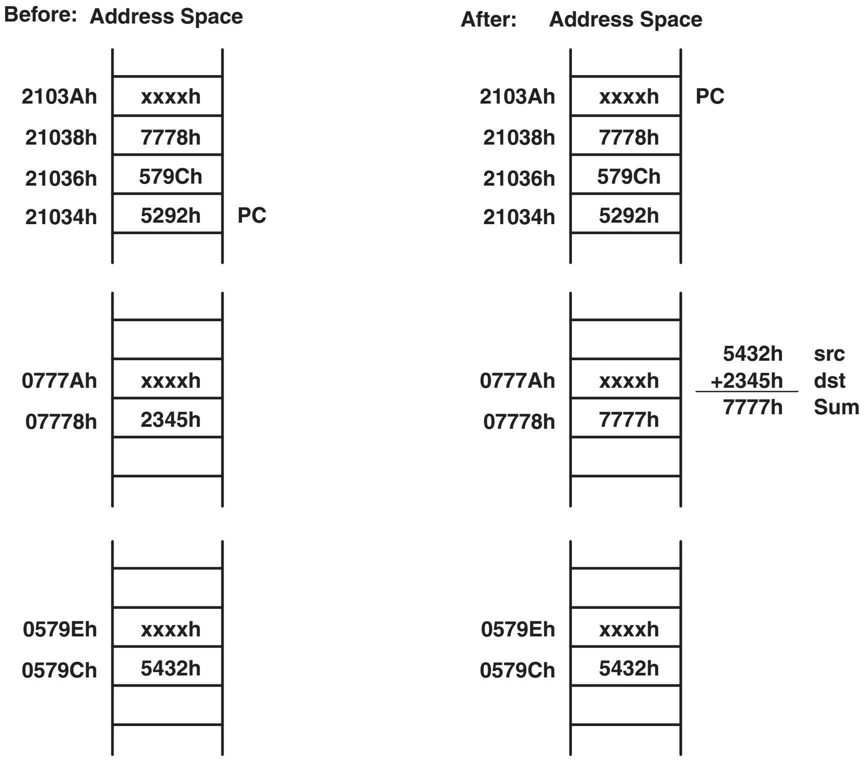
The Absolute mode uses the contents of the word following the instruction as the address of the operand. The Absolute mode has two addressing possibilities:

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

4.4.4.1 Absolute Mode in Lower 64KB

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 64KB 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.

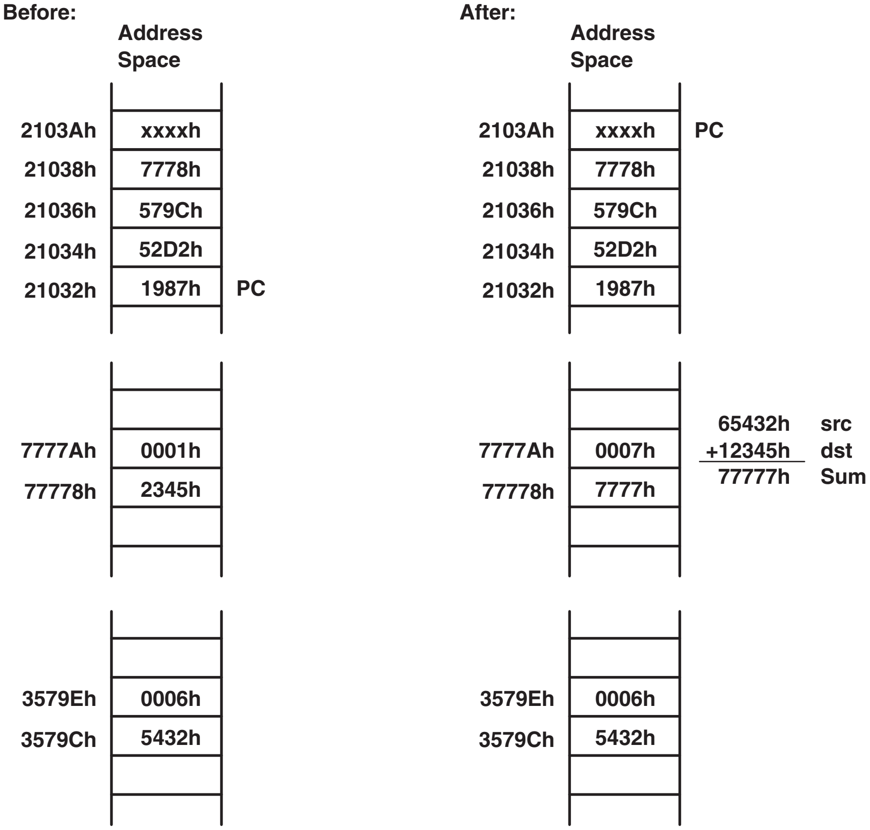
Length:	Two or three words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the index from 0 and inserts it.
Example:	`ADD.W &EDE,&TONI ;` This instruction adds the 16-bit data contained in the absolute source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

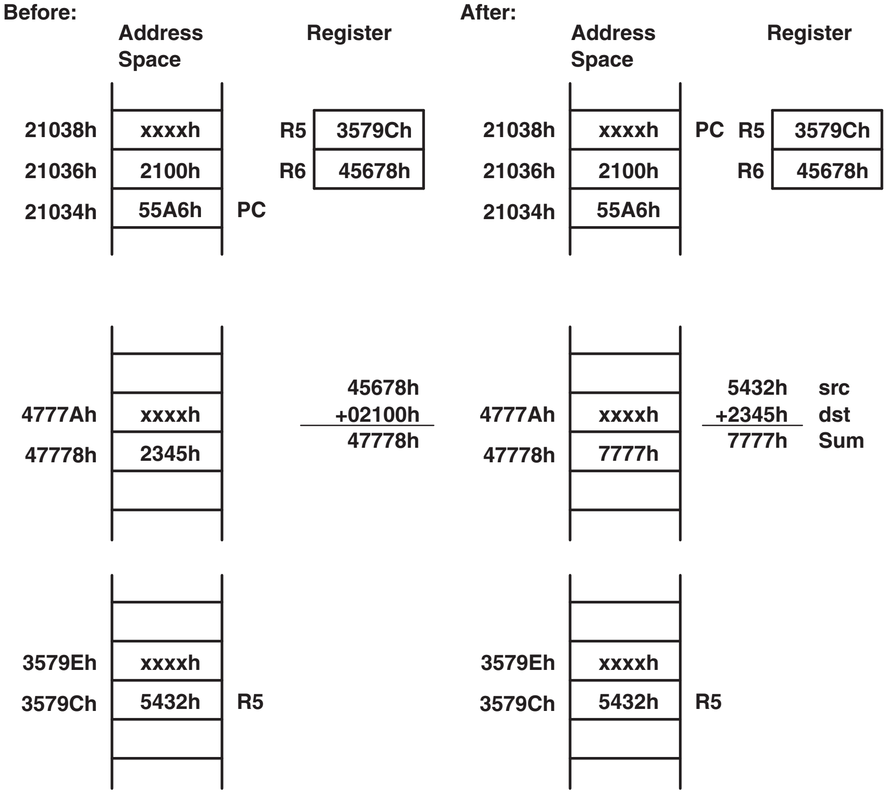
Comment: Valid only for the source operand. The substitute for the destination operand is 0(Rdst).

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.

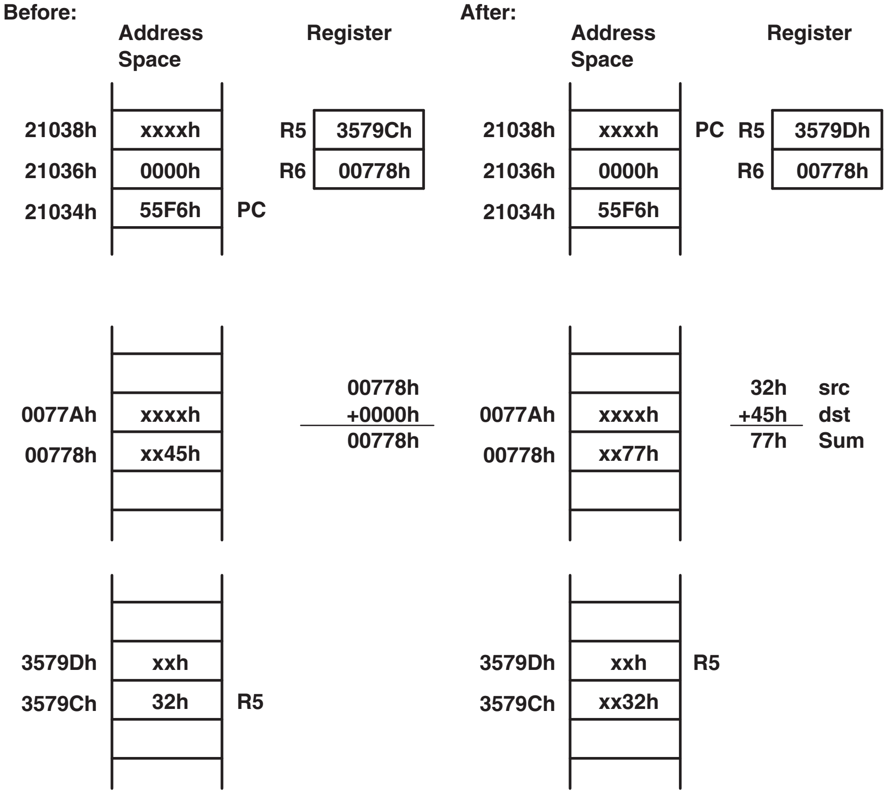
Destination: Word pointed to by R6 + 2100h, which results in address 45678h + 2100h = 7778h



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

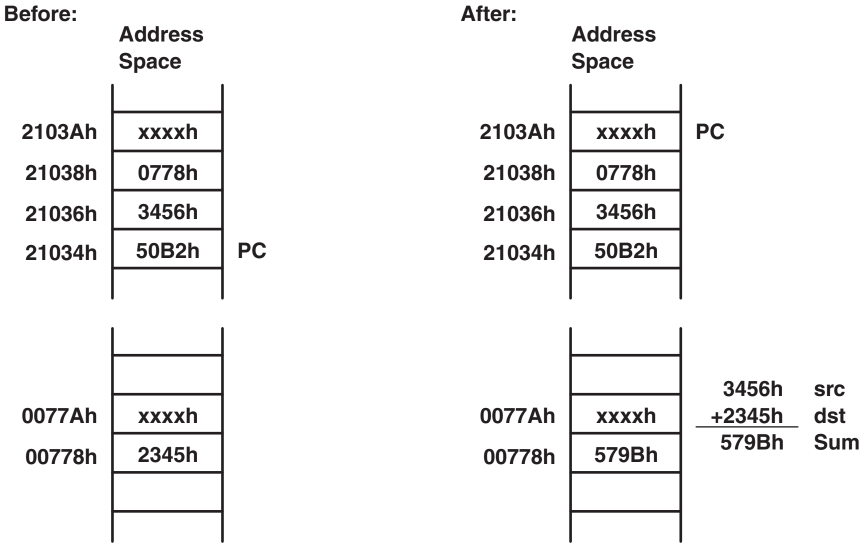
The Immediate mode allows accessing constants as operands by including the constant in the memory location following the instruction. The PC is used with the Indirect Autoincrement mode. The PC points to the immediate value contained in the next word. After the fetching of the immediate operand, the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

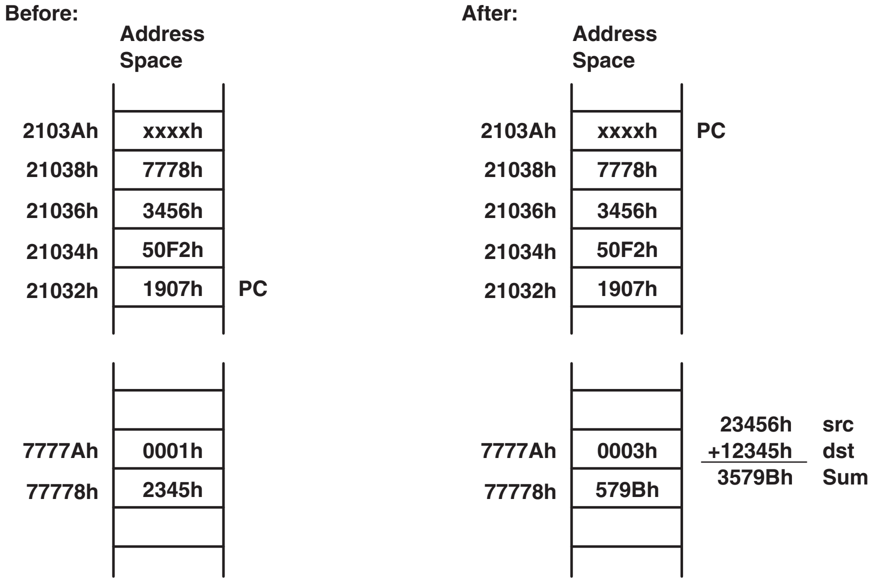
Comment: Valid only for the source operand

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 64KB. 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 + 32KB.
- 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.

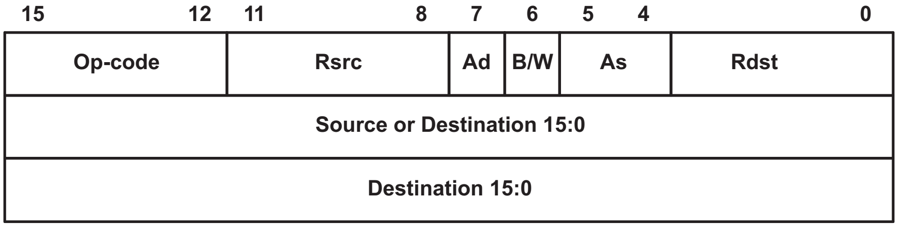
Section 4.5.1 lists and describes the MSP430 instructions, and Section 4.5.2 lists and describes the MSP430X instructions.

4.5.1 MSP430 Instructions

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

4.5.1.1 MSP430 Double-Operand (Format I) Instructions

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



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

Figure 4-22. MSP430 Double-Operand Instruction Format

Table 4-4. MSP430 Double-Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits[1]			
			V	N	Z	C
`MOV(.B)`	src,dst	src → dst	–	–	–	–
`ADD(.B)`	src,dst	src + dst → dst	*	*	*	*
`ADDC(.B)`	src,dst	src + dst + C → dst	*	*	*	*
`SUB(.B)`	src,dst	dst + .not.src + 1 → dst	*	*	*	*
`SUBC(.B)`	src,dst	dst + .not.src + C → dst	*	*	*	*
`CMP(.B)`	src,dst	dst - src	*	*	*	*
`DADD(.B)`	src,dst	src + dst + C → dst (decimally)	*	*	*	*
`BIT(.B)`	src,dst	src .and. dst	0	*	*	$\overline{Z}$
`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.

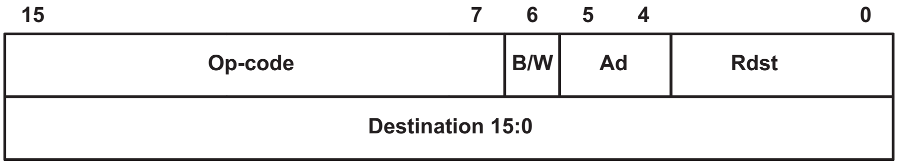


Figure 4-23. MSP430 Single-Operand Instructions

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 64KB	–	–	–	–
`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.

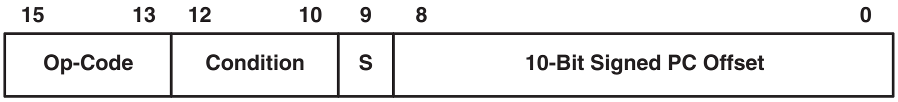


Figure 4-24. Format of Conditional Jump Instructions

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`	*	*	*	*

[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
`INV(.B) dst`	Invert dst	`XOR(.B) #–1,dst`	*	*	*	*
`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.

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

Figure 4-25. Extension Word for Register Modes

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

Bit	Description		
15:11	Extension word op-code. Op-codes 1800h to 1FFFh are extension words.		
10:9	Reserved		
ZC	Zero carry		
	0	The executed instruction uses the status of the carry bit C.	
	1	The executed instruction uses the carry bit as 0. The carry bit is defined by the result of the final operation after instruction execution.	
#	Repetition		
	0	The number of instruction repetitions is set by extension word bits 3:0.	
	1	The number of 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.

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

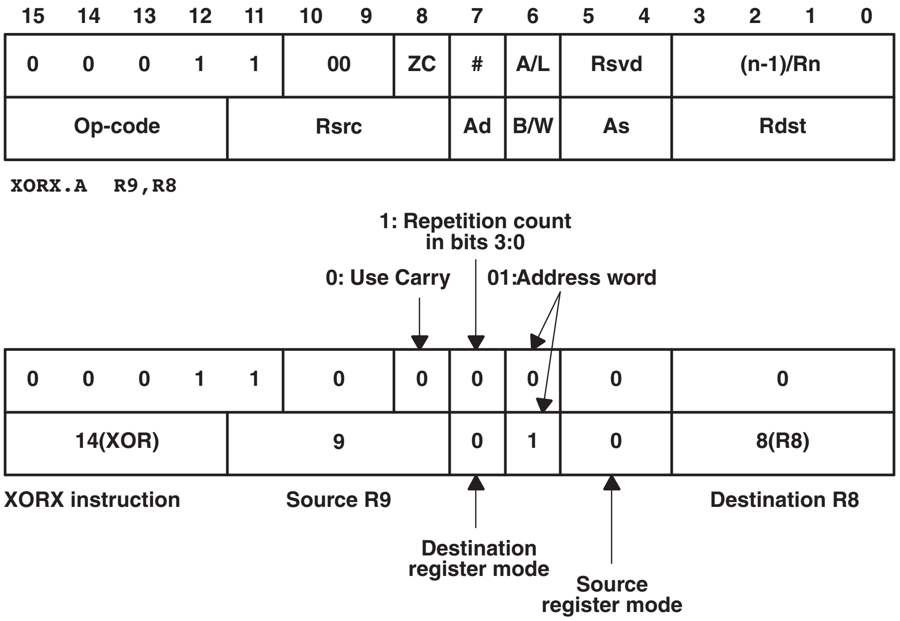
Figure 4-26. Extension Word for Non-Register Modes

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

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

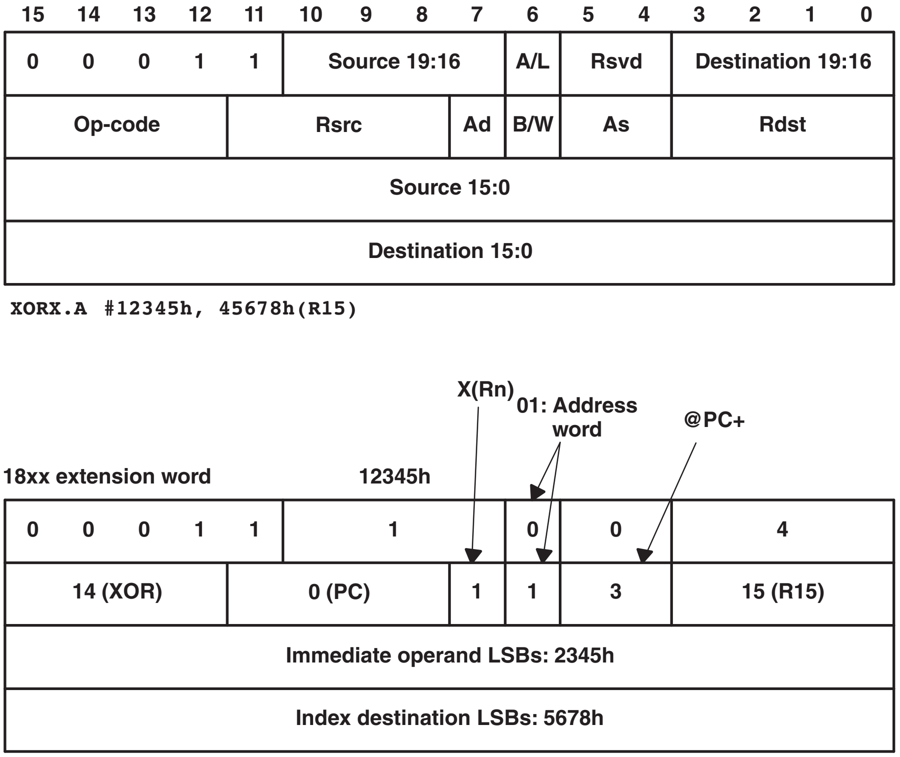
NOTE: B/W and A/L bit settings for SWPBX and SXTX

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



```
XORX.A   R9,R8
```

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



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

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.

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

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

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

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

Figure 4-29. Extended Format I Instruction Formats

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

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

Figure 4-30. 20-Bit Addresses in Memory

4.5.2.4 Extended Single-Operand (Format II) Instructions

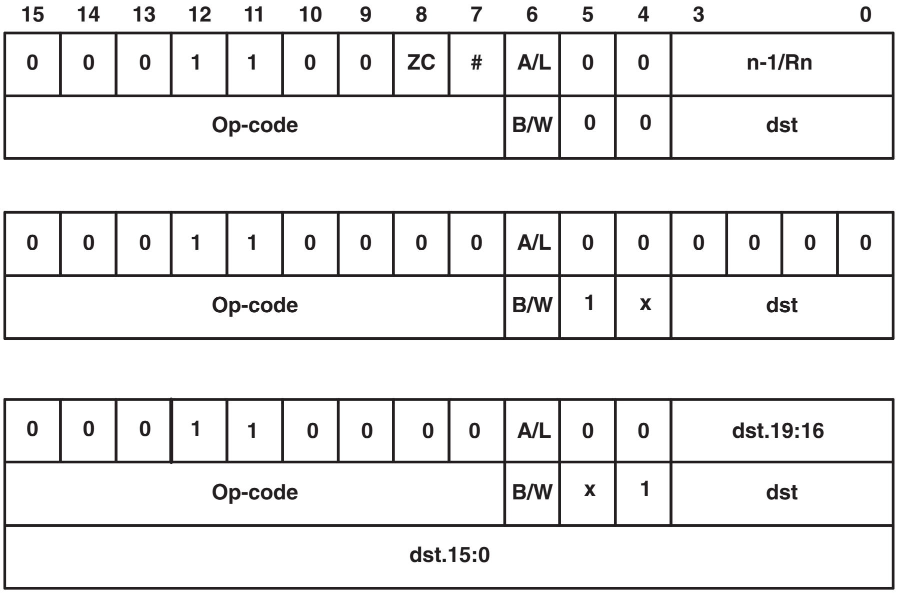
Extended MSP430X Format II instructions are listed in Table 4-14.

Table 4-14. Extended Single-Operand Instructions

Mnemonic	Operands	Operation	n	Status Bits[1]			
				V	N	Z	C
`CALLA`	dst	Call indirect to subroutine (20-bit address)		–	–	–	–
`POPM.A`	#n,Rdst	Pop n 20-bit registers from stack	1 to 16	–	–	–	–
`POPM.W`	#n,Rdst	Pop n 16-bit registers from stack	1 to 16	–	–	–	–
`PUSHM.A`	#n,Rsrc	Push n 20-bit registers to stack	1 to 16	–	–	–	–
`PUSHM.W`	#n,Rsrc	Push n 16-bit registers to stack	1 to 16	–	–	–	–
`PUSHX(.B,.A)`	src	Push 8-, 16-, 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.



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

0	0	0	1	1	0	0	0	0	A/L	0	0	0	0	0	0
Op-code									B/W	1	x	dst			

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

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.

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

Figure 4-32. PUSHM and POPM Instruction Format

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

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

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

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

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

Figure 4-34. BRA Instruction Format

15 4	3 0
Op-code	Rdst

Op-code	Rdst
index15:0	

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

Figure 4-35. CALLA Instruction Format

4.5.2.5 Extended Emulated Instructions

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

Table 4-15. Extended Emulated Instructions

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

4.5.2.6 MSP430X Address Instructions

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

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

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

[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	Extension word for Format I and Format II instructions															
1Cxx																
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 64KB. X is within ±32KB.

```
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.

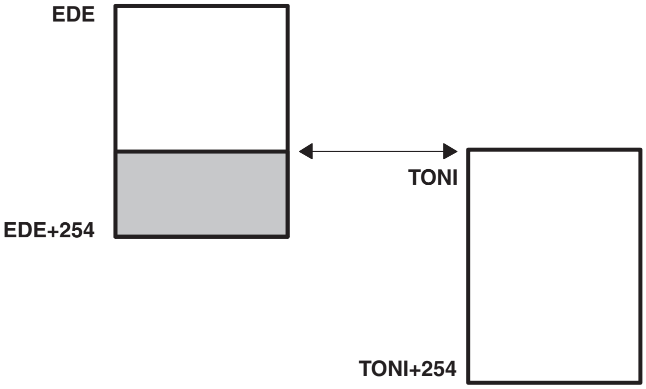


Figure 4-36. Decrement Overlap

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
```

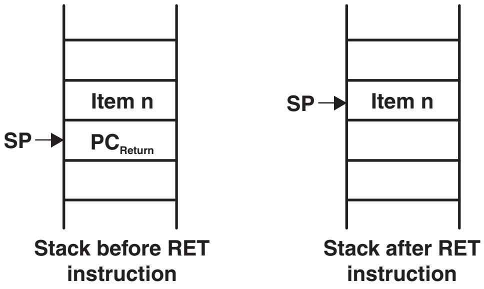


Figure 4-37. Stack After a RET Instruction

4.6.2.37 RETI

RETI Return from interrupt

Syntax `RETI`

Operation

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

Description The SR is restored to the value at the beginning of the interrupt service routine. This includes the four MSBs of the PC.19:16. The SP is incremented by two afterward.

The 20-bit PC is restored from PC.19:16 (from same stack location as the status bits) and PC.15:0. The 20-bit PC is restored to the value at the beginning of the interrupt service routine. The program continues at the address following the last executed instruction when the interrupt was granted. The SP is incremented by two afterward.

Status Bits

N: Restored from stack

C: Restored from stack

Z: Restored from stack

V: Restored from stack

Mode Bits OSCOFF, CPUOFF, and GIE are restored from stack.

Example Interrupt handler in the lower 64 K. A 20-bit return address is stored on the stack.

```
INTRPT  PUSHM.A   #2,R14    ; Save R14 and R13 (20-bit data)
        ...                 ; Interrupt handler code
        POPM.A    #2,R14    ; Restore R13 and R14 (20-bit data)
        RETI                ; Return to 20-bit address in full memory range
```

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.

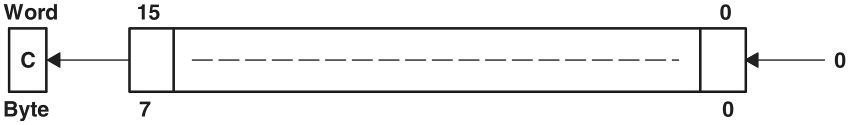


Figure 4-38. Destination Operand—Arithmetic Shift Left

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).

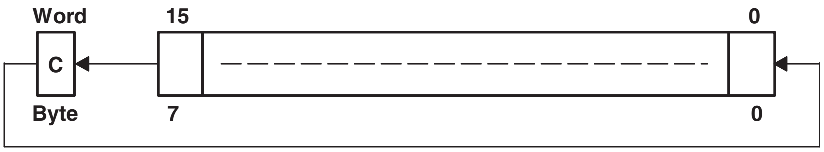


Figure 4-39. Destination Operand—Carry Left Shift

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
```

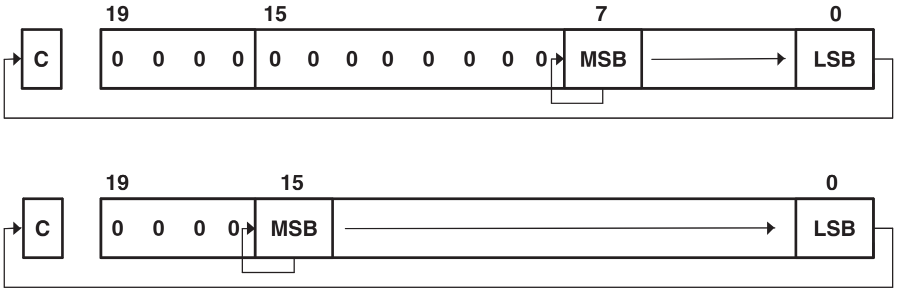


Figure 4-40. Rotate Right Arithmetically RRA.B and RRA.W

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
```

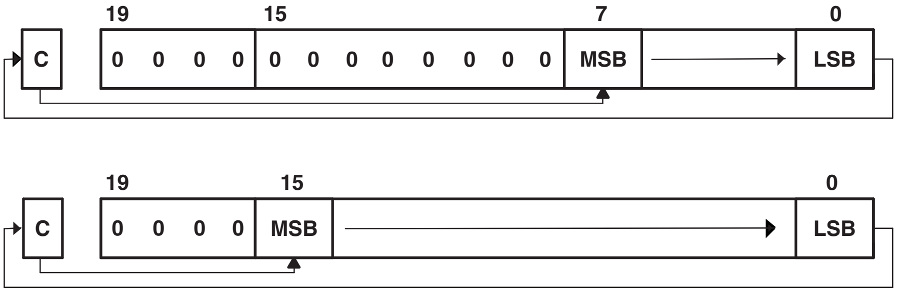


Figure 4-41. Rotate Right Through Carry RRC.B and RRC.W

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
```

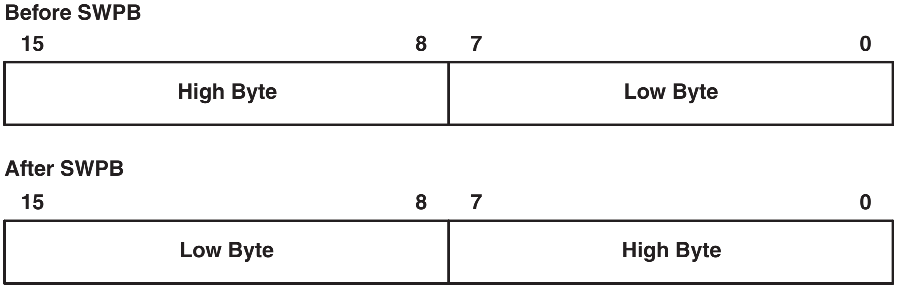


Figure 4-42. Swap Bytes in Memory

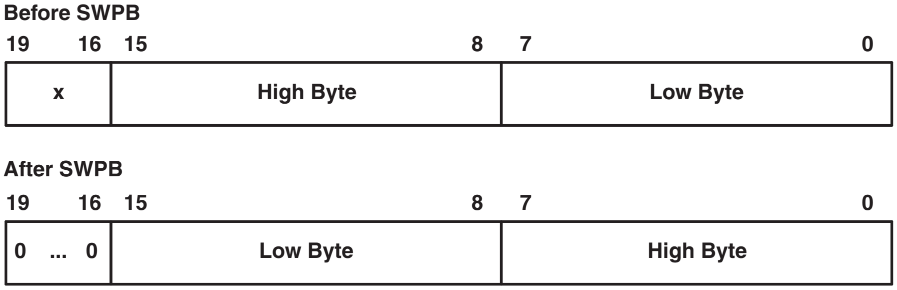


Figure 4-43. Swap Bytes in a Register

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
```

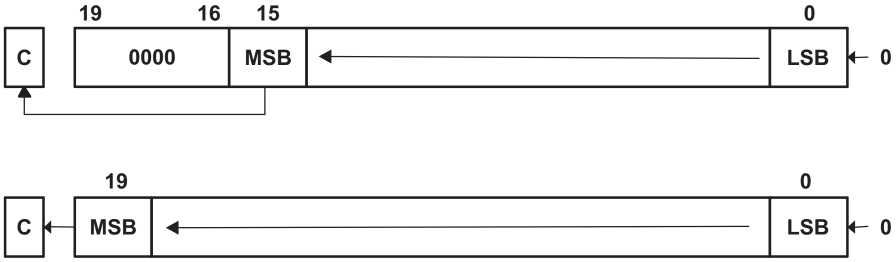


Figure 4-44. Rotate Left Arithmetically—RLAM[.W] and RLAM.A

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)
```

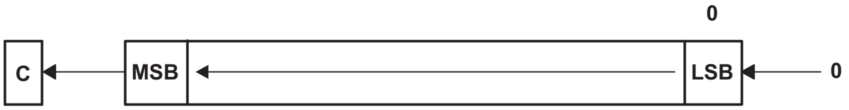


Figure 4-45. Destination Operand-Arithmetic Shift Left

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)
```

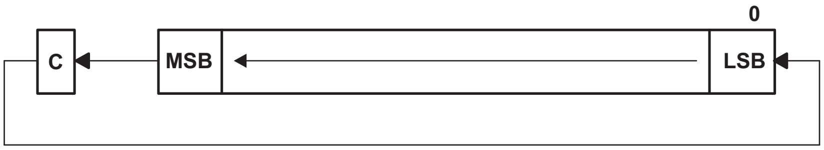


Figure 4-46. Destination Operand-Carry Left Shift

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
```

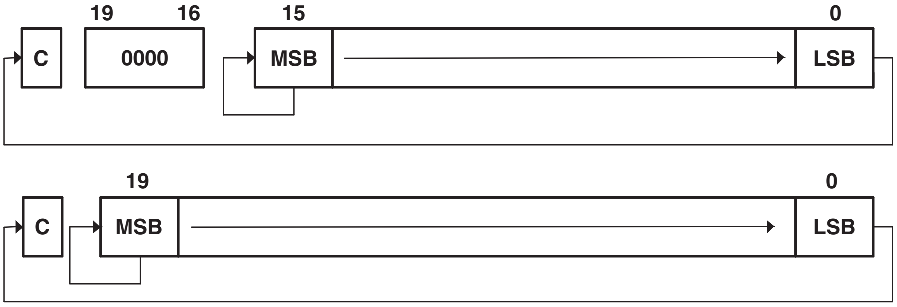


Figure 4-47. Rotate Right Arithmetically RRAM[.W] and RRAM.A

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
```

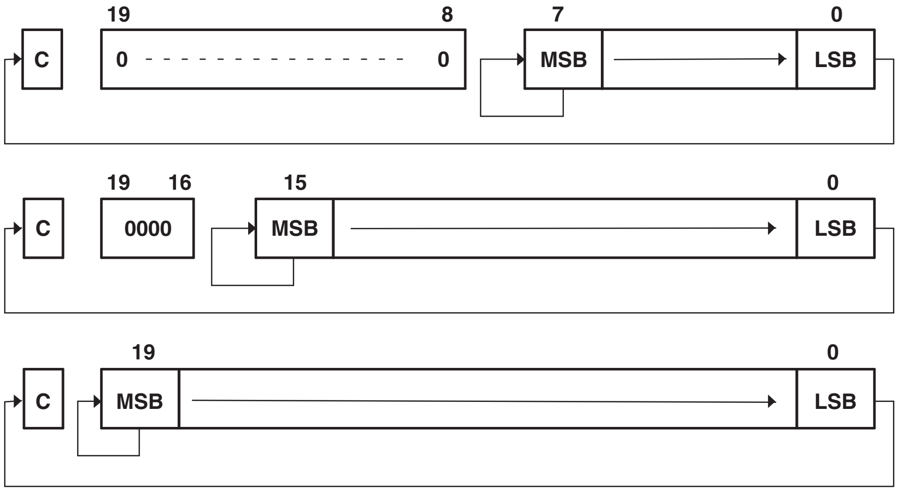


Figure 4-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode

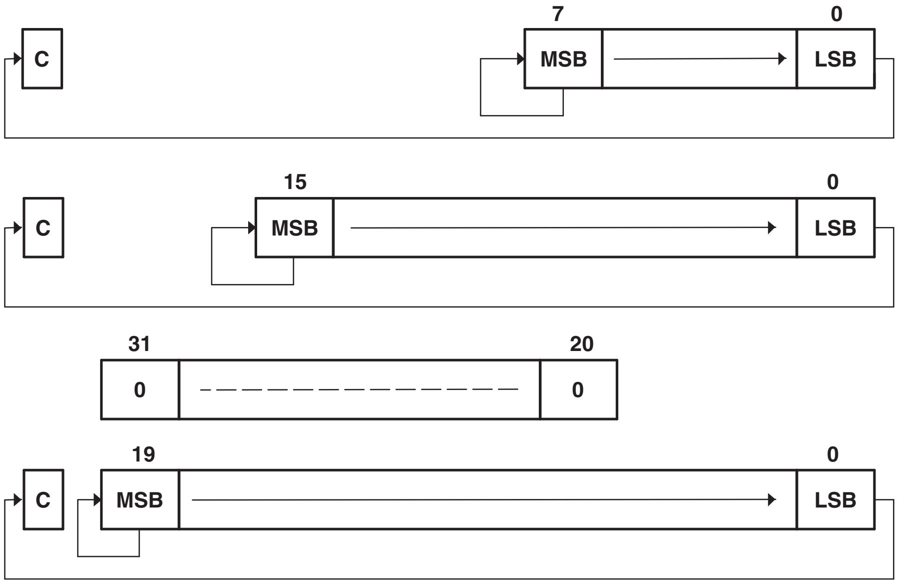


Figure 4-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode

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
```

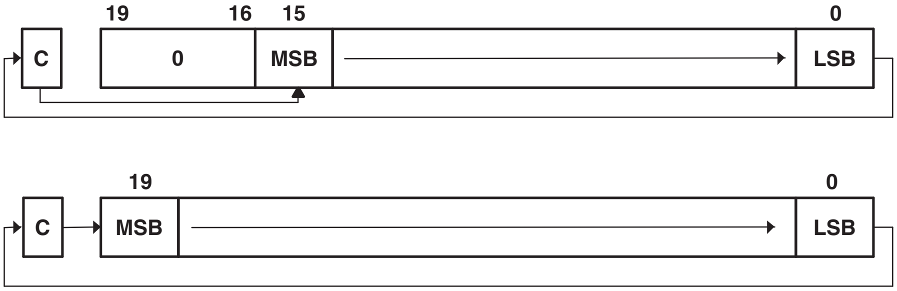


Figure 4-50. Rotate Right Through Carry RRCM[.W] and RRCM.A

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
```

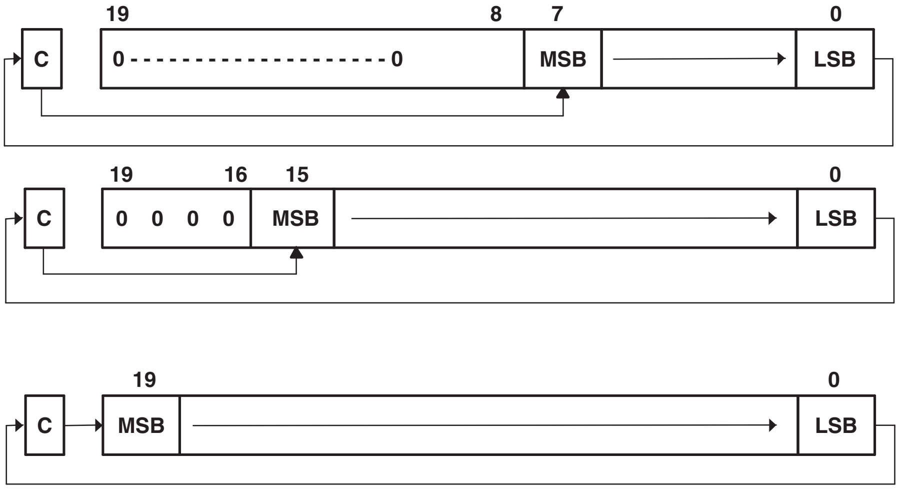


Figure 4-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode

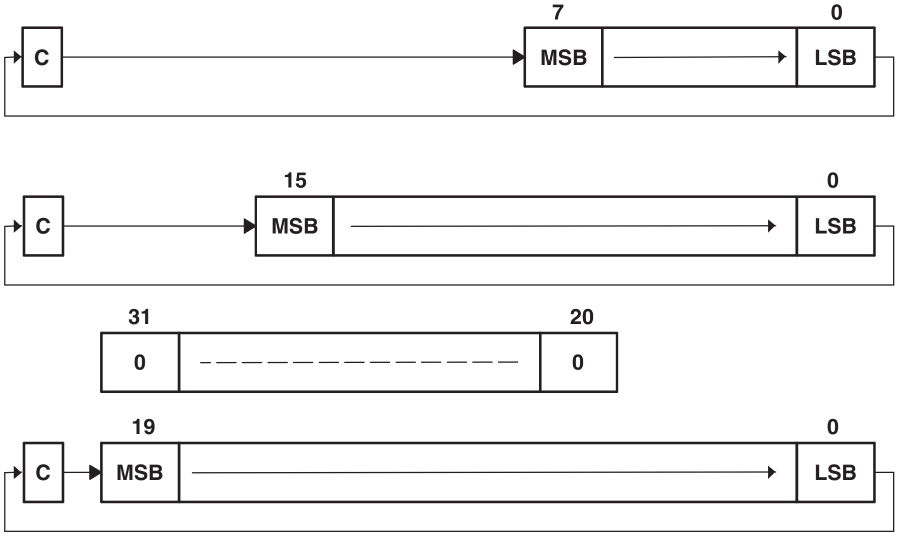


Figure 4-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode

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
```

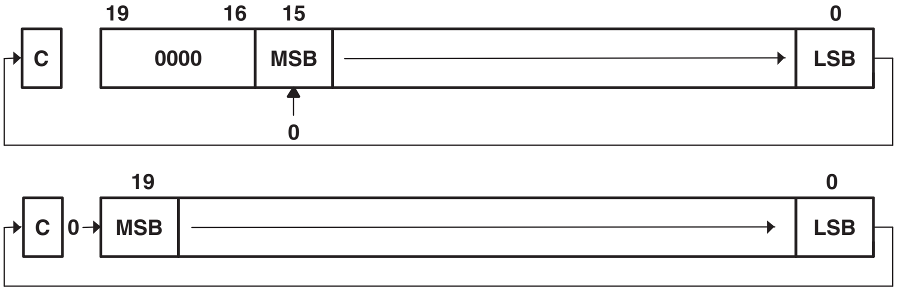


Figure 4-53. Rotate Right Unsigned RRUM[.W] and RRUM.A

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
```

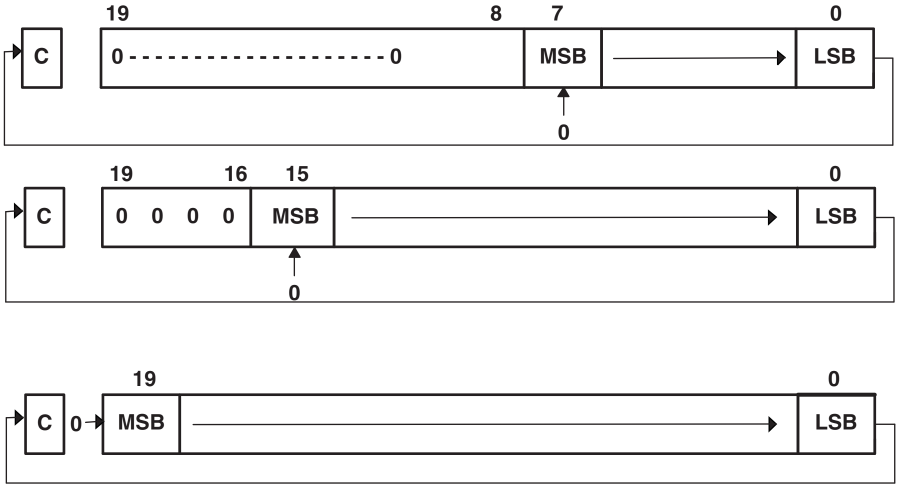


Figure 4-54. Rotate Right Unsigned RRUX(.B,.A) – Register Mode

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
```

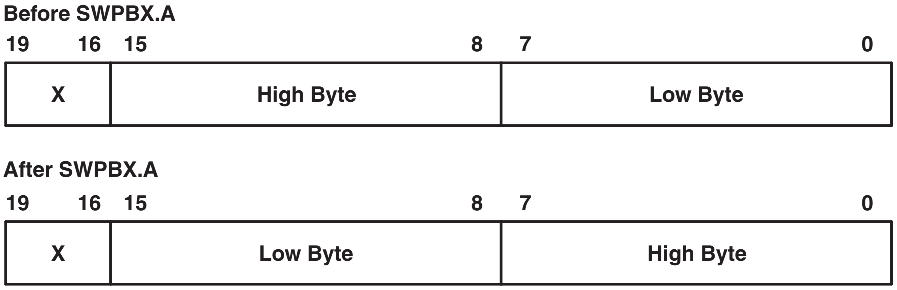


Figure 4-55. Swap Bytes SWPBX.A Register Mode

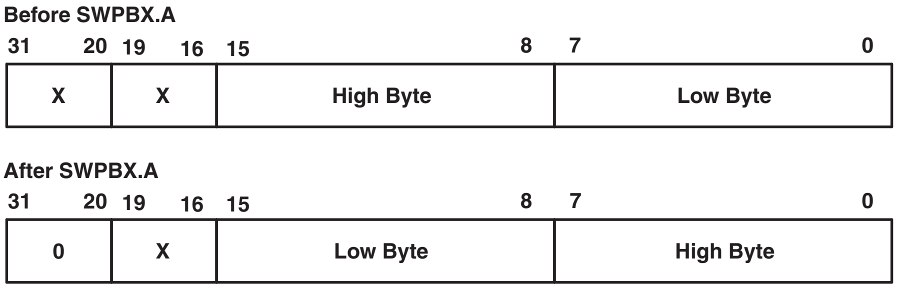


Figure 4-56. Swap Bytes SWPBX.A In Memory

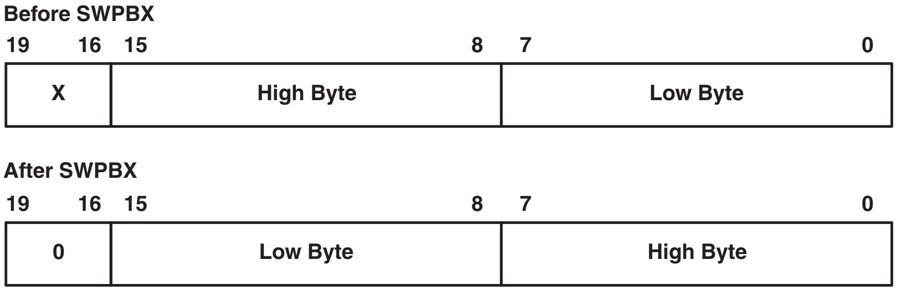


Figure 4-57. Swap Bytes SWPBX[.W] Register Mode

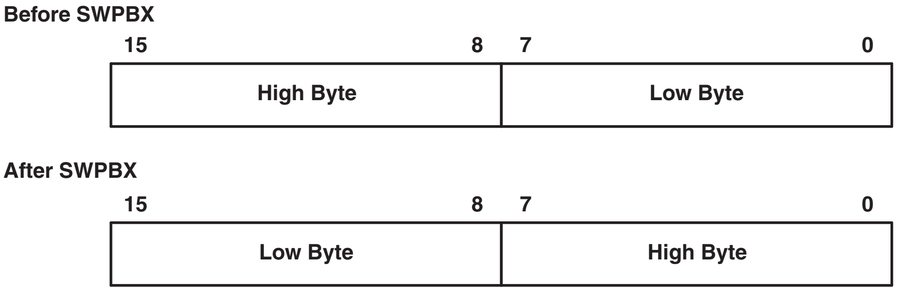


Figure 4-58. Swap Bytes SWPBX[.W] In Memory

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
```

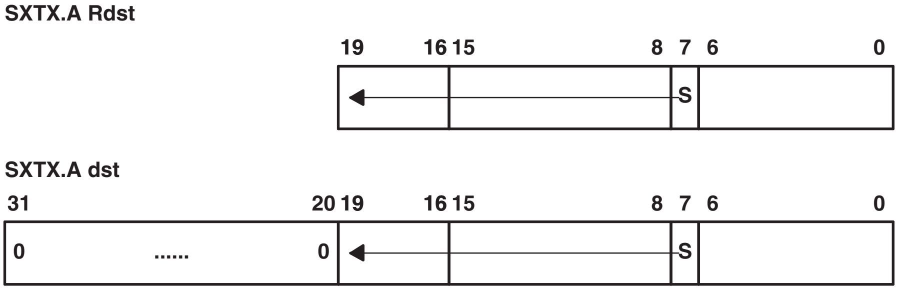


Figure 4-59. Sign Extend SXTX.A

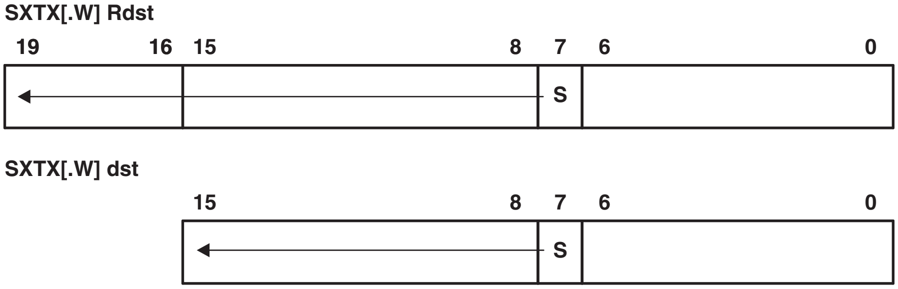


Figure 4-60. Sign Extend SXTX[.W]

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
```

Chapter 5

SLAU445D – October 2014 – Revised December 2015

FRAM Controller (FRCTL)

This chapter describes the operation of the FRAM controller.

Topic **Page**

5.1 FRAM Introduction 270
5.2 FRAM Organization 270
5.3 FRCTL Module Operation 270
5.4 Programming FRAM Devices 271
5.5 Wait State Control 271
5.6 FRAM ECC 272
5.7 FRAM Write Back 272
5.8 FRAM Power Control 272
5.9 FRAM Cache 273
5.10 FRCTL Registers 274

5.1 FRAM Introduction

FRAM is a nonvolatile memory that reads and writes like standard SRAM. The MSP430 FRAM 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 (ECC) 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 5-1 shows the block diagram of the FRAM Controller.

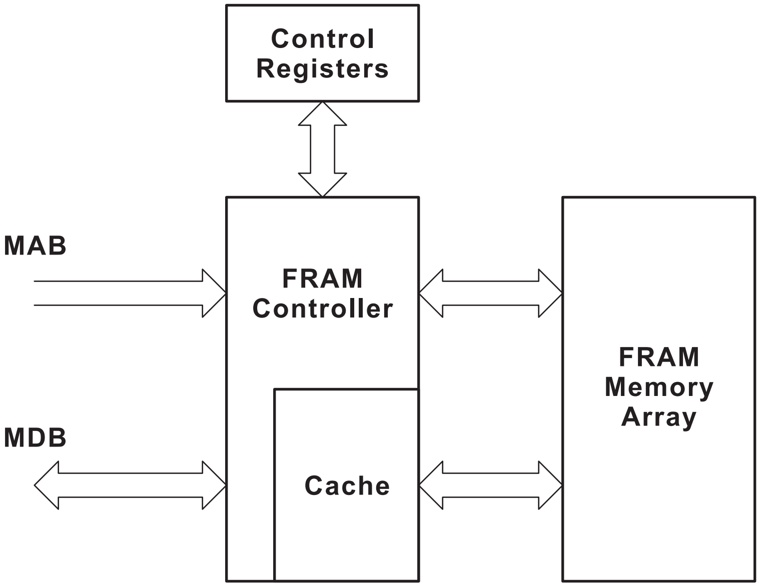


Figure 5-1. FRAM Controller Block Diagram

5.2 FRAM Organization

The FRAM 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.

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 (ECC) logic that can correct bit errors and detect multiple bit errors. Two flags are available that indicate the presence of an error.

The CBDIFG is set when a correctable bit error is 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 is 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 Devices

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

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

5.4.1 Programming FRAM Through JTAG or Spy-Bi-Wire

Devices can be programmed through 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 Through the Bootloader (BSL)

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

5.4.3 Programming FRAM Through a Custom Solution

The ability of the CPU to write to its own FRAM 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. 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.

5.5 Wait State Control

The system clock for the CPU 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 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 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 array. By setting FRPWR = 0, the FRAM 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. A second control bit FRLPMPWR is used to delay the power-up of the FRAM 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.

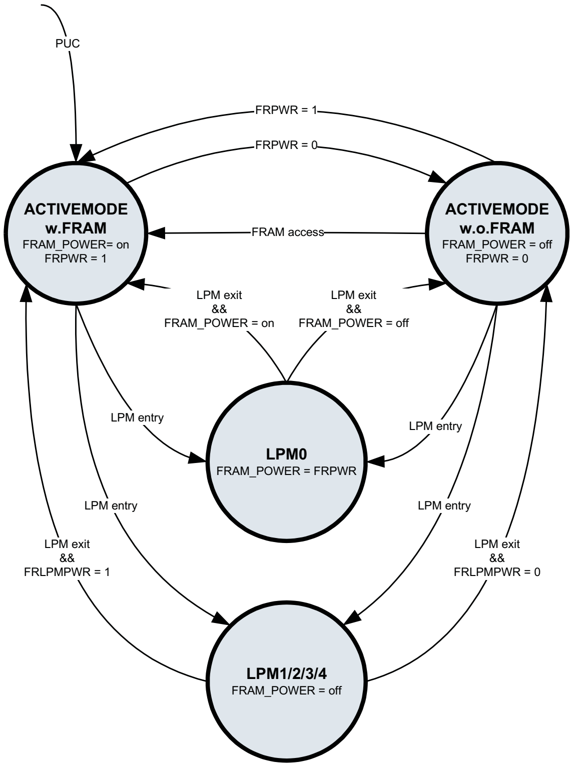


Figure 5-2. FRAM Power Control Diagram

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

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td colspan="8">FRCTLPW</td></tr>
<tr><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Reserved</td><td colspan="3">NWAITS</td><td colspan="4">Reserved</td></tr>
<tr><td>r-0</td><td>rw-[0]</td><td>rw-[0]</td><td>rw-[0]</td><td>r-0</td><td>r-0</td><td>r-0</td><td>r-0</td></tr>
</table>

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 reads as 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 reads as 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	rw-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 reads as 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 reads as 0.
3	Reserved	RW	0h	Reserved. Must be written as 0.
2	FRPWR	RW	1h	FRAM power control. Writing to the register enables or disables the FRAM power supply. The read 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 reads as 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	rw-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 reads as 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. **Note:** The ACCTEIFG bit may be set in debug mode when the system frequency is configured to be greater than 8 MHz, regardless of the wait states (NWAITS). In the case, it is not an FRAM access violation. The ACCTEIFG bit does not trigger a PUC or change the SYSRSTIV register value. The ACCTEIFG bit is cleared only by writing 0. It is recommended to use SYSRESTIV register to check FRAM access violation error to avoid confusion.
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 reads as 0.

Backup Memory (BKMEM)

The Backup Memory provides up to 256 bytes that are retained during LPM3.5. The size of the Backup Memory varies by device–see the device-specific data sheet for details. This chapter describes the Backup Memory functionality and features.

Topic **Page**

6.1 Backup Memory Introduction 279
6.2 BKMEM Registers 279

6.1 Backup Memory Introduction

Features of the Backup Memory include:

- Configurable from 32 bytes to 256 bytes
- Supports modes from AM to LPM3.5
- Supports word or byte access

6.2 BKMEM Registers

Table 6-1 lists the Backup Memory registers. The base address of the Backup Memory module can be found in the device-specific data sheet.

Table 6-1. BKMEM Registers

Offset	Acronym	Register Name	Type	Access	Reset
00h	BAKMEM0	Backup Memory 0	Read/write	Word	Undefined
00h	BAKMEM0_L		Read/write	Byte	
01h	BAKMEM0_H		Read/write	Byte	
02h	BAKMEM1	Backup Memory 1 [1]	Read/write	Word	Undefined
02h	BAKMEM1_L		Read/write	Byte	
03h	BAKMEM1_H		Read/write	Byte	

[1] Words 2 to 127, if available, follow the same format. See the device-specific data sheet for more details.

Chapter 7

SLAU445D–October 2014–Revised December 2015

Digital I/O

This chapter describes the operation of the digital I/O ports in all devices.

Topic **Page**

7.1 Digital I/O Introduction 281
7.2 Digital I/O Operation 282
7.3 I/O Configuration 285
7.4 Digital I/O Registers 288

7.1 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 fewer lines (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 fewer 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).

7.2 Digital I/O Operation

The digital I/Os are configured with user software. The setup and operation of the digital I/Os are discussed in the following sections.

7.2.1 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.

7.2.2 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

7.2.3 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

7.2.4 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 7-1 summarizes the use of PxDIR, PxREN, and PxOUT for proper I/O configuration.

Table 7-1. 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

7.2.5 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 functions. Table 7-3 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. A device in this family may have only PxSEL0 or both PxSEL0 and PxSEL1.

Table 7-2. I/O Function Selection for Devices With Only One Selection Bit – PxSEL0

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

Table 7-3. I/O Function Selection for Devices With Two Selection Bits – PxSEL0 and PxSEL1

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.

7.2.6 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 in addition to 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.

7.2.6.1 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
```

7.2.6.2 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

7.2.6.3 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

7.3 I/O Configuration

7.3.1 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 7.3.2) 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 wakeup from LPMx.5. After clearing LOCKLPM5, all interrupt flags should be cleared (note, this is different from the flow for wakeup from LPMx.5). 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 disabled. To prevent floating inputs, all port pins including unused ones (Section 7.3.2) should be configured according to the application needs as early as possible during the initialization procedure.

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

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

7.3.2 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.

7.3.3 Configuration for LPMx.5 Low-Power Modes

NOTE: See Section 1.4.3, *Low-Power Modes LPM3.5 and LPM4.5 (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/Os, 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 make sure 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 makes sure that each pin is in a safe condition before 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 before 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 wake up from a port interrupt if its respective port interrupt flag is already asserted. TI recommends clearing the flags before entering LPMx.5. TI also recommends setting GIE = 1 before entry into LPMx.5. This allows any pending flags to be serviced before LPMx.5 entry.

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

During LPMx.5 the I/O pin states are held and locked based on the settings before 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 before 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.

7.4 Digital I/O Registers

The digital I/O registers are listed in Table 7-4. 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 7-4.

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

Table 7-4. Digital I/O Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
0Eh	P1IV	Port 1 Interrupt Vector	Read only	Word	0000h	Section 7.4.1
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 7.4.2
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 7.4.3
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 7.4.4
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 7.4.5
02h	P1OUT or PAOUT_L	Port 1 Output	Read/write	Byte	undefined	Section 7.4.6
04h	P1DIR or PADIR_L	Port 1 Direction	Read/write	Byte	00h	Section 7.4.7
06h	P1REN or PAREN_L	Port 1 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Ah	P1SEL0 or PASEL0_L	Port 1 Select 0	Read/write	Byte	00h	Section 7.4.9
0Ch	P1SEL1 or PASEL1_L	Port 1 Select 1	Read/write	Byte	00h	Section 7.4.10
16h	P1SELC or PASELC_L	Port 1 Complement Selection	Read/write	Byte	00h	Section 7.4.11
18h	P1IES or PAIES_L	Port 1 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Ah	P1IE or PAIE_L	Port 1 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Ch	P1IFG or PAIFG_L	Port 1 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14

Table 7-4. 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 7.4.5
03h	P2OUT or PAOUT_H	Port 2 Output	Read/write	Byte	undefined	Section 7.4.6
05h	P2DIR or PADIR_H	Port 2 Direction	Read/write	Byte	00h	Section 7.4.7
07h	P2REN or PAREN_H	Port 2 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Bh	P2SEL0 or PASEL0_H	Port 2 Select 0	Read/write	Byte	00h	Section 7.4.9
0Dh	P2SEL1 or PASEL1_H	Port 2 Select 1	Read/write	Byte	00h	Section 7.4.10
17h	P2SELC or PASELC_L	Port 2 Complement Selection	Read/write	Byte	00h	Section 7.4.11
19h	P2IES or PAIES_H	Port 2 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Bh	P2IE or PAIE_H	Port 2 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Dh	P2IFG or PAIFG_H	Port 2 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14
00h	P3IN or PBIN_L	Port 3 Input	Read only	Byte	undefined	Section 7.4.5
02h	P3OUT or PBOUT_L	Port 3 Output	Read/write	Byte	undefined	Section 7.4.6
04h	P3DIR or PBDIR_L	Port 3 Direction	Read/write	Byte	00h	Section 7.4.7
06h	P3REN or PBREN_L	Port 3 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Ah	P3SEL0 or PBSEL0_L	Port 3 Select 0	Read/write	Byte	00h	Section 7.4.9
0Ch	P3SEL1 or PBSEL1_L	Port 3 Select 1	Read/write	Byte	00h	Section 7.4.10
16h	P3SELC or PBSELC_L	Port 3 Complement Selection	Read/write	Byte	00h	Section 7.4.11
18h	P3IES or PBIES_L	Port 3 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Ah	P3IE or PBIE_L	Port 3 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Ch	P3IFG or PBIFG_L	Port 3 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14

Table 7-4. 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 7.4.5
03h	P4OUT or PBOUT_H	Port 4 Output	Read/write	Byte	undefined	Section 7.4.6
05h	P4DIR or PBDIR_H	Port 4 Direction	Read/write	Byte	00h	Section 7.4.7
07h	P4REN or PBREN_H	Port 4 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Bh	P4SEL0 or PBSEL0_H	Port 4 Select 0	Read/write	Byte	00h	Section 7.4.9
0Dh	P4SEL1 or PBSEL1_H	Port 4 Select 1	Read/write	Byte	00h	Section 7.4.10
17h	P4SELC or PBSELC_L	Port 4 Complement Selection	Read/write	Byte	00h	Section 7.4.11
19h	P4IES or PBIES_H	Port 4 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Bh	P4IE or PBIE_H	Port 4 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Dh	P4IFG or PBIFG_H	Port 4 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14
00h	P5IN or PCIN_L	Port 5 Input	Read only	Byte	undefined	Section 7.4.5
02h	P5OUT or PCOUT_L	Port 5 Output	Read/write	Byte	undefined	Section 7.4.6
04h	P5DIR or PCDIR_L	Port 5 Direction	Read/write	Byte	00h	Section 7.4.7
06h	P5REN or PCREN_L	Port 5 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Ah	P5SEL0 or PCSEL0_L	Port 5 Select 0	Read/write	Byte	00h	Section 7.4.9
0Ch	P5SEL1 or PCSEL1_L	Port 5 Select 1	Read/write	Byte	00h	Section 7.4.10
16h	P5SELC or PCSELC_L	Port 5 Complement Selection	Read/write	Byte	00h	Section 7.4.11
18h	P5IES or PCIES_L	Port 5 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Ah	P5IE or PCIE_L	Port 5 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Ch	P5IFG or PCIFG_L	Port 5 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14

Table 7-4. 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 7.4.5
03h	P6OUT or PCOUT_H	Port 6 Output	Read/write	Byte	undefined	Section 7.4.6
05h	P6DIR or PCDIR_H	Port 6 Direction	Read/write	Byte	00h	Section 7.4.7
07h	P6REN or PCREN_H	Port 6 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Bh	P6SEL0 or PCSEL0_H	Port 6 Select 0	Read/write	Byte	00h	Section 7.4.9
0Dh	P6SEL1 or PCSEL1_H	Port 6 Select 1	Read/write	Byte	00h	Section 7.4.10
17h	P6SELC or PCSELC_L	Port 6 Complement Selection	Read/write	Byte	00h	Section 7.4.11
19h	P6IES or PCIES_H	Port 6 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Bh	P6IE or PCIE_H	Port 6 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Dh	P6IFG or PCIFG_H	Port 6 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14
00h	P7IN or PDIN_L	Port 7 Input	Read only	Byte	undefined	Section 7.4.5
02h	P7OUT or PDOUT_L	Port 7 Output	Read/write	Byte	undefined	Section 7.4.6
04h	P7DIR or PDDIR_L	Port 7 Direction	Read/write	Byte	00h	Section 7.4.7
06h	P7REN or PDREN_L	Port 7 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Ah	P7SEL0 or PDSEL0_L	Port 7 Select 0	Read/write	Byte	00h	Section 7.4.9
0Ch	P7SEL1 or PDSEL1_L	Port 7 Select 1	Read/write	Byte	00h	Section 7.4.10
16h	P7SELC or PDSELC_L	Port 7 Complement Selection	Read/write	Byte	00h	Section 7.4.11
18h	P7IES or PDIES_L	Port 7 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Ah	P7IE or PDIE_L	Port 7 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Ch	P7IFG or PDIFG_L	Port 7 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14

Table 7-4. 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 7.4.5
03h	P8OUT or PDOUT_H	Port 8 Output	Read/write	Byte	undefined	Section 7.4.6
05h	P8DIR or PDDIR_H	Port 8 Direction	Read/write	Byte	00h	Section 7.4.7
07h	P8REN or PDREN_H	Port 8 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Bh	P8SEL0 or PDSEL0_H	Port 8 Select 0	Read/write	Byte	00h	Section 7.4.9
0Dh	P8SEL1 or PDSEL1_H	Port 8 Select 1	Read/write	Byte	00h	Section 7.4.10
17h	P8SELC or PDSELC_L	Port 8 Complement Selection	Read/write	Byte	00h	Section 7.4.11
19h	P8IES or PDIES_H	Port 8 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Bh	P8IE or PDIE_H	Port 8 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Dh	P8IFG or PDIFG_H	Port 8 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14
00h	P9IN or PEIN_L	Port 9 Input	Read only	Byte	undefined	Section 7.4.5
02h	P9OUT or PEOUT_L	Port 9 Output	Read/write	Byte	undefined	Section 7.4.6
04h	P9DIR or PEDIR_L	Port 9 Direction	Read/write	Byte	00h	Section 7.4.7
06h	P9REN or PEREN_L	Port 9 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Ah	P9SEL0 or PESEL0_L	Port 9 Select 0	Read/write	Byte	00h	Section 7.4.9
0Ch	P9SEL1 or PESEL1_L	Port 9 Select 1	Read/write	Byte	00h	Section 7.4.10
16h	P9SELC or PESELC_L	Port 9 Complement Selection	Read/write	Byte	00h	Section 7.4.11
18h	P9IES or PEIES_L	Port 9 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Ah	P9IE or PEIE_L	Port 9 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Ch	P9IFG or PEIFG_L	Port 9 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14

Table 7-4. 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 7.4.5
03h	P10OUT or PEOUT_H	Port 10 Output	Read/write	Byte	undefined	Section 7.4.6
05h	P10DIR or PEDIR_H	Port 10 Direction	Read/write	Byte	00h	Section 7.4.7
07h	P10REN or PEREN_H	Port 10 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Bh	P10SEL0 or PESEL0_H	Port 10 Select 0	Read/write	Byte	00h	Section 7.4.9
0Dh	P10SEL1 or PESEL1_H	Port 10 Select 1	Read/write	Byte	00h	Section 7.4.10
17h	P10SELC or PESELC_L	Port 10 Complement Selection	Read/write	Byte	00h	Section 7.4.11
19h	P10IES or PEIES_H	Port 10 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Bh	P10IE or PEIE_H	Port 10 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Dh	P10IFG or PEIFG_H	Port 10 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14
00h	P11IN or PFIN_L	Port 11 Input	Read only	Byte	undefined	Section 7.4.5
02h	P11OUT or PFOUT_L	Port 11 Output	Read/write	Byte	undefined	Section 7.4.6
04h	P11DIR or PFDIR_L	Port 11 Direction	Read/write	Byte	00h	Section 7.4.7
06h	P11REN or PFREN_L	Port 11 Resistor Enable	Read/write	Byte	00h	Section 7.4.8
0Ah	P11SEL0 or PFSEL0_L	Port 11 Select 0	Read/write	Byte	00h	Section 7.4.9
0Ch	P11SEL1 or PFSEL1_L	Port 11 Select 1	Read/write	Byte	00h	Section 7.4.10
16h	P11SELC or PFSELC_L	Port 11 Complement Selection	Read/write	Byte	00h	Section 7.4.11
18h	P11IES or PFIES_L	Port 11 Interrupt Edge Select	Read/write	Byte	undefined	Section 7.4.12
1Ah	P11IE or PFIE_L	Port 11 Interrupt Enable	Read/write	Byte	00h	Section 7.4.13
1Ch	P11IFG or PFIFG_L	Port 11 Interrupt Flag	Read/write	Byte	00h	Section 7.4.14

Table 7-4. 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 7-4. 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 7-4. 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 7-4. 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 7-4. 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 7-4. 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 7-4. 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	

7.4.1 P1IV Register

Port 1 Interrupt Vector Register

Figure 7-1. 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 7-5. 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 10h = Interrupt Source: Port 1.7 interrupt; Interrupt Flag: P1IFG.7; Interrupt Priority: Lowest

7.4.2 P2IV Register

Port 2 Interrupt Vector Register

Figure 7-2. 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 7-6. 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 10h = Interrupt Source: Port 2.7 interrupt; Interrupt Flag: P2IFG.7; Interrupt Priority: Lowest

7.4.3 P3IV Register

Port 3 Interrupt Vector Register

Figure 7-3. P3IV Register

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td colspan="8">P3IV</td></tr>
<tr><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">P3IV</td></tr>
<tr><td>r0</td><td>r0</td><td>r0</td><td>r-0</td><td>r-0</td><td>r-0</td><td>r-0</td><td>r0</td></tr>
</table>

Table 7-7. 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 10h = Interrupt Source: Port 3.7 interrupt; Interrupt Flag: P3IFG.7; Interrupt Priority: Lowest

7.4.4 P4IV Register

Port 4 Interrupt Vector Register

Figure 7-4. P4IV Register

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td colspan="8">P4IV</td></tr>
<tr><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">P4IV</td></tr>
<tr><td>r0</td><td>r0</td><td>r0</td><td>r-0</td><td>r-0</td><td>r-0</td><td>r-0</td><td>r0</td></tr>
</table>

Table 7-8. 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 10h = Interrupt Source: Port 4.7 interrupt; Interrupt Flag: P4IFG.7; Interrupt Priority: Lowest

7.4.5 PxIN Register

Port x Input Register

Figure 7-5. PxIN Register

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

Table 7-9. PxIN Register Description

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

7.4.6 PxOUT Register

Port x Output Register

Figure 7-6. PxOUT Register

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

Table 7-10. 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

7.4.7 PxDIR Register

Port x Direction Register

Figure 7-7. 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 7-11. 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

7.4.8 PxREN Register

Port x Pullup or Pulldown Resistor Enable Register

Figure 7-8. 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 7-12. 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

7.4.9 PxSEL0 Register

Port x Function Selection Register 0

Figure 7-9. 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 7-13. 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.

7.4.10 PxSEL1 Register

Port x Function Selection Register 1

Figure 7-10. 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 7-14. 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

7.4.11 PxSELC Register

Port x Complement Selection

Figure 7-11. 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 7-15. 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.

7.4.12 PxIES Register

Port x Interrupt Edge Select Register

Figure 7-12. PxIES Register

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

Table 7-16. 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

7.4.13 PxIE Register

Port x Interrupt Enable Register

Figure 7-13. 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 7-17. 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

7.4.14 PxIFG Register

Port x Interrupt Flag Register

Figure 7-14. 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 7-18. 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 I/O

This chapter describes the functionality of the Capacitive Touch I/Os and related control.

Topic **Page**

8.1 Capacitive Touch I/O Introduction 308
8.2 Capacitive Touch I/O Operation 309
8.3 CapTouch Registers 310

8.1 Capacitive Touch I/O Introduction

The Capacitive Touch I/O 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 8-1 shows the capacitive touch I/O principle.

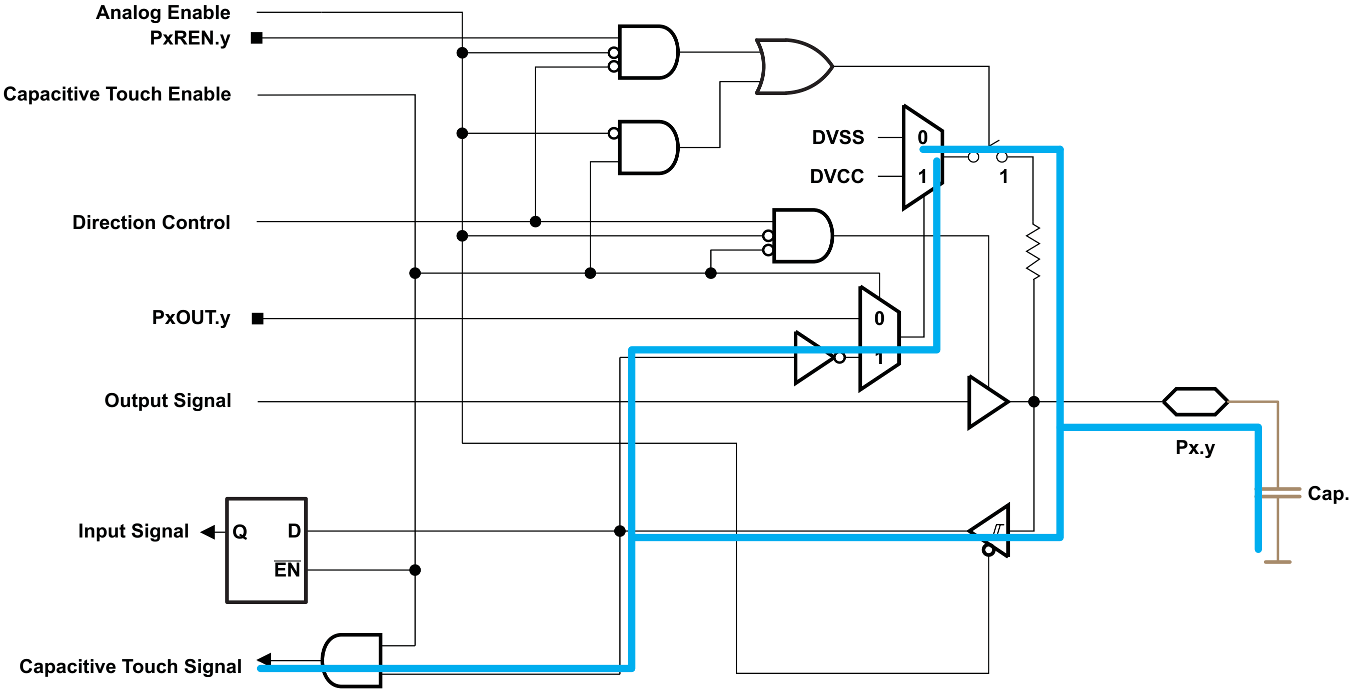


Figure 8-1. Capacitive Touch I/O Principle

Figure 8-2 shows the block diagram of the Capacitive Touch I/O module.

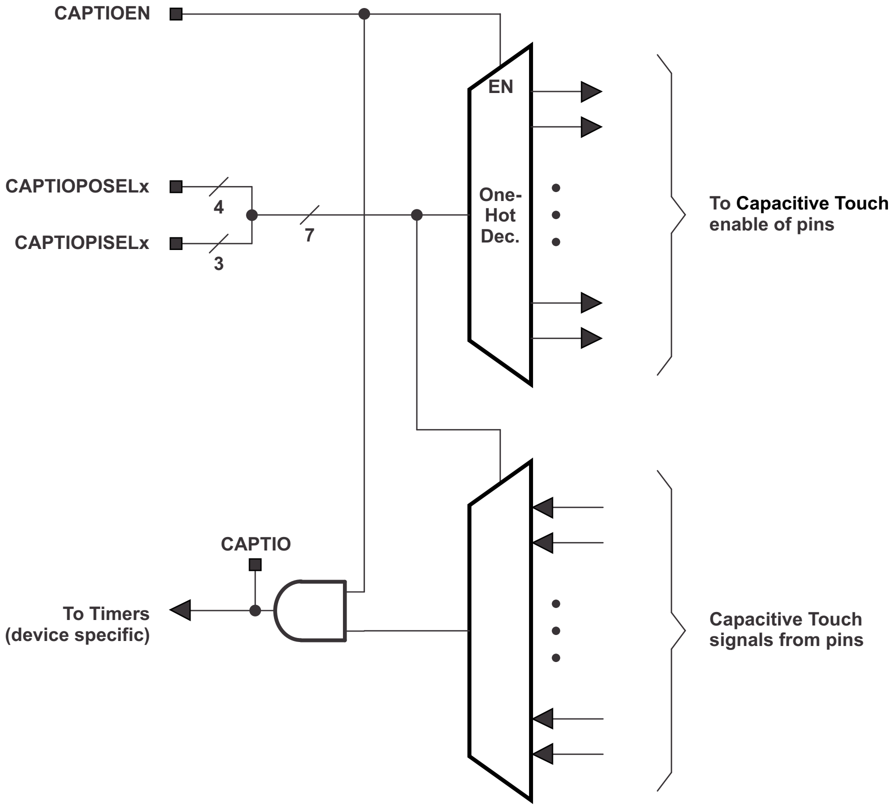


Figure 8-2. Capacitive Touch I/O Block Diagram

8.2 Capacitive Touch I/O Operation

Enable the Capacitive Touch I/O 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 I/O control register (CAPTIOCTL_L) by 2.

8.3 CapTouch Registers

The Capacitive Touch I/O registers and their address offsets are listed in Table 8-1. In a given device, multiple Capacitive Touch I/O registers might be available. The base address of each Capacitive Touch I/O 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 8-1. CapTouch Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
0Eh	CAPTIOxCTL	Capacitive Touch I/O x control register	Read/write	Word	0000h	Section 8.3.1
0Eh	CAPTIOxCTL_L		Read/write	Byte	00h	
0Fh	CAPTIOxCTL_H		Read/write	Byte	00h	

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

Capacitive Touch I/O x Control Register

Figure 8-3. 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 8-2. CAPTIOxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads 0.
9	CAPTIO	R	0h	Capacitive Touch I/O state. Reports the current state of the selected Capacitive Touch I/O. Reads 0 when Capacitive Touch I/O is disabled. 0b = Curent state 0 or Capacitive Touch I/O is disabled 1b = Current state 1
8	CAPTIOEN	RW	0h	Capacitive Touch I/O enable 0b = All Capacitive Touch I/Os are disabled. Signal toward timers is 0. 1b = Selected Capacitive Touch I/O is enabled
7-4	CAPTIOPOSELx	RW	0h	Capacitive Touch I/O 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 I/O 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.

Chapter 9

SLAU445D – October 2014 – Revised December 2015

CapTIvate™ Module

This chapter introduces the CapTIvate module. For additional documentation, examples, and other information, see the CapTIvate Technology Guide.

Topic **Page**

9.1 CapTIvate Introduction 313
9.2 CapTIvate Overview 313
9.3 CapTIvate Registers 318

9.1 CapTIvate Introduction

The CapTIvate module supports relative capacitance measurements for detecting capacitance changes.

Features of CapTIvate include:

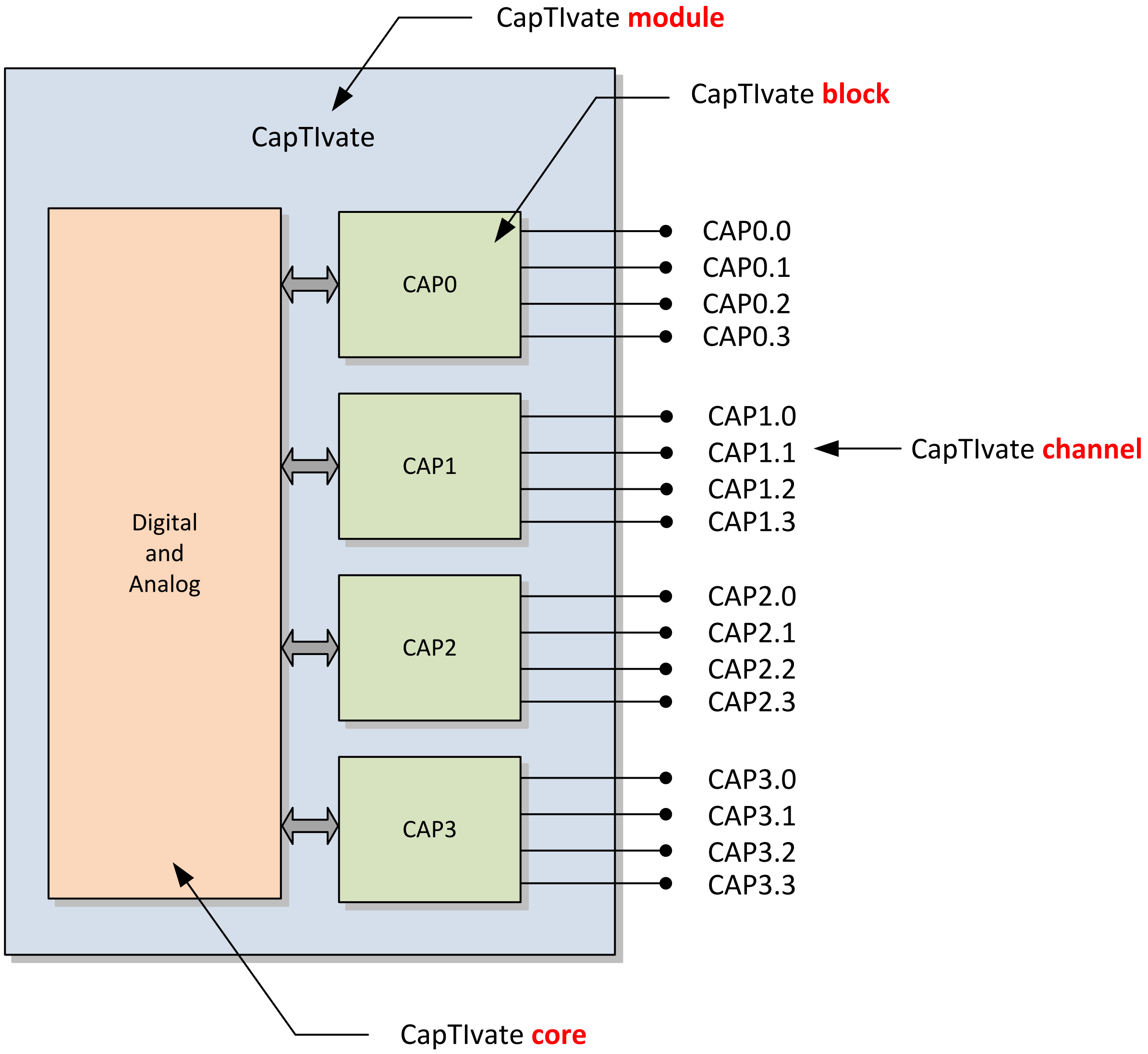
- Charge-transfer capacitance measurement technology
- Wake on touch: finite state machine to automate detection and environmental compensation without CPU interaction
- Each channel can be configured independently as either a receiver (Rx) or a transmitter (Tx) to support both mutual- and self-capacitance measurements
- Configuration allows simultaneous or sequential capacitance measurements
- Each CapTIvate module can support up to 12 measurement blocks and 8 I/O channels per block (see data sheet for device-specific configuration)
- Signal conditioning to provide signal gain
- Signal conditioning to provide offset compensation for parasitic capacitance
- Integrated calibration capacitors
- Frequency shift to provide immunity (EMI) to radiated and conducted emissions
- Frequency modulation to reduce emissions and provide compatibility (EMC) with other electronic devices
- Synchronized conversion to move (in time) the conversion away from noise related events (trigger conversion on zero-crossing event)
- Integrated timer for timer triggered conversions
- Integrated LDO for increased immunity to power supply noise
- Integrated oscillator
- Processing logic to perform measurement filtering, environmental compensation, and threshold detection

9.2 CapTIvate Overview

9.2.1 Declarations

The following declarations, or terms, are used to describe parts of the CapTIvate module:

module	Contains the **core** and several **blocks**
core	Contains common digital and analog functions like oscillator, timing generator, V_{REG} generator required to supply multiple blocks
block	Contains the corresponding analog and digital functions Nomenclature: CAPx (x = 0 to 11, this example shows only four blocks.)
channel	Physical connection to external electrode and charge-transfer switches The following nomenclature is used to refer to a specific channel: CAPx.y, where x is the block number and y is the channel number (FR2xxx: x = 0 to 3, y = 0 to 3) (maximum configuration: x = 0 to 11, y = 0 to 7)



NOTE: This figure shows the implementation in the MSP430FR26xx devices.

Figure 9-1. CapTIvate Declarations

9.2.2 Terms

The following is a list of abbreviations and application-specific terms.

CT	Charge transfer
CAPx.y	CapTIvate channel, x = 0 to 11, y = 0 to 7
CRx	CapTIvate channel configured as receive input, x = 0 to 15
CTx	CapTIvate channel configured as transmit output, x = 0 to 15
Vdd	Voltage supply to the device
Vreg	Regulated voltage supply
C_S capacitor	A capacitor used to accumulate charge from an unknown capacitor (C_X), usually the antenna or touch pad. C_S stems from 'sampling capacitor'. This capacitor is implemented on the chip.

C_X capacitor
In self-capacitance mode, this is a capacitor of unknown value referred to ground (which is the electrode or touch pad). The term is used to refer to the touch pad (when viewed as a capacitor), the track and touch pad, or the pin leading to the electrode or touch pad.

C_M capacitor
In mutual-capacitance mode, this is the capacitance between the transmit antenna (CTx) and receive antenna (CRx).

Channel
A channel is a 'CapTIvate Block' output. A single channel can be used to measure the self-capacitance of an electrode or two channels can be used to measure the mutual capacitance between two electrodes (receive electrode and transmit electrode).

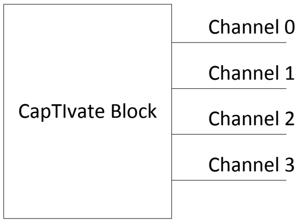


Figure 9-2. Channel Definition

Element
The description of a single touch instance. This instance can be realized with a single channel in self mode or with two channels in mutual mode.

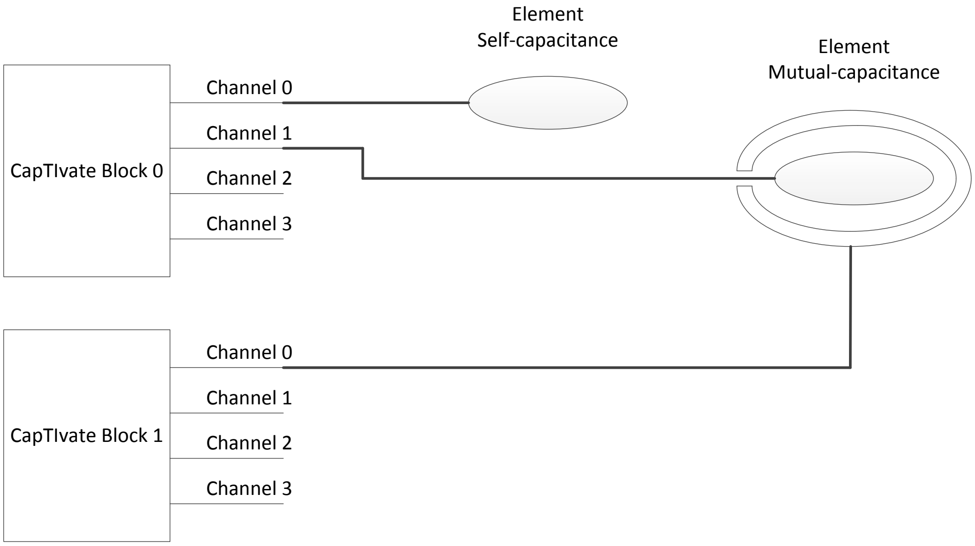


Figure 9-3. Element Definition

Time slot
A time slot represents a single time period during which a number of elements are converted in parallel. For example, if there are four CapTIvate Blocks then four elements can be measured in parallel during one time slot.

Wake on touch
No CPU interaction is required to periodically measure, filter, and analyze a touch element. If the analysis determines an event that requires further processing or decision making, then an interrupt is provided to wake up the CPU.

Conversion
The time required to make a capacitance measurement of all the elements within a time slot. The measurement of the elements is in parallel, so the dominant element (element with the longest measurement time) dictates the time of the conversion. (From a hardware perspective, the conversion is the number of charge transfers required to charge the C_S capacitor to the V_{trip} voltage.)

9.2.3 CapTIvate Configurations

The CapTIvate module can have different numbers of blocks and channels per block. Only one configuration is possible per device. Figure 9-4 shows the minimum, maximum, and one example implementation of the CapTIvate module.

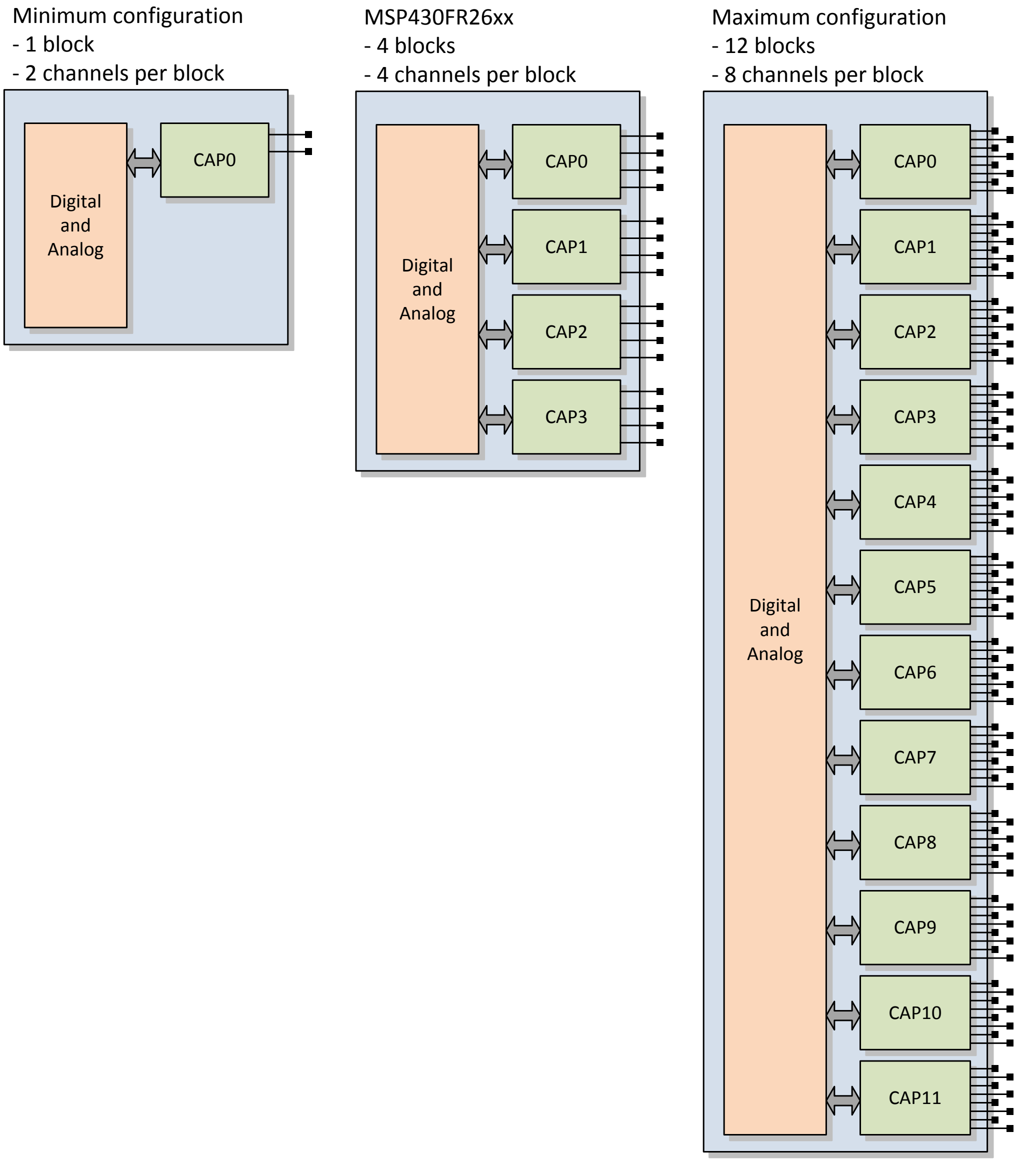


Figure 9-4. CapTIvate Configurations

9.2.4 Operation Modes

9.2.4.1 Self-Capacitance Mode

A capacitor (C_X) of unknown value (referenced to earth) is formed with a single electrode. As the user approaches (or touches) the single electrode, the capacitance of the electrode increases (see Figure 9-5). This change in capacitance is typically compared to the value of the capacitance on the electrode without a user nearby. Should a threshold set by firmware be exceeded, the condition is reported as a 'proximity' or 'contact' event, depending on the level of change detected. Self-capacitive techniques can be employed for buttons, sliders, wheels, proximity sensors, and single-touch panels.

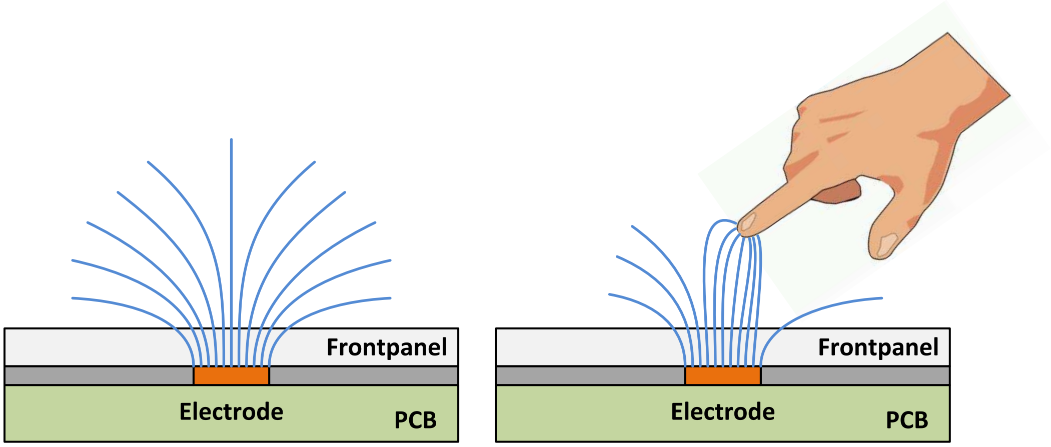


Figure 9-5. Self-Capacitance Mode

9.2.4.2 Mutual-Capacitance Mode

A capacitor (C_M) is formed between two electrodes and with the two electrodes connected to different pins of the controller IC. When an object approaches or touches the capacitor structure, the electric field between the two electrodes is modified and the capacitance is reduced (see Figure 9-6). This change is measured and compared to a reference value. If the variance exceeds the threshold set by firmware, a contact or proximity event is reported.

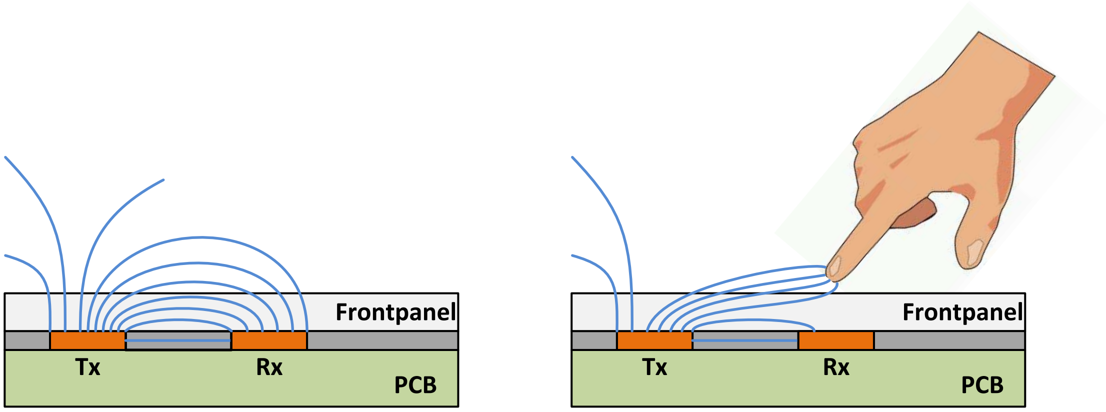


Figure 9-6. Mutual-Capacitance Mode

9.3 CapTIvate Registers

Table 9-1 lists the registers for the CapTIvate module. For more information, see the CapTIvate™ Technology Guide.

Table 9-1. CapTIvate Registers

Offset	Acronym	Register Name	Section
0x120	CAPIE	CapTIvate Interrupt Enable Register	Section 9.3.1
0x122	CAPIFG	CapTIvate Interrupt Flag Register	Section 9.3.2
0x124	CAPIV	CapTIvate Interrupt Vector Register	Section 9.3.3

9.3.1 CAPIE Register

CapTIvate Interrupt Enable Register

Figure 9-7. CAPIE Register

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td colspan="7">Reserved</td><td>CAPMAXIEN</td></tr>
<tr><td>r(0)</td><td>r(0)</td><td>r(0)</td><td>r(0)</td><td>r(0)</td><td>r(0)</td><td>r(0)</td><td>rw-(0)</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="4">Reserved</td><td>CAPCNTRIEN</td><td>CAPTIEN</td><td>CAPDTCTIEN</td><td>EOCIEN</td></tr>
<tr><td>r(0)</td><td>r(0)</td><td>r(0)</td><td>r(0)</td><td>rw-(0)</td><td>rw-(0)</td><td>rw-(0)</td><td>rw-(0)</td></tr>
</table>

Table 9-2. CAPIE Register Description

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0b	Reserved. Always reads as 0.
8	CAPMAXIEN	RW	0b	CapTIvate maximum count interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
7-4	Reserved	R	0b	Reserved. Always reads as 0.
3	CAPCNTRIEN	RW	0b	CapTIvate conversion counter interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
2	CAPTIEN	RW	0b	CapTIvate timer interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
1	CAPDTCTIEN	RW	0b	CapTIvate detection interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
0	EOCIEN	RW	0b	End of conversion interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled When enabled, an interrupt is called when EOCIFG = 1; that is, at the end of each conversion. EOCIFG must be cleared during the interrupt service routine.

9.3.2 CAPIFG Register

CapTIvate Interrupt Flag Register

Figure 9-8. CAPIFG Register

15	14	13	12	11	10	9	8
Reserved							CAPMAXIFG
r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	rw-(0)
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved				CAPCNTRIFG	CAPTIFG	CAPDTCTIFG	EOCIFG
r(0)	r(0)	r(0)	r(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 9-3. CAPIFG Register Description

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0b	Reserved. Always reads as 0.
8	CAPMAXIFG	RW	0b	CapTIvate maximum count interrupt flag 0b = Maximum count not reached 1b = Maximum count reached
7-4	Reserved	R	0b	Reserved. Always reads as 0.
3	CAPCNTRIFG	RW	0b	specified number of conversion have been reached 0b = No interrupt pending 1b = Interrupt pending
2	CAPTIFG	RW	0b	CapTIvate timer interrupt flag 0b = No interrupt pending 1b = Interrupt pending
1	CAPDTCTIFG	RW	0b	CapTIvate detection interrupt flag 0b = No interrupt pending 1b = Interrupt pending
0	EOCIFG	RW	0b	End of conversion interrupt flag 0b = No end of conversion has occurred 1b = End of conversion has occurred This bit is set by hardware when each of the enabled CRx channels has finished converting and its results are ready. This bit is cleared by hardware when a conversion is launched (when CIPF becomes 1) or when CAPPWR = 0. If EOCITEN = 1, the CapTIvate interrupt occurs when EOCIFG transitions to 1. EOCIFG must be cleared by software before exiting the interrupt service routine.

9.3.3 CAPIV Register

CapTIvate Interrupt Vector Register

Figure 9-9. CAPIV Register

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

Table 9-4. CAPIV Register Description

Bit	Field	Type	Reset	Description
15-0	CAPIVx	R	0	CapTIvate interrupt vector value. Generates a value that can be used as address offset for fast interrupt service routine handling. 0000h = No interrupt pending 0002h = Interrupt source: End of conversion interrupt, Flag = EOCIFG 0004h = Interrupt source: Detection interrupt, Flag = CAPDTCTIFG 0006h = Interrupt source: CapTIvate timer interrupt, Flag = CAPTIFG 0008h = Interrupt source: CapTIvate counter interrupt, Flag = CAPCNTRIFG 000Ah = Interrupt source: max count value reached, Flag = CAPMAXIFG 000Ch to FFFEh = Reserved Read will clear highest priority interrupt. Write will clear all pending interrupts.

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**

10.1 Cyclic Redundancy Check (CRC) Module Introduction 323
10.2 CRC Standard and Bit Order 323
10.3 CRC Checksum Generation 324
10.4 CRC Registers 327

10.1 Cyclic Redundancy Check (CRC) Module Introduction

The CRC module produces a signature for a given sequence of data values. The signature is generated through a feedback path from data bits 0, 4, 11, and 15 (see Figure 10-1). The CRC signature is based on the polynomial given in the CRC-CCITT-BR polynomial (see Equation 11) .

$$f(x) = x^{16} + x^{12} + x^{5} +1 \tag{11}$$

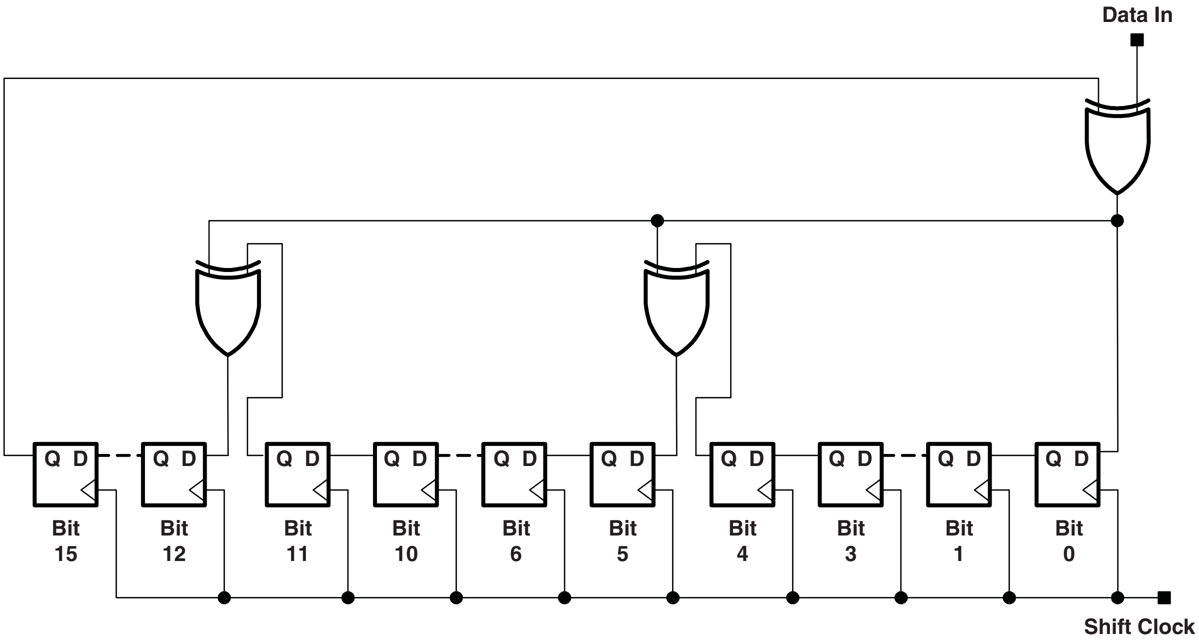


Figure 10-1. LFSR Implementation of CRC-CCITT Standard, Bit 0 is the MSB of the Result

Identical input data sequences result in identical signatures when the CRC is initialized with a fixed seed value, whereas different sequences of input data, in general, result in different signatures.

10.2 CRC 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 Figure 10-1, the bit convention shown is as given in the original standards (that is, 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.

10.3 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.

10.3.1 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 must be started by writing a seed to the CRCINIRES register to initialize the register. Software or hardware 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.

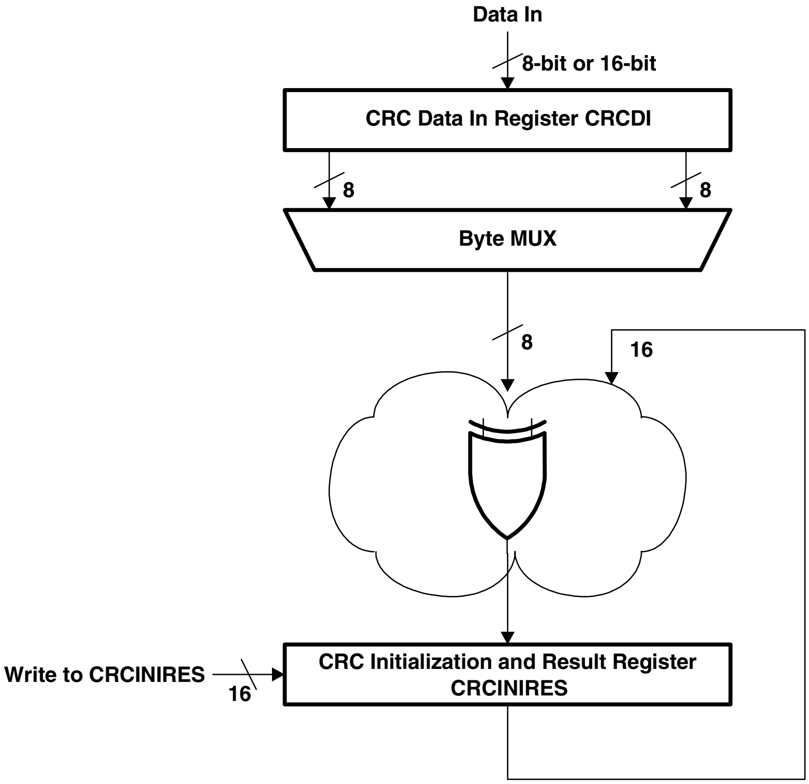

Data In
8-bit or 16-bit
CRC Data In Register CRCDI
8
8
Byte MUX
8
16
Write to CRCINIRES
16
CRC Initialization and Result Register CRCINIRES

Figure 10-2. Implementation of CRC-CCITT Using the CRCDI and CRCINIRES Registers

10.3.2 Assembler Examples

Example 10-1 demonstrates the operation of the on-chip CRC.

Example 10-1. 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 10-2 using word or byte accesses and the CRC data-in as well as the CRC data-in reverse byte registers.

Example 10-2. 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
```

10.4 CRC Registers

The CRC module registers are listed in Table 10-1. The base address can be found in the device-specific data sheet. The address offset is given in Table 10-1.

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

Table 10-1. CRC Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	CRCDI	CRC Data In	Read/write	Word	0000h	Section 10.4.1
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 10.4.2
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 10.4.3
04h	CRCINIRES_L		Read/write	Byte	FFh	
05h	CRCINIRES_H		Read/write	Byte	FFh	
06h	CRCRESR	CRC Result Reverse	Read only	Word	FFFFh	Section 10.4.4
06h	CRCRESR_L		Read/write	Byte	FFh	
07h	CRCRESR_H		Read/write	Byte	FFh	

10.4.1 CRCDI Register

CRC Data In Register

Figure 10-3. 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 10-2. 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.

10.4.2 CRCDIRB Register

CRC Data In Reverse Register

Figure 10-4. 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 10-3. 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.

10.4.3 CRCINIRES Register

CRC Initialization and Result Register

Figure 10-5. 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 10-4. 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.

10.4.4 CRCRESR Register

CRC Reverse Result Register

Figure 10-6. 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 10-5. 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).

Chapter 11

SLAU445D–October 2014–Revised December 2015

Watchdog Timer (WDT_A)

The watchdog timer is a 32-bit timer that can be used as a watchdog or as an interval timer. This chapter describes the watchdog timer. The enhanced watchdog timer, WDT_A, is implemented in all devices.

Topic **Page**

11.1 WDT_A Introduction **331**
11.2 WDT_A Operation **333**
11.3 WDT_A Registers **335**

11.1 WDT_A Introduction

The primary function of the watchdog timer (WDT_A) module is to perform a controlled system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can be configured as an interval timer and can generate interrupts at selected time intervals.

Features of the watchdog timer module include:

- Eight software-selectable time intervals
- Watchdog mode
- Interval mode
- Password-protected access to Watchdog Timer Control (WDTCTL) register
- Selectable clock source
- Can be stopped to conserve power
- Clock fail-safe feature

Figure 11-1 shows the watchdog timer block diagram.

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.

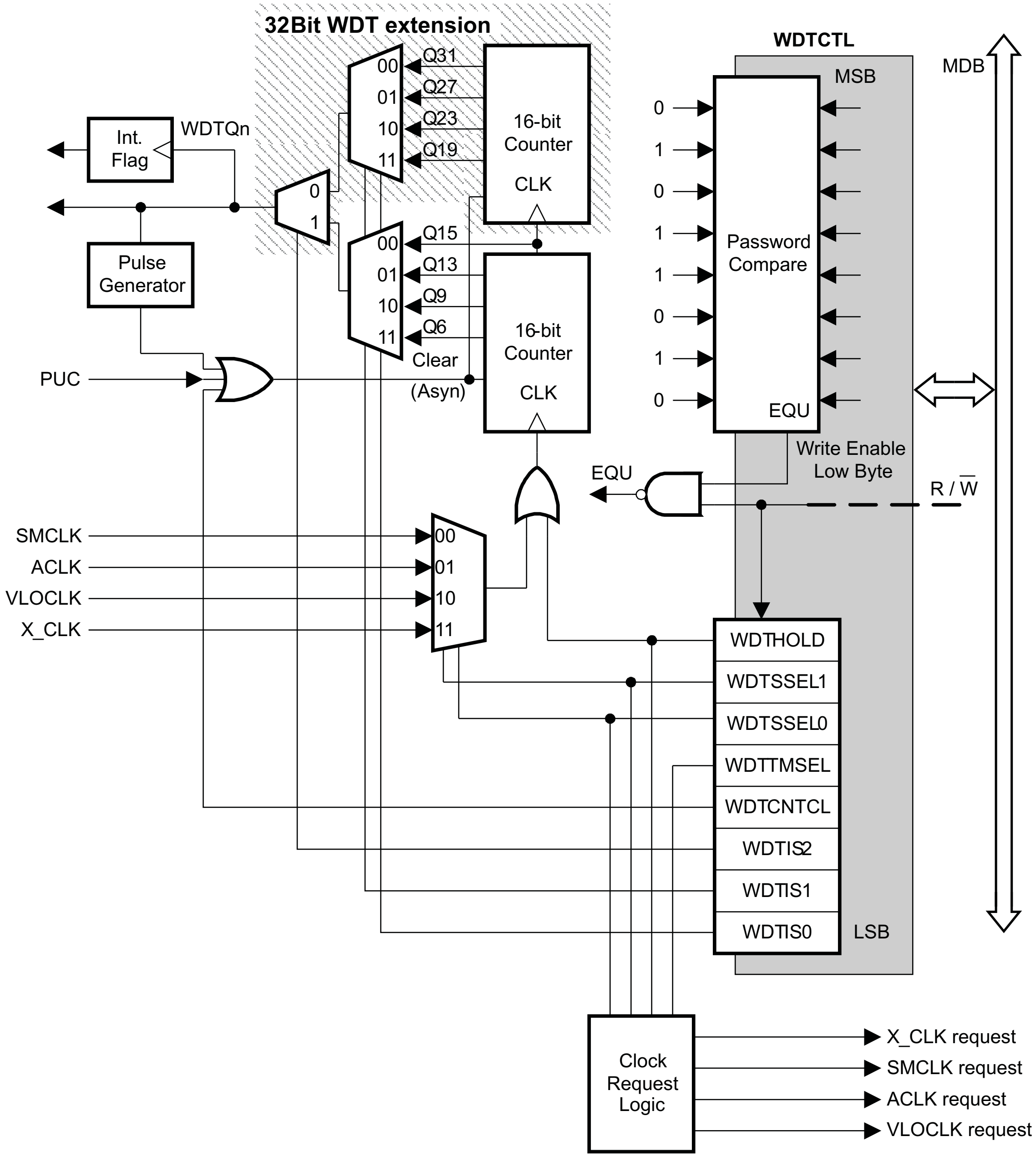


Figure 11-1. Watchdog Timer Block Diagram

11.2 WDT_A Operation

The watchdog timer module can be configured as either a watchdog or interval timer with the WDTCTL register. WDTCTL is a 16-bit password-protected read/write register. Any read or write access must use word instructions, and write accesses must include the write password 05Ah in the upper byte. 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. A byte read of the WDTCTL high or low byte returns the value of the low byte. Writing a byte wide to upper or lower byte of WDTCTL results in a PUC.

11.2.1 Watchdog Timer Counter (WDTCNT)

The WDTCNT is a 32-bit up counter that is not directly accessible by software. The WDTCNT is controlled and its time intervals are selected through the Watchdog Timer Control (WDTCTL) register. The WDTCNT can be sourced from SMCLK, ACLK, VLOCLK, or X_CLK on some devices. The clock source is selected with the WDTSSEL bits. The timer interval is selected with the WDTIS bits.

11.2.2 Watchdog Mode

After a PUC condition, the WDT module is configured in the watchdog mode with an initial 32-ms (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.

11.2.3 Interval Timer Mode

Setting the WDTTMSEL bit to 1 selects the interval timer mode. This mode can be used to provide periodic interrupts. In interval timer mode, the WDTIFG flag is set at the expiration of the selected time interval. A PUC is not generated in interval timer mode at expiration of the selected timer interval, and the WDTIFG enable bit WDTIE remains unchanged.

When the WDTIE bit and the GIE bit are set, the WDTIFG flag requests an interrupt. The WDTIFG interrupt flag is automatically reset when its interrupt request is serviced, or may be reset by software. The interrupt vector address in interval timer mode is different from that in watchdog mode.

NOTE: Modifying the watchdog timer

The watchdog timer interval should be changed together with WDTCNTCL = 1 in a single instruction to avoid an unexpected immediate PUC or interrupt. The watchdog timer should be halted before changing the clock source to avoid a possible incorrect interval.

11.2.4 Watchdog Timer Interrupts

The watchdog timer uses two bits in the SFRs for interrupt control:

- WDT interrupt flag, WDTIFG, located in SFRIFG1.0
- WDT interrupt enable, WDTIE, located in SFRIE1.0

When using the watchdog timer in the watchdog mode, the WDTIFG flag sources a reset vector interrupt. The WDTIFG can be used by the reset interrupt service routine to determine if the watchdog caused the device to reset. If the flag is set, the watchdog timer initiated the reset condition, either by timing out or by a password violation. If WDTIFG is cleared, the reset was caused by a different source.

When using the watchdog timer in interval timer mode, the WDTIFG flag is set after the selected time interval and requests a watchdog timer interval timer interrupt if the WDTIE and the GIE bits are set. The interval timer interrupt vector is different from the reset vector used in watchdog mode. In interval timer mode, the WDTIFG flag is reset automatically when the interrupt is serviced, or can be reset with software.

11.2.5 Clock Fail-Safe Feature

The WDT_A provides a fail-safe clocking feature, ensuring the clock to the WDT_A cannot be disabled while in watchdog mode. This means the low-power modes may be affected by the choice for the WDT_A clock.

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

When the WDT_A module is used in interval timer mode, there is no fail-safe feature within WDT_A for the clock source.

11.2.6 Operation in Low-Power Modes

MSP 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 using 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 11-1).

Example 11-1. 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
```

11.3 WDT_A Registers

The watchdog timer module registers are listed in Table 11-1. 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 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. WDT_A Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	WDTCTL	Watchdog Timer Control	Read/write	Word	6904h	Section 11.3.1
00h	WDTCTL_L		Read/write	Byte	04h	
01h	WDTCTL_H		Read/write	Byte	69h	

11.3.1 WDTCTL Register

Watchdog Timer Control Register

Figure 11-2. 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 11-2. WDTCTL Register Description

Bit	Field	Type	Reset	Description
15-8	WDTPW	RW	69h	Watchdog timer password. Always reads 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
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	4h	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

12.1 Timer_A Introduction 338
12.2 Timer_A Operation 340
12.3 Timer_A Registers 353

12.1 Timer_A Introduction

Timer_A is a 16-bit timer/counter with up to seven capture/compare registers. Timer_A can support multiple captures or 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

Figure 12-1 shows the block diagram of Timer_A.

NOTE: Use of the word *count*

Count is used throughout this chapter. It means the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, an associated action does not take place.

NOTE: Nomenclature

There may be multiple instantiations of Timer_A on a given device. The prefix TAx is used, where x is a greater than equal to zero indicating the Timer_A instantiation. For devices with one instantiation, x = 0. The suffix n, where n = 0 to 6, represents the specific capture/compare registers associated with the Timer_A instantiation.

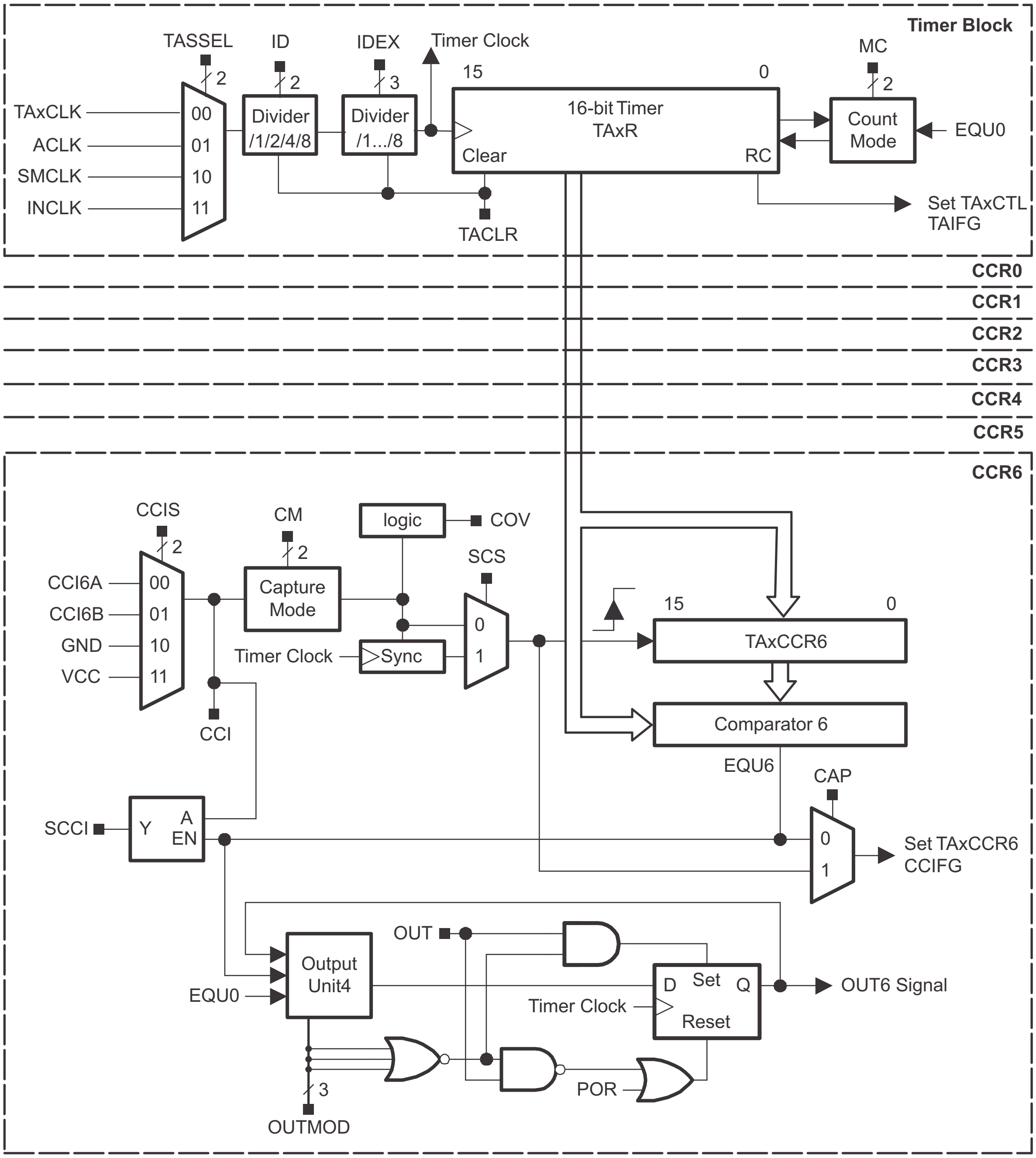


Figure 12-1. Timer_A Block Diagram

12.2 Timer_A Operation

The Timer_A module is configured with user software. The setup and operation of Timer_A are discussed in the following sections.

12.2.1 16-Bit Timer Counter

The 16-bit timer/counter register, TAxR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TAxR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.

NOTE: Accessing TAxR

Care must be taken when accessing TAxR. If TAxR is accessed (read or write) by the CPU while the timer is running, the value read from TAxR or the value written to TAxR could be unpredictable. To avoid this uncertainty, the timer should be stopped by writing the MC bits to zero before accessing TAxR. For read, alternatively TAxR can be read multiple times while the timer is running, and a majority vote taken in software to determine the correct reading.

12.2.1.1 Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally from 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

The timer clock dividers are reset by the TACLR bit. The clock divider is implemented as a down counter. To reset the down counter's state, write one to the TACLR bit in Stop mode. When the timer starts counting, the timer clock begins clocking at the first rising edge of the Timer_A clock source selected with the TASSEL bits and continues clocking at the divider setting set by the ID and TAIDEX bits.

The clock divider (ID bits and TAIDEX bits) should not be changed while the timer is running. It could cause unexpected behaviors. Stop the timer first (MC = 0) when changing the ID bits or TAIDEX bits.

12.2.2 Starting the Timer

When the device is out of reset (BOR or POR), the timer is at stop condition and all registers have default values. To start the timer from the default condition, perform the following steps:

1. Write 1 to the TACLR bit (TACLR = 1) to clear TAxR, clock divider state, and the counter direction.
2. If necessary, write initial counter value to TAxR.
3. Initialize TAxCCRn.
4. Apply desired configuration to TAxIV, TAIDEX and TAxCCTLn.
5. Apply desired configuration to TAxCTL including to MC bits.

12.2.3 Timer Mode Control

The timer has four modes of operation: stop, up, continuous, and up/down (see Table 12-1). The operating mode is selected with the MC bits.

Table 12-1. Timer Modes

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.

To move from one mode to another, first stop the timer by writing zero to the MC bits (MC = 0), then set the MC bits to the desired mode (see Table 12-1 for details).

12.2.3.1 Up Mode

The Up mode is used if the timer period must be different from 0FFFFh counts. The timer repeatedly counts up to the value of compare register TAxCCR0, which defines the period (see Figure 12-2). The number of timer counts in the period is TAxCCR0 + 1. When the timer value equals TAxCCR0, the timer restarts counting from zero. If Up mode is selected when the timer value is greater than TAxCCR0, the timer immediately restarts counting from zero.

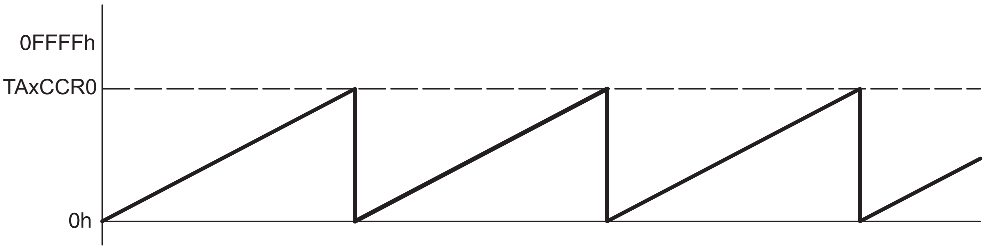


Figure 12-2. Up Mode

The TAxCCR0 CCIFG interrupt flag is set when the timer *counts* to the TAxCCR0 value. The TAIFG interrupt flag is set when the timer *counts* from TAxCCR0 to zero. Figure 12-3 shows the flag set cycle.

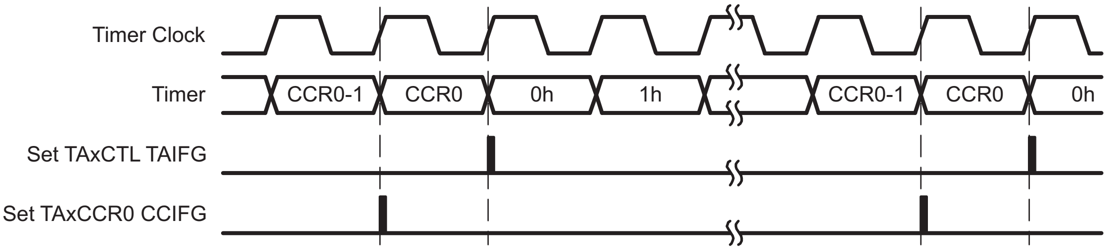


Figure 12-3. Up Mode Flag Setting

12.2.3.1.1 Changing Period Register TAxCCR0

When the MC bits are configured to Up mode (MC = 1) from Stop mode (MC = 0), the timer starts counting up from the value in TAxR if the TAxCCR0 is greater than TAxR. If TAxCCR0 is less than TAxR or equal to TAxR, the timer rolls back to zero and then counts up to TAxCCR0. One additional count may occur before the counter rolls to zero.

Changing TAxCCR0 while the timer is running may result in unexpected behaviors. To avoid the uncertainty, TAxCCR0 should be updated in Stop mode (MC = 0).

12.2.3.2 Continuous Mode

In the Continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in Figure 12-4. The capture/compare register TAxCCR0 works the same way as the other capture/compare registers.

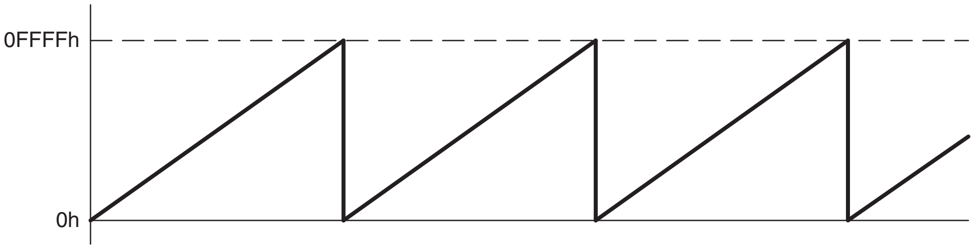


Figure 12-4. Continuous Mode

The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. Figure 12-5 shows the flag set cycle.

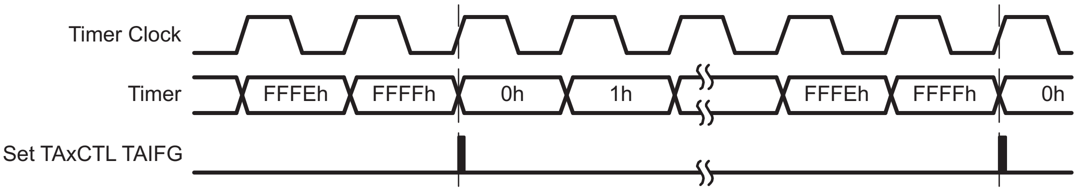


Figure 12-5. Continuous Mode Flag Setting

12.2.3.3 Use of Continuous Mode

The Continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TAxCCRn register in the interrupt service routine. Figure 12-6 shows two separate time intervals, t_0 and t_1, being added to the capture/compare registers. In this usage, the time interval is controlled by hardware, not software, without impact from interrupt latency. Up to n (where n = 0 to 6), independent time intervals or output frequencies can be generated using capture/compare registers.

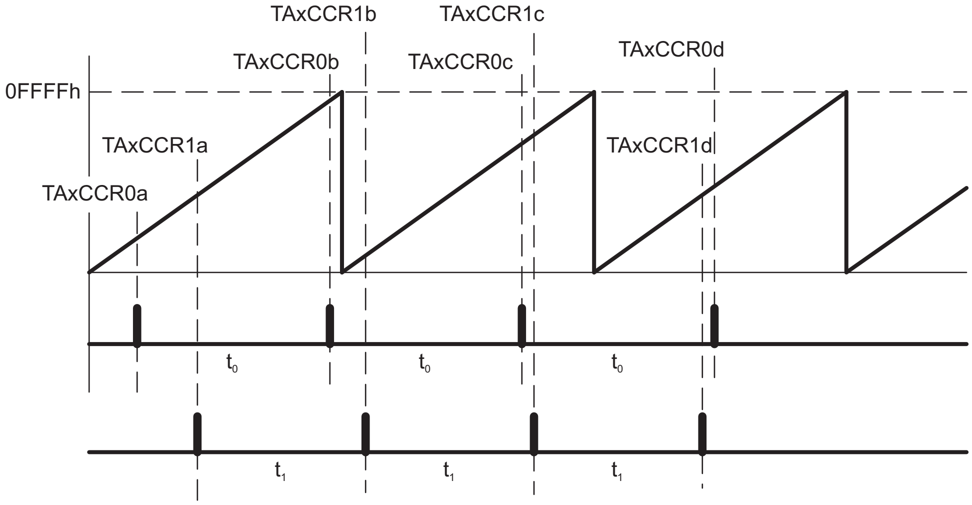


Figure 12-6. Continuous Mode Time Intervals

Time intervals can be produced with other modes as well, where TAxCCR0 is used as the period register. Their handling is more complex because the sum of the old TAxCCRn data and the new period can be higher than the TAxCCR0 value. When the previous TAxCCRn value plus t_x is greater than the TAxCCR0 data, the TAxCCR0 value must be subtracted to obtain the correct time interval.

12.2.3.4 Up/Down Mode

The Up/Down mode is used if the timer period must be different from 0FFFFh counts, and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare register TAxCCR0 and back down to zero (see Figure 12-7). The period is twice the value in TAxCCR0.

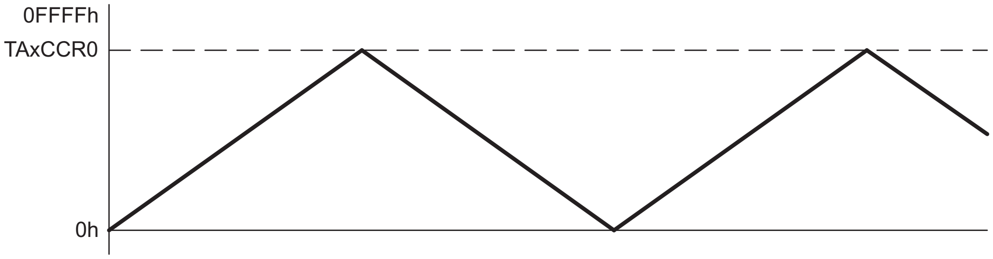


Figure 12-7. Up/Down Mode

The count direction is latched. This allows the timer to be stopped and then restarted in the same direction it was counting before it was stopped. If this is not desired, the TACLR bit must be set in Stop mode to clear the direction. The TACLR bit also clears the TAxR value and the timer clock divider (the divider setting remains unchanged).

In Up/Down mode, the TAxCCR0 CCIFG interrupt flag and the TAIFG interrupt flag are set only once during a period, separated by one-half the timer period. The TAxCCR0 CCIFG interrupt flag is set when the timer *counts* from TAxCCR0 – 1 to TAxCCR0, and TAIFG is set when the timer completes *counting* down from 0001h to 0000h. Figure 12-8 shows the flag set cycle.

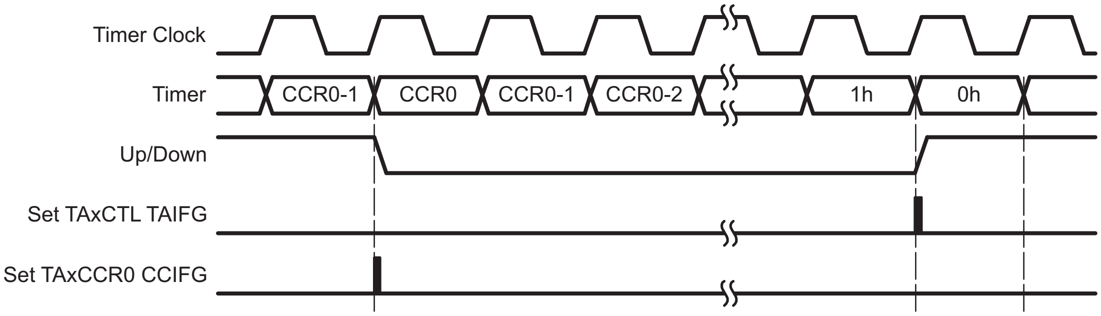


Figure 12-8. Up/Down Mode Flag Setting

12.2.3.4.1 Changing Period Register TAxCCR0

When the MC bits is configured to Up/Down mode (MC = 3) from Stop mode, the timer starts counting up or down depending on the previous direction. The timer keeps the previous direction regardless of the previous mode. The direction can be forced to up direction by setting to TACLR bit in Stop mode, but the direction cannot be forced to down direction when the timer starts with up direction, if TAxCCR0 is greater than TAxR, the timer will count up to TAxCC0. If TAxCCR0 is less than TAxR, or equal to TAxR, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

In Up/Down mode, updating TAxCCR0 while the timer is running may result in unexpected behaviors. To avoid the uncertainly, TAxCCR0 should be updated in Stop mode (MC = 0).

12.2.3.5 Use of Up/Down Mode

The Up/Down mode supports applications that require dead times between output signals (see Section 12.2.5). 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 12-9, 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

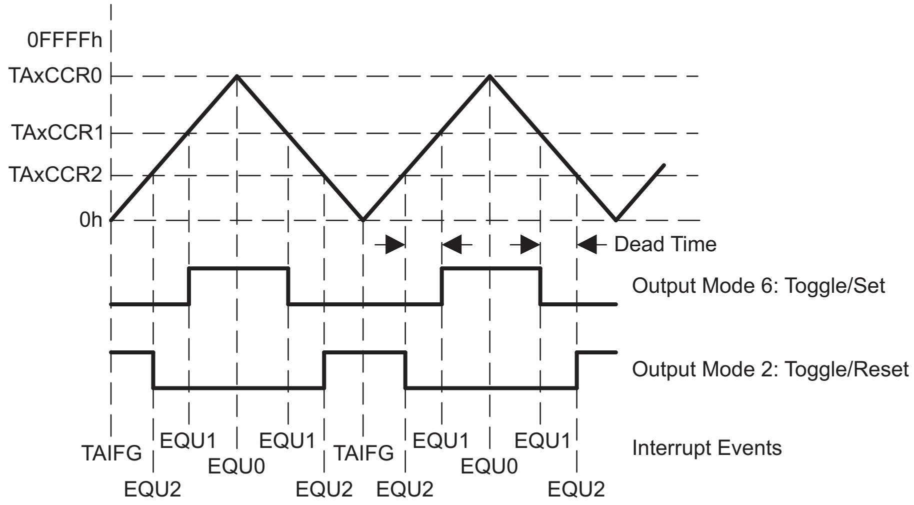


Figure 12-9. Output Unit in Up/Down Mode

12.2.4 Capture/Compare Blocks

Up to seven identical capture/compare blocks, TAxCCRn (where n = 0 to 7), are present in Timer_A. Any of the blocks may be used to capture the timer data or to generate time intervals.

12.2.4.1 Capture Mode

The capture mode is selected when CAP = 1. Capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CCIxA and CCIxB are connected to external pins or internal signals and are selected with the CCIS bits. The CM bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture occurs:

- The timer value is copied into the TAxCCRn register.
- The interrupt flag CCIFG is set.

The input signal level can be read at any time through 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.

NOTE: Reading TAxCCRn in Capture mode

In Capture mode, if TAxCCRn is ready by the CPU while the timer counter value is being copied into TAxCCRn at a capture event, the value ready by the CPU could be invalid. To avoid this undesired result, TAxCCRn must be read after the CCIFG flag is set and before the next capture event occurs.

The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit synchronizes the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended (see Figure 12-10).

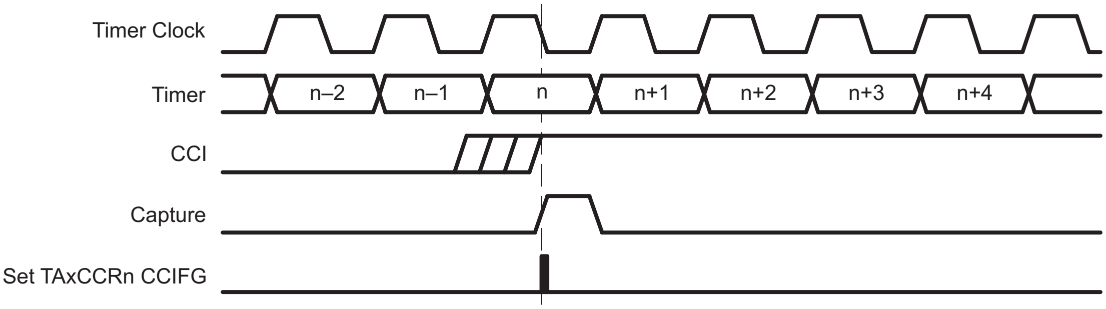


Figure 12-10. Capture Signal (SCS = 1)

NOTE: Changing Capture Input source (CCIS bits)

Switching between CCIxA and CCIxB 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). Note that switching between GND and VCC can be performed at any time. See Section 12.2.4.1.1 for details.

Overflow logic is provided in each capture/compare register to indicate if a second capture was performed before the value from the first capture was read. Bit COV is set when this occurs as shown in Figure 12-11. COV must be reset with software.

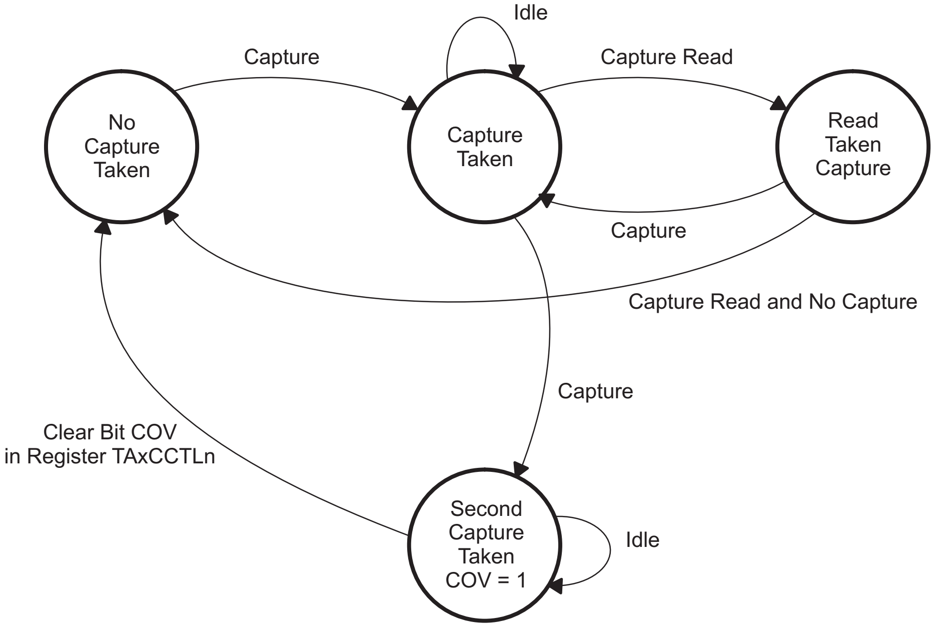

Idle
Capture
Capture Read
No Capture Taken
Capture Taken
Read Taken Capture
Capture
Capture Read and No Capture
Capture
Clear Bit COV in Register TAxCCTLn
Second Capture Taken COV = 1
Idle

Figure 12-11. Capture Cycle

12.2.4.1.1 Capture Initiated by Software

Captures can be initiated by software. The CM bits can be set for capture on both edges. Software then sets 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
```

12.2.4.2 Compare Mode

The compare mode is selected when CAP = 0. The compare mode is used to generate PWM output signals or interrupts at specific time intervals. When TAxR *counts* to the value in a TAxCCRn, where n represents the specific capture/compare register.

- Interrupt flag CCIFG is set.
- Internal signal EQUn = 1.
- EQUn affects the output according to the output mode.
- The input signal CCI is latched into SCCI.

NOTE: Updating TAxCCRn registers

In Compare mode, the timer should be stopped by writing the MC bits to zero (MC = 0) before writing new data to TAxCCRn. Updating TAxCCRn while the timer is running could result in unexpected behaviors.

12.2.5 Output Unit

Each capture/compare block contains an output unit. The output unit is used to generate output signals, such as PWM signals. Each output unit has eight operating modes that generate signals based on the EQU0 and EQUn signals.

12.2.5.1 Output Modes

The output modes are defined by the OUTMOD bits and are described in Table 12-2. The OUTn signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0 because EQUn = EQU0.

Table 12-2. Output Modes

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 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.

12.2.5.1.1 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. Figure 12-12 shows an example using TAxCCR0 and TAxCCR1.

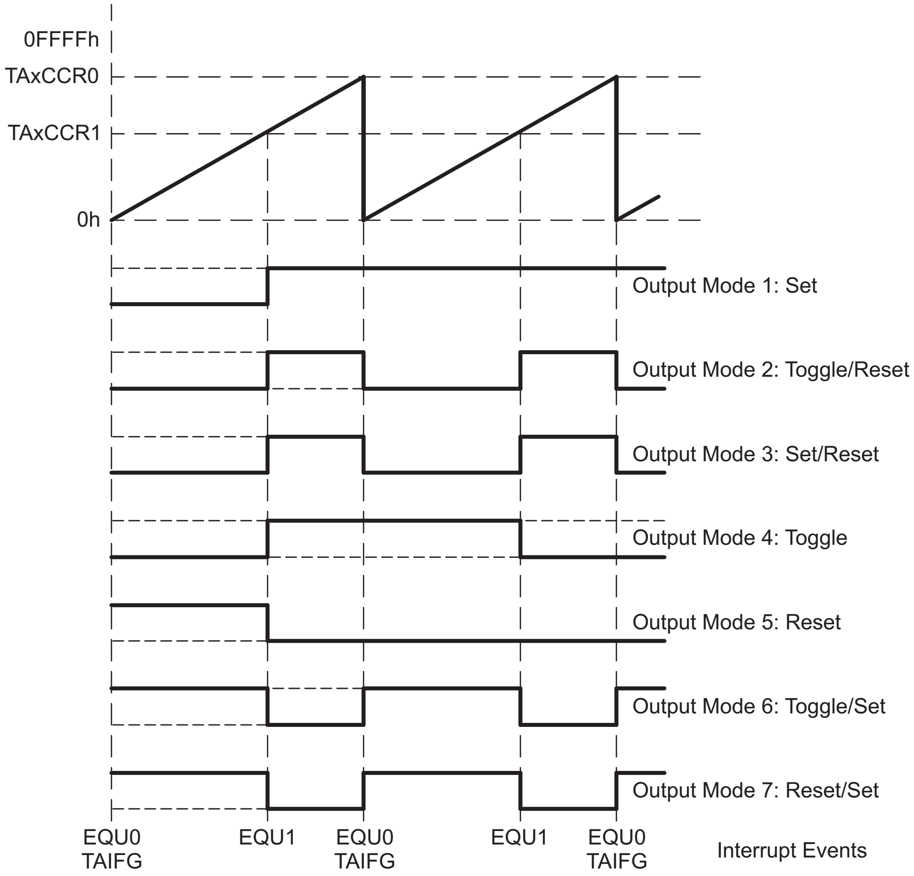


Figure 12-12. Output Example – Timer in Up Mode

12.2.5.1.2 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 12-13 using TAxCCR0 and TAxCCR1.

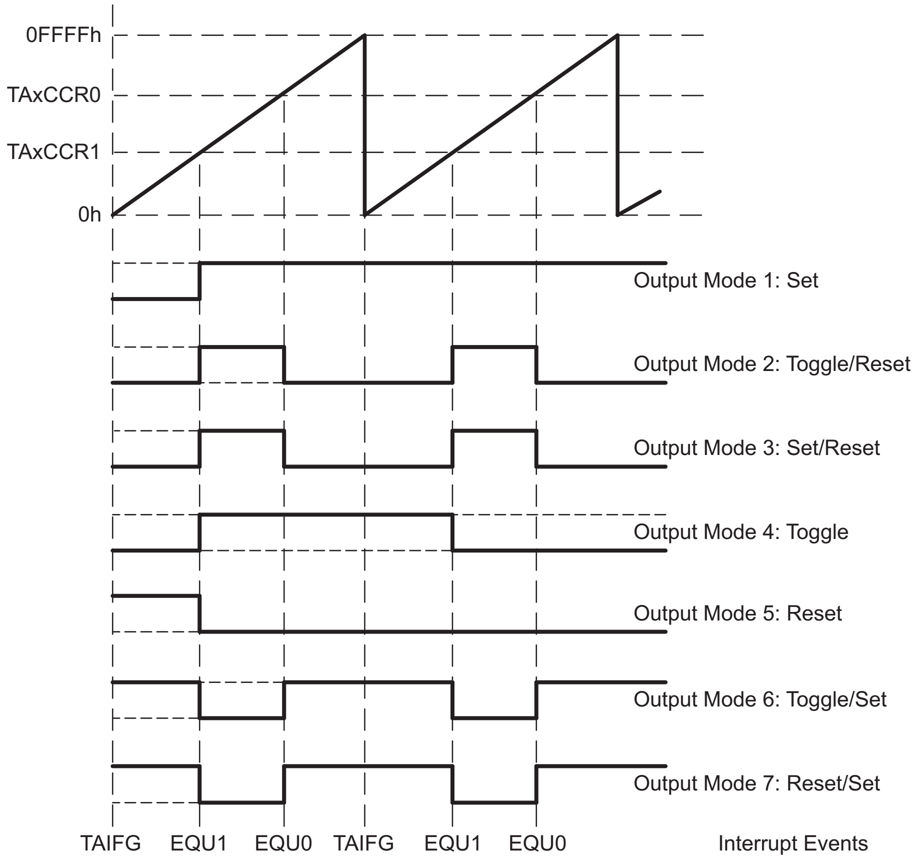


Figure 12-13. Output Example – Timer in Continuous Mode

12.2.5.1.3 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. Figure 12-14 shows an example using TAxCCR0 and TAxCCR2.

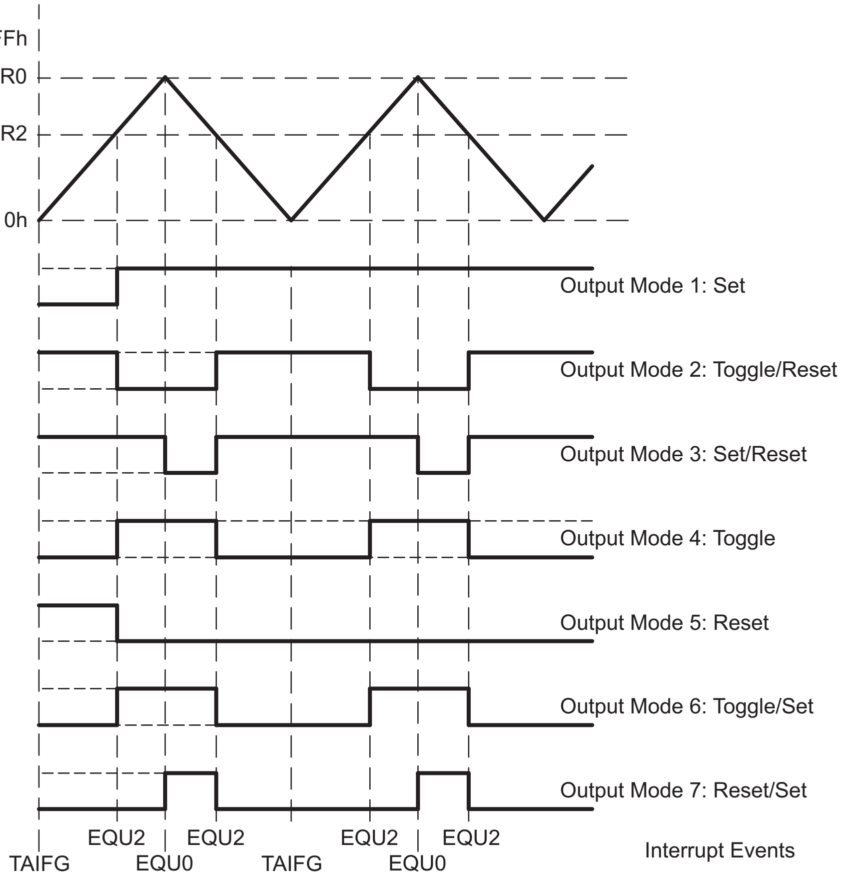


Figure 12-14. Output Example – Timer in Up/Down Mode

NOTE: Switching between output modes

TI recommends stopping the timer (MC = 0) before changing the OUTMOD bits. However, if it is necessary to change OUTMOD bits while the timer is running, one of the OUTMOD bits should remain set during the transition, unless switching to mode 0. Otherwise, output glitching can occur, because a NOR gate decodes output mode 0. A safe method for switching between output modes is to use output mode 7 as a transition state:

```
BIS   #OUTMOD_7,&TA0CCTL1      ; Set output mode=7
BIC   #OUTMOD,&TA0CCTL1        ; Clear unwanted bits
```

12.2.6 Timer_A Interrupts

Two interrupt vectors are associated with the 16-bit Timer_A module:

- TAxCCR0 interrupt vector for TAxCCR0 CCIFG
- TAxIV interrupt vector for all other CCIFG flags and TAIFG

In capture mode, any CCIFG flag is set when a timer value is captured in the associated TAxCCRn register. In compare mode, any CCIFG flag is set if TAxR *counts* to the associated TAxCCRn value. Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set.

12.2.6.1 TAxCCR0 Interrupt

The TAxCCR0 CCIFG flag has the highest Timer_A interrupt priority and has a dedicated interrupt vector as shown in Figure 12-15. The TAxCCR0 CCIFG flag is automatically reset when the TAxCCR0 interrupt request is serviced.

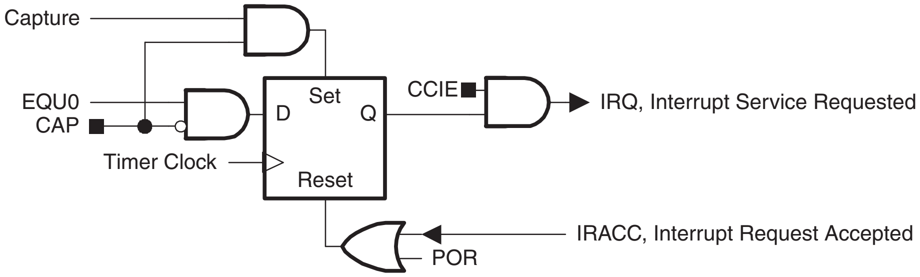

Capture
EQU0
CAP
Timer Clock
Set
D
Q
Reset
CCIE
IRQ, Interrupt Service Requested
IRACC, Interrupt Request Accepted
POR

Figure 12-15. Capture/Compare TAxCCR0 Interrupt Flag

12.2.6.2 TAxIV, Interrupt Vector Generator

The TAxCCRy CCIFG flags and TAIFG flags are prioritized and combined to source a single interrupt vector. The interrupt vector register TAxIV is used to determine which flag requested an interrupt.

The highest-priority enabled interrupt generates a number in the TAxIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Timer_A interrupts do not affect the TAxIV value.

Any access, read or write, of the TAxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, if the TAxCCR1 and TAxCCR2 CCIFG flags are set when the interrupt service routine accesses the TAxIV register, TAxCCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TAxCCR2 CCIFG flag generates another interrupt.

12.2.6.2.1 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
```

NOTE: Changing Timer Clock source

TI recommends stopping the timer before modifying its operation while it is running.

A delay of at least 1.5 timer clocks is required to resynchronize before restarting the timer if the timer clock source is asynchronous to MCLK, because the timer state machine takes this time to synchronize the clock source as the new configuration. (Assuming the timer uses a 1-MHz clock, it is recommended to have a 1.5-µs delay before starting the timer.)

12.2.7 Updating Timer_A Configuration

Care must be taken when applying new configuration to TAxCTL, TAxCTLn, or TAxEX0. The control bits listed are designed not to be dynamically updated while the timer is running, Changing the controls listed below while the timer is running could result in unexpected behaviors. Note that the control bits that are not listed below can be read or updated while the timer is running.

- TAxCTL register
  - Clock source select (TASSEL)
  - Input divider (ID)
  - Mode control (MC) (Note: Switching to Stop mode can be performed at any time)
  - Timer_A clear (TACLR)
- TAxCCTLn registers
  - Capture mode (CM) (Note: Switching to no capture mode can be performed any time)
  - Capture/compare input select (CCIS) (Note: Switching between GND an VCC can be performed at any time)
  - Synchronize capture source (SCS)
  - Capture mode (CAP)
  - Output mode (OUTMOD)
- TAxEX0 register
  - Input divider expansion (TAIDEX)

Follow these steps to update Timer_A configuration:

1. Write zero to the mode control bits (MC = 0) (Note: Do not use TACLR bit to reset the mode control bits).
2. If necessary, write 1 to the TACLR bit (TACLR = 1) to clear TAxR, clock divider state, and the counter direction.
3. If necessary, update counter value to TAxR.
4. If updating the CM, CCIS, SCS bits or the TAxCCRn registers and the timer is in capture mode, disable capture mode first by writing zero to the CAP bit (CAP = 0) or the CM bits (CM = 0).
5. Apply desired configuration to TAxCCRn, TAIDEX, and TAxCCTLn.
6. Apply desired configuration to TAxCTL including the MC bits.

12.3 Timer_A Registers

Timer_A registers are listed in Table 12-3 for the largest configuration available. The base address can be found in the device-specific data sheet.

Table 12-3. Timer_A Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TAxCTL	Timer_Ax Control	Read/write	Word	0000h	Section 12.3.1
02h	TAxCCTL0	Timer_Ax Capture/Compare Control 0	Read/write	Word	0000h	Section 12.3.3
04h	TAxCCTL1	Timer_Ax Capture/Compare Control 1	Read/write	Word	0000h	Section 12.3.3
06h	TAxCCTL2	Timer_Ax Capture/Compare Control 2	Read/write	Word	0000h	Section 12.3.3
08h	TAxCCTL3	Timer_Ax Capture/Compare Control 3	Read/write	Word	0000h	Section 12.3.3
0Ah	TAxCCTL4	Timer_Ax Capture/Compare Control 4	Read/write	Word	0000h	Section 12.3.3
0Ch	TAxCCTL5	Timer_Ax Capture/Compare Control 5	Read/write	Word	0000h	Section 12.3.3
0Eh	TAxCCTL6	Timer_Ax Capture/Compare Control 6	Read/write	Word	0000h	Section 12.3.3
10h	TAxR	Timer_Ax Counter	Read/write	Word	0000h	Section 12.3.2
12h	TAxCCR0	Timer_Ax Capture/Compare 0	Read/write	Word	0000h	Section 12.3.4
14h	TAxCCR1	Timer_Ax Capture/Compare 1	Read/write	Word	0000h	Section 12.3.4
16h	TAxCCR2	Timer_Ax Capture/Compare 2	Read/write	Word	0000h	Section 12.3.4
18h	TAxCCR3	Timer_Ax Capture/Compare 3	Read/write	Word	0000h	Section 12.3.4
1Ah	TAxCCR4	Timer_Ax Capture/Compare 4	Read/write	Word	0000h	Section 12.3.4
1Ch	TAxCCR5	Timer_Ax Capture/Compare 5	Read/write	Word	0000h	Section 12.3.4
1Eh	TAxCCR6	Timer_Ax Capture/Compare 6	Read/write	Word	0000h	Section 12.3.4
2Eh	TAxIV	Timer_Ax Interrupt Vector	Read only	Word	0000h	Section 12.3.5
20h	TAxEX0	Timer_Ax Expansion 0	Read/write	Word	0000h	Section 12.3.6

12.3.1 TAxCTL Register

Timer_Ax Control Register

Figure 12-16. 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 12-4. 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 MC = 0 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 (the divider setting remains unchanged), and the count direction. The TACLR bit is automatically reset and always reads 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

12.3.2 TAxR Register

Timer_Ax Counter Register

Figure 12-17. 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 12-5. TAxR Register Description

Bit	Field	Type	Reset	Description
15-0	TAxR	RW	0h	Timer_A counter. The TAxR register is the count of Timer_A.

12.3.3 TAxCCTLn Register

Timer_Ax Capture/Compare Control n Register

Figure 12-18. 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 12-6. 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 from 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 OUTMOD = 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high

Table 12-6. 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

12.3.4 TAxCCRn Register

Timer_A Capture/Compare n Register

Figure 12-19. 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 12-7. TAxCCRn Register Description

Bit	Field	Type	Reset	Description
15-0	TAxCCRn	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, TAxR, is copied into the TAxCCRn register when a capture is performed.

12.3.5 TAxIV Register

Timer_Ax Interrupt Vector Register

Figure 12-20. 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 12-8. 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

12.3.6 TAxEX0 Register

Timer_Ax Expansion 0 Register

Figure 12-21. 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 12-9. 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

Real-Time Clock (RTC) Counter

The Real-Time Clock (RTC) counter is a 16-bit counter that is functional in active mode (AM) and several low-power modes (LPMs). RTC counter accepts multiple clock sources, which are selected by control registers settings, to generate timing from less than 1 µs up to many hours. This chapter describes the operation and use of the RTC counter module.

Topic **Page**

13.1 RTC Counter Introduction **361**
13.2 RTC Counter Operation **362**
13.3 RTC Counter Registers **364**

13.1 RTC Counter Introduction

The RTC counter is a 16-bit counter that functions in AM and all LPMs except LPM4 and LPM4.5. This module can accept any one of three clock sources:

1. Device specific: SMCLK (maximum operating frequency depends on device configuration) or ACLK (approximately 32 kHz)
2. XT1CLK (approximately 32 kHz)
3. VLOCLK (approximately 10 kHz)

In LPM3.5, RTC counter accepts only XT1CLK or VLOCLK as its clock source to periodically wake up the device. The selected clock source can be predivided before driving the main 16-bit counter. The 16-bit counter supports continuous tick by a 16-bit modulo register that is user accessible and a 16-bit shadow register that is not user accessible. The RTC counter can generate an interrupt when the counter value overflows at the preset shadow register value. RTC counter features include:

- 16-bit modulo counter architecture
  - 16-bit basic counter
  - 16-bit modulo register that is user accessible for read and write
  - 16-bit shadow register that is not user accessible to support continued operation when the modulo value is updated
  - 16-bit compare logic to detect counter overflow at the boundary of the shadow register value
- Three possible clock sources that are selected by setting the RTCSS bits: XT1CLK, VLOCLK, or device specific (SMCLK or ACLK)
  - SMCLK is functional in AM and LPM0 only
  - ACLK is functional in AM to LPM3.
  - XT1CLK and VLOCLK are functional in AM and LPMs, excluding LPM4 and LPM4.5
- Configurable predivider for the source clock input is set by the RTCPS bits
  - Passthrough: ÷1; the input clock source directly drives the 16-bit counter
  - Predivider: ÷10, ÷100, ÷1000, ÷16, ÷64, ÷256, or ÷1024; the input clock source is divided by the selected value before it drives the 16-bit counter
- A hardware interrupt is triggered (if enabled by the RTCIE bit) when the counter value reaches the shadow register value.
- An overflow event can be a trigger in hardware for other modules. See the device-specific data sheet for details on which modules support this trigger.
- Software can reset the counter by setting the RTCSR bit.

Figure 13-1 shows the block diagram of the RTC counter.

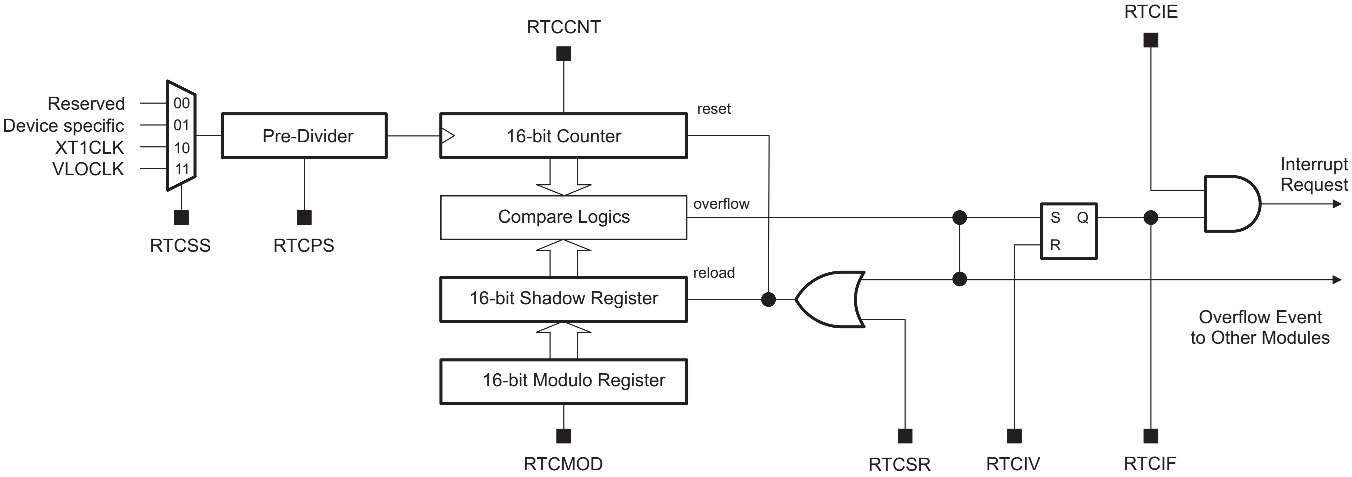


Figure 13-1. RTC Counter Block Diagram

13.2 RTC Counter Operation

The RTC counter module is configured with user software. The setup and operation of RTC counter is described in the following sections.

13.2.1 16-Bit Timer Counter

The 16-bit timer counter register, RTCCNT, increments with each rising edge of the source clock signal. RTCCNT is read only with software. When the counter value reaches the value of the shadow register, the RTC counter generates an overflow signal, the counter value resets to zero, and the counter continues to tick without interruption. As long as the counter clock source that is specified by the RTCSS bit is active, the counter is operational.

RTCCNT is cleared by the overflow event, or it can be reset by software writing logic 1 to the RTCSR bit in the RTCCTL register. If the counter is reset by software, the shadow register is updated by the value in the modulo register at the next cycle of the divided clock, but no overflow event or interrupt is generated.

The maximum input frequency to the counter during LPM3.5 is 40 kHz. Therefore, the predivider must be configured so that the divided clock frequency does not exceed 40 kHz.

13.2.2 Clock Source Select and Divider

In AM and LPM0, the RTC counter clock can be sourced from device-specific (SMCLK or ACLK), XT1CLK, or VLOCLK. In LPM3, ACLK, XT1CLK, or VLOCLK can be selected. In LPM3.5, only XT1CLK or VLOCLK can be selected. The clock source is specified by the RTCSS bits in the RTCCTL register. After reset, RTCSS defaults to 00b (disabled), which means that no clock source is selected.

The selected clock source can be predivided before it is used by the counter. If the passthrough mode (÷1) is selected, the predivider is bypassed and the selected clock source directly sources the counter. The predivider options of ÷16, ÷64, ÷256, and ÷1024 allow simple division of clock source frequencies that are powers of 2, such as from 32768-Hz crystals. The predivider options of ÷10, ÷100, and ÷1000 allow simple division of clock source frequencies that are multiples of 10, such as from 4-MHz or 8-MHz clock inputs.

NOTE: Selected Clock Source in LPM3.5

In LPM3.5, the RTC counter is very low power, and the divided clock source that drives the counter can have a maximum frequency of 40 kHz.

13.2.3 Modulo Register (RTCMOD) and Shadow Register

The modulo register (RTCMOD) is a 16-bit register that is set by user software. The value in RTCMOD is latched and does not take effect until it is loaded into the shadow register. The shadow register is also a 16-bit register, and it stores the modulo value that the RTC counter logic compares with the counter value. The shadow register acts as a buffer to the RTCMOD register, so that software can set a new modulo value without interrupting the counter. The RTCMOD register is read and write accessible by the user. The shadow register is not accessible the user.

The value in RTCMOD is loaded to the shadow register under two conditions:

1. When the counter reaches the value in the shadow register, which also triggers an overflow signal and clears the counter value.
2. When a software reset is triggered by software writing logic 1 to the RTCSR bit in the RTCCTL register.

Because the shadow register always updates its value from RTCMOD, software must set RTCMOD before the hardware overflow occurs. Using the software reset lets software immediately set the target modulo value into shadow register without waiting for the next overflow. If the value in RTCMOD is not updated when the hardware overflow occurs, the shadow register fetches the previous modulo value stored in RTCMOD. If RTCMOD is changed multiple times before the overflow, only the last value is loaded to the shadow register.

RTC counter always generates an overflow when the RTCMOD is set to either 0x0000 or 0x0001.

Care should be taken when setting RTCMOD so that the overflow events do not happen too quickly to be serviced. When the selected RTC counter source frequency is close to the CPU clock frequency, the modulo value must be long enough that the CPU is able to respond to the RTC counter interrupt service routine (ISR) in time before the next RTC counter interrupt occurs. In addition, frequent writes to the RTCSR bit (software reset) could lead to an overflow event being missed, as the count is reset each time, and the RTCMOD setting overwrites the current shadow register setting.

Figure 13-2 shows a hardware overflow event loading the new value (0x2000) from RTCMOD into the shadow register, replacing the previous value (0x4000).

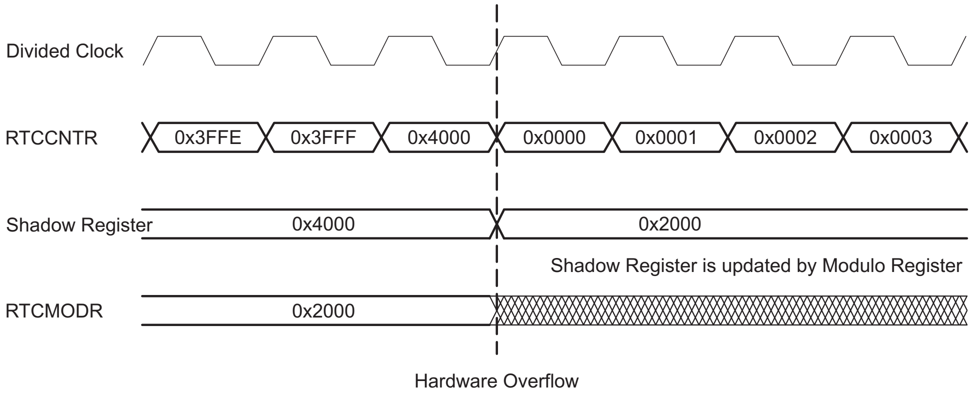


Figure 13-2. Shadow Register Example

13.2.4 RTC Counter Interrupt and External Event/Trigger

There is an interrupt vector (RTCIV) associated with the 16-bit RTC counter module interrupt flag (RTCIFG).

When an overflow occurs, the RTCIFG bit in the RTCCTL register is set until it is cleared by a read of the RTCIV register. At the same time, an interrupt is submitted to the CPU for post-processing, if the RTCIE bit in the RTCCTL register is set. Reading RTCIV register clears the interrupt flag.

TI recommends clearing the RTCIFG bit by reading the RTCIV register before enabling the RTC counter interrupt. Otherwise, an interrupt might be generated if the RTCIFG was already set by a previous overflow.

In addition to the interrupt, the hardware overflow also submits an external trigger to other on-chip modules as a synchronous signal. Refer to the device-specific data sheet for more information on module triggers that are available on particular devices.

13.3 RTC Counter Registers

Table 13-1 lists the RTC counter registers and the address offset for each register. Refer to the device-specific data sheet for the base address of the module.

Table 13-1. RTC Counter Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	RTCCTL	Real-Time Clock Control	Read/write	Word	0000h	Section 13.3.1
00h	RTCCTL_L		Read/write	Byte	00h	
01h	RTCCTL_H		Read/write	Byte	00h	
04h	RTCIV	Real-Time Clock Interrupt Vector	Read/write	Word	0000h	Section 13.3.2
04h	RTCIV_L		Read/write	Byte	00h	
05h	RTCIV_H		Read/write	Byte	00h	
08h	RTCMOD	Real-Timer Clock Modulo	Read/write	Word	BEEFh	Section 13.3.3
08h	RTCMOD_L		Read/write	Byte	EFh	
09h	RTCMOD_H		Read/write	Byte	BEh	
0Ch	RTCCNT	Real-Time Clock Counter	Read	Word	0000h	Section 13.3.4
0Ch	RTCCNT_L		Read	Byte	00h	
0Dh	RTCCNT_H		Read	Byte	00h	

13.3.1 RTCCTL Register

RTC Counter Control Register

Figure 13-3. RTCCTL Register

15	14	13	12	11	10	9	8
Reserved		RTCSS		Reserved	RTCPS		
r0	r0	rw-{0}	rw-{0}	r0	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
Reserved	RTCSR	Reserved				RTCIE	RTCIFG
r0	w-{0}	r0	r0	r0	r0	rw-{0}	r-{0}

Table 13-2. RTCCTL Register Description

Bit	Field	Type	Reset	Description
15-14	Reserved	R	0h	Reserved
13-12	RTCSS	RW	0h	Real-time clock source select 00b = Reserved 01b = Device specific 10b = XT1CLK 11b = VLOCLK
11	Reserved	R	0h	Reserved
10-8	RTCPS	RW	0h	Real-time clock predivider select 000b = /1 001b = /10 010b = /100 011b = /1000 100b = /16 101b = /64 110b = /256 111b = /1024
7	Reserved	R	0h	Reserved
6	RTCSR	W	0h	Real-time software reset. This is a write-only bit and is always read with logic 0. 0b = Write 0 has no effect 1b = Write 1 to this bit clears the counter value and reloads the shadow register value from the modulo register at the next tick of the selected source clock. No overflow event or interrupt is generated.
5-2	Reserved	R	0h	Reserved
1	RTCIE	RW	0h	Real-time interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
0	RTCIFG	R	0h	Real-time interrupt flag. This bit reports the status of a pending interrupt. This read only bit can be cleared by reading RTCIV register. 0b = No interrupt pending 1b = Interrupt pending

13.3.2 RTCIV Register

RTC Counter Interrupt Vector Register

Figure 13-4. RTCIV Register

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

Table 13-3. RTCIV Register Description

Bit	Field	Type	Reset	Description
15-0	RTCIV	R	0h	Low-power-counter interrupt vector. 00h = No interrupt pending 02h = Interrupt Source: RTC Counter Overflow; Interrupt Flag: RTCIFG

13.3.3 RTCMOD Register

RTC Counter Modulo Register

Figure 13-5. RTCMOD Register

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

Table 13-4. RTCMOD Register Description

Bit	Field	Type	Reset	Description
15-0	RTCMOD	RW	BEEFh	RTC modulo value

13.3.4 RTCCNT Register

RTC Counter Register

Figure 13-6. RTCCNT Register

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

Table 13-5. RTCCNT Register Description

Bit	Field	Type	Reset	Description
15-0	RTCCNT	R	0h	RTC counter. This is a read-only register and reflects the current counter value.

Chapter 14

SLAU445D–October 2014–Revised December 2015

32-Bit Hardware Multiplier (MPY32)

This chapter describes the 32-bit hardware multiplier (MPY32). The MPY32 module is implemented in all devices.

Topic **Page**

14.1 32-Bit Hardware Multiplier (MPY32) Introduction 369
14.2 MPY32 Operation 371
14.3 MPY32 Registers 383

14.1 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 14-1.

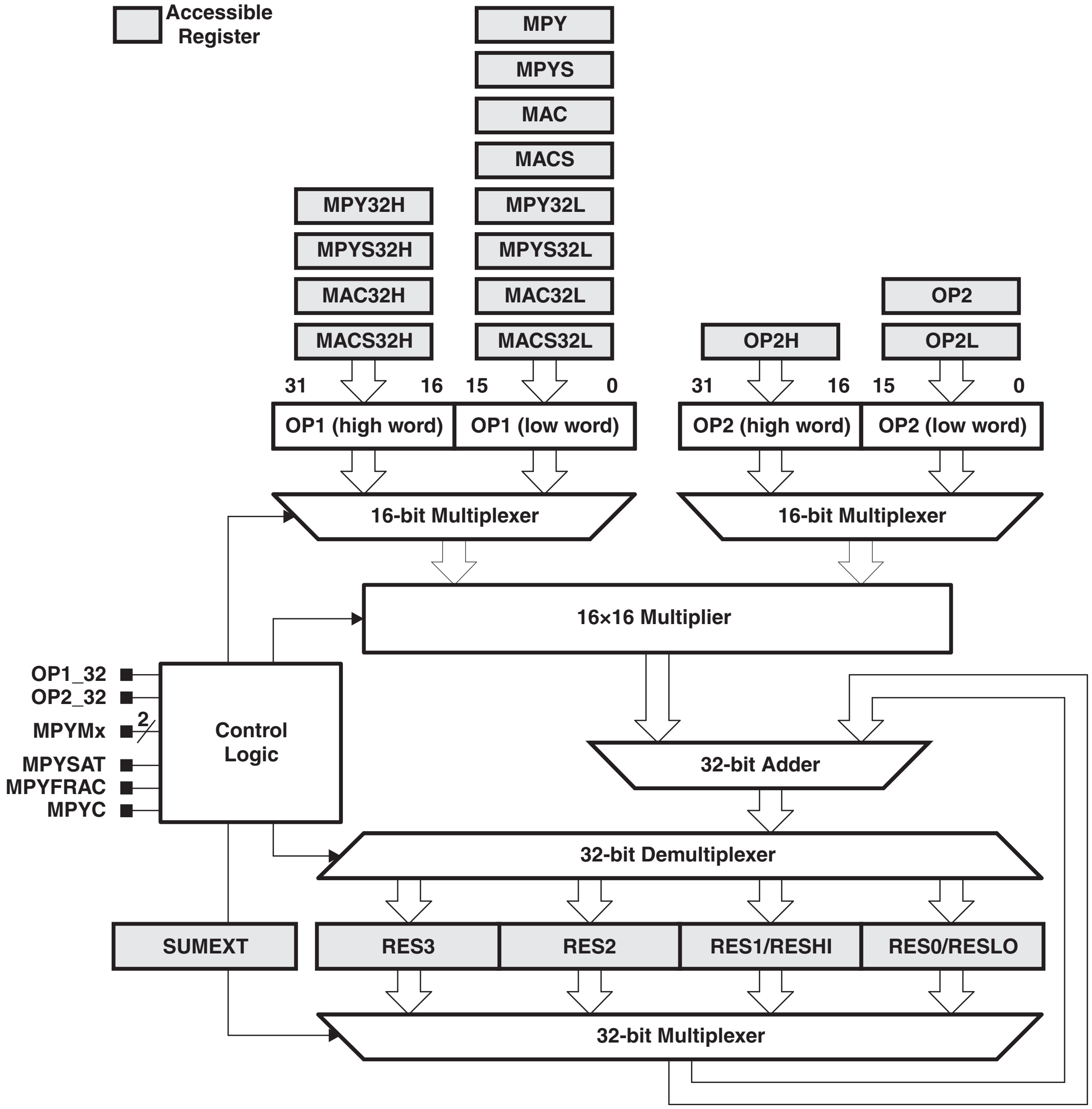


Figure 14-1. MPY32 Block Diagram

14.2 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 to which the first operand is written.

The hardware multiplier has two 32-bit operand registers – operand one (OP1) and operand two (OP2), and a 64-bit result register accessible through registers RES0 to RES3. For compatibility with the 16×16 hardware multiplier, the result of a 8-bit or 16-bit operation is also accessible through RESLO, RESHI, and SUMEXT. 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 14-1 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 14-1. 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

14.2.1 Operand Registers

Operand one (OP1) has 12 registers (see Table 14-2) that are used to load data into the multiplier and also to 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 14-2. 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 14-3. 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 14-1 for how many CPU cycles are needed until a certain result register is ready and valid for each of the different modes.

14.2.2 Result Registers

The multiplication result is always 64 bits wide. It is accessible through 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 through 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 14-4. 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 14-4 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 14-4. 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

14.2.2.1 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.

14.2.3 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
```

14.2.4 Fractional Numbers

The MPY32 provides support for fixed-point signal processing. In fixed-point signal processing, fractional numbers 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 14-2 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.

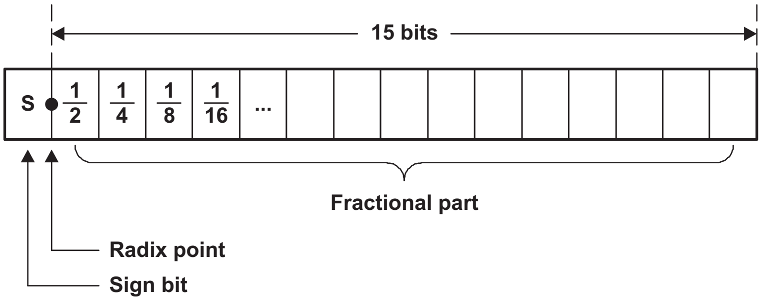


Figure 14-2. Q15 Format Representation

The range can be increased by shifting the radix point to the right as shown in Figure 14-3. The signed Q14 format with 16 bits gives a range from –2.0 to 1.999938965 ≈ 2.0.

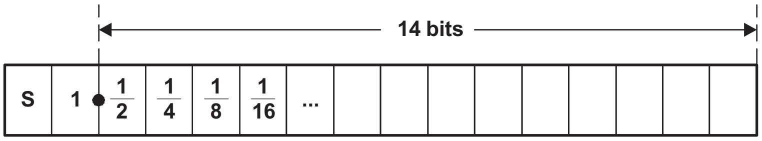


Figure 14-3. Q14 Format Representation

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.

14.2.4.1 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 14-5. 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

14.2.4.2 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 14-6. 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 14-4 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 unshifted result, that is, the result that is read with fractional mode disabled.

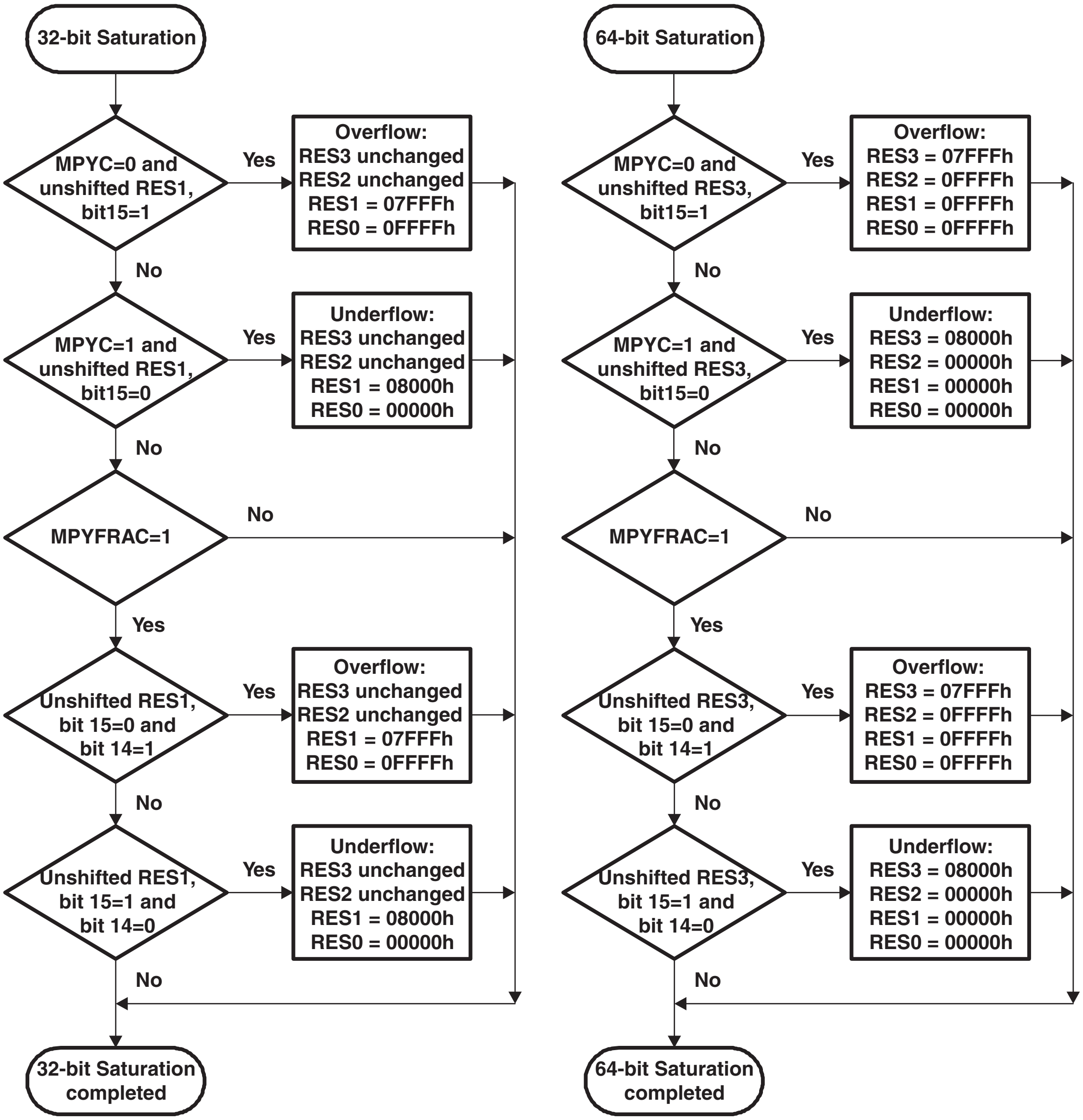

32-bit Saturation
MPYC=0 and unshifted RES1, bit15=1
Yes
Overflow: RES3 unchanged RES2 unchanged RES1 = 07FFFh RES0 = 0FFFFh
No
MPYC=1 and unshifted RES1, bit15=0
Yes
Underflow: RES3 unchanged RES2 unchanged RES1 = 08000h RES0 = 00000h
No
MPYFRAC=1
No
Yes
Unshifted RES1, bit 15=0 and bit 14=1
Yes
Overflow: RES3 unchanged RES2 unchanged RES1 = 07FFFh RES0 = 0FFFFh
No
Unshifted RES1, bit 15=1 and bit 14=0
Yes
Underflow: RES3 unchanged RES2 unchanged RES1 = 08000h RES0 = 00000h
No
32-bit Saturation completed
64-bit Saturation
MPYC=0 and unshifted RES3, bit15=1
Yes
Overflow: RES3 = 07FFFh RES2 = 0FFFFh RES1 = 0FFFFh RES0 = 0FFFFh
No
MPYC=1 and unshifted RES3, bit15=0
Yes
Underflow: RES3 = 08000h RES2 = 00000h RES1 = 00000h RES0 = 00000h
No
MPYFRAC=1
No
Yes
Unshifted RES3, bit 15=0 and bit 14=1
Yes
Overflow: RES3 = 07FFFh RES2 = 0FFFFh RES1 = 0FFFFh RES0 = 0FFFFh
No
Unshifted RES3, bit 15=1 and bit 14=0
Yes
Underflow: RES3 = 08000h RES2 = 00000h RES1 = 00000h RES0 = 00000h
No
64-bit Saturation completed

Figure 14-4. 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 before the result is converted into a fractional number, it 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 14-4.

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.

14.2.5 Putting It All Together

Figure 14-5 shows the complete multiplication flow, depending on the various selectable modes for the MPY32 module.

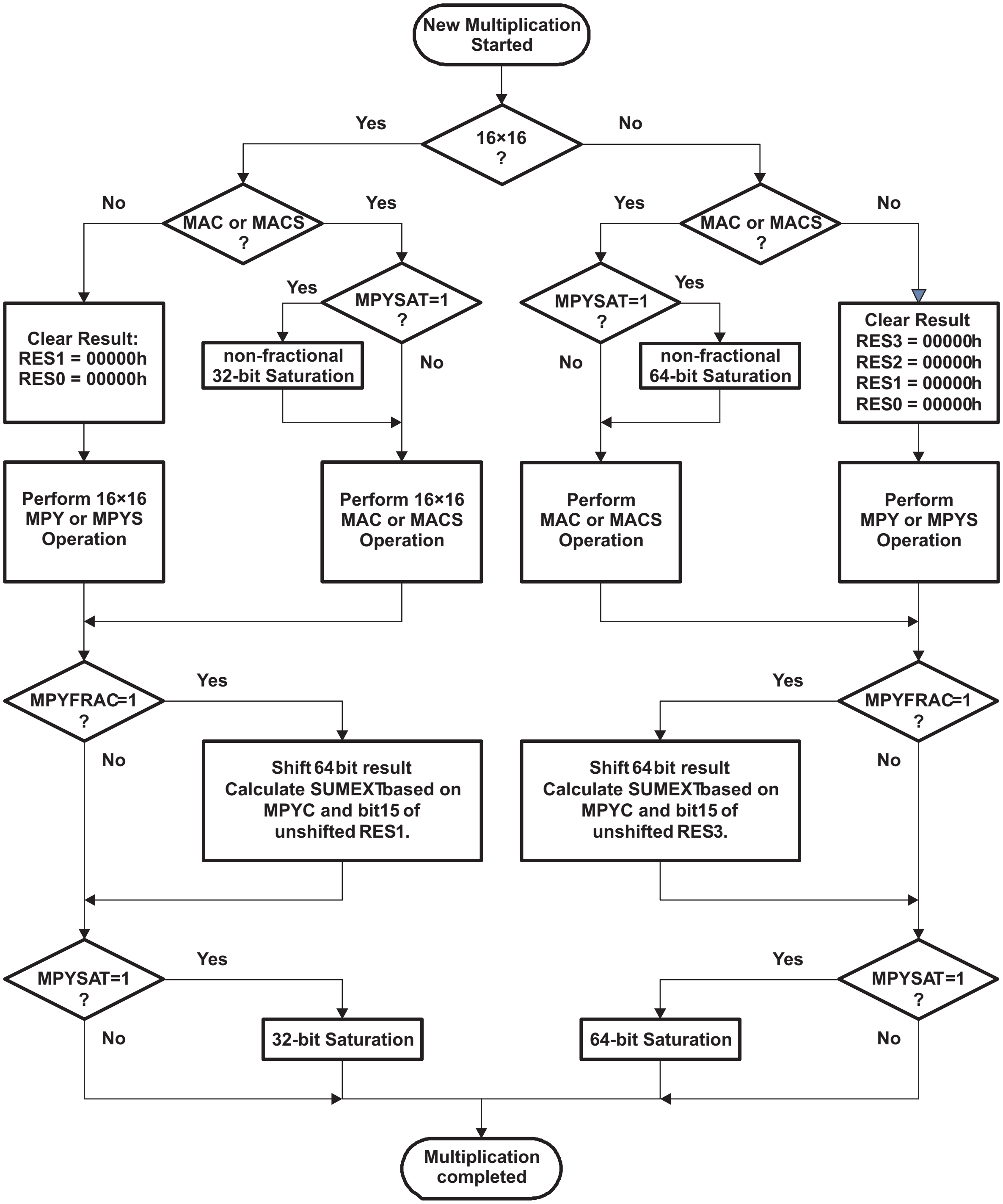

New Multiplication Started
16×16 ?
Yes
No
MAC or MACS ?
No
Yes
MAC or MACS ?
Yes
No
MPYSAT=1 ?
Yes
No
MPYSAT=1 ?
Yes
No
Clear Result: RES1 = 00000h RES0 = 00000h
non-fractional 32-bit Saturation
non-fractional 64-bit Saturation
Clear Result RES3 = 00000h RES2 = 00000h RES1 = 00000h RES0 = 00000h
Perform 16×16 MPY or MPYS Operation
Perform 16×16 MAC or MACS Operation
Perform MAC or MACS Operation
Perform MPY or MPYS Operation
MPYFRAC=1 ?
Yes
No
MPYFRAC=1 ?
Yes
No
Shift 64bit result Calculate SUMEXTbased on MPYC and bit15 of unshifted RES1.
Shift 64bit result Calculate SUMEXTbased on MPYC and bit15 of unshifted RES3.
MPYSAT=1 ?
Yes
No
MPYSAT=1 ?
Yes
No
32-bit Saturation
64-bit Saturation
Multiplication completed

Figure 14-5. 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 or 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.

14.2.6 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 14-1, 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
```

14.2.7 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
```

14.2.7.1 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
```

14.2.8 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 by DMA. The signal into the DMA controller is 'Multiplier ready' (see the DMA Controller chapter for details).

14.3 MPY32 Registers

MPY32 registers are listed in Table 14-7. The base address can be found in the device-specific data sheet. The address offsets are listed in Table 14-7.

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 14-7. 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 14-7. 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 14-8 are treated equally.

Table 14-8. 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

14.3.1 MPY32CTL0 Register

32-Bit Hardware Multiplier Control 0 Register

Figure 14-6. 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 14-9. 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

LCD_E Controller

The LCD_E controller drives static and 2-mux to 8-mux LCDs. This chapter describes the LCD_E controller. The differences between LCD_B, LCD_C and LCD_E are listed in Table 15-1.

Topic **Page**

15.1 LCD_E Introduction **387**
15.2 LCD_E Operation **389**
15.3 LCD_E Registers **412**

15.1 LCD_E Introduction

The LCD_E controller directly drives LCD displays by automatically creating the ac segment and common voltage signals. The LCD_E controller can support static and 2-mux to 8-mux LCD glasses.

The LCD_E controller features are:

- Display memory
- Supports LPM3.5
- Configurable SEG and COM pins
- Automatic signal generation
- Configurable frame frequency
- Blinking of individual segments with separate blinking memory for static, and 2- to 4-mux LCDs
- Blinking of complete display for 5- to 8-mux LCDs
- Regulated charge pump up to 3.44 V (typical)
- Contrast control by software
- Support for the following types of LCDs
  - Static
  - 2-mux, 1/3 bias
  - 3-mux, 1/3 bias
  - 4-mux, 1/3 bias
  - 5-mux, 1/3 bias
  - 6-mux, 1/3 bias
  - 7-mux, 1/3 bias
  - 8-mux, 1/3 bias

Table 15-1 lists the differences between LCD_B, LCD_C, and LCD_E.

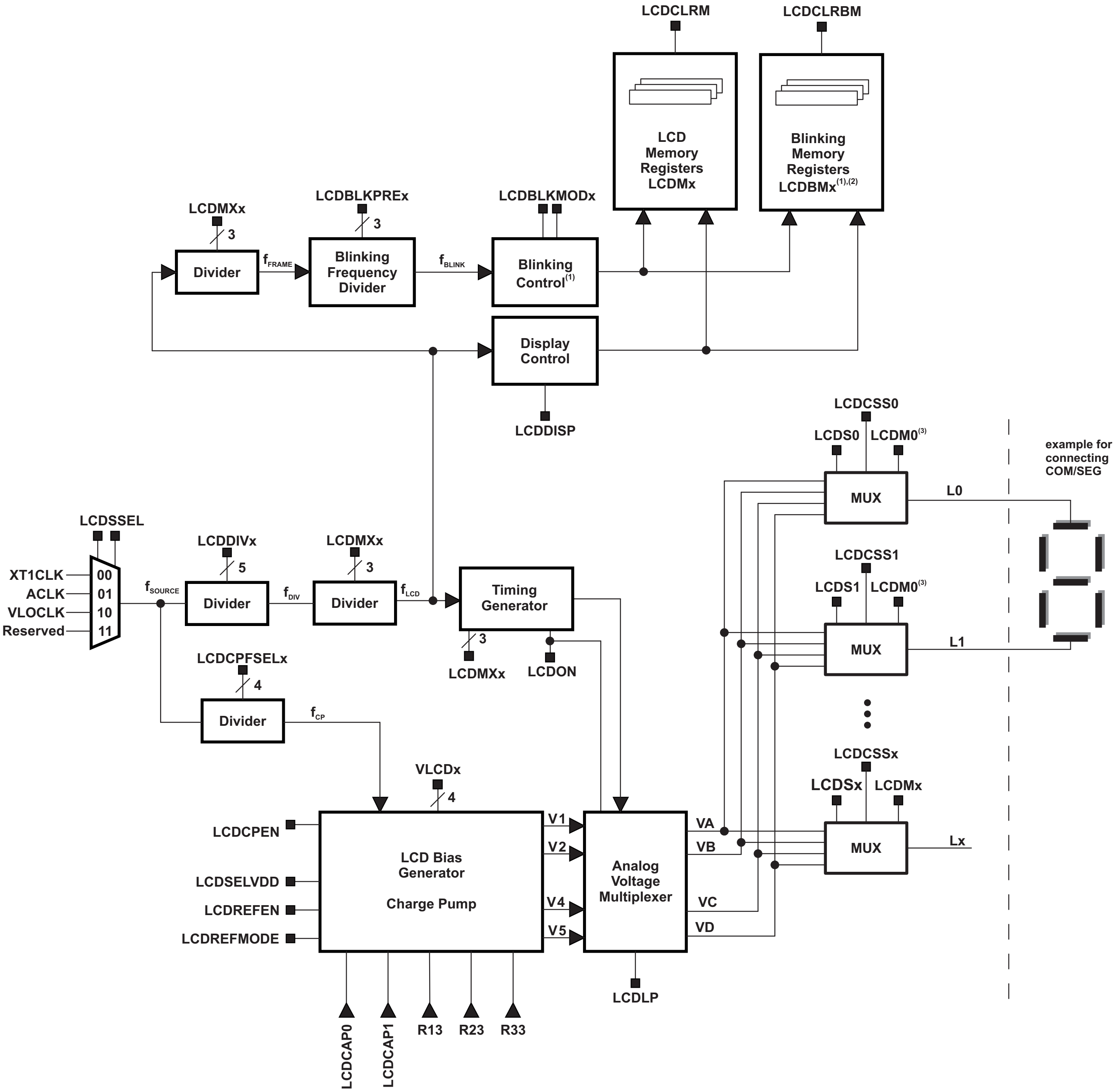
Table 15-1. Differences Between LCD_B, LCD_C, and LCD_E

Feature	LCD_B	LCD_C	LCD_E
Supported types of LCDs	Static, 2-, 3-, 4-mux	Static, 2-, 3-, 4-, 5-, 6-, 7, 8-mux	Static, 2-, 3-, 4-mux 5-, 6-, 7, 8-mux (device specific)
LCD bias modes	1/2 bias and 1/3 bias	1/2 bias and 1/3 bias	1/3 bias
LCD Blinking Memory	yes	yes	device specific
SEG/COM mux	COM fixed	COM fixed	each LCD drive pin
External Pins	R03, R13, R23, R33	R03, R13, R23, R33	R13, R23, R33, LCDCAP0, LCDCAP1
LPM3.5	not supported	not supported	supported
Maximum VLCDx settings	001111b	001111b	001111b
Maximum LCD voltage ($V_{LCD,typ}$)	3.44 V	3.44 V	3.44 V
Number of LCD pins	up to 4 x 46	up to 4 x 50 or 8 x 46	up to 4 x 60 or 8 x 56

Figure 15-1 shows the LCD controller block diagram.

NOTE: Maximum LCD Segment Control

The maximum number of segment lines and memory registers available differs with device. See the device-specific data sheet for available segment pins and the maximum number of segments supported.



(1) device specific
(2) only static, 2- to 4-mux)
(3) used LCDMx depends on selected MUX mode (LCDMXx)

Figure 15-1. LCD Controller Block Diagram

15.2 LCD_E Operation

The LCD controller is configured with user software. The setup and operation of the LCD controller is discussed in the following sections.

15.2.1 LCD Memory

The LCD memory organization differs slightly depending on the mode. Each memory bit corresponds to one LCD segment, LCD common or is not used, depending on the mode. To turn on an LCD segment, its corresponding memory bit is set. The memory can also be accessed word-wise using the even addresses starting at LCDM0W, LCDM2W, and so on. Setting the bit LCDCLRM clears all LCD display memory registers at the next frame boundary. It is reset automatically after the registers are cleared.

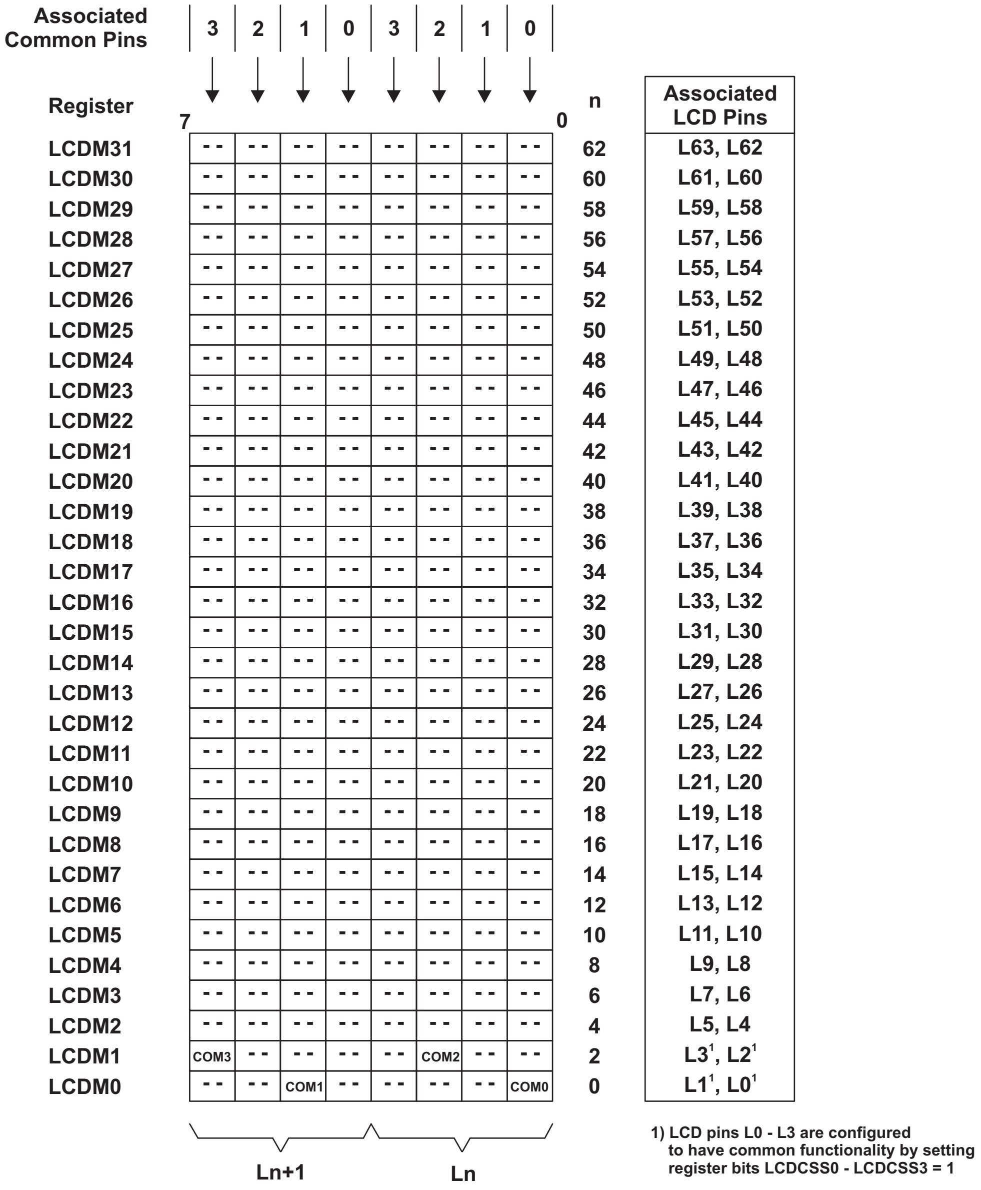
15.2.1.1 Static and 2-Mux to 4-Mux Mode

For static and 2-mux to 4-mux modes, one byte of the LCD memory contains the information for two segment lines.

In static and 2-mux to 4-mux modes, the following maximum LCD segments are possible:

- Static: up to 63 segments (one COM line)
- 2-mux: up to 124 segments (two COM lines)
- 3-mux: up to 183 segments (three COM lines)
- 4-mux: up to 240 segments (four COM lines)

Figure 15-2 shows an example LCD memory map for 4-mux mode with 240 segments.



Associated Common Pins	3	2	1	0	3	2	1	0		
Register	7							0	n	Associated LCD Pins
LCDM31	- -	- -	- -	- -	- -	- -	- -	- -	62	L63, L62
LCDM30	- -	- -	- -	- -	- -	- -	- -	- -	60	L61, L60
LCDM29	- -	- -	- -	- -	- -	- -	- -	- -	58	L59, L58
LCDM28	- -	- -	- -	- -	- -	- -	- -	- -	56	L57, L56
LCDM27	- -	- -	- -	- -	- -	- -	- -	- -	54	L55, L54
LCDM26	- -	- -	- -	- -	- -	- -	- -	- -	52	L53, L52
LCDM25	- -	- -	- -	- -	- -	- -	- -	- -	50	L51, L50
LCDM24	- -	- -	- -	- -	- -	- -	- -	- -	48	L49, L48
LCDM23	- -	- -	- -	- -	- -	- -	- -	- -	46	L47, L46
LCDM22	- -	- -	- -	- -	- -	- -	- -	- -	44	L45, L44
LCDM21	- -	- -	- -	- -	- -	- -	- -	- -	42	L43, L42
LCDM20	- -	- -	- -	- -	- -	- -	- -	- -	40	L41, L40
LCDM19	- -	- -	- -	- -	- -	- -	- -	- -	38	L39, L38
LCDM18	- -	- -	- -	- -	- -	- -	- -	- -	36	L37, L36
LCDM17	- -	- -	- -	- -	- -	- -	- -	- -	34	L35, L34
LCDM16	- -	- -	- -	- -	- -	- -	- -	- -	32	L33, L32
LCDM15	- -	- -	- -	- -	- -	- -	- -	- -	30	L31, L30
LCDM14	- -	- -	- -	- -	- -	- -	- -	- -	28	L29, L28
LCDM13	- -	- -	- -	- -	- -	- -	- -	- -	26	L27, L26
LCDM12	- -	- -	- -	- -	- -	- -	- -	- -	24	L25, L24
LCDM11	- -	- -	- -	- -	- -	- -	- -	- -	22	L23, L22
LCDM10	- -	- -	- -	- -	- -	- -	- -	- -	20	L21, L20
LCDM9	- -	- -	- -	- -	- -	- -	- -	- -	18	L19, L18
LCDM8	- -	- -	- -	- -	- -	- -	- -	- -	16	L17, L16
LCDM7	- -	- -	- -	- -	- -	- -	- -	- -	14	L15, L14
LCDM6	- -	- -	- -	- -	- -	- -	- -	- -	12	L13, L12
LCDM5	- -	- -	- -	- -	- -	- -	- -	- -	10	L11, L10
LCDM4	- -	- -	- -	- -	- -	- -	- -	- -	8	L9, L8
LCDM3	- -	- -	- -	- -	- -	- -	- -	- -	6	L7, L6
LCDM2	- -	- -	- -	- -	- -	- -	- -	- -	4	L5, L4
LCDM1	COM3	- -	- -	- -	- -	COM2	- -	- -	2	L3[1], L2[1]
LCDM0	- -	- -	COM1	- -	- -	- -	- -	COM0	0	L1[1], L0[1]

1) LCD pins L0 - L3 are configured to have common functionality by setting register bits LCDCSS0 - LCDCSS3 = 1

Figure 15-2. LCD Memory for Static to 4-Mux Mode – Example for 4-Mux Mode With 240 Segments

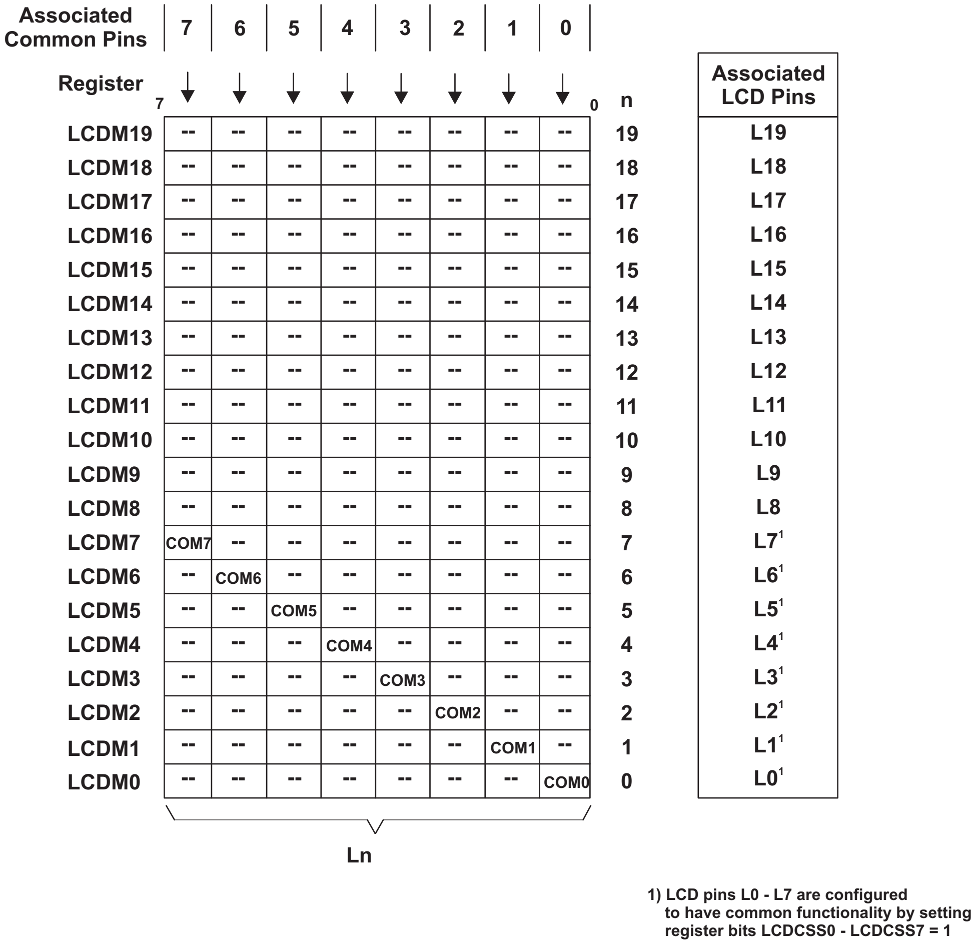
15.2.1.2 5-Mux to 8-Mux Mode

For 5-mux to 8-mux modes, one byte of the LCD memory contains the information for one segment line.

In 5-mux to 8-mux modes the following maximum LCD segments are possible:

- 5-mux: up to 295 segments (five COM lines)
- 6-mux: up to 348 segments (six COM lines)
- 7-mux: up to 399 segments (seven COM lines)
- 8-mux: up to 448 segments (eight COM lines)

Figure 15-3 shows an example LCD memory map for 8-mux mode with 96 segments.



Associated Common Pins	7	6	5	4	3	2	1	0		
Register	7							0	n	Associated LCD Pins
LCDM19	--	--	--	--	--	--	--	--	19	L19
LCDM18	--	--	--	--	--	--	--	--	18	L18
LCDM17	--	--	--	--	--	--	--	--	17	L17
LCDM16	--	--	--	--	--	--	--	--	16	L16
LCDM15	--	--	--	--	--	--	--	--	15	L15
LCDM14	--	--	--	--	--	--	--	--	14	L14
LCDM13	--	--	--	--	--	--	--	--	13	L13
LCDM12	--	--	--	--	--	--	--	--	12	L12
LCDM11	--	--	--	--	--	--	--	--	11	L11
LCDM10	--	--	--	--	--	--	--	--	10	L10
LCDM9	--	--	--	--	--	--	--	--	9	L9
LCDM8	--	--	--	--	--	--	--	--	8	L8
LCDM7	COM7	--	--	--	--	--	--	--	7	L7[1]
LCDM6	--	COM6	--	--	--	--	--	--	6	L6[1]
LCDM5	--	--	COM5	--	--	--	--	--	5	L5[1]
LCDM4	--	--	--	COM4	--	--	--	--	4	L4[1]
LCDM3	--	--	--	--	COM3	--	--	--	3	L3[1]
LCDM2	--	--	--	--	--	COM2	--	--	2	L2[1]
LCDM1	--	--	--	--	--	--	COM1	--	1	L1[1]
LCDM0	--	--	--	--	--	--	--	COM0	0	L0[1]

1) LCD pins L0 - L7 are configured to have common functionality by setting register bits LCDCSS0 - LCDCSS7 = 1

Figure 15-3. LCD Memory for 5-Mux to 8-Mux Mode – Example for 8-Mux Mode With 96 Segments

15.2.2 Configuration of Port Pin as LCD Output

LCD segments and common functions are multiplexed with digital I/O functions. These pins can function either as digital I/O or as LCD functions. The LCD segment/common functions, when multiplexed with digital I/O, are selected using the LCDSx bits in the LCDPCTLx registers. Setting LCDSx bits select the LCD function for each pin. When LCDSx = 0, a multiplexed pin is set to digital I/O function. When LCDSx = 1, a multiplexed pin is selected as LCD function. See the port schematic section of the device-specific data sheet for details on controlling the pin functionality.

NOTE: LCDSx Bits Do Not Affect Dedicated LCD Segment/Common Pins

The LCDSx bits only affect pins with multiplexed LCD segment/common functions and digital I/O functions. Dedicated LCD segment/common pins are not affected by the LCDSx bits.

15.2.3 Configuration of LCD Pin as COM or SEG

To simplify board layout and routing of segment and common lines, each LCD pin can be either defined as LCD segment (SEG) or as common line (COM). The LCDCSSx bits define how the content of the LCDMx registers are interpreted. By setting LCDCSSx=0 the LCD pins Lx are working as LCD segments. When setting LCDCSSx = 1, the contents of LCDMx define which common line (COM0 to COM7) is used at the corresponding LCD pin Lx.

The use of this functionality is described in the following sections.

15.2.3.1 Defining LCD Pin as Segment

Static, 2-, 3-, 4-mux Mode

In static, 2-, 3-, and 4-mux mode, the LCDMx register contains the memory for two segment pins. For example LCDM1 contains L3 and L2 (see Section 15.2.1). To define the LCD Pin as LCD segment the corresponding bit in LCDCSSELx register must be set to 0 (default). With this the LCDMx registers are used to enable or disable LCD segments. For example, to define LCD pin L14 as LCD segment, set LCDCSS14 = 0 in the LCDCSSEL0 register.

5-, 6-, 7- and 8-mux Mode

In 5-, 6-, 7- and 8-mux mode, each LCDMx register contains the memory for one segment pin. To define the LCD pin as LCD segment, the corresponding bit in LCDCSSELx register must be set to 0 (default). With this the LCDMx registers are used to enable or disable LCD segments. For instance LCDM7 contains memory for L7, LCDM29 for L29, and so on.

NOTE: See the device-specific data sheet to determine whether or not 5-, 6-, 7-, or 8-mux mode is available on a device.

15.2.3.2 Defining LCD Pin as Common Line

NOTE: Only one common (COMx) pin per LCD pin can be selected. Assigning two or more common functions to one LCD pin can lead to unpredicted behavior.

To define the LCD pin to have LCD common functionality, the corresponding bit in LCDCSSELx register must be set to 1. By this the LCDMx register is used to configure the associated LCD pin to have COMx functionality.

The LCDMx setting behaves differently, depending on whether static- to 4-mux mode or 5-mux to 8-mux mode is used. The differences are described in the following sections.

15.2.3.2.1 COM Assignment in Static, 2-, 3-, or 4-Mux Mode

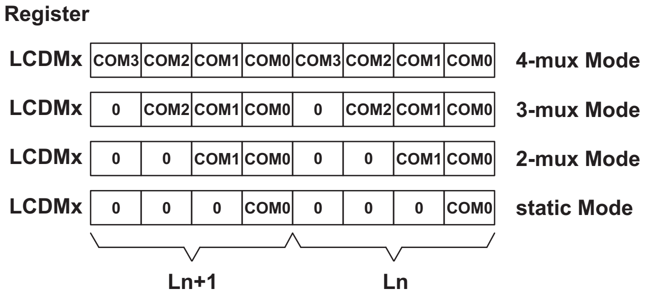
In static, 2-, 3-, or 4-mux mode, each LCDMx is used to control the common functionality of two LCD pins. Similar to the segment functionality described in Section 15.2.1, the lower nibble of LCDMx is used to control even-numbered LCD pins (L0, L2, ...). Odd-numbered LCD pins (L1, L3, ...) are controlled by the upper nibble of LCDMx. Selecting two or more COM pins per LCD pin can lead to unpredicted behavior of the LCD and must be avoided.

In static mode only COM0 is available.

In 2-mux mode COM0 and COM1 can be selected.

In 3-mux mode COM0, COM1, and COM2 can be selected.

In 4-mux mode COM0, COM1, COM2, and COM3 can be selected.



Register									
LCDMx	COM3	COM2	COM1	COM0	COM3	COM2	COM1	COM0	4-mux Mode
LCDMx	0	COM2	COM1	COM0	0	COM2	COM1	COM0	3-mux Mode
LCDMx	0	0	COM1	COM0	0	0	COM1	COM0	2-mux Mode
LCDMx	0	0	0	COM0	0	0	0	COM0	static Mode
	Ln+1				Ln				

SEG_{MAX} = Number of Segment Pins

$n = 0 \ldots \frac{SEG_{MAX} - 1}{2}$

x = n

Figure 15-4. LCDMx in Static, 2-, 3-, or 4-Mux Mode

Examples:

To use LCD pin L4 as COM2, make the following configuration:

```
LCDPCTL0 = BIT4;   // configure I/O pad as LCD pin
LCDCSSEL0 = BIT4;  // configure LCD pin L4 as COM
LCDM2 = BIT2;      // define L4 as COM2
```

To use LCD pin L23 as COM0, make the following configuration:

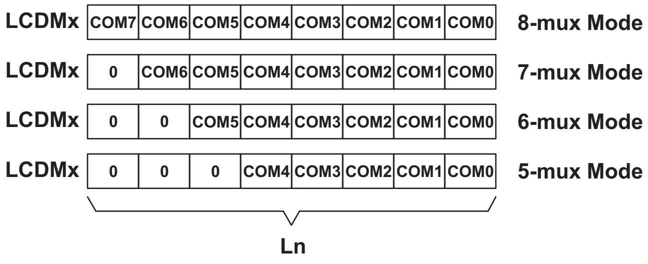
```
LCDPCTL1 = BIT7;   // configure I/O pad as LCD pin
LCDCSSEL1 = BIT7;  // configure LCD pin L23 as COM
LCDM11 = BIT4;     // define L23 as COM0
```

15.2.3.2.2 COM Assignment in 5-, 6-, 7-, or 8-Mux Mode

In 5-, 6-, 7- and 8-mux mode, each LCDMx is used to control the common functionality of one LCD pin.

To define a LCD pin as LCD common, the corresponding bit in LCDCSSELx register must be set to 1. In 5-, 6-, 7- and 8-mux mode, each LCDMx register controls the common functionality of one LCD pin. Selecting two or more COM pins per LCD pin can lead to unpredicted behavior of the LCD and must be avoided.

NOTE: See the device-specific data sheet to determine whether or not 5-, 6-, 7-, or 8-mux mode is available on a device.



LCDMx	COM7	COM6	COM5	COM4	COM3	COM2	COM1	COM0	8-mux Mode
LCDMx	0	COM6	COM5	COM4	COM3	COM2	COM1	COM0	7-mux Mode
LCDMx	0	0	COM5	COM4	COM3	COM2	COM1	COM0	6-mux Mode
LCDMx	0	0	0	COM4	COM3	COM2	COM1	COM0	5-mux Mode
	Ln								

SEG_{MAX} = Number of Segment Pins

$n = 0 \ldots SEG_{MAX} - 1$

x = n

Figure 15-5. LCDMx in 5-, 6-, 7-, or 8-Mux Mode

Examples:

To use LCD pin L4 as COM6, make the following configuration:

```
LCDPCTL0 |= BIT4;   // configure I/O pad as LCD pin
LCDCSSEL0 |= BIT4;   // configure LCD pin L4 as COM
LCDM4 = BIT6;       // define L4 as COM6
```

To use LCD pin L23 as COM5, make the following configuration:

```
LCDPCTL1 = BIT7;    // configure I/O pad as LCD pin
LCDCSSEL1 = BIT7;    // configure LCD pin L23 as COM
LCDM23 = BIT5;      // define L23 as COM5
```

15.2.4 LCD Timing Generation

The LCD_E controller uses the f_{LCD} signal from the integrated clock divider to generate the timing for common and segment lines. The LCDSSEL bit sets the source frequency f_{SOURCE} to ACLK (30 kHz to 40 kHz), XT1CLK (32.768 kHz), or VLOCLK (≈ 10 kHz). The f_{LCD} frequency is selected with the LCDDIVx and LCDMXx bits, and depends on the selected mux mode. The divider corresponding to the mux-mode is listed in Table 15-2.

Table 15-2. Divider depending on MUX-Mode

MUX Mode	MUXDIVIDER
1 (Static)	64
2	32
3	16
4	16
5	12
6	8
7	8
8	8

The resulting f_{LCD} frequency is calculated by:

$$f_{LCD} = \frac{f_{SOURCE}}{(LCDDIVx + 1) \times MUXDIVDER}$$

EXAMPLE 1:

The proper f_{LCD} frequency depends on the LCD's requirement for framing frequency and the LCD multiplex rate. To avoid ghosting effects on the LCD, f_{LCD} should be in the range of approximately 30 Hz to 60 Hz. It is calculated by:

$f_{LCD} = 2 \times mux \times f_{FRAME}$

For example, to calculate f_{LCD} for a 3-mux LCD with a frame frequency of 25 Hz to 80 Hz:

f_{FRAME} (from LCD data sheet) = 25 Hz to 80 Hz

$f_{LCD} = 2 \times 3 \times f_{FRAME}$

f_{LCD}(min) = 150 Hz

f_{LCD}(max) = 480 Hz

With f_{SOURCE} = 32768 Hz, LCDDIVx = 01101, and LCDMXx = 010:

f_{LCD} = 32768 Hz / ((13+1) × 16) = 32768 Hz / 224 = 146 Hz

With LCDDIVx = 00100 and LCDMXx = 010:

f_{LCD} = 32768 Hz / ((4+1) × 16) = 32768 Hz / 56 = 409 Hz

The lowest frequency has the lowest current consumption. The highest frequency has the least flicker.

EXAMPLE 2:

Table 15-3 shows the possible f_{LCD}, f_{FRAME}, and f_{BLINK} frequencies for a given f_{SOURCE} = 32.768 kHz depending on the selected mux mode.

Table 15-3. Example for Possible LCD Frequencies

f_{SOURCE} (Hz)	Mux Mode	LCDDIVx[1]	f_{DIV} (Hz)	f_{LCD} (Hz)	f_{FRAME} (Hz)	f_{BLINK} (Hz)
32768	Static	4-16	8192 ... 2048	(8192 ... 2048) / 64 = (128 ... 32)	(128 ... 32) / 2 / 1 = (64 ... 16)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	2	4-16	8192 ... 2048	(8192 ... 2048) / 32 = (256 ... 64)	(256 ... 64) / 2 / 2 = (64 ... 16)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	3	4-16	8192 ... 2048	(8192 ... 2048) / 16 = (512 ... 128)	(512 ... 128) / 2 / 3 = (85 ... 21)	f_{LCD} / ((LCDMx + 1) × 2 ^ (LCDBLKPREx + 2))
32768	4	4-16	8192 ... 2048	(8192 ... 2048) / 16 = (512 ... 128)	(512 ... 128) / 2 / 4 = (64 ... 16)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	5	4-16	8192 ... 2048	(8192 ... 2048) / 12 = (683 ... 171)	(682 ... 172) / 2 / 5 = (68 ... 17)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	6	4-16	8192 ... 2048	(8192 ... 2048) / 8 = (1024 ... 256)	(1024 ... 256) / 2 / 6 = (85 ... 21)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	7	4-16	8192 ... 2048	(8192 ... 2048) / 8 = (1024 ... 256)	(1024 ... 256) / 2 / 7 = (73 ... 18)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	8	4-16	8192 ... 2048	(8192 ... 2048) / 8 = (1024 ... 256)	(1024 ... 256) / 2 / 8 = (64 ... 16)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)

[1] LCDDIVx < 4 is not recommended, as it would result in higher frequencies for f_{LCD} , f_{FRAME}, and f_{BLINK}

15.2.5 Blanking the LCD

The LCD controller allows blanking the complete LCD. The LCDSON bit is ANDed with each segment's memory bit. When LCDSON = 1, each segment is on or off according to its bit value. When LCDSON = 0, each LCD segment is off.

15.2.6 LCD Blinking

The LCD controller also supports blinking. In static and 2-mux to 4-mux mode, the blinking mode LCDBLKMODx = 01 allows blinking of individual segments; with LCDBLKMODx = 10 all segments are blinking; and with LCDBLKMODx = 00 blinking is disabled. In 5-mux mode and above, only blinking mode LCDBLKMODx = 10 that allows blinking of all segments is available; if another mode is selected, blinking is disabled.

15.2.6.1 Blinking Memory

In static and 2-mux to 4-mux mode, a separate blinking memory is implemented to select the blinking segments. To enable individual segments for blinking, the corresponding bit in the blinking memory LCDBMx registers must be set. The memory uses the same structure as the LCD memory shown in Figure 15-2. Each memory bit corresponds to one LCD segment or is not used, depending on the multiplexing mode LCDMXx. To enable blinking for a LCD segment, its corresponding memory bit is set.

The blinking memory can also be accessed word-wise using the even addresses starting at LCDBM0W, LCDBM2W, and so on.

Setting the bit LCDCLRBM clears all blinking memory registers at the next frame boundary. It is automatically reset after the registers are cleared.

15.2.6.2 COM Configuration in Blinking Mode

Special care must be taken, if LCD segments are configured for blinking. As in Section 15.2.3.2 described, a part of the display memory LCDMx is used for COM configuration. It depends on selected blinking mode LCDBLKMODx, how display memory LCDMx and blinking memory LCDBMx have to be configured. See Table 15-4 for details.

Table 15-4. Overview on COM Configuration in Blinking Mode

Blinking Mode LCDBLKMOXx	Description
00b	Blinking disabled, the user can select which memory to be displayed by setting LCDDISP bit in LCDMEMCTL register **LCDMx**: the COM related configuration bits should be set accordingly **LCDBMx**: the COM related configuration bits should be set according to LCDMx configuration
01b	Blinking of individual segments as enabled in blinking memory register LCDBMx **LCDMx**: the COM related memory bits should be set accordingly **LCDBMx**: the COM related memory bits should be set to 0
10b	Blinking of all segments **LCDMx**: the COM related memory bits should be set accordingly **LCDBMx**: this memory is not used in this blinking mode, no programming of LCDBMx necessary
11b	Switching between display contents as stored in LCDMx and LCDBMx memory registers **LCDMx**: the COM related memory bits should be set accordingly **LCDBMx**: the COM related memory bits should be set according to LCDMx configuration

By saying LCDBMx must be configured according to LCDMx it means that the same memory number "x" must be used. For example if LCDM2 = 02h (LCD pin L2 = COM1), then LCDBM2 has also to be programmed to 02h.

Example:

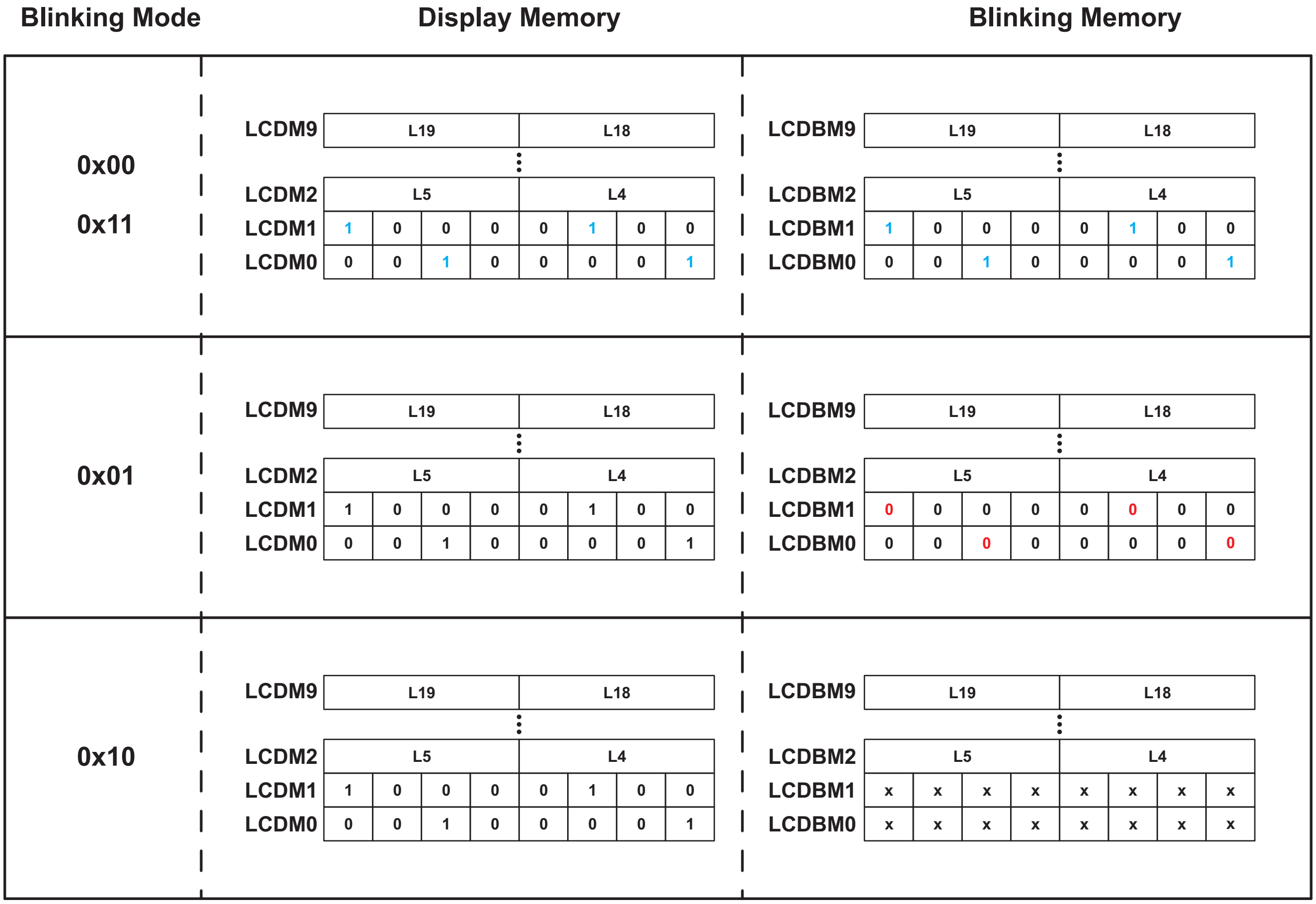
LCD configured in 4-MUX mode, 20 LCD pins, 4 configured as common, 16 configured as segment

L0 = COM0, L1 = COM1, L2 = COM2, L3 = COM3; L4 ... L19 = SEG0 ... SEG19

The following configuration must be done:

```
LCDPCTL0 = 0xFFFF;  // configure I/O pad of L0 to L15 as LCD pin
LCDPCTL1 = 0x000F;  // configure I/O pad of L16 to L19 as LCD pin
LCDCSSEL0 = 0x000F; // configure LCD pin L0-L3 as common
```

Figure 15-6 shows how to configure the display memory LCDMx and the blinking memory LCDBMx during different blinking modes.



	Display Memory								Blinking Memory								
0x00 0x11	LCDM9	L19				L18			LCDBM9	L19				L18			
	LCDM2	L5				L4			LCDBM2	L5				L4			
	LCDM1	1	0	0	0	0	1	0	0	LCDBM1	1	0	0	0	0	1	0	0
	LCDM0	0	0	1	0	0	0	0	1	LCDBM0	0	0	1	0	0	0	0	1
0x01	LCDM9	L19				L18			LCDBM9	L19				L18			
	LCDM2	L5				L4			LCDBM2	L5				L4			
	LCDM1	1	0	0	0	0	1	0	0	LCDBM1	0	0	0	0	0	0	0	0
	LCDM0	0	0	1	0	0	0	0	1	LCDBM0	0	0	0	0	0	0	0	0
0x10	LCDM9	L19				L18			LCDBM9	L19				L18			
	LCDM2	L5				L4			LCDBM2	L5				L4			
	LCDM1	1	0	0	0	0	1	0	0	LCDBM1	x	x	x	x	x	x	x	x
	LCDM0	0	0	1	0	0	0	0	1	LCDBM0	x	x	x	x	x	x	x	x

Note: x = don't care

Figure 15-6. Example LCDMx and LCDBMx Configuration in Different Blinking Modes

15.2.6.3 Blinking Frequency

The blinking frequency f_{BLINK} is selected with the CDBLKDIVx and LCDMXx bits, thus depending on selected mux-mode. The resulting f_{BLINK} frequency is calculated by:

$$f_{BLINK} = \frac{f_{LCD}}{(LCDMXx + 1) \times 2^{LCDBLKPREx+2}}$$

The divider generating the blinking frequency f_{BLINK} is reset when LCDBLKMODx = 00. After a blinking mode LCDBLKMODx = 01 or 10 is selected, the enabled segments or all segments go blank at the next frame boundary and stay off for half of a BLKCLK period. Then they go active at the next frame boundary and stay on for another half BLKCLK period before they go blank again at a frame boundary.

NOTE: Blinking Frequency Restrictions

The blinking frequency must be smaller than the frame frequency f_{FRAME}.

The blinking frequency should only be changed when LCDBLKMODx = 00.

15.2.6.4 Dual Display Memory

In static- to 4-mux mode, the blinking memory LCDBMx can also be used as a secondary display memory when no blinking mode (LCDBLKMODx = 01 or 10) is selected.

With LCDBLKMODx = 00 the LCDDISP bit can be used to manually select the memory to be displayed. With LCDDISP = 0, the LCD memory LCDMx is selected, and with LCDDISP = 1 the blinking memory LCDBMx is selected as display memory. Switching between the memories is synchronized to the frame boundaries.

With LCDBLKMODx = 11 the LCD controller switches automatically between the memories using the divider to generate the blinking frequency. The LCDDISP bit can be used as status bit, indicating the selected memory. After LCDBLKMODx = 11 is selected, the memory to be displayed for the first half a BLKCLK period is the LCD memory. In the second half, the blinking memory is used as display memory. Switching between the memories is synchronized to the frame boundaries.

15.2.7 LCD Voltage and Bias Generation

The LCD_E module allows selectable sources for the peak output waveform voltage, V1, as well as the fractional LCD biasing voltages V2, V4, and V5. V_{LCD} may be sourced from V_{CC}, an internal charge pump, or externally.

15.2.7.1 LCD Voltage Selection

V_{LCD} is sourced from V_{CC} when LCDSELVDD = 1 and LCDREFEN = 0. V_{LCD} is sourced from the internal charge pump when LCDSELVDD = 0 and LCDCPEN = 1. The internal charge pump either sourced by V_{EXT} or V_{DD} through R33 or from external reference voltage $V_{REF,EXT}$ or internal reference voltage through R13. The VLCDx bits provide a software selectable LCD voltage from 2.6 V to 3.5 V (typical) independent of V_{DD}. See the device-specific data sheet for specifications.

When the internal charge pump is used, a 100-nF or larger capacitor must be connected between the LCDCAP0 and LCDCAP1 pins. The charge pump may be temporarily disabled by setting LCDCPEN = 0 with VLCDx > 0 to reduce system noise. It can be automatically disabled during certain periods by setting the corresponding bits in the LCDVCTL register. In this case, the voltage present at the external capacitor is used for the LCD voltages until the charge pump is re-enabled.

NOTE: Capacitor Required For Internal Charge Pump

A 100-nF or larger capacitor must be connected from the LCDCAP0 and LCDCAP1 pins when the internal charge pump is enabled.

15.2.7.2 LCD Bias Generation

The fractional LCD biasing voltages, V2 and V4 can be generated internally or externally, independent of the source for V_{LCD}. V5 is always connected to ground. The bias generation block diagram for LCD_E static and 2-mux to 8-mux modes is shown in Figure 15-7.

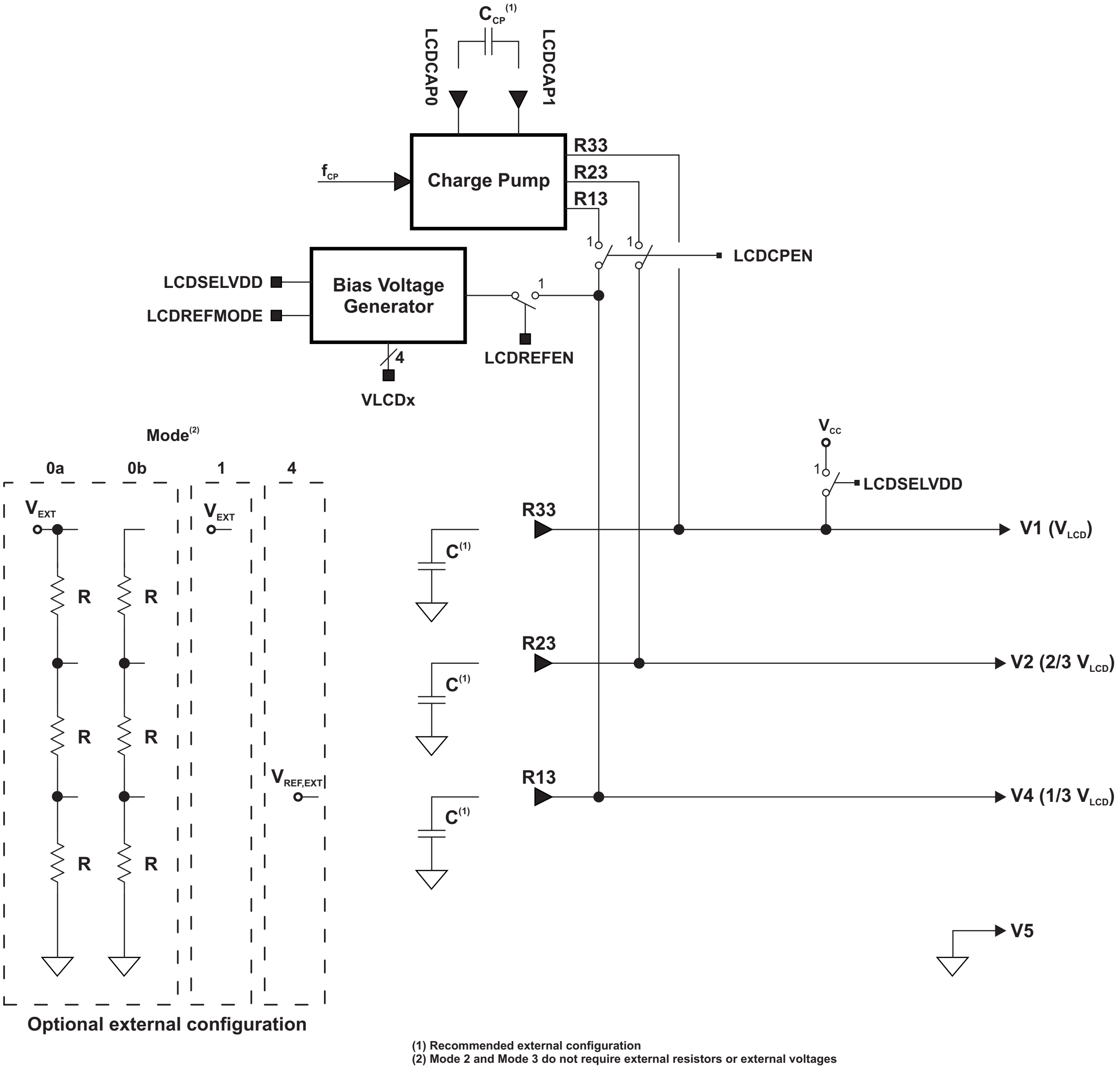


Figure 15-7. Bias Generation

The bias voltages V1, V2 and V4 are available on pin R33, R23 and R13. To source the bias voltages V1, V2 and V4 externally, an equally weighted resistor divider is used with resistors ranging from a few k? to 1 M?, depending on the size of the display. When using the internal charge pump it is possible to derive the bias voltages V1, V2 and V4 from several sources. It is possible to connect either an external voltage V_{EXT} or internally V_{DD} to R33 to generate V2 and V4. See section Section 15.2.8.1 (Mode 1 and Mode 2). The third possibility is to source R13 either externally or internally. See section Section 15.2.8.2 (Mode 3) and Section 15.2.8.3 (Mode 4).

15.2.7.3 LCD Contrast Control

The peak voltage of the output waveforms together with the selected mode and biasing determine the contrast and the contrast ratio of the LCD. The LCD contrast can be controlled in software by adjusting the LCD voltage generated by the integrated charge pump using the VLCDx settings.

The contrast ratio depends on the used LCD display. Table 15-5 shows the biasing configurations that apply to the different modes together with the RMS voltages for the segments turned on ($V_{RMS,ON}$) and turned off ($V_{RMS,OFF}$) as functions of V_{LCD}. It also shows the resulting contrast ratios between the on and off states.

Table 15-5. LCD Voltage and Biasing Characteristics

Mode	Bias Config	LCDMXx	COM Lines	Voltage Levels	$V_{RMS,OFF}$/ V_{LCD}	$V_{RMS,ON}$/ V_{LCD}	Contrast Ratio $V_{RMS,ON}$/ $V_{RMS,OFF}$
Static	Static	0000	1	V1, V5	0	1	1/0
2-mux	1/3	0001	2	V1, V2, V4, V5	0.333	0.745	2.236
3-mux	1/3	0010	3	V1, V2, V4, V5	0.333	0.638	1.915
4-mux	1/3	0011	4	V1, V2, V4, V5	0.333	0.577	1.732
5-mux	1/3	0100	5	V1, V2, V4, V5	0.333	0.537	1.612
6-mux	1/3	0101	6	V1, V2, V4, V5	0.333	0.509	1.528
7-mux	1/3	0110	7	V1, V2, V4, V5	0.333	0.488	1.464
8-mux	1/3	0111	8	V1, V2, V4, V5	0.333	0.471	1.414

A typical approach to determine the required V_{LCD} is by equating $V_{RMS,OFF}$ with a defined LCD threshold voltage, typically when the LCD exhibits approximately 10% contrast ($V_{th,10\%}$): $V_{RMS,OFF} = V_{th,10\%}$. Using the values for $V_{RMS,OFF}/V_{LCD}$ provided in the table results in $V_{LCD} = V_{th,10\%}/(V_{RMS,OFF}/V_{LCD})$. In the static mode, a suitable choice is V_{LCD} greater than or equal to three times $V_{th,10\%}$.

15.2.8 LCD Operation Modes

This section describes the different modes in which the LCD can be operated.

15.2.8.1 Internal Charge Pump Enabled, Internal V_{REF} Disabled (Mode 1, Mode 2)

In Figure 15-8 Mode 1 is depicted. LCD voltages are sourced from an external voltage V_{EXT}, which is connected to pin R33. The internal charge pump is used to generate the LCD voltages V1, V2, V4 and V5. Contrast can be adjusted by changing V_{EXT}.

```
LCDSELVDD  = 0;       // Pin R33 is connected to external supply voltage
LCDCPEN    = 1;       // internal charge pump enabled
LCDREFEN   = 0;       // internal reference voltage at R13 is disabled
LCDCPFSELx = 0b1111;  // charge pump frequency select, slowest value
VLCDx      = 0b0000;  // not used, set to reset value
LCDON      = 1;       // enable LCD
```

Mode 2 is shown in Figure 15-9. R33 is connected to internal supply voltage V_{DD}. The internal charge pump generates the LCD voltages V1, V2, V4, and V5. Contrast can be adjusted by changing V_{DD} from 3.6 V to 1.8 V.

```
LCDSELVDD  = 1;       // Pin R33 is connected to internal supply voltage
LCDCPEN    = 1;       // internal charge pump enabled
LCDREFEN   = 0;       // internal reference voltage at R13 is disabled
LCDCPFSELx = 0b1111;  // charge pump frequency select, slowest value
VLCDx      = 0b0000;  // not used, set to reset value
LCDON      = 1;       // enable LCD
```

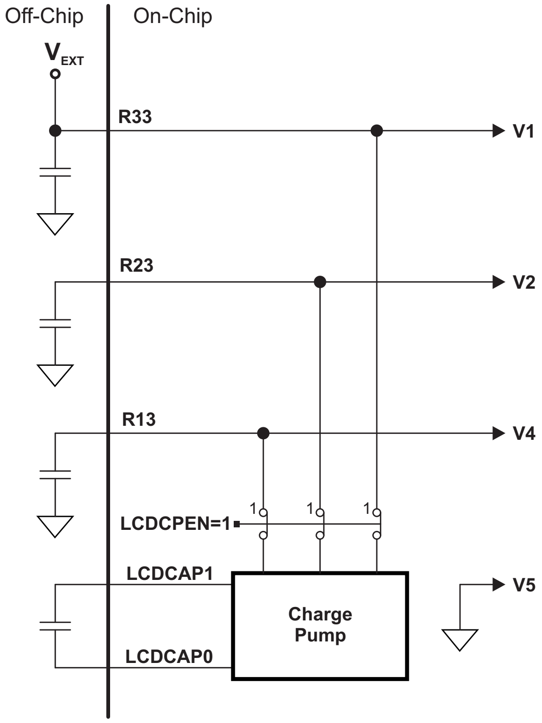


Figure 15-8. LCD Operation Mode 1

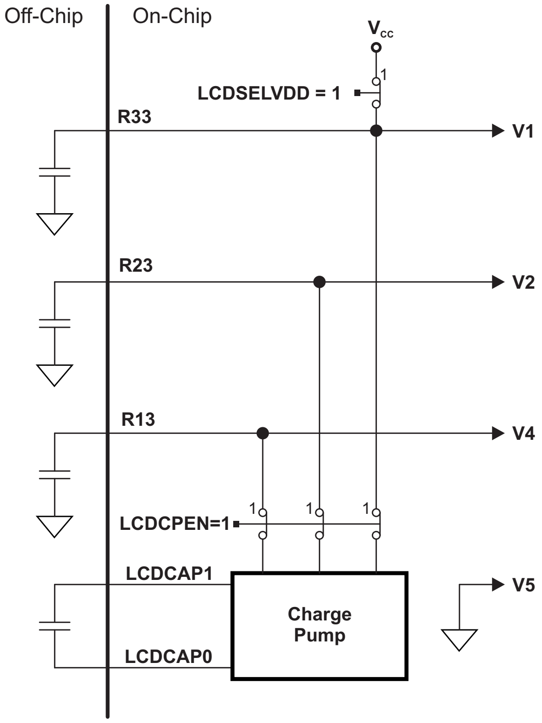


Figure 15-9. LCD Operation Mode 2

15.2.8.2 Internal Charge Pump Enabled, Internal V_{REF} Enabled (Mode 3)

In Figure 15-10 Mode 3 is depicted. LCD voltages are derived from Bias Voltage Generator, which is connected to pin R13. The internal charge pump is used to generate the LCD voltages V1, V2, V5 is connected to ground. Contrast can be adjusted in software by changing VLCDx bits in LCDVCTL register. By setting LCDREFMODE = 1, the bias voltage generator is in switch mode. Thus the bias voltage generator is on for 1 clock cycle and off for another 256 clock cycles to save power. Setting LCDREFMODE = 0 sets the bias generator to static mode to be able to drive larger LCD panels.

```
LCDSELVDD  = 0;       // Pin R33 is connected to external supply voltage
LCDCPEN    = 1;       // internal charge pump enabled
LCDREFEN   = 1;       // internal reference voltage at R13 is enabled
LCDCPFSELx = 0b1111;  // charge pump frequency select, slowest value
VLCDx      = 0b1000;  // VLCDx set to mid position
LCDON      = 1;       // enable LCD
```

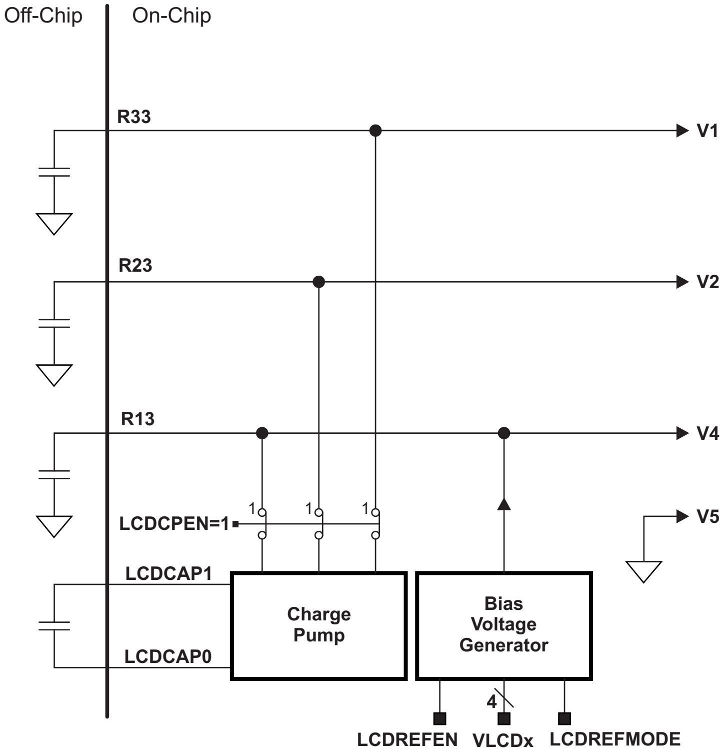


Figure 15-10. LCD Operation Mode 3

NOTE: Mode 3 is the recommended operating mode, as this provides the lowest external component cost and very low operating currents.

15.2.8.3 Internal Charge Pump Enabled, Internal V_{REF} Disabled (Mode 4)

Figure 15-11 shows Mode 4. LCD voltages are derived from external reference voltage, which is connected to pin R13. The internal charge pump is used to generate the LCD biasing voltages V1, V2. V5 is connected to ground. Contrast can be adjusted by changing external voltage $V_{REF,EXT}$ from 0.8 V to 1.2 V.

```
LCDSELVDD  = 0;       // Pin R33 is connected to external supply voltage
LCDCPEN    = 1;       // internal charge pump enabled
LCDREFEN   = 0;       // internal reference voltage at R13 is disabled
LCDCPFSELx = 0b1111;  // charge pump frequency select, slowest value
VLCDx      = 0b0000;  // not used, set to reset value
LCDON      = 1;       // enable LCD
```

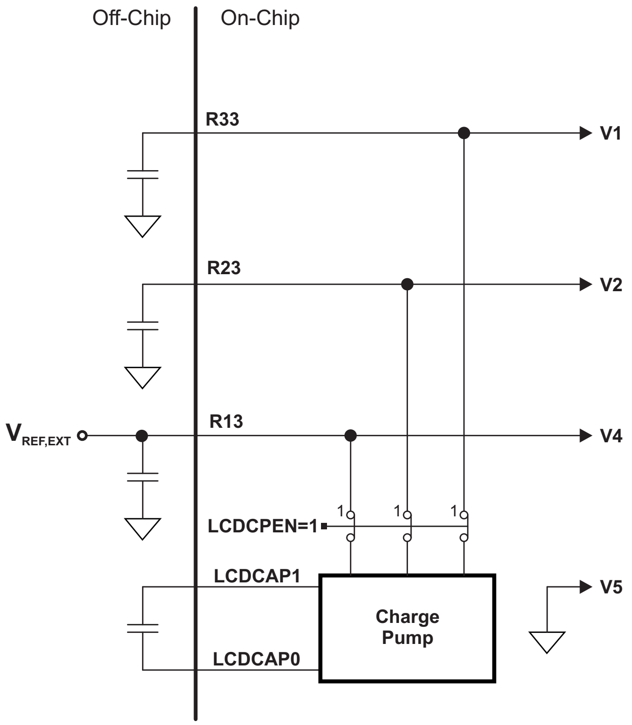


Figure 15-11. LCD Operation Mode 4

15.2.9 LCD Interrupts

The LCD_E module has three interrupt sources available, each with independent enables and flags.

The three interrupt flags, namely LCDFRMIFG, LCDBLKOFFIFG and LCDBLKONIFG are prioritized and combined to source a single interrupt vector. The interrupt vector register LCDIV is used to determine which flag requested an interrupt.

The highest priority enabled interrupt generates a number in the LCDIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled LCD interrupts do not affect the LCDIV value.

Any read access of the LCDIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. A write access to the LCDIV register automatically resets all pending interrupt flags. In addition, all flags can be cleared by software.

The LCDBLKONIFG is set at the BLKCLK rising edge and LCD switches to blinking status when blinking is enabled with LCDBLKMODx = 01 or 10. It is also set at the BLKCLK edge that selects the blinking memory as display memory when LCDBLKMODx = 11. It is automatically cleared when a LCD or blinking memory register is written. Setting the LCDBLKONIE bit enables the interrupt.

The LCDBLKOFFIFG is set at the BLKCLK falling edge and LCD switches to non-blinking status when blinking is enabled with LCDBLKMODx = 01 or 10. It is also set at the BLKCLK edge that selects the LCD memory as display memory when LCDBLKMODx = 11. It is automatically cleared when a LCD or blinking memory register is written. Setting the LCDBLKOFFIE bit enables the interrupt.

The LCDFRMIFG is set at a frame boundary. It is automatically cleared when a LCD or blinking memory register is written. Setting the LCDFRMIFGIE bit enables the interrupt.

15.2.9.1 LCDIV Software Example

The following software example shows the recommended use of LCDIV and the handling overhead. The LCDIV 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 LCD_E interrupt flags.
LCDB_HND             ; Interrupt latency         6
   ADD &LCDBIV,PC    ; Add offset to Jump table   3
   RETI              ; Vector 0: No interrupt     5
   JMP LCDBLKON_HND  ; Vector 4: LCDBLKONIFG      2
   JMP LCDBLKOFF_HND ; Vector 6: LCDBLKOFFIFG     2
LCDFRM_HND           ; Vector 8: LCDFRMIFG
   ...    ; Task starts here
   RETI                                          5
LCDBLKON_HND ; Vector 4: LCDBLKONIFG
   ... ; Task starts here
   RETI ; Back to main program                   5
LCDBLKOFF_HND ; Vector 6: LCDBLKOFFIFG
   ... ; Task starts here
   RETI ; Back to main program                   5
```

15.2.10 Static Mode

In static mode, each MSP430 segment pin drives one LCD segment, and one common line (COM0) is used. Figure 15-12 shows some example static waveforms.

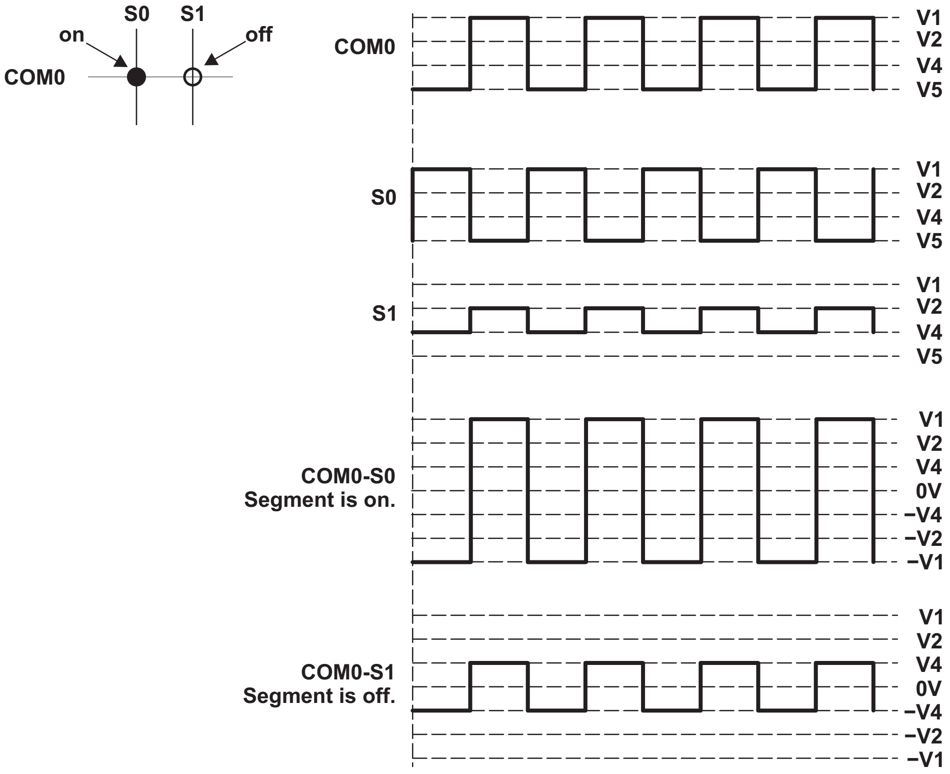


Figure 15-12. Example Static Waveforms

15.2.11 2-Mux Mode

In 2-mux mode, each MSP430 segment pin drives two LCD segments, and two common lines (COM0 and COM1) are used. Figure 15-13 shows some example 2-mux 1/3-bias waveforms.

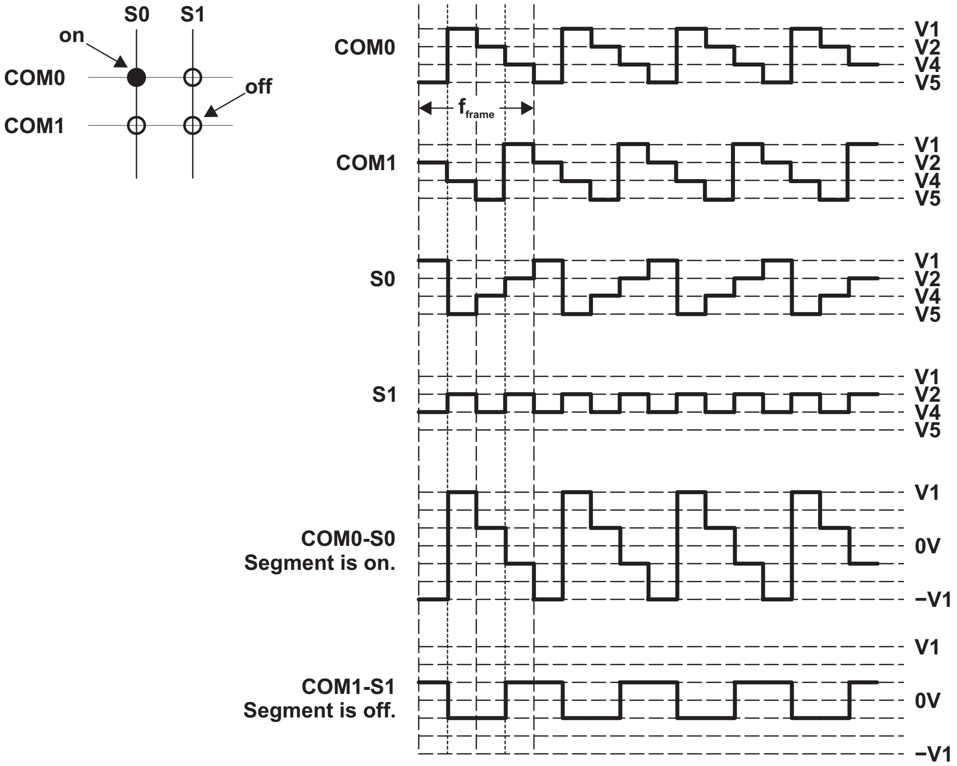


Figure 15-13. Example 2-Mux Waveforms

15.2.12 3-Mux Mode

In 3-mux mode, each MSP430 segment pin drives three LCD segments, and three common lines (COM0, COM1, and COM2) are used. Figure 15-14 shows some example 3-mux 1/3-bias waveforms.

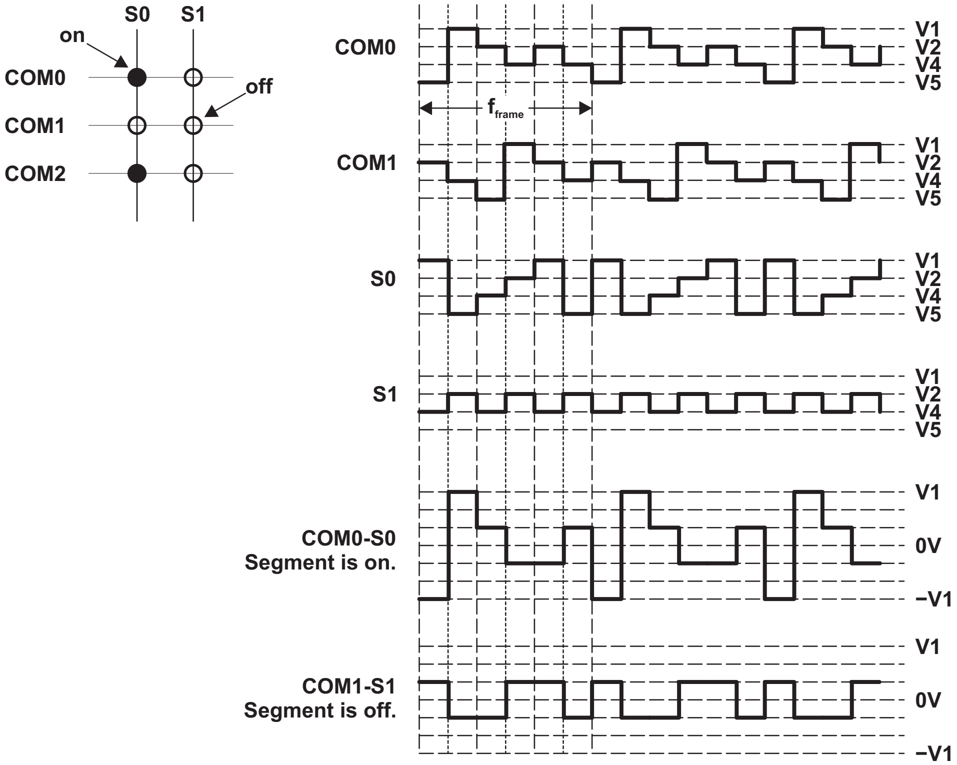


Figure 15-14. Example 3-Mux Waveforms

15.2.13 4-Mux Mode

In 4-mux mode, each MSP430 segment pin drives four LCD segments and four common lines (COM0, COM1, COM2, and COM3) are used. Figure 15-15 shows some example 4-mux 1/3-bias waveforms.

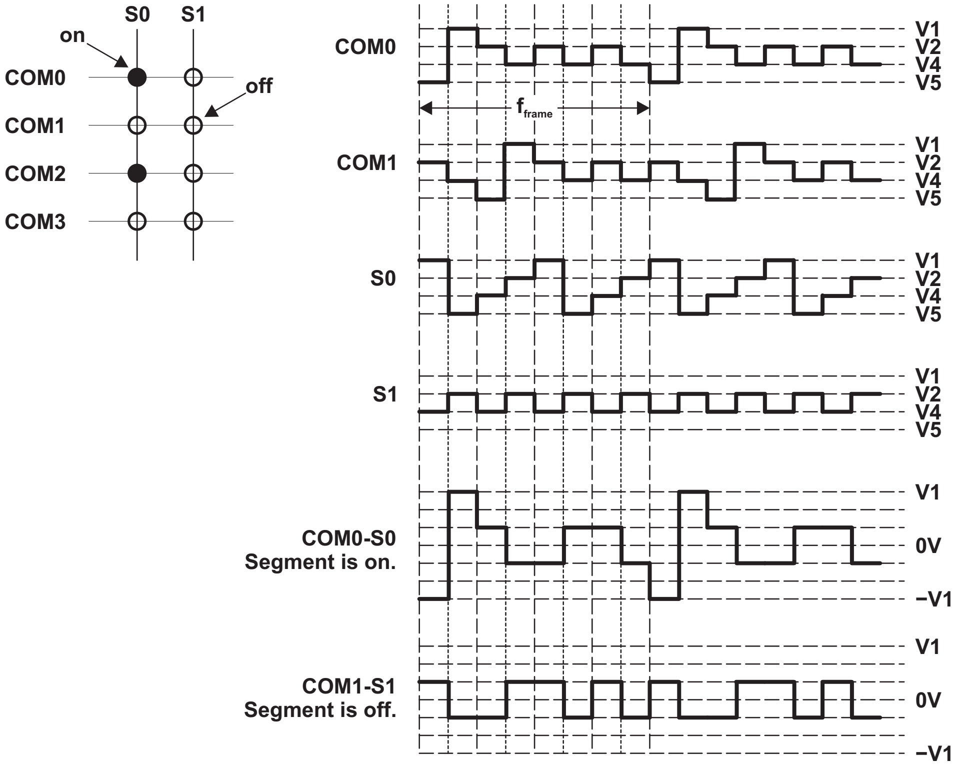


Figure 15-15. Example 4-Mux Waveforms

15.2.14 6-Mux Mode

In 6-mux mode, each MSP430 segment pin drives six LCD segments, and six common lines (COM0, COM1, COM2, COM3, COM4, and COM5) are used. Figure 15-16 shows some example 6-mux 1/3-bias waveforms.

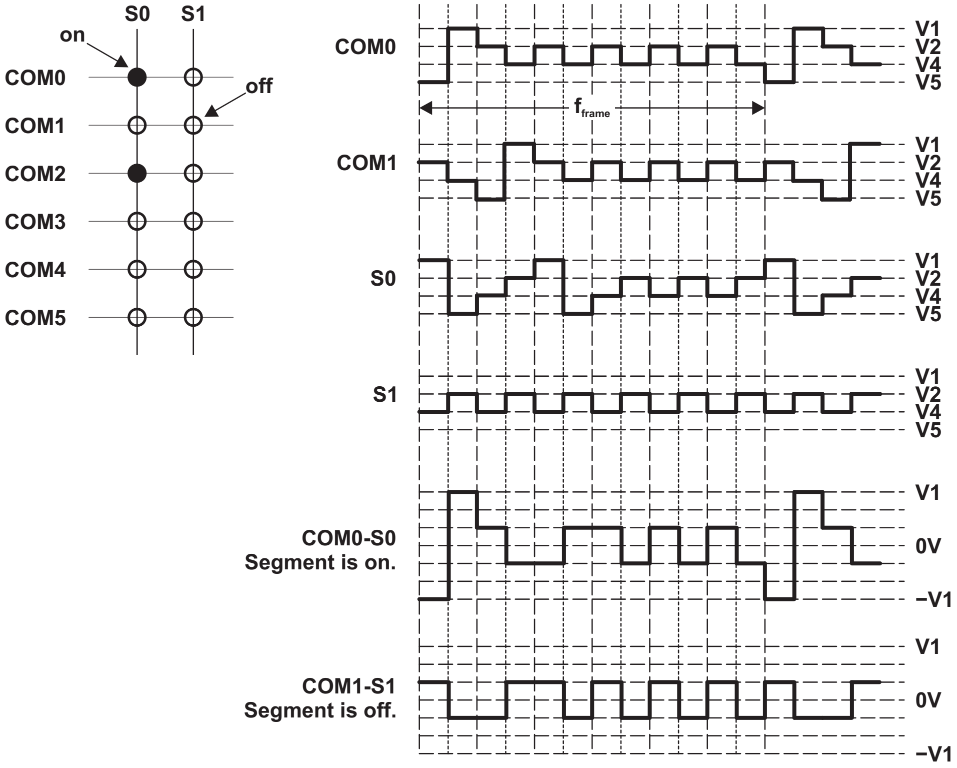


Figure 15-16. Example 6-Mux Waveforms

15.2.15 8-Mux Mode

In 8-mux mode, each MSP430 segment pin drives eight LCD segments, and eight common lines (COM0 through COM7) are used. Figure 15-17 shows some example 8-mux 1/3-bias waveforms.

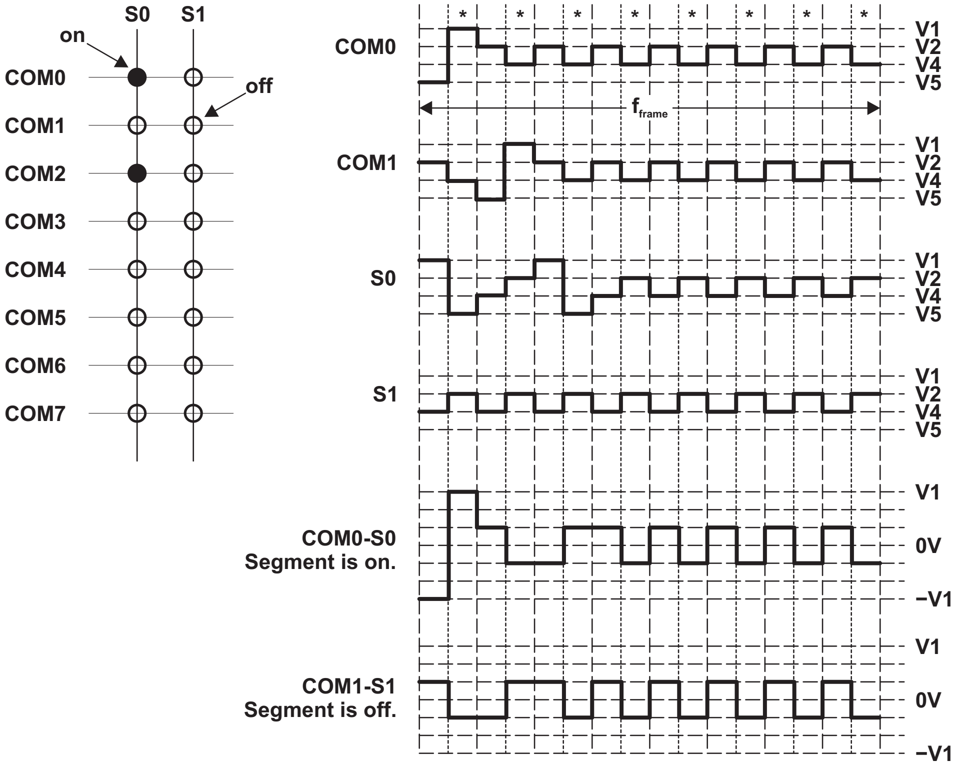


Figure 15-17. Example 8-Mux, 1/3 Bias Waveforms (LCDLP = 0)

Figure 15-18 shows some example 8-mux 1/3-bias waveforms with LCDLP = 1. With LCDLP = 1, the voltage sequence compared to the non-low power waveform is reshuffled; that is, all of the timeslots marked with "*" in Figure 15-17 are grouped together. The same principle applies to all mux modes.

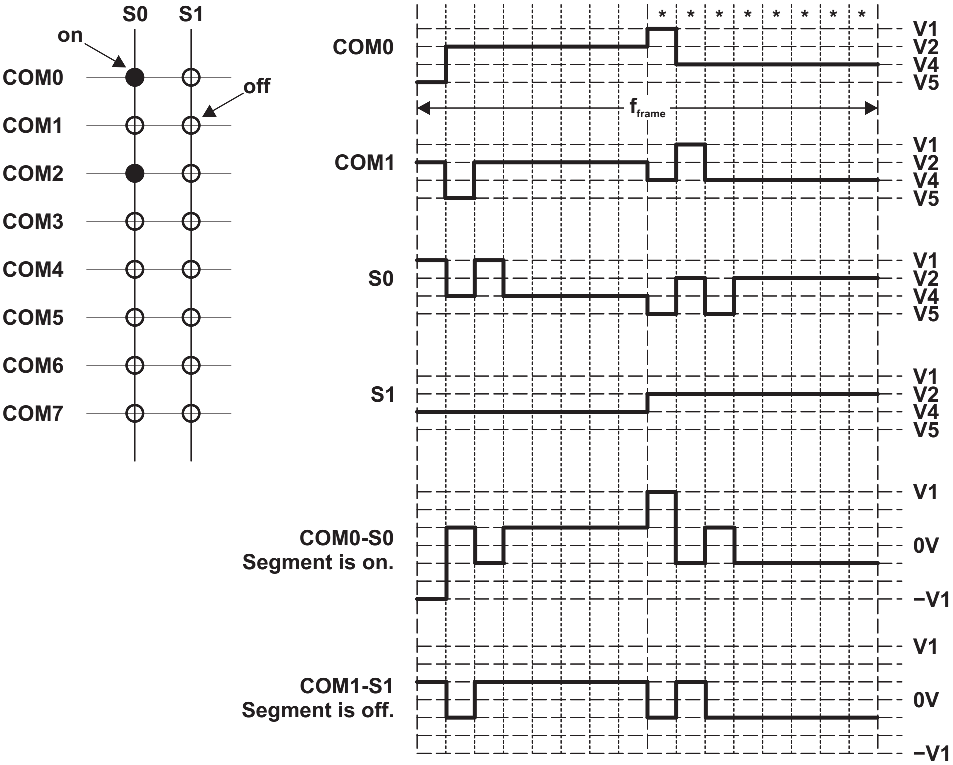


Figure 15-18. Example 8-Mux, 1/3 Bias Low-Power Waveforms (LCDLP = 1)

15.3 LCD_E Registers

The LCD_E controller registers are listed in Table 15-6 to Table 15-9. The LCD memory and blinking memory registers can also be accessed as word.

The number of available memory registers on a given device depends on the number of available segment pins (see the device-specific data sheet).

Table 15-6. LCD_E Registers

Offset	Acronym	Register Name	Type	Reset	Section
000h	LCDCTL0	LCD_E control register 0	Read/write	3800h	Section 15.3.1
002h	LCDCTL1	LCD_E control register 1	Read/write	0000h	Section 15.3.2
004h	LCDBLKCTL	LCD_E blinking control register	Read/write	0000h	Section 15.3.3
006h	LCDMEMCTL	LCD_E memory control register	Read/write	0000h	Section 15.3.4
008h	LCDVCTL	LCD_E voltage control register	Read/write	0000h	Section 15.3.5
00Ah	LCDPCTL0	LCD_E port control 0	Read/write	0000h	Section 15.3.6
00Ch	LCDPCTL1	LCD_E port control 1	Read/write	0000h	Section 15.3.7
00Eh	LCDPCTL2	LCD_E port control 2 (=256 segments)	Read/write	0000h	Section 15.3.8
010h	LCDPCTL3	LCD_E port control 3 (384 segments)	Read/write	0000h	Section 15.3.9
012h		Reserved	Read/write	0000h	
014h	LCDCSSEL0	LCD_E COM/SEG select register 0	Read/write	0000h	Section 15.3.10
016h	LCDCSSEL1	LCD_E COM/SEG select register 1	Read/write	0000h	Section 15.3.11
018h	LCDCSSEL2	LCD_E COM/SEG select register 2	Read/write	0000h	Section 15.3.12
01Ah	LCDCSSEL3	LCD_E COM/SEG select register 3	Read/write	0000h	Section 15.3.13
01Ch		Reserved	Read/write	0000h	
01Eh	LCDIV	LCD_E interrupt vector	Read only	0000h	Section 15.3.16

Table 15-7. LCD Memory Registers for Static and 2-Mux to 4-Mux Modes[1] [2]

Offset	Acronym	Register Name	Type	Reset
020h	LCDM0W	LCD memory 0 Word (S3, S2, S1, S0)	Read/write	Unchanged
020h	LCDM0	LCD memory 0 (S1, S0)	Read/write	Unchanged
021h	LCDM1	LCD memory 1 (S3, S2)	Read/write	Unchanged
022h	LCD2W	LCD memory 2 Word (S7, S6, S5, S4)	Read/write	Unchanged
022h	LCDM2	LCD memory 2 (S5, S4)	Read/write	Unchanged
023h	LCDM3	LCD memory 3 (S7, S6)	Read/write	Unchanged
024h	LCD4W	LCD memory 4 Word (S11, S10, S9, S8)	Read/write	Unchanged
024h	LCDM4	LCD memory 4 (S9, S8)	Read/write	Unchanged
025h	LCDM5	LCD memory 5 (S11, S10)	Read/write	Unchanged
026h	LCDM6W	LCD memory 6 Word (S15, S14, S13, S12)	Read/write	Unchanged
026h	LCDM6	LCD memory 6 (S13, S12)	Read/write	Unchanged
027h	LCDM7	LCD memory 7 (S15, S14)	Read/write	Unchanged
028h	LCDM8W	LCD memory 8 Word (S19, S18, S17, S16)	Read/write	Unchanged
028h	LCDM8	LCD memory 8 (S17, S16)	Read/write	Unchanged
029h	LCDM9	LCD memory 9 (S19, S18)	Read/write	Unchanged
02Ah	LCDM10W	LCD memory 10 Word (S23, S22, S21, S20)	Read/write	Unchanged
02Ah	LCDM10	LCD memory 10 (S21, S20)	Read/write	Unchanged
02Bh	LCDM11	LCD memory 11 (S23, S22)	Read/write	Unchanged
02Ch	LCDM12W	LCD memory 12 Word (S27, S26, S25, S24)	Read/write	Unchanged
02Ch	LCDM12	LCD memory 12 (S25, S24)	Read/write	Unchanged
02Dh	LCDM13	LCD memory 13 (S27, S26)	Read/write	Unchanged
02Eh	LCDM14W	LCD memory 14 Word (S31, S30, S29, S28)	Read/write	Unchanged
02Eh	LCDM14	LCD memory 14 (S29, S28)	Read/write	Unchanged
02Fh	LCDM15	LCD memory 15 (S31, S30)	Read/write	Unchanged
030h	LCDM16W	LCD memory 16 Word (S35, S34, S33, S32)	Read/write	Unchanged
030h	LCDM16	LCD memory 16 (S33, S32)	Read/write	Unchanged
031h	LCDM17	LCD memory 17 (S35, S34)	Read/write	Unchanged
032h	LCDM18W	LCD memory 18 Word (S39, S38, S37, S36)	Read/write	Unchanged
032h	LCDM18	LCD memory 18 (S37, S36)	Read/write	Unchanged
033h	LCDM19	LCD memory 19 (S39, S38)	Read/write	Unchanged
034h	LCDM20W	LCD memory 20 Word (S43, S42, S41, S40)	Read/write	Unchanged
034h	LCDM20	LCD memory 20 (S41, S40)	Read/write	Unchanged
035h	LCDM21	LCD memory 21 (S43, S42)	Read/write	Unchanged
036h	LCDM22W	LCD memory 22 Word (S47, S46, S45, S44)	Read/write	Unchanged
036h	LCDM22	LCD memory 22 (S45, S44)	Read/write	Unchanged
037h	LCDM23	LCD memory 23 (S47, S46)	Read/write	Unchanged
038h	LCDM24W	LCD memory 24 Word (S51, S50, S49, S48)	Read/write	Unchanged
038h	LCDM24	LCD memory 24 (S49, S48)	Read/write	Unchanged
039h	LCDM25	LCD memory 25 (S51, S50)	Read/write	Unchanged
03Ah	LCDM26W	LCD memory 26 Word (S55, S54, S53, S52)	Read/write	Unchanged
03Ah	LCDM26	LCD memory 26 (S53, S52)	Read/write	Unchanged
03Bh	LCDM27	LCD memory 27 (S55, S54)	Read/write	Unchanged
03Ch	LCDM28W	LCD memory 28 Word (S59, S58, S57, S56)	Read/write	Unchanged
03Ch	LCDM28	LCD memory 28 (S57, S56)	Read/write	Unchanged
03Dh	LCDM29	LCD memory 29 (S59, S58)	Read/write	Unchanged

[1] The LCD memory registers can also be accessed as word.

[2] The number of available memory registers on a given device depends on the amount of available segment pins. See the device-specific data sheet.

Table 15-7. LCD Memory Registers for Static and 2-Mux to 4-Mux Modes[1] [2] (continued)

Offset	Acronym	Register Name	Type	Reset
03Eh	LCDM30W	LCD memory 30 Word (S63, S62, S61, S60)	Read/write	Unchanged
03Eh	LCDM30	LCD memory 30 (S61, S60)	Read/write	Unchanged
03Fh	LCDM31	LCD memory 31 (S63, S62)	Read/write	Unchanged

Table 15-8. LCD Blinking Memory Registers for Static and 2-Mux to 4-Mux Modes[1][2]

Offset	Acronym	Register Name	Type	Reset
040h	LCDBM0W	LCD blinking memory 0 Word	Read/write	Unchanged
040h	LCDBM0	LCD blinking memory 0	Read/write	Unchanged
041h	LCDBM1	LCD blinking memory 1	Read/write	Unchanged
042h	LCDBM2W	LCD blinking memory 2 Word	Read/write	Unchanged
042h	LCDBM2	LCD blinking memory 2	Read/write	Unchanged
043h	LCDBM3	LCD blinking memory 3	Read/write	Unchanged
044h	LCDBM4W	LCD blinking memory 4 Word	Read/write	Unchanged
044h	LCDBM4	LCD blinking memory 4	Read/write	Unchanged
045h	LCDBM5	LCD blinking memory 5	Read/write	Unchanged
046h	LCDBM6W	LCD blinking memory 6 Word	Read/write	Unchanged
046h	LCDBM6	LCD blinking memory 6	Read/write	Unchanged
047h	LCDBM7	LCD blinking memory 7	Read/write	Unchanged
048h	LCDBM8W	LCD blinking memory 8 Word	Read/write	Unchanged
048h	LCDBM8	LCD blinking memory 8	Read/write	Unchanged
049h	LCDBM9	LCD blinking memory 9	Read/write	Unchanged
04Ah	LCDBM10W	LCD blinking memory 10 Word	Read/write	Unchanged
04Ah	LCDBM10	LCD blinking memory 10	Read/write	Unchanged
04Bh	LCDBM11	LCD blinking memory 11	Read/write	Unchanged
04Ch	LCDBM12W	LCD blinking memory 12 Word	Read/write	Unchanged
04Ch	LCDBM12	LCD blinking memory 12	Read/write	Unchanged
04Dh	LCDBM13	LCD blinking memory 13	Read/write	Unchanged
04Eh	LCDBM14W	LCD blinking memory 14 Word	Read/write	Unchanged
04Eh	LCDBM14	LCD blinking memory 14	Read/write	Unchanged
04Fh	LCDBM15	LCD blinking memory 15	Read/write	Unchanged
050h	LCDBM16W	LCD blinking memory 16 Word	Read/write	Unchanged
050h	LCDBM16	LCD blinking memory 16	Read/write	Unchanged
051h	LCDBM17	LCD blinking memory 17	Read/write	Unchanged
052h	LCDBM18W	LCD blinking memory 18 Word	Read/write	Unchanged
052h	LCDBM18	LCD blinking memory 18	Read/write	Unchanged
053h	LCDBM19	LCD blinking memory 19	Read/write	Unchanged
054h	LCDBM20W	LCD blinking memory 20 Wrod	Read/write	Unchanged
054h	LCDBM20	LCD blinking memory 20	Read/write	Unchanged
055h	LCDBM21	LCD blinking memory 21	Read/write	Unchanged
056h	LCDBM22W	LCD blinking memory 22 Word	Read/write	Unchanged
056h	LCDBM22	LCD blinking memory 22	Read/write	Unchanged
057h	LCDBM23	LCD blinking memory 23	Read/write	Unchanged
058h	LCDBM24W	LCD blinking memory 24 Word	Read/write	Unchanged
058h	LCDBM24	LCD blinking memory 24	Read/write	Unchanged
059h	LCDBM25	LCD blinking memory 25	Read/write	Unchanged
05Ah	LCDBM26W	LCD blinking memory 26 Word	Read/write	Unchanged
05Ah	LCDBM26	LCD blinking memory 26	Read/write	Unchanged
05Bh	LCDBM27	LCD blinking memory 27	Read/write	Unchanged
05Ch	LCDBM28W	LCD blinking memory 28 Word	Read/write	Unchanged
05Ch	LCDBM28	LCD blinking memory 28	Read/write	Unchanged
05Dh	LCDBM29	LCD blinking memory 29	Read/write	Unchanged

[1] The LCD blinking memory registers can also be accessed as word.

[2] The number of available memory registers on a given device depends on the amount of available segment pins (see the device-specific data sheet).

Table 15-8. LCD Blinking Memory Registers for Static and 2-Mux to 4-Mux Modes[1][2] (continued)

Offset	Acronym	Register Name	Type	Reset
05Eh	LCDBM30W	LCD blinking memory 30 Word	Read/write	Unchanged
05Eh	LCDBM30	LCD blinking memory 30	Read/write	Unchanged
05Fh	LCDBM31	LCD blinking memory 31	Read/write	Unchanged

Table 15-9. LCD Memory Registers for 5-Mux to 8-Mux Modes[1][2]

Offset	Acronym	Register Name	Type	Reset
020h	LCDM0W	LCD memory 0 Word (S1, S0)	Read/write	Unchanged
020h	LCDM0	LCD memory 0 (S0)	Read/write	Unchanged
021h	LCDM1	LCD memory 1 (S1)	Read/write	Unchanged
022h	LCDM2W	LCD memory 2 Word (S3, S2)	Read/write	Unchanged
022h	LCDM2	LCD memory 2 (S2)	Read/write	Unchanged
023h	LCDM3	LCD memory 3 (S3)	Read/write	Unchanged
024h	LCDM4W	LCD memory 4 Word (S5, S4)	Read/write	Unchanged
024h	LCDM4	LCD memory 4 (S4)	Read/write	Unchanged
025h	LCDM5	LCD memory 5 (S5)	Read/write	Unchanged
026h	LCDM6W	LCD memory 6 Word (S7, S6)	Read/write	Unchanged
026h	LCDM6	LCD memory 6 (S6)	Read/write	Unchanged
027h	LCDM7	LCD memory 7 (S7)	Read/write	Unchanged
028h	LCDM8W	LCD memory 8 Word (S9, S8)	Read/write	Unchanged
028h	LCDM8	LCD memory 8 (S8)	Read/write	Unchanged
029h	LCDM9	LCD memory 9 (S9)	Read/write	Unchanged
02Ah	LCDM10W	LCD memory 10 Word (S11, S10)	Read/write	Unchanged
02Ah	LCDM10	LCD memory 10 (S10)	Read/write	Unchanged
02Bh	LCDM11	LCD memory 11 (S11)	Read/write	Unchanged
02Ch	LCDM12W	LCD memory 12 Word (S13, S12)	Read/write	Unchanged
02Ch	LCDM12	LCD memory 12 (S12)	Read/write	Unchanged
02Dh	LCDM13	LCD memory 13 (S13)	Read/write	Unchanged
02Eh	LCDM14W	LCD memory 14 Word (S15, S14)	Read/write	Unchanged
02Eh	LCDM14	LCD memory 14 (S14)	Read/write	Unchanged
02Fh	LCDM15	LCD memory 15 (S15)	Read/write	Unchanged
030h	LCDM16W	LCD memory 16 Word (S17, S16)	Read/write	Unchanged
030h	LCDM16	LCD memory 16 (S16)	Read/write	Unchanged
031h	LCDM17	LCD memory 17 (S17)	Read/write	Unchanged
032h	LCDM18W	LCD memory 18 Word (S19, S18)	Read/write	Unchanged
032h	LCDM18	LCD memory 18 (S18)	Read/write	Unchanged
033h	LCDM19	LCD memory 19 (S19)	Read/write	Unchanged
034h	LCDM20W	LCD memory 20 Word (S21, S20)	Read/write	Unchanged
034h	LCDM20	LCD memory 20 (S20)	Read/write	Unchanged
035h	LCDM21	LCD memory 21 (S21)	Read/write	Unchanged
036h	LCDM22W	LCD memory 22 Word (S23, S22)	Read/write	Unchanged
036h	LCDM22	LCD memory 22 (S22)	Read/write	Unchanged
037h	LCDM23	LCD memory 23 (S23)	Read/write	Unchanged
038h	LCDM24W	LCD memory 24 Word (S25, S24)	Read/write	Unchanged
038h	LCDM24	LCD memory 24 (S24)	Read/write	Unchanged
039h	LCDM25	LCD memory 25 (S25)	Read/write	Unchanged
03Ah	LCDM26W	LCD memory 26 Word (S27, S26)	Read/write	Unchanged
03Ah	LCDM26	LCD memory 26 (S26)	Read/write	Unchanged
03Bh	LCDM27	LCD memory 27 (S27)	Read/write	Unchanged
03Ch	LCDM28W	LCD memory 28 Word (S29, S28)	Read/write	Unchanged
03Ch	LCDM28	LCD memory 28 (S28)	Read/write	Unchanged
03Dh	LCDM29	LCD memory 29 (S29)	Read/write	Unchanged

[1] The LCD memory registers can also be accessed as word.

[2] The number of available memory registers on a given device depends on the number of available segment pins (see the device-specific data sheet).

Table 15-9. LCD Memory Registers for 5-Mux to 8-Mux Modes[1][2] (continued)

Offset	Acronym	Register Name	Type	Reset
03Eh	LCDM30W	LCD memory 30 Word (S31, S30)	Read/write	Unchanged
03Eh	LCDM30	LCD memory 30 (S30)	Read/write	Unchanged
03Fh	LCDM31	LCD memory 31 (S31)	Read/write	Unchanged
040h	LCDM32W	LCD memory 32 Word (S33, S32)	Read/write	Unchanged
040h	LCDM32	LCD memory 32 (S32)	Read/write	Unchanged
041h	LCDM33	LCD memory 33 (S33)	Read/write	Unchanged
042h	LCDM34W	LCD memory 34 Word (S35, S34)	Read/write	Unchanged
042h	LCDM34	LCD memory 34 (S34)	Read/write	Unchanged
043h	LCDM35	LCD memory 35 (S35)	Read/write	Unchanged
044h	LCDM36W	LCD memory 36 Word (S37, S36)	Read/write	Unchanged
044h	LCDM36	LCD memory 36 (S36)	Read/write	Unchanged
045h	LCDM37	LCD memory 37 (S37)	Read/write	Unchanged
046h	LCDM38W	LCD memory 38 Word (S39, S38)	Read/write	Unchanged
046h	LCDM38	LCD memory 38 (S38)	Read/write	Unchanged
047h	LCDM39	LCD memory 39 (S39)	Read/write	Unchanged
048h	LCDM40W	LCD memory 40 Word (S41, S40)	Read/write	Unchanged
048h	LCDM40	LCD memory 40 (S40)	Read/write	Unchanged
049h	LCDM41	LCD memory 41 (S41)	Read/write	Unchanged
04Ah	LCDM42W	LCD memory 42 Word (S43, S42)	Read/write	Unchanged
04Ah	LCDM42	LCD memory 42 (S42)	Read/write	Unchanged
04Bh	LCDM43	LCD memory 43 (S43)	Read/write	Unchanged
04Ch	LCDM44W	LCD memory 44 Word (S45, S44)	Read/write	Unchanged
04Ch	LCDM44	LCD memory 44 (S44)	Read/write	Unchanged
04Dh	LCDM45	LCD memory 45 (S45)	Read/write	Unchanged
04Eh	LCDM46W	LCD memory 46 Word (S47, S46)	Read/write	Unchanged
04Eh	LCDM46	LCD memory 46 (S46)	Read/write	Unchanged
04Fh	LCDM47	LCD memory 47 (S47)	Read/write	Unchanged
050h	LCDM48W	LCD memory 48 Word (S49, S48)	Read/write	Unchanged
050h	LCDM48	LCD memory 48 (S48)	Read/write	Unchanged
051h	LCDM49	LCD memory 49 (S49)	Read/write	Unchanged
052h	LCDM50W	LCD memory 50 Word (S51, S50)	Read/write	Unchanged
052h	LCDM50	LCD memory 50 (S50)	Read/write	Unchanged
053h	LCDM51	LCD memory 51 (S51)	Read/write	Unchanged
054h	LCDM52W	LCD memory 52 Word (S53, S52)	Read/write	Unchanged
054h	LCDM52	LCD memory 52 (S52)	Read/write	Unchanged
055h	LCDM53	LCD memory 53 (S53)	Read/write	Unchanged
056h	LCDM54W	LCD memory 54 Word (S55, S54)	Read/write	Unchanged
056h	LCDM54	LCD memory 54 (S54)	Read/write	Unchanged
057h	LCDM55	LCD memory 55 (S55)	Read/write	Unchanged
058h	LCDM56W	LCD memory 56 Word (S57, S56)	Read/write	Unchanged
058h	LCDM56	LCD memory 56 (S56)	Read/write	Unchanged
059h	LCDM57	LCD memory 57 (S57)	Read/write	Unchanged
05Ah	LCDM58W	LCD memory 58 Word (S59, S58)	Read/write	Unchanged
05Ah	LCDM58	LCD memory 58 (S58)	Read/write	Unchanged
05Bh	LCDM59	LCD memory 59 (S59)	Read/write	Unchanged
05Ch	LCDM60W	LCD memory 60 Word (S61, S60)	Read/write	Unchanged
05Ch	LCDM60	LCD memory 60 (S60)	Read/write	Unchanged

Table 15-9. LCD Memory Registers for 5-Mux to 8-Mux Modes[1][2] (continued)

Offset	Acronym	Register Name	Type	Reset
05Dh	LCDM61	LCD memory 61 (S61)	Read/write	Unchanged
05Eh	LCDM62W	LCD memory 62 Word (S63, S62)	Read/write	Unchanged
05Eh	LCDM62	LCD memory 62 (S62)	Read/write	Unchanged
05Fh	LCDM63	LCD memory 63 (S63)	Read/write	Unchanged

15.3.1 LCDCTL0 Register

LCD_E Control Register 0

Figure 15-19. LCDCTL0 Register

15	14	13	12	11	10	9	8
LCDDIVx					Reserved		
rw-{0}	rw-{0}	rw-{1}	rw-{1}	rw-{1}	r-{0}	r-{0}	r-{0}
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
LCDSSEL		LCDMXx			LCDSON	LCDLP	LCDON
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-10. LCDCTL0 Register Description

Bit	Field	Type	Reset	Description
15-11	LCDDIVx	RW	00111b	LCD frequency divider. Together with LCDMXx, the LCD frequency f_{LCD} is calculated as $f_{LCD} = f_{SOURCE} / ((LCDDIVx + 1) \times Value[LCDMXx])$. Change only while LCDON = 0. 00000b = Divide by 1 00001b = Divide by 2 ⋮ 11110b = Divide by 31 11111b = Divide by 32
10-8	Reserved	R	0h	Reserved
7-6	LCDSSEL	RW	0h	Clock source f_{SOURCE} select for LCD and blinking frequency. Change only while LCDON = 0. 00b = XT1CLK 01b = ACLK (30 kHz to 40 kHz) 10b = VLOCLK 11b = Reserved
5-3	LCDMXx	RW	0h	LCD mux rate. These bits select the LCD mode. Change only while LCDON = 0. 000b = Static 001b = 2-mux 010b = 3-mux 011b = 4-mux 100b = 5-mux 101b = 6-mux 110b = 7-mux 111b = 8-mux
2	LCDSON	RW	0h	LCD segments on. This bit supports flashing LCD applications by turning off all segment lines, while leaving the LCD timing generator and R33 enabled. 0b = All LCD segments are off. 1b = All LCD segments are enabled and on or off according to their corresponding memory location.
1	LCDLP	RW	0h	LCD low-power waveform 0b = Standard LCD waveforms on segment and common lines selected. 1b = Low-power LCD waveforms on segment and common lines selected.
0	LCDON	RW	0h	LCD on. This bit turns the LCD_E module on or off. 0b = LCD_E module off 1b = LCD_E module on

15.3.2 LCDCTL1 Register

LCD_E Control Register 1

Figure 15-20. LCDCTL1 Register

15	14	13	12	11	10	9	8
Reserved					LCDBLKONIE	LCDBLKOFFIE	LCDFRMIE
r0	r0	r0	r0	r0	rw-{0}	rw-{0}	rw-{0}
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved					LCDBLKONIFG	LCDBLKOFFIFG	LCDFRMIFG
r0	r0	r0	r0	r0	rw-{0}	rw-{0}	rw-{0}

Table 15-11. LCDCTL1 Register Description

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved
10	LCDBLKONIE	RW	0h	LCD blinking interrupt enable, segments switched on 0b = Interrupt disabled 1b = Interrupt enabled
9	LCDBLKOFFIE	RW	0h	LCD blinking interrupt enable, segments switched off 0b = Interrupt disabled 1b = Interrupt enabled
8	LCDFRMIE	RW	0h	LCD frame interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
7-3	Reserved	R	0h	Reserved
2	LCDBLKONIFG	RW	0h	LCD blinking interrupt flag, set at the rising edge of BLKCLK. Automatically cleared when data is written into a memory register. 0b = No interrupt pending 1b = Interrupt pending
1	LCDBLKOFFIFG	RW	0h	LCD blinking interrupt flag, set at the falling edge of BLKCLK. Automatically cleared when data is written into a memory register. 0b = No interrupt pending 1b = Interrupt pending
0	LCDFRMIFG	RW	0h	LCD frame interrupt flag. Automatically cleared when data is written into a memory register. 0b = No interrupt pending 1b = Interrupt pending

15.3.3 LCDBLKCTL Register

LCD_E Blink Control Register

Figure 15-21. LCDBLKCTL 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			LCDBLKPREx			LCDBLKMODx	
r0	r0	r0	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-12. LCDBLKCTL Register Description

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	
4-2	LCDBLKPREx	RW	0h	Clock prescaler for blinking frequency. Together with LCDMXx, the blinking frequency f_{BLINK} is calculated as $f_{BLINK} = f_{LCD} / ((LCDMXx + 1) \times 2^{(LCDBLKPREx + 2)})$. Settings for LCDMXx and LCDBLKPREx should only be changed while LCDBLKMODx = 00. 000b = Divide by 4 001b = Divide by 8 010b = Divide by 16 011b = Divide by 32 100b = Divide by 64 101b = Divide by 128 110b = Divide by 256 111b = Divide by 512
1-0	LCDBLKMODx	RW	0h	Blinking mode 00b = Blinking disabled. 01b = Blinking of individual segments as enabled in blinking memory register LCDBMx. In mux mode >5 blinking is disabled. 10b = Blinking of all segments 11b = Switching between display contents as stored in LCDMx and LCDBMx memory registers. In mux mode >5 blinking is disabled.

15.3.4 LCDMEMCTL Register

LCD_E Memory Control Register

Figure 15-22. LCDMEMCTL 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					LCDCLRBM	LCDCLRM	LCDDISP
r0	r0	r0	r0	r0	rw-{0}	rw-{0}	rw-{0}

Table 15-13. LCDMEMCTL Register Description

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved
2	LCDCLRBM	RW	0h	Clear LCD blinking memory Clears all blinking memory registers LCDBMx. The bit is automatically reset when the blinking memory is cleared. Setting this bit in 5-mux mode and above has no effect. It is immediately reset again. 0b = Contents of blinking memory registers LCDBMx remain unchanged 1b = Clear content of all blinking memory registers LCDBMx
1	LCDCLRM	RW	0h	Clear LCD memory Clears all LCD memory registers LCDMx. The bit is automatically reset when the LCD memory is cleared. 0b = Contents of LCD memory registers LCDMx remain unchanged 1b = Clear content of all LCD memory registers LCDMx
0	LCDDISP	RW	0h	Select LCD memory registers for display When LCDBLKMODx = 00, LCDDISP can be set by software. The bit is cleared in LCDBLKMODx = 01 and LCDBLKMODx = 10 or if a mux mode ≥5 is selected and cannot be changed by software. When LCDBLKMODx = 11, this bit reflects the currently displayed memory but cannot be changed by software. When returning to LCDBLKMODx = 00 the bit is cleared. 0b = Display content of LCD memory registers LCDMx 1b = Display content of LCD blinking memory registers LCDBMx

15.3.5 LCDVCTL Register

LCD_E Voltage Control Register

Figure 15-23. LCDVCTL Register

15	14	13	12	11	10	9	8
LCDCPFSELx				VLCDx			
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
LCDCPEN	LCDREFEN	LCDSELVDD	Reserved				LCDREFMODE
rw-{0}	rw-{0}	rw-{0}	r0	r0	r0	r0	rw-{0}

Table 15-14. LCDVCTL Register Description

Bit	Field	Type	Reset	Description
15-12	LCDCPFSELx	RW	0h	Charge pump frequency selection. Clock source can be XT1, ACLK, VLO (4-bit, if f_{SOURCE} = f_{ACLK} = 32.768 kHz) 0000b = 32.768 kHz / 1 / 8 = 4.096 kHz 0001b = 32.768 kHz / 2 / 8 = 2.048 kHz 0010b = 32.768 kHz / 3 / 8 = 1.365 kHz 0011b = 32.768 kHz / 4 / 8 = 1.024 kHz 0100b = 32.768 kHz / 5 / 8 = 819 Hz 0101b = 32.768 kHz / 6 / 8 = 682 Hz 0110b = 32.768 kHz / 7 / 8 = 585 Hz 0111b = 32.768 kHz / 8 / 8 = 512 Hz 1000b = 32.768 kHz / 9 / 8 = 455 Hz 1001b = 32.768 kHz / 10 / 8 = 409 Hz 1010b = 32.768 kHz / 11 / 8 = 372 Hz 1011b = 32.768 kHz / 12 / 8 = 341 Hz 1100b = 32.768 kHz / 13 / 8 = 315 Hz 1101b = 32.768 kHz / 14 / 8 = 292 Hz 1110b = 32.768 kHz / 15 / 8 = 273 Hz 1111b = 32.768 kHz / 16 / 8 = 256 Hz
11-8	VLCDx	RW	0h	Internal reference voltage select on R13. Only valuable when LCDCPEN = 1 and LCDREFEN = 1. 0000b = 2.60 V 0001b = 2.66 V 0010b = 2.72 V 0011b = 2.78 V 0100b = 2.84 V 0101b = 2.90 V 0110b = 2.96 V 0111b = 3.02 V 1000b = 3.08 V 1001b = 3.14 V 1010b = 3.20 V 1011b = 3.26 V 1100b = 3.32 V 1101b = 3.38 V 1110b = 3.44 V 1111b = 3.50 V
7	LCDCPEN	RW	0h	Charge pump enable 0b = Charge pump disabled[1] 1b = Charge pump enabled when V_{LCD} is generated internally (VLCDEXT = 0) and VLCDx > 0 or VLCDREFx > 0.

[1] To use LCD, an external resistor divider must be connected to R13, R23, and R33.

Table 15-14. LCDVCTL Register Description (continued)

Bit	Field	Type	Reset	Description
6	LCDREFEN	RW	0h	Internal reference voltage enable on R13 0b = Internal reference voltage disabled 1b = Internal reference voltage enabled
5	LCDSELVDD	RW	0h	Selects if R33 is supplied either from V_{CC} internally or from charge pump 0b = R33 connected to external supply 1b = R33 internally connected to V_{CC}
4-1	Reserved	R	0h	Reserved
0	LCDREFMODE	RW	0h	Selects whether R13 voltage is switched or in static mode 0b = Static mode 1b = Switched mode

15.3.6 LCDPCTL0 Register

LCD_E Port Control Register 0

Settings for LCDSx should only be changed while LCDON = 0.

Figure 15-24. LCDPCTL0 Register

15	14	13	12	11	10	9	8
LCDS15	LCDS14	LCDS13	LCDS12	LCDS11	LCDS10	LCDS9	LCDS8
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDS7	LCDS6	LCDS5	LCDS4	LCDS3	LCDS2	LCDS1	LCDS0
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-15. LCDPCTL0 Register Description

Bit	Field	Type	Reset	Description
15	LCDS15	RW	0h	LCD pin 15 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
14	LCDS14	RW	0h	LCD pin 14 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
13	LCDS13	RW	0h	LCD pin 13 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
12	LCDS12	RW	0h	LCD pin 12 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
11	LCDS11	RW	0h	LCD pin 11 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
10	LCDS10	RW	0h	LCD pin 10 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
9	LCDS9	RW	0h	LCD pin 9 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
8	LCDS8	RW	0h	LCD pin 8 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
7	LCDS7	RW	0h	LCD pin 7 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
6	LCDS6	RW	0h	LCD pin 6 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.

Table 15-15. LCDPCTL0 Register Description (continued)

Bit	Field	Type	Reset	Description
5	LCDS5	RW	0h	LCD pin 5 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
4	LCDS4	RW	0h	LCD pin 4 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
3	LCDS3	RW	0h	LCD pin 3 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
2	LCDS2	RW	0h	LCD pin 2 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
1	LCDS1	RW	0h	LCD pin 1 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
0	LCDS0	RW	0h	LCD pin 0 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.

15.3.7 LCDPCTL1 Register

LCD_E Port Control Register 1

Settings for LCDSx should only be changed while LCDON = 0.

Figure 15-25. LCDPCTL1 Register

15	14	13	12	11	10	9	8
LCDS31	LCDS30	LCDS29	LCDS28	LCDS27	LCDS26	LCDS25	LCDS24
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDS23	LCDS22	LCDS21	LCDS20	LCDS19	LCDS18	LCDS17	LCDS16
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-16. LCDPCTL1 Register Description

Bit	Field	Type	Reset	Description
15	LCDS31	RW	0h	LCD pin 31 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
14	LCDS30	RW	0h	LCD pin 30 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
13	LCDS29	RW	0h	LCD pin 29 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
12	LCDS28	RW	0h	LCD pin 28 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
11	LCDS27	RW	0h	LCD pin 27 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
10	LCDS26	RW	0h	LCD pin 26 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
9	LCDS25	RW	0h	LCD pin 25 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
8	LCDS24	RW	0h	LCD pin 24 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
7	LCDS23	RW	0h	LCD segment line 23 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
6	LCDS22	RW	0h	LCD segment line 22 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.

Table 15-16. LCDPCTL1 Register Description (continued)

Bit	Field	Type	Reset	Description
5	LCDS21	RW	0h	LCD segment line 21 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
4	LCDS20	RW	0h	LCD segment line 20 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
3	LCDS19	RW	0h	LCD segment line 19 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
2	LCDS18	RW	0h	LCD segment line 18 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
1	LCDS17	RW	0h	LCD segment line 17 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
0	LCDS16	RW	0h	LCD segment line 16 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.

15.3.8 LCDPCTL2 Register

LCD_E Port Control Register 2 (= 256 Segments)

Settings for LCDSx should only be changed while LCDON = 0.

Figure 15-26. LCDPCTL2 Register

15	14	13	12	11	10	9	8
LCDS47	LCDS46	LCDS45	LCDS44	LCDS43	LCDS42	LCDS41	LCDS40
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDS39	LCDS38	LCDS37	LCDS36	LCDS35	LCDS34	LCDS33	LCDS32
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-17. LCDPCTL2 Register Description

Bit	Field	Type	Reset	Description
15	LCDS47	RW	0h	LCD pin 47 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
14	LCDS46	RW	0h	LCD pin 46 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
13	LCDS45	RW	0h	LCD pin 45 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
12	LCDS44	RW	0h	LCD pin 44 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
11	LCDS43	RW	0h	LCD pin 43 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
10	LCDS42	RW	0h	LCD pin 42 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
9	LCDS41	RW	0h	LCD pin 41 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
8	LCDS40	RW	0h	LCD pin 40 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
7	LCDS39	RW	0h	LCD pin 39 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
6	LCDS38	RW	0h	LCD pin 38 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.

Table 15-17. LCDPCTL2 Register Description (continued)

Bit	Field	Type	Reset	Description
5	LCDS37	RW	0h	LCD pin 37 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
4	LCDS36	RW	0h	LCD pin 36 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
3	LCDS35	RW	0h	LCD pin 35 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
2	LCDS34	RW	0h	LCD pin 34 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
1	LCDS33	RW	0h	LCD pin 33 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
0	LCDS32	RW	0h	LCD pin 32 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.

15.3.9 LCDPCTL3 Register

LCD_E Port Control Register 3 (384 Segments, COMs Shared With Segments)

Settings for LCDSx should only be changed while LCDON = 0.

Figure 15-27. LCDPCTL3 Register

15	14	13	12	11	10	9	8
LCDS63	LCDS62	LCDS61	LCDS60	LCDS59	LCDS58	LCDS57	LCDS56
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDS55	LCDS54	LCDS53	LCDS52	LCDS51	LCDS50	LCDS49	LCDS48
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-18. LCDPCTL3 Register Description

Bit	Field	Type	Reset	Description
15	LCDS63	RW	0h	LCD pin 63 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
14	LCDS62	RW	0h	LCD pin 62 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
13	LCDS61	RW	0h	LCD pin 61 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
12	LCDS60	RW	0h	LCD pin 60 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
11	LCDS59	RW	0h	LCD pin 59 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
10	LCDS58	RW	0h	LCD pin 58 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
9	LCDS57	RW	0h	LCD pin 57 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
8	LCDS56	RW	0h	LCD pin 56 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
7	LCDS55	RW	0h	LCD pin 55 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
6	LCDS54	RW	0h	LCD pin 54 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.

Table 15-18. LCDPCTL3 Register Description (continued)

Bit	Field	Type	Reset	Description
5	LCDS53	RW	0h	LCD pin 53 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
4	LCDS52	RW	0h	LCD pin 52 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
3	LCDS51	RW	0h	LCD pin 51 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
2	LCDS50	RW	0h	LCD pin 50 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
1	LCDS49	RW	0h	LCD pin 49 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.
0	LCDS48	RW	0h	LCD pin 48 enable. This bit affects only pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0b = Multiplexed pins are port functions. 1b = Pins are LCD functions.

15.3.10 LCDCSSEL0 Register

LCD_E COM/SEG Select Register 0

Figure 15-28. LCDCSSEL0 Register

15	14	13	12	11	10	9	8
LCDCSS15	LCDCSS14	LCDCSS13	LCDCSS12	LCDCSS11	LCDCSS10	LCDCSS9	LCDCSS8
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDCSS7	LCDCSS6	LCDCSS5	LCDCSS4	LCDCSS3	LCDCSS2	LCDCSS1	LCDCSS0
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-19. LCDCSSEL0 Register Description

Bit	Field	Type	Reset	Description
15	LCDCSS15	RW	0h	Selects pin L15 as either common or segment line. 0b = Segment line 1b = Common line
14	LCDCSS14	RW	0h	Selects pin L14 as either common or segment line. 0b = Segment line 1b = Common line
13	LCDCSS13	RW	0h	Selects pin L13 as either common or segment line. 0b = Segment line 1b = Common line
12	LCDCSS12	RW	0h	Selects pin L12 as either common or segment line. 0b = Segment line 1b = Common line
11	LCDCSS11	RW	0h	Selects pin L11 as either common or segment line. 0b = Segment line 1b = Common line
10	LCDCSS10	RW	0h	Selects pin L10 as either common or segment line. 0b = Segment line 1b = Common line
9	LCDCSS9	RW	0h	Selects pin L9 as either common or segment line. 0b = Segment line 1b = Common line
8	LCDCSS8	RW	0h	Selects pin L8 as either common or segment line. 0b = Segment line 1b = Common line
7	LCDCSS7	RW	0h	Selects pin L7 as either common or segment line. 0b = Segment line 1b = Common line
6	LCDCSS6	RW	0h	Selects pin L6 as either common or segment line. 0b = Segment line 1b = Common line
5	LCDCSS5	RW	0h	Selects pin L5 as either common or segment line. 0b = Segment line 1b = Common line
4	LCDCSS4	RW	0h	Selects pin L4 as either common or segment line. 0b = Segment line 1b = Common line
3	LCDCSS3	RW	0h	Selects pin L3 as either common or segment line. 0b = Segment line 1b = Common line

Table 15-19. LCDCSSEL0 Register Description (continued)

Bit	Field	Type	Reset	Description
2	LCDCSS2	RW	0h	Selects pin L2 as either common or segment line. 0b = Segment line 1b = Common line
1	LCDCSS1	RW	0h	Selects pin L1 as either common or segment line. 0b = Segment line 1b = Common line
0	LCDCSS0	RW	0h	Selects pin L0 as either common or segment line. 0b = Segment line 1b = Common line

15.3.11 LCDCSSEL1 Register

LCD_E COM/SEG Select Register 1

Figure 15-29. LCDCSSEL1 Register

15	14	13	12	11	10	9	8
LCDCSS31	LCDCSS30	LCDCSS29	LCDCSS28	LCDCSS27	LCDCSS26	LCDCSS25	LCDCSS24
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDCSS23	LCDCSS22	LCDCSS21	LCDCSS20	LCDCSS19	LCDCSS18	LCDCSS17	LCDCSS16
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-20. LCDCSSEL1 Register Description

Bit	Field	Type	Reset	Description
15	LCDCSS31	RW	0h	Selects pin L31 as either common or segment line. 0b = Segment line 1b = Common line
14	LCDCSS30	RW	0h	Selects pin L30 as either common or segment line. 0b = Segment line 1b = Common line
13	LCDCSS29	RW	0h	Selects pin L29 as either common or segment line. 0b = Segment line 1b = Common line
12	LCDCSS28	RW	0h	Selects pin L28 as either common or segment line. 0b = Segment line 1b = Common line
11	LCDCSS27	RW	0h	Selects pin L27 as either common or segment line. 0b = Segment line 1b = Common line
10	LCDCSS26	RW	0h	Selects pin L26 as either common or segment line. 0b = Segment line 1b = Common line
9	LCDCSS25	RW	0h	Selects pin L25 as either common or segment line. 0b = Segment line 1b = Common line
8	LCDCSS24	RW	0h	Selects pin L24 as either common or segment line. 0b = Segment line 1b = Common line
7	LCDCSS23	RW	0h	Selects pin L23 as either common or segment line. 0b = Segment line 1b = Common line
6	LCDCSS22	RW	0h	Selects pin L22 as either common or segment line. 0b = Segment line 1b = Common line
5	LCDCSS21	RW	0h	Selects pin L21 as either common or segment line. 0b = Segment line 1b = Common line
4	LCDCSS20	RW	0h	Selects pin L20 as either common or segment line. 0b = Segment line 1b = Common line
3	LCDCSS19	RW	0h	Selects pin L19 as either common or segment line. 0b = Segment line 1b = Common line

Table 15-20. LCDCSSEL1 Register Description (continued)

Bit	Field	Type	Reset	Description
2	LCDCSS18	RW	0h	Selects pin L18 as either common or segment line. 0b = Segment line 1b = Common line
1	LCDCSS17	RW	0h	Selects pin L17 as either common or segment line. 0b = Segment line 1b = Common line
0	LCDCSS16	RW	0h	Selects pin L16 as either common or segment line. 0b = Segment line 1b = Common line

15.3.12 LCDCSSEL2 Register

LCD_E COM/SEG Select Register 0

Figure 15-30. LCDCSSEL2 Register

15	14	13	12	11	10	9	8
LCDCSS47	LCDCSS46	LCDCSS45	LCDCSS44	LCDCSS43	LCDCSS42	LCDCSS41	LCDCSS40
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDCSS39	LCDCSS38	LCDCSS37	LCDCSS36	LCDCSS35	LCDCSS34	LCDCSS33	LCDCSS32
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-21. LCDCSSEL2 Register Description

Bit	Field	Type	Reset	Description
15	LCDCSS47	RW	0h	Selects pin L47 as either common or segment line. 0b = Segment line 1b = Common line
14	LCDCSS46	RW	0h	Selects pin L46 as either common or segment line. 0b = Segment line 1b = Common line
13	LCDCSS45	RW	0h	Selects pin L45 as either common or segment line. 0b = Segment line 1b = Common line
12	LCDCSS44	RW	0h	Selects pin L44 as either common or segment line. 0b = Segment line 1b = Common line
11	LCDCSS43	RW	0h	Selects pin L43 as either common or segment line. 0b = Segment line 1b = Common line
10	LCDCSS42	RW	0h	Selects pin L42 as either common or segment line. 0b = Segment line 1b = Common line
9	LCDCSS41	RW	0h	Selects pin L41 as either common or segment line. 0b = Segment line 1b = Common line
8	LCDCSS40	RW	0h	Selects pin L40 as either common or segment line. 0b = Segment line 1b = Common line
7	LCDCSS39	RW	0h	Selects pin L39 as either common or segment line. 0b = Segment line 1b = Common line
6	LCDCSS38	RW	0h	Selects pin L38 as either common or segment line. 0b = Segment line 1b = Common line
5	LCDCSS37	RW	0h	Selects pin L37 as either common or segment line. 0b = Segment line 1b = Common line
4	LCDCSS36	RW	0h	Selects pin L36 as either common or segment line. 0b = Segment line 1b = Common line
3	LCDCSS35	RW	0h	Selects pin L35 as either common or segment line. 0b = Segment line 1b = Common line

Table 15-21. LCDCSSEL2 Register Description (continued)

Bit	Field	Type	Reset	Description
2	LCDCSS34	RW	0h	Selects pin L34 as either common or segment line. 0b = Segment line 1b = Common line
1	LCDCSS33	RW	0h	Selects pin L33 as either common or segment line. 0b = Segment line 1b = Common line
0	LCDCSS32	RW	0h	Selects pin L32 as either common or segment line. 0b = Segment line 1b = Common line

15.3.13 LCDCSSEL3 Register

LCD_E COM/SEG Select Register 0

Figure 15-31. LCDCSSEL3 Register

15	14	13	12	11	10	9	8
LCDCSS63	LCDCSS62	LCDCSS61	LCDCSS60	LCDCSS59	LCDCSS58	LCDCSS57	LCDCSS56
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDCSS55	LCDCSS54	LCDCSS53	LCDCSS52	LCDCSS51	LCDCSS50	LCDCSS49	LCDCSS48
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-22. LCDCSSEL3 Register Description

Bit	Field	Type	Reset	Description
15	LCDCSS63	RW	0h	Selects pin L63 as either common or segment line. 0b = Segment line 1b = Common line
14	LCDCSS62	RW	0h	Selects pin L62 as either common or segment line. 0b = Segment line 1b = Common line
13	LCDCSS61	RW	0h	Selects pin L61 as either common or segment line. 0b = Segment line 1b = Common line
12	LCDCSS60	RW	0h	Selects pin L60 as either common or segment line. 0b = Segment line 1b = Common line
11	LCDCSS59	RW	0h	Selects pin L59 as either common or segment line. 0b = Segment line 1b = Common line
10	LCDCSS58	RW	0h	Selects pin L58 as either common or segment line. 0b = Segment line 1b = Common line
9	LCDCSS57	RW	0h	Selects pin L57 as either common or segment line. 0b = Segment line 1b = Common line
8	LCDCSS56	RW	0h	Selects pin L56 as either common or segment line. 0b = Segment line 1b = Common line
7	LCDCSS55	RW	0h	Selects pin L55 as either common or segment line. 0b = Segment line 1b = Common line
6	LCDCSS54	RW	0h	Selects pin L54 as either common or segment line. 0b = Segment line 1b = Common line
5	LCDCSS53	RW	0h	Selects pin L53 as either common or segment line. 0b = Segment line 1b = Common line
4	LCDCSS52	RW	0h	Selects pin L52 as either common or segment line. 0b = Segment line 1b = Common line
3	LCDCSS51	RW	0h	Selects pin L51 as either common or segment line. 0b = Segment line 1b = Common line

Table 15-22. LCDCSSEL3 Register Description (continued)

Bit	Field	Type	Reset	Description
2	LCDCSS50	RW	0h	Selects pin L50 as either common or segment line. 0b = Segment line 1b = Common line
1	LCDCSS49	RW	0h	Selects pin L49 as either common or segment line. 0b = Segment line 1b = Common line
0	LCDCSS48	RW	0h	Selects pin L48 as either common or segment line. 0b = Segment line 1b = Common line

15.3.14 LCDM[index] Register – Static, 2-Mux, 3-Mux, 4-Mux Mode

LCD_E Memory [_index_] Register

For Static, 2-Mux, 3-Mux, 4-Mux Mode: **_index = 0 to 31_**

Figure 15-32. LCDM[_index_] Register

7	6	5	4	3	2	1	0
MBIT7	MBIT6	MBIT5	MBIT4	MBIT3	MBIT2	MBIT1	MBIT0
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-23. LCDM[_index_] Register Description

Bit	Field	Type	Reset	Description
7	MBIT7	RW	0h	If LCD pin L[2*_index+1_] is selected as segment line (LCDCSS[_2*index+1_] = 0b) and LCD mux rate is 4-mux (LCDMXx=011b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[_2*index+1_] is selected as common line (LCDCSS[_2*index_+1] = 1b): 0b = Pin L[2*_index_+1] not used as COM3 1b = Pin L[_2*index+1_] is used as COM3
6	MBIT6	RW	0h	If LCD pin L[2*_index+1_] is selected as segment line (LCDCSS[_2*index+1_] = 0b) and LCD mux rate is 3- or 4-mux (010b <= LCDMXx <= 011b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[_2*index_+1] is selected as common line (LCDCSS[_2*index_+1] = 1b): 0b = Pin L[2*_index+1_] not used as COM2 1b = Pin L[2*_index_+1] is used as COM2
5	MBIT5	RW	0h	If LCD pin L[2*_index+1_] is selected as segment line (LCDCSS[2*_index+1_] = 0b) and LCD mux rate is 2-, 3- or 4-mux (001b <= LCDMXx <= 011b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[2*_index+1_] is selected as common line (LCDCSS[_2*index_+1] = 1b): 0b = Pin L[2*_index_+1] not used as COM1 1b = Pin L[_2*index+1_] is used as COM1
4	MBIT4	RW	0h	If LCD pin L[_2*index+1_] is selected as segment line (LCDCSS[_2*index+1_] = 0b) and LCD mux rate is static, 2-, 3- or 4-mux (000b <= LCDMXx <= 011b) 0b = LCD segment off 1b = LCD segment on If LCD pin L[_2*index+1_] is selected as common line (LCDCSS[_2*index_+1] = 1b): 0b = Pin L[_2*ndex_+1] not used as COM0 1b = Pin L[_2*ndex+1_] is used as COM0
3	MBIT3	RW	0h	If LCD pin L[_index_] is selected as segment line (LCDCSS[_index_] = 0b) and LCD mux rate is 4-mux (LCDMXx=011b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[_2*index_] is selected as common line (LCDCSS[2*_index_] = 1b): 0b = Pin L[_2*index_] not used as COM3 1b = Pin L[2*_index_] is used as COM3
2	MBIT2	RW	0h	If LCD pin L[2*_index_] is selected as segment line (LCDCSS[2*_index_] = 0b) and LCD mux rate is 3- or 4-mux (010b <= LCDMXx <= 011b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[2*_index_] is selected as common line (LCDCSS[2*_index_] = 1b): 0b = Pin L[_2*index_] not used as COM2 1b = Pin L[2*_index_] is used as COM2

Table 15-23. LCDM[*index*] Register Description (continued)

Bit	Field	Type	Reset	Description
1	MBIT1	RW	0h	If LCD pin L[2**index*] is selected as segment line (LCDCSS[2**index*] = 0b) and LCD mux rate is 2-, 3- or 4-mux (001b <= LCDMXx <= 011b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[2**index*] is selected as common line (LCDCSS[2**index*] = 1b): 0b = Pin L[2**index*] not used as COM1 1b = Pin L[2**index*] is used as COM1
0	MBIT0	RW	0h	If LCD L[2**index*] is selected as segment line (LCDCSS[2**index*] = 0b) and LCD mux rate is static, 2-, 3- or 4-mux (000b <= LCDMXx <= 011b) 0b = LCD segment off 1b = LCD segment on If LCD pin L[2**index*] is selected as common line (LCDCSS[2**index*] = 1b): 0b = Pin L[2**index*] not used as COM0 1b = Pin L[2**index*] is used as COM0

15.3.15 LCDM[index] Register – 5-Mux, 6-Mux, 7-Mux, 8-Mux Mode

LCD_E Memory [*index*] Register

5-Mux, 6-Mux, 7-Mux, 8-Mux Mode: ***index = 0 to 63***

Figure 15-33. LCDM[*index*] Register

7	6	5	4	3	2	1	0
MBIT7	MBIT6	MBIT5	MBIT4	MBIT3	MBIT2	MBIT1	MBIT0
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

Table 15-24. LCDM[*index*] Register Description

Bit	Field	Type	Reset	Description
7	MBIT7	RW	0h	If LCD pin L[*index*] is selected as segment line (LCDCSS[*index*] = 0b) and LCD mux rate is 8-mux (LCDMXx = 111b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[*index*] is selected as common line (LCDCSS[*index*] = 1b): 0b = Pin L[*index*] not used as COM7 1b = Pin L[*index*] is used as COM7
6	MBIT6	RW	0h	If LCD pin L[*index*] is selected as segment line (LCDCSS[*index*] = 0b) and LCD mux rate is 7- or 8-mux (LCDMXx >= 110b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[*index*] is selected as common line (LCDCSS[*index*] = 1b): 0b = Pin L[*index*] not used as COM6 1b = Pin L[*index*] is used as COM6
5	MBIT5	RW	0h	If LCD pin L[*index*] is selected as segment line (LCDCSS[*index*] = 0b) and LCD mux rate is 6-, 7- or 8-mux (LCDMXx >= 101b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[*index*] is selected as common line (LCDCSS[*index*] = 1b): 0b = Pin L[*index*] not used as COM5 1b = Pin L[*index*] is used as COM5
4	MBIT4	RW	0h	If LCD pin L[*index*] is selected as segment line (LCDCSS[*index*] = 0b) and LCD mux rate is 5-, 6-, 7- or 8-mux (LCDMXx >= 100b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[*index*] is selected as common line (LCDCSS[*index*] = 1b): 0b = Pin L[*index*] not used as COM4 1b = Pin L[*index*] is used as COM4
3	MBIT3	RW	0h	If LCD pin L[*index*] is selected as segment line (LCDCSS[*index*] = 0b) and LCD mux rate is 5-, 6-, 7- or 8-mux (LCDMXx >= 100b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[*index*] is selected as common line (LCDCSS[*index*] = 1b): 0b = Pin L[*index*] not used as COM3 1b = Pin L[*index*] is used as COM3
2	MBIT2	RW	0h	If LCD pin L[*index*] is selected as segment line (LCDCSS[*index*] = 0b) and LCD mux rate is 5-, 6-, 7- or 8-mux (LCDMXx >= 100b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[*index*] is selected as common line (LCDCSS[*index*] = 1b): 0b = Pin L[*index*-1] not used as COM2 1b = Pin L[*index*-1] is used as COM2

Table 15-24. LCDM[*index*] Register Description (continued)

Bit	Field	Type	Reset	Description
1	MBIT1	RW	0h	If LCD pin L[*index*] is selected as segment line (LCDCSS[*index*] = 0b) and LCD mux rate is 5-, 6-, 7- or 8-mux (LCDMXx >= 100b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[*index*] is selected as common line (LCDCSS[*index*] = 1b): 0b = Pin L[*index*] not used as COM1 1b = Pin L[*index*] is used as COM1
0	MBIT0	RW	0h	If LCD pin L[*index*] is selected as segment line (LCDCSS[*index*] = 0b) and LCD mux rate is 5-, 6-, 7- or 8-mux (LCDMXx >= 100b): 0b = LCD segment off 1b = LCD segment on If LCD pin L[*index*] is selected as common line (LCDCSS[*index*] = 1b): 0b = Pin L[*index*] not used as COM0 1b = Pin L[*index*] is used as COM0

15.3.16 LCDIV Register

LCD_E Interrupt Vector Register

Figure 15-34. LCDIV Register

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

Table 15-25. LCDIV Register Description

Bit	Field	Type	Reset	Description
15-0	LCDIVx	R	0h	LCD_E interrupt vector value 00h = No interrupt pending 04h = Interrupt Source: Blink, segments off; Interrupt Flag: LCDBLKOFFIFG; Interrupt Priority: Highest 06h = Interrupt Source: Blink, segments on; Interrupt Flag: LCDBLKONIFG 08h = Interrupt Source: Frame interrupt; Interrupt Flag: LCDFRMIFG; Interrupt Priority: Lowest

ADC Module

The ADC module is a high-performance 10-bit or 12-bit analog-to-digital converter (ADC). See the device-specific data sheet to determine the resolution supported by a device. This chapter describes the operation of the ADC module.

Topic Page

16.1 ADC Introduction 448
16.2 ADC Operation 450
16.3 ADC Registers 467

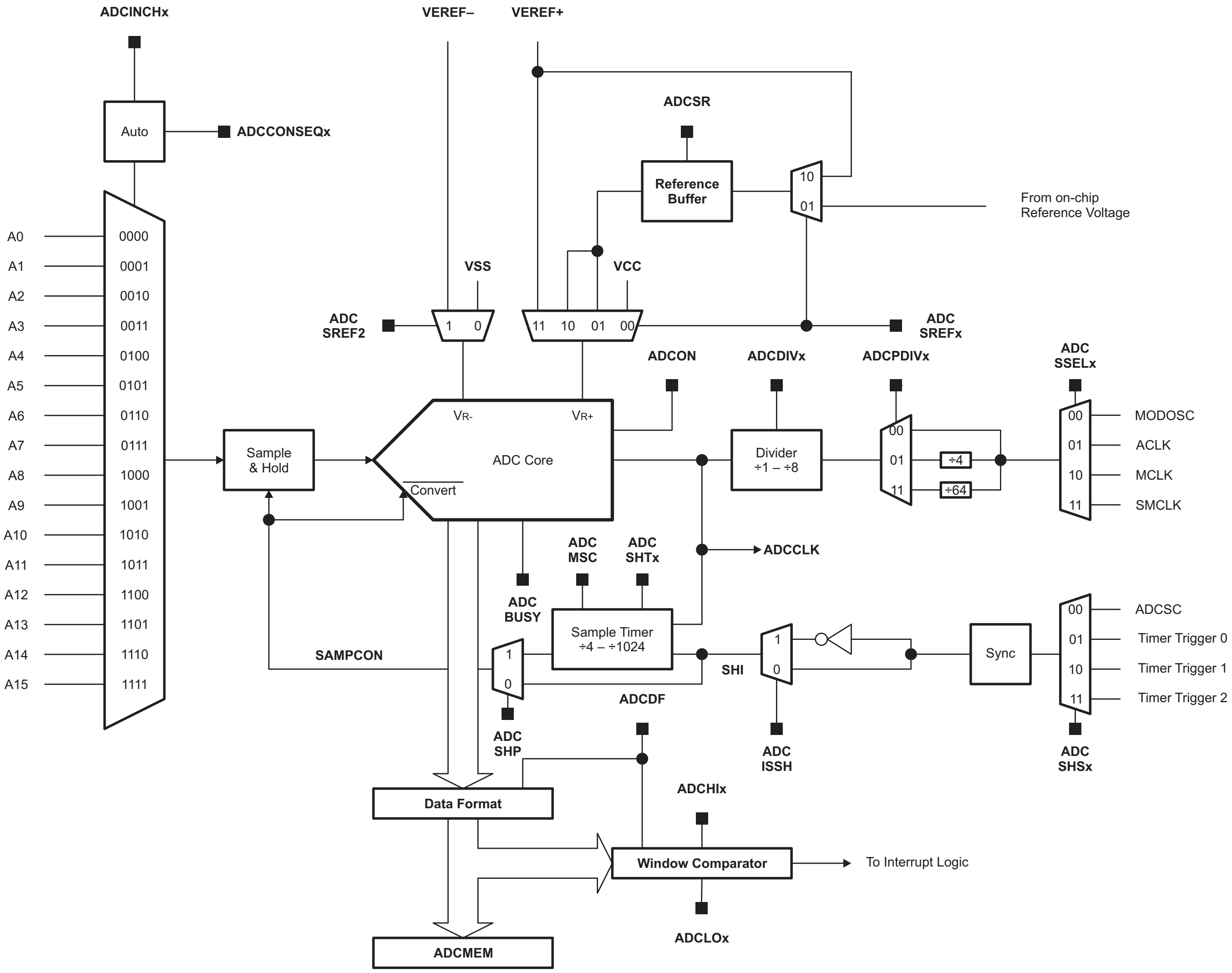
16.1 ADC Introduction

The ADC module supports fast 10-bit or 12-bit analog-to-digital conversions. The module implements a 10-bit or 12-bit SAR core together, sample select control, and a window comparator.

ADC features include:

- Greater than 200-ksps maximum conversion rate
- Monotonic 10-bit or 12-bit converter with no missing codes
- Sample-and-hold with programmable sampling periods controlled by software or timers
- Conversion initiation by software or different timers
- Software-selectable on-chip reference or external reference
- Twelve individually configurable external input channels
- Conversion channel for on-chip temperature sensor
- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence, and repeat-sequence conversion modes
- Window comparator for low-power monitoring of input signals
- Interrupt vector register for fast decoding of six ADC interrupts (ADCIFG0, ADCTOVIFG, ADCOVIFG, ADCLOIFG, ADCINIFG, ADCHIIFG)

Figure 16-1 shows the block diagram of the ADC module.



A The MODOSC is part of the Clock System. See the Clock System chapter for more information.

B When using ADCSHP = 0, no synchronisation of the trigger input is done.

Figure 16-1. ADC Block Diagram

16.2 ADC Operation

The ADC module is configured with user software. The setup and operation of the ADC is described in the following sections.

16.2.1 ADC Core

The ADC core converts an analog input to its 10-bit or 12-bit digital representation and stores the result in the conversion register ADCMEM0. 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 (03FFh for 10-bit, and 0FFFh for 12-bit) when the input signal is equal to or higher than V_{R+}. The output 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. The conversion formula for the ADC result N_{ADC} is:

$$\text{10-bit:} \quad N_{ADC} = 1023 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

$$\text{12-bit:} \quad N_{ADC} = 4095 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

The ADC core is configured by the control registers ADCCTL0, ADCCTL1, and ADCCTL2. The core is enabled with the ADCON bit. The ADC can be turned off when not in use to save power. With few exceptions, the ADC control bits can be modified only when ADCENC = 0. ADCENC must be set to 1 before any conversion can take place.

16.2.1.1 Conversion Clock Selection

The ADCCLK is used as the conversion clock and also to generate the sampling period when the pulse sampling mode is selected. The ADC source clock is selected using the ADCSSELx bits. Possible ADCCLK sources are SMCLK, ACLK, and MODOSC. The input clock can be divided from 1 to 512 using the ADCDIVx bits and the ADCPDIVx bits.

MODOSC, generated in the clock system, is approximately 5 MHz but varies with individual devices, supply voltage, and temperature. See the device-specific data sheet for the specifications of the MODOSC.

The clock that is chosen to source ADCCLK must remain 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.

16.2.2 ADC Inputs and Multiplexer

The channel for conversion is selected by the analog input multiplexer from 12 external and 4 internal analog signals. The input multiplexer is a break-before-make type to reduce input-to-input noise injection that can result from channel switching (see Figure 16-2). 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 (AV_{SS}), so that the stray capacitance is grounded to eliminate crosstalk.

The ADC 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 conversions.

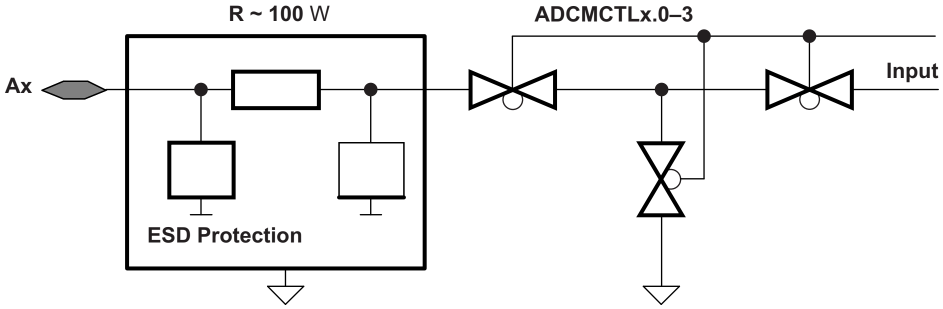


Figure 16-2. Analog Multiplexer

16.2.2.1 Analog Port Selection

The ADC 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 ADC function
```

16.2.3 Voltage Reference Generator

The ADC module can use either the on-chip reference voltage or an external reference voltage supplied on external pins.

See the device-specific data sheet for the specifications of the on-chip reference voltage.

External references may be supplied for V_{R+} and $V_{R–}$through pins VEREF+ and VEREF-, respectively.

16.2.3.1 Internal Reference Low-Power Features

The on-chip reference is designed for low-power applications. This reference includes a band-gap voltage source in the PMM module. The current consumption is specified in the device-specific data sheet. The ADC also contains an internal buffer for reference voltages. This buffer is automatically enabled when the internal reference is selected for V_{REF+}, and it is also optionally available for Ve_{REF+}. The on-chip reference from the PMM module must be enabled by software. Its settling time is ≤30 µs. See the PMM module chapter for more information on the on-chip reference.

The reference buffer of the ADC also has selectable speed versus power settings. When the maximum conversion rate is below 50 ksps, setting ADCSR = 1 reduces the current consumption of the buffer by approximately 50%.

16.2.4 Automatic Power Down

The ADC is designed for low-power applications. When the ADC is not actively converting, the core is automatically disabled, and it is automatically reenabled when needed. The MODOSC is also automatically enabled when needed and disabled when not needed.

16.2.5 Sample and Conversion Timing

An analog-to-digital conversion is initiated with a rising edge of the sample input signal SHI. The source for SHI is selected with the ADCSHSx bits and can be chosen from the following:

- ADCSC bit
- Three timer outputs

The polarity of the SHI signal source can be inverted with the ADCISSH bit. The SAMPCON signal controls the sample period and start of conversion. When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the analog-to-digital conversion, which requires 11 ADCCLK cycles in 10-bit resolution mode, and 13 ADCCLK cycles in 12-bit resolution mode. One additional ADCCLK cycle is needed to synchronize the input trigger source clock and ADC clock. The window comparator needs one additional ADCCLK cycle.

Two sample-timing methods are selected by control bit ADCSHP: extended sample mode and pulse mode.

16.2.5.1 Extended Sample Mode

The extended sample mode is selected when ADCSHP = 0. The SHI signal directly controls SAMPCON and defines the length of the sample period t_{sample}. When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the conversion after synchronization with ADCCLK (see 10-bit mode Figure 16-3 or 12-bit mode Figure 16-4). The SHI signal requires at least 4 ADCCLK cycles.

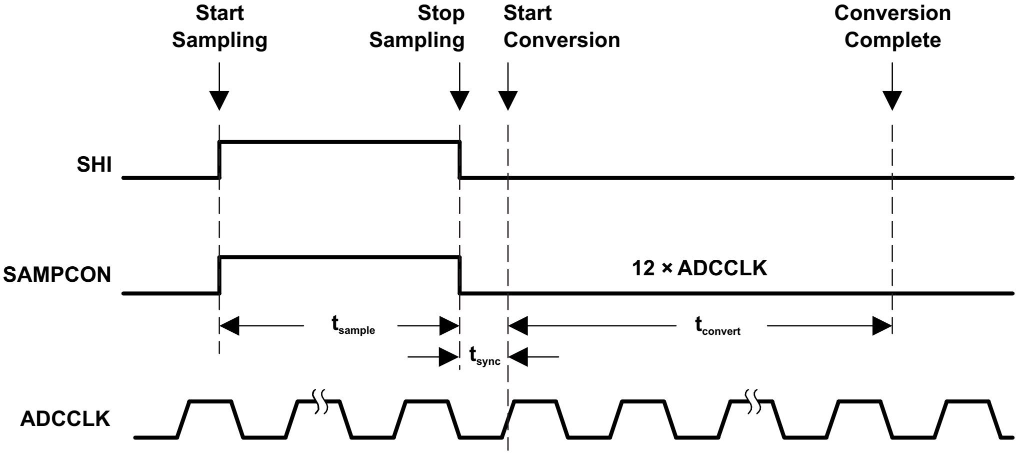


Figure 16-3. Extended Sample Mode in 10-bit mode

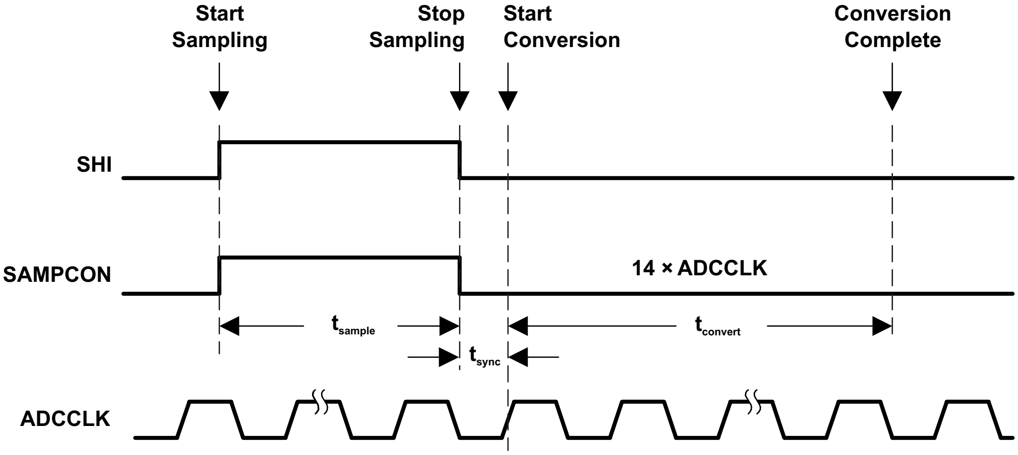


Figure 16-4. Extended Sample Mode in 12-bit mode

16.2.5.2 Pulse Sample Mode

The pulse sample mode is selected when ADCSHP = 1. The SHI signal triggers the sampling timer. The ADCSHTx bits in the ADCCTL0 register control the interval of the sampling timer that defines the SAMPCON sample period t_{sample}. The sampling timer keeps SAMPCON high after synchronization with ADCCLK for a programmed interval, t_{sample}. The total sampling time is t_{sample} plus t_{sync} (see 10-bit mode Figure 16-5 or 12-bit mode Figure 16-6).

The ADCSHTx bits select the sampling time in 4× multiples of ADCCLK. The ADCSC bit is automatically cleared if it is used as the sample-and-hold source in this mode.

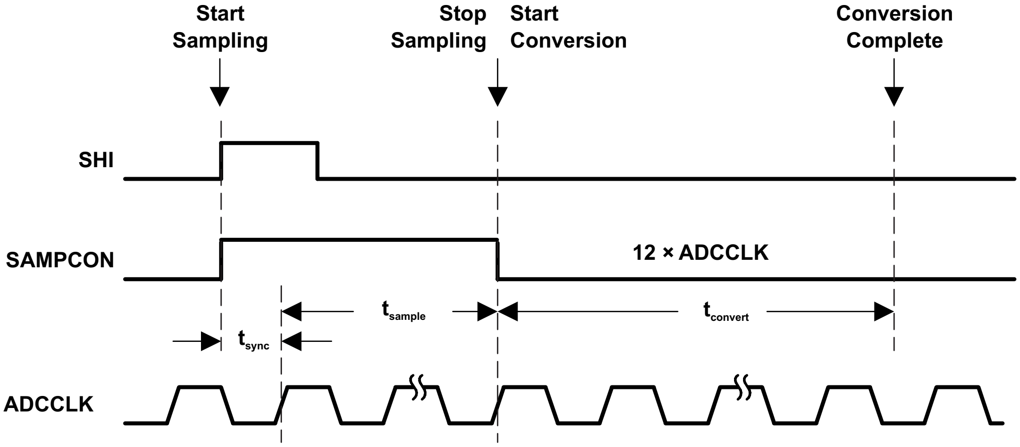


Figure 16-5. Pulse Sample Mode in 10-bit mode

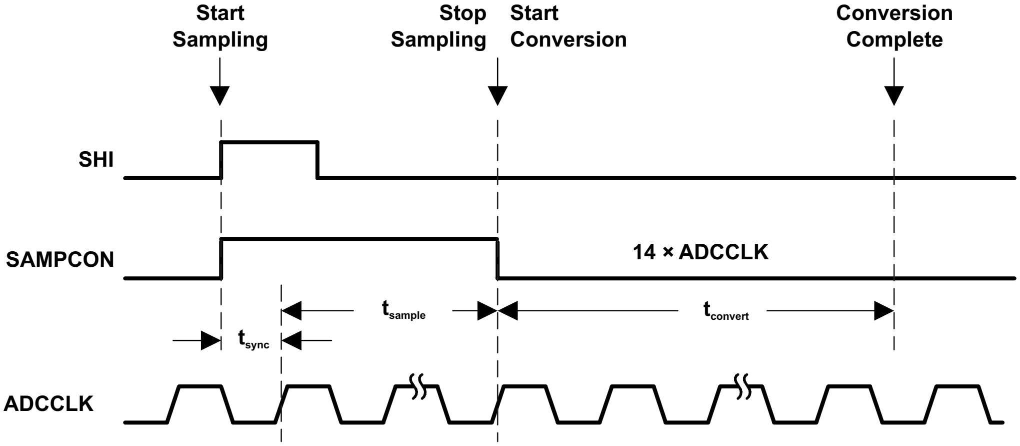
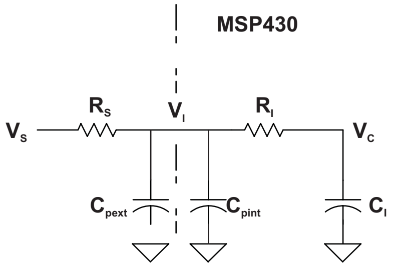


Figure 16-6. Pulse Sample Mode in 12-bit mode

16.2.5.3 Sample Timing Considerations

When SAMPCON = 0, all Ax inputs are high impedance. When SAMPCON = 1, the selected Ax input can be modeled as an RC low-pass filter during the sampling time t_{sample} (see Figure 16-7). An internal MUX-on input resistance R_I (see device-specific data sheet) in series with capacitor C_I (see 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 10-bit or 12-bit conversion.



V_I = Input voltage at pin Ax
V_S = External source voltage
R_S = External source resistance
R_I = Internal MUX-on input resistance
C_I = Input capacitance
C_{pint} = Parasitic capacitance, internal
C_{Pext} = Parasitic capacitance, external
V_C = Capacitance-charging voltage

Figure 16-7. Analog Input Equivalent Circuit

The resistance of the sources (R_S and R_I) affects t_{sample}. See the device-specific data sheet for the t_{sample} limits.

16.2.6 Conversion Result

For all conversion modes, the conversion result is accessible by reading the ADCMEM0 register. When a conversion result is written to ADCMEM0, the ADCIFG0 interrupt flag is set.

16.2.7 ADC Conversion Modes

The ADC has four operating modes, selected by the CONSEQx bits (see Table 16-1).

Table 16-1. Conversion Mode Summary

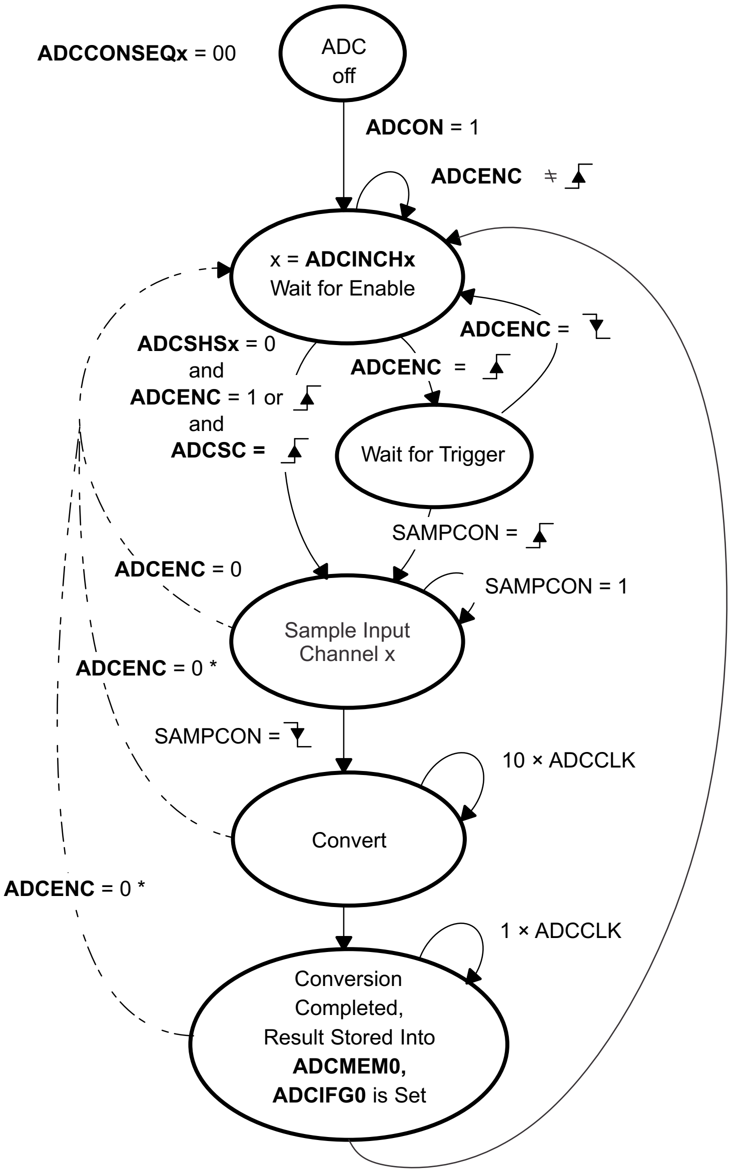
ADCCONSEQx	Mode	Operation
00	Single-channel single-conversion	A single channel is converted once.
01	Sequence-of-channels	A sequence of channels is converted once.
10	Repeat-single-channel	A single channel is converted repeatedly.
11	Repeat-sequence-of-channels	A sequence of channels is converted repeatedly.

16.2.7.1 Single-Channel Single-Conversion Mode

A single channel selected by ADCINCHx is sampled and converted once. The ADC result is written to ADCMEM0. 10-bit mode Figure 16-8 or 12-bit mode Figure 16-9 shows the flow of the single-channel single-conversion mode.

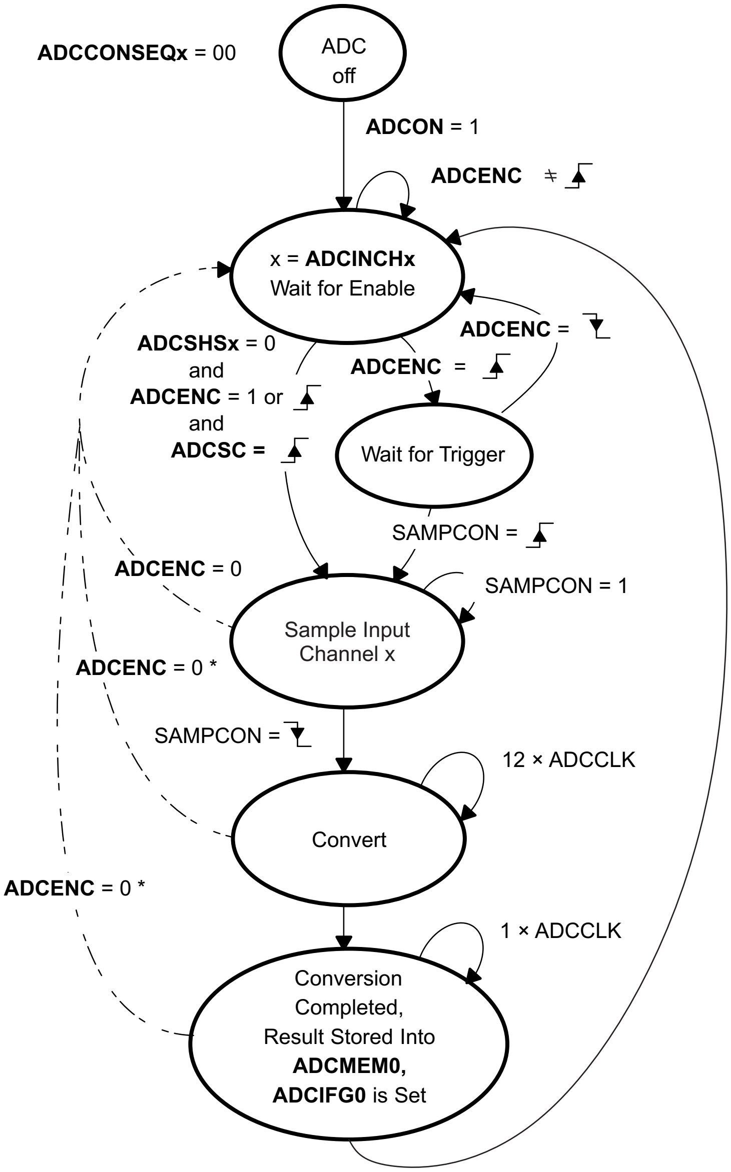
When ADCSC triggers a conversion, successive conversions can be triggered by the ADCSC bit. When any other trigger source is used, ADCENC must be toggled between each conversion.

Resetting ADCON bit within a conversion causes the ADC to go back into "ADC off" state. In this case, the value of the conversion register and the value of the interrupt flags is unpredictable.



* = Conversion result is unpredictable
x = Pointer to the selected ADC channel defined by **ADCINCHx**
All bit and register names are in bold font; signals are in normal font.

Figure 16-8. Single-Channel Single-Conversion Mode in 10-bit mode



* = Conversion result is unpredictable
x = Pointer to the selected ADC channel defined by **ADCINCHx**
All bit and register names are in bold font; signals are in normal font.

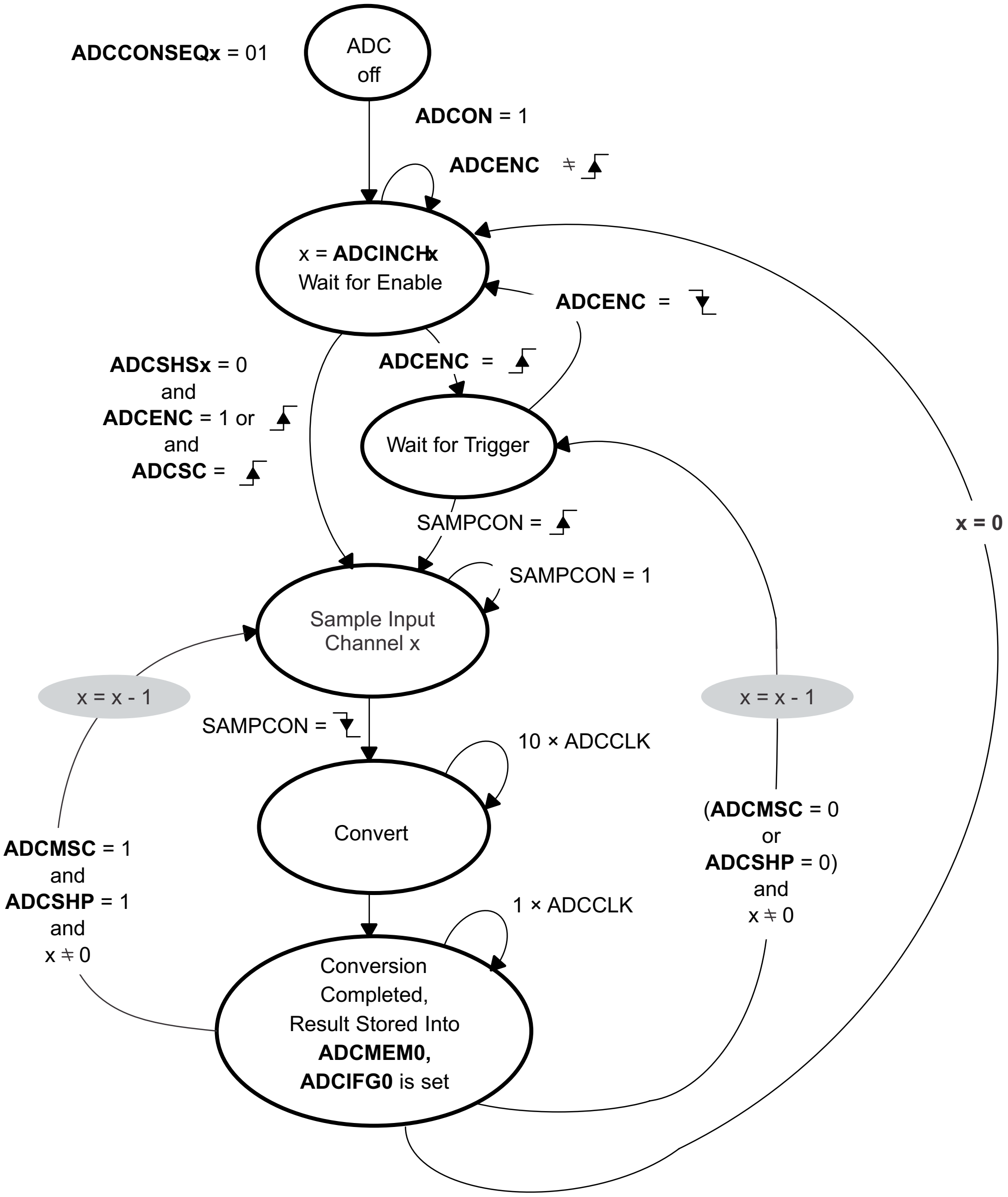
Figure 16-9. Single-Channel Single-Conversion Mode in 12-bit mode

16.2.7.2 Sequence-of-Channels Mode

A sequence of channels is sampled and converted once. The sequence begins with the channel selected by the ADCINCHx bits and decrements to channel A0. Each ADC result is written to ADCMEM0. The sequence stops after conversion of channel A0. 10-bit mode Figure 16-10 or 12-bit mode Figure 16-11 shows the sequence-of-channels mode.

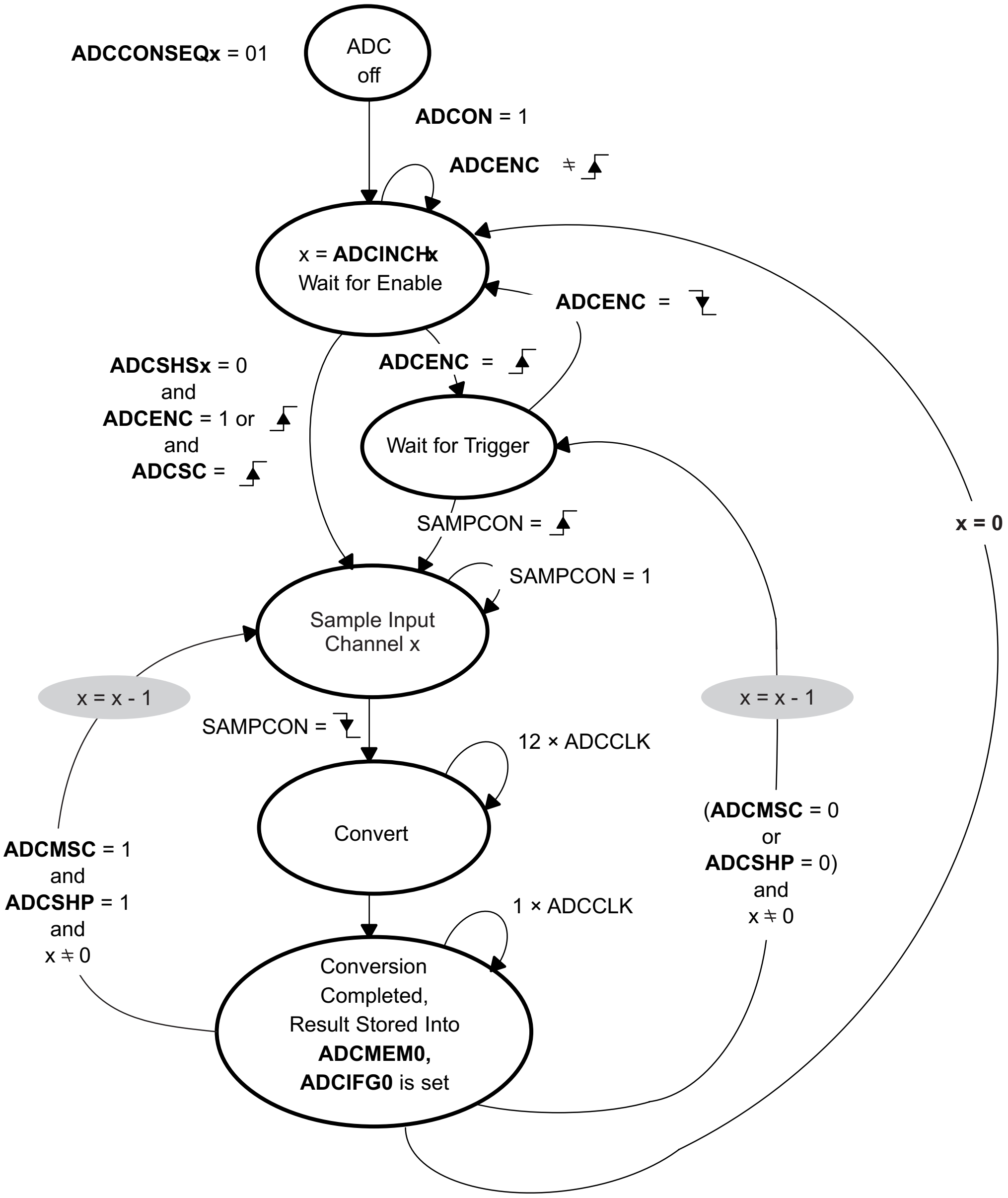
When ADCSC triggers a sequence, successive sequences can be triggered by the ADCSC bit. When any other trigger source is used, ADCENC must be toggled between each sequence.

As in all conversion modes, resetting ADCON bit within a conversion causes the ADC to go back into "ADC off" state.



x = Input channel Ax
All bit and register names are in bold font; signals are in normal font.

Figure 16-10. Sequence-of-Channels Mode in 10-bit mode

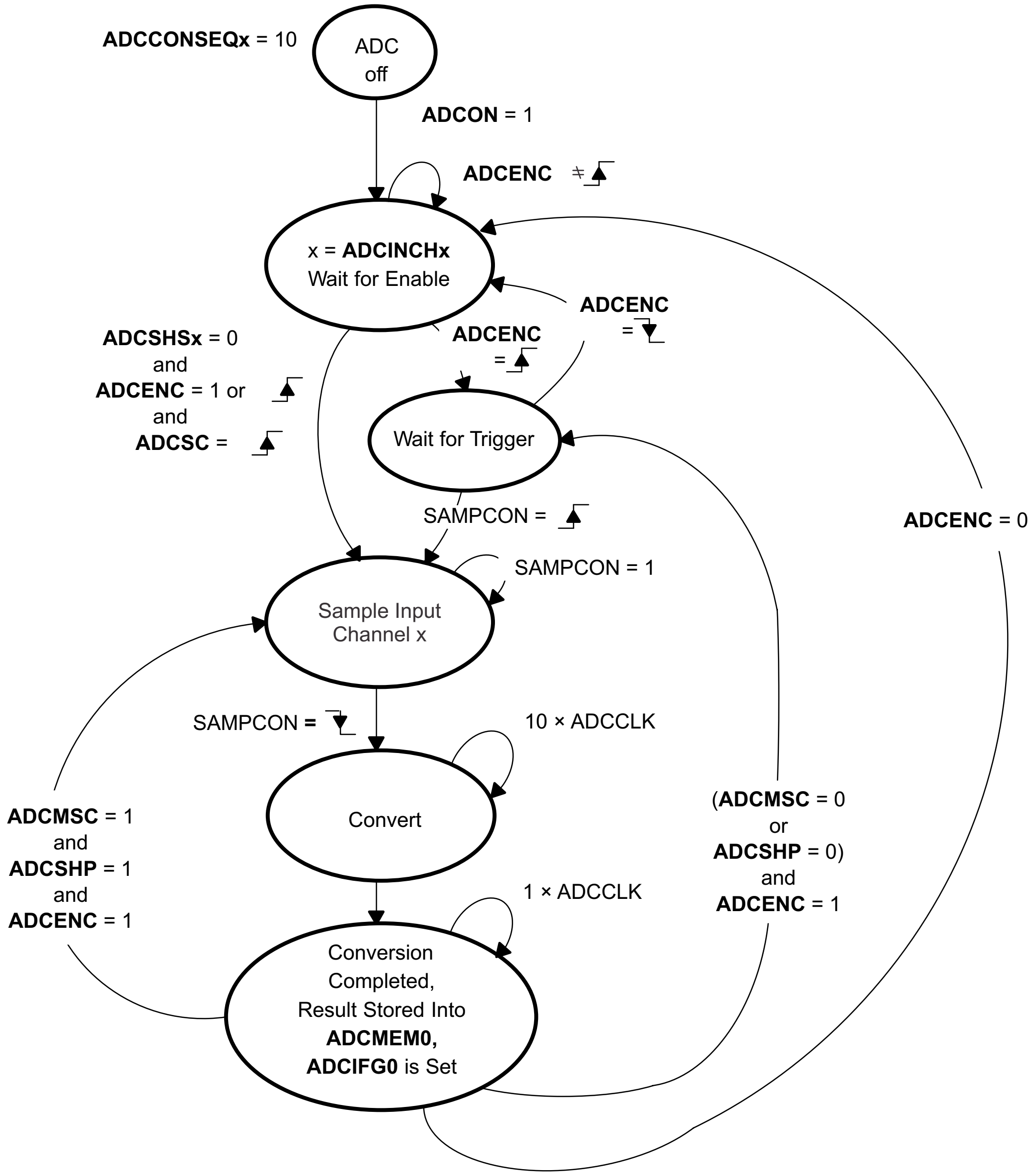


x = Input channel Ax
All bit and register names are in bold font; signals are in normal font.

Figure 16-11. Sequence-of-Channels Mode in 12-bit mode

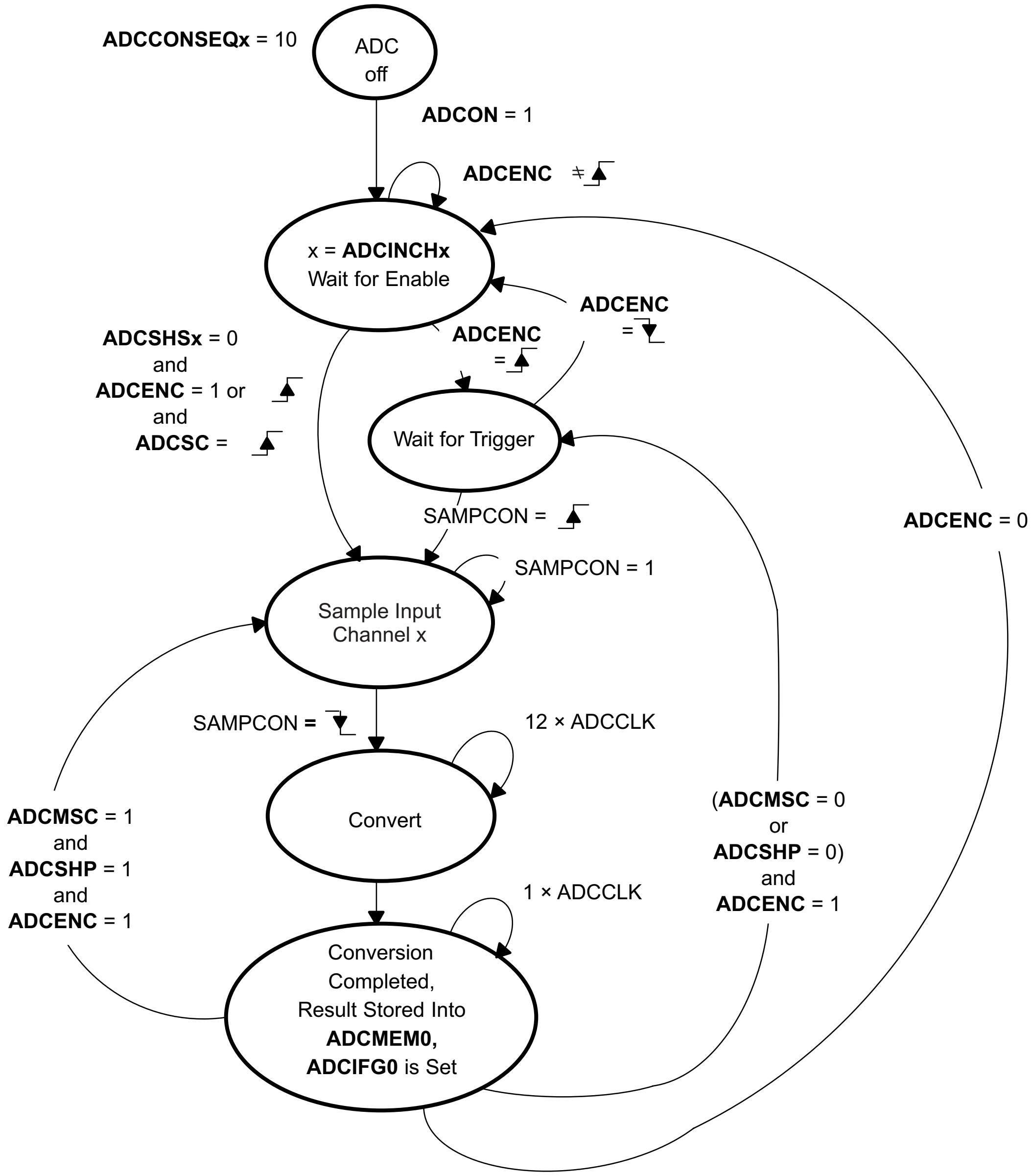
16.2.7.3 Repeat-Single-Channel Mode

A single channel selected by ADCINCHx is sampled and converted continuously. Each ADC result is written to ADCMEM0. 10-bit mode Figure 16-12 or 12-bit mode Figure 16-13 shows the repeat-single-channel mode.



x = Pointer to the selected ADC channel defined by **ADCINCHx**
All bit and register names are in bold font; signals are in normal font.

Figure 16-12. Repeat-Single-Channel Mode in 10-bit mode

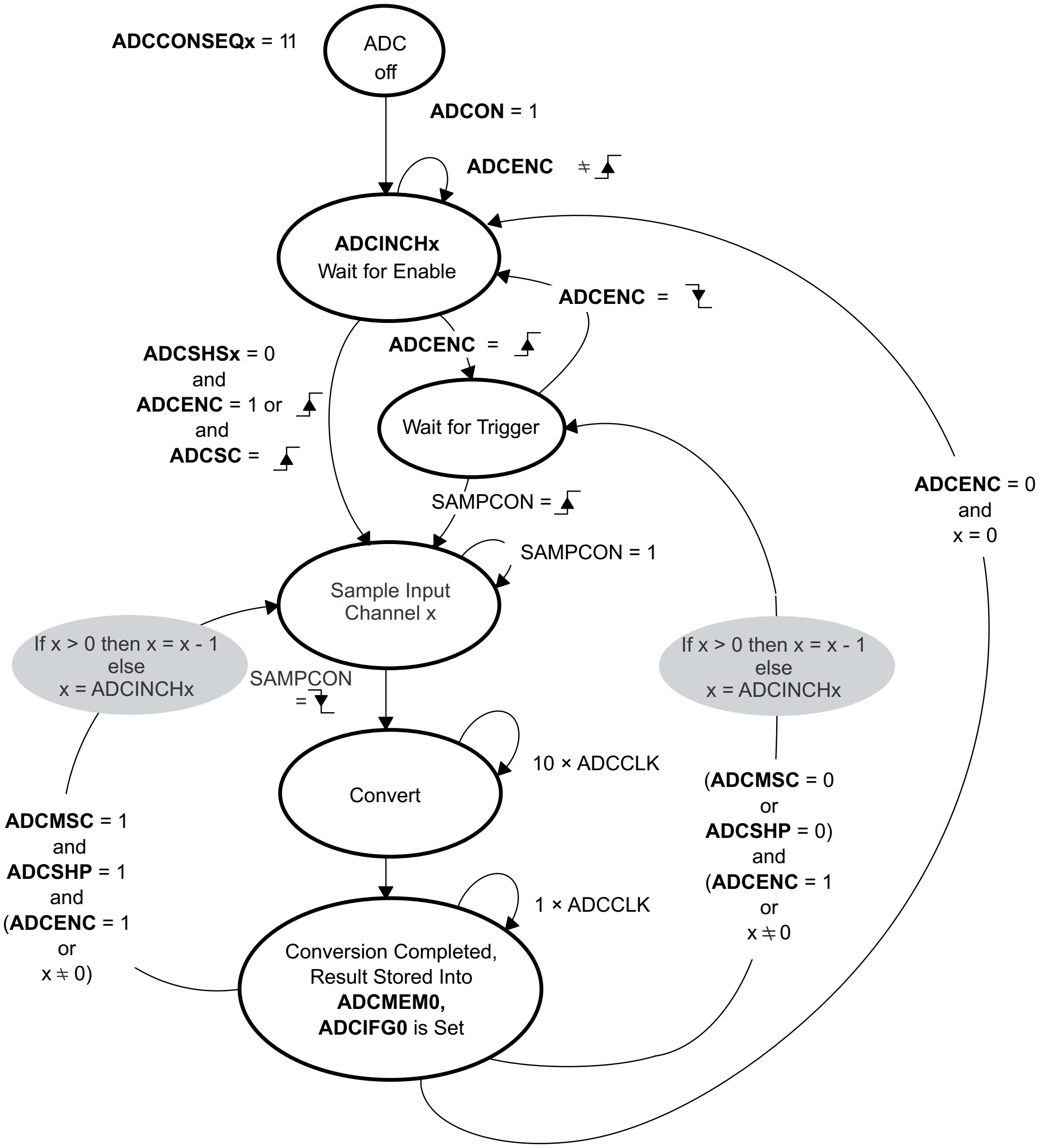

ADCCONSEQx = 10
ADC off
ADCON = 1
ADCENC ≠
x = ADCINCHx Wait for Enable
ADCENC =
ADCENC =
ADCSHSx = 0 and ADCENC = 1 or and ADCSC =
Wait for Trigger
SAMPCON =
SAMPCON = 1
ADCENC = 0
Sample Input Channel x
SAMPCON =
12 × ADCCLK
Convert
ADCMSC = 1 and ADCSHP = 1 and ADCENC = 1
(ADCMSC = 0 or ADCSHP = 0) and ADCENC = 1
1 × ADCCLK
Conversion Completed, Result Stored Into ADCMEM0, ADCIFG0 is Set

x = Pointer to the selected ADC channel defined by **ADCINCHx**
All bit and register names are in bold font; signals are in normal font.

Figure 16-13. Repeat-Single-Channel Mode in 12-bit mode

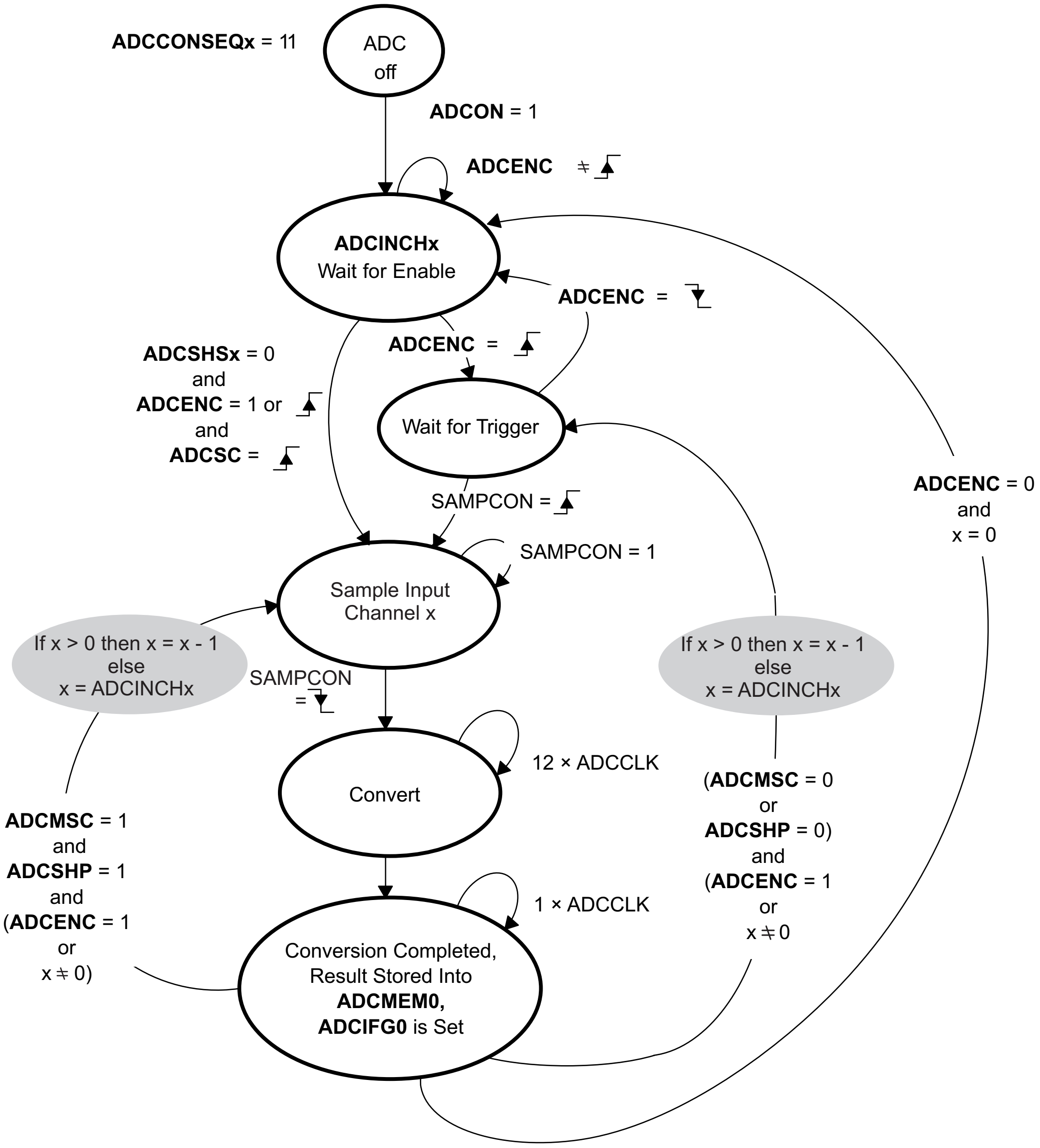
16.2.7.4 Repeat-Sequence-of-Channels Mode

A sequence of channels is sampled and converted repeatedly. The sequence begins with the channel selected by ADCINCHx and decrements to channel A0. Each ADC result is written to ADCMEM0. The sequence ends after conversion of channel A0, and the next trigger signal restarts the sequence. 10-bit mode Figure 16-14 or 12-bit mode Figure 16-15 shows the repeat-sequence-of-channels mode.


ADCCONSEQx = 11
ADC off
ADCON = 1
ADCENC ≠ ↑
ADCINCHx Wait for Enable
ADCENC = ↓
ADCENC = ↑
ADCSHSx = 0 and **ADCENC** = 1 or ↑ and **ADCSC** = ↑
Wait for Trigger
SAMPCON = ↑
SAMPCON = 1
Sample Input Channel x
If x > 0 then x = x - 1 else x = ADCINCHx
SAMPCON = ↓
If x > 0 then x = x - 1 else x = ADCINCHx
ADCENC = 0 and x = 0
Convert
10 × ADCCLK
ADCMSC = 1 and **ADCSHP** = 1 and (**ADCENC** = 1 or x ≠ 0)
(**ADCMSC** = 0 or **ADCSHP** = 0) and (**ADCENC** = 1 or x ≠ 0
1 × ADCCLK
Conversion Completed, Result Stored Into **ADCMEM0**, **ADCIFG0** is Set

x = Input channel Ax
All bit and register names are in bold font; signals are in normal font.

Figure 16-14. Repeat-Sequence-of-Channels Mode in 10-bit mode


ADCCONSEQx = 11
ADC off
ADCON = 1
ADCENC ≠
ADCINCHx Wait for Enable
ADCENC =
ADCENC =
ADCSHSx = 0 and ADCENC = 1 or and ADCSC =
Wait for Trigger
SAMPCON =
SAMPCON = 1
ADCENC = 0 and x = 0
Sample Input Channel x
If x > 0 then x = x - 1 else x = ADCINCHx
SAMPCON =
If x > 0 then x = x - 1 else x = ADCINCHx
Convert
12 × ADCCLK
ADCMSC = 1 and ADCSHP = 1 and (ADCENC = 1 or x ≠ 0)
(ADCMSC = 0 or ADCSHP = 0) and (ADCENC = 1 or x ≠ 0
1 × ADCCLK
Conversion Completed, Result Stored Into ADCMEM0, ADCIFG0 is Set

x = Input channel Ax
All bit and register names are in bold font; signals are in normal font.

Figure 16-15. Repeat-Sequence-of-Channels Mode in 12-bit mode

16.2.7.5 Using the Multiple Sample and Convert (ADCMSC) Bit

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is available. When ADCMSC = 1, CONSEQx > 0, and the sample timer is used, the first rising edge of the SHI signal triggers the first conversion. Successive conversions are triggered automatically as soon as the prior conversion is completed. Additional rising edges on SHI are ignored until the sequence is completed in the single-sequence mode, or until the ADCENC bit is toggled in repeat-single-channel or repeated-sequence modes. The function of the ADCENC bit is unchanged when using the ADCMSC bit.

16.2.7.6 Stopping Conversions

Stopping ADC activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Setting ADCENC = 0 in single-channel single-conversion mode stops a conversion immediately and the results are unpredictable. For correct results, poll the busy bit until reset before clearing ADCENC.
- Setting ADCENC = 0 during repeat-single-channel operation stops the converter at the end of the current conversion.
- Setting ADCENC = 0 during a sequence or repeat-sequence mode stops the converter at the end of the sequence.
- Any conversion mode can be stopped immediately by setting CONSEQx = 0 and setting ADCENC = 0. Conversion data are unreliable.

16.2.7.7 Window Comparator

The window comparator allows the ADC to monitor analog signals without any CPU interaction. The window comparator triggers the following interrupt flags:

- The ADCLO interrupt flag (ADCLOIFG) is set if the current result of the ADC conversion is below the low threshold defined in register ADCLO.
- The ADCHI interrupt flag (ADCHIIFG) is set if the current result of the ADC conversion is greater than the high threshold defined in register ADCHI.
- The ADCIN interrupt flag (ADCINIFG) is set if the current result of the ADC conversion is between the low threshold defined in register ADCLO and the high threshold defined in ADCHI.

These interrupts are generated independent of the selected conversion mode.

The values in the ADCHI and ADCLO registers must be in the correct data format. For example, if the binary data format is selected (ADCDF = 0), then the thresholds in the threshold registers ADCHI and ADCLO also must be binary coded. Changing the ADCDF or ADCRES bit resets the threshold registers.

The interrupt flags must be reset by software. The ADC updates the flags when a new value is available in the ADCMEM0. This update is only to set the corresponding interrupt flag. When the window comparator is used, software must reset the flags according to the application needs.

16.2.7.8 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, select the analog input channel ADCINCHx = 1100b. Any other configuration is done as if an external channel were selected, including reference selection, conversion-mode selection, and all other settings. The temperature sensor must be activated by software.

Figure 16-16 shows the typical temperature sensor transfer function. When using the temperature sensor, the sample period must be greater than 30 µs. The temperature sensor offset error can be large and must be calibrated for most applications (see the device-specific data sheet for parameters).

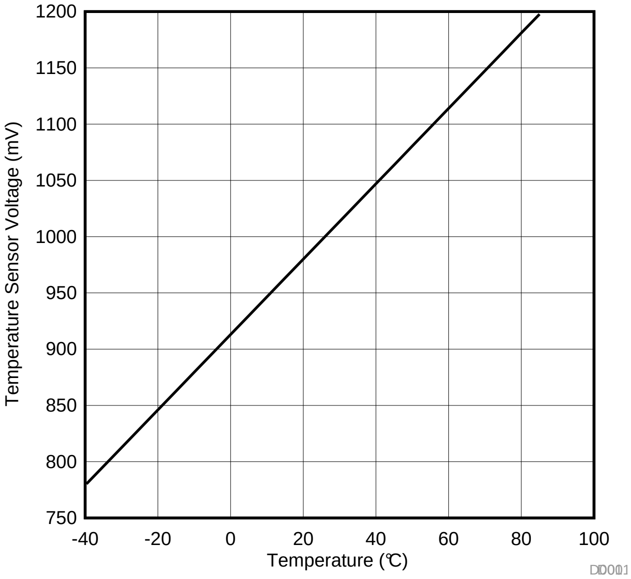


Figure 16-16. Typical Temperature Sensor Transfer Function

16.2.7.9 ADC Grounding and Noise Considerations

As with any high-resolution ADC, appropriate printed-circuit-board layout and grounding techniques must 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. Figure 16-17 shows connections that can prevent ground loops.

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. TI recommends a noise-free design using separate analog and digital ground planes with a single-point connection to achieve high accuracy.

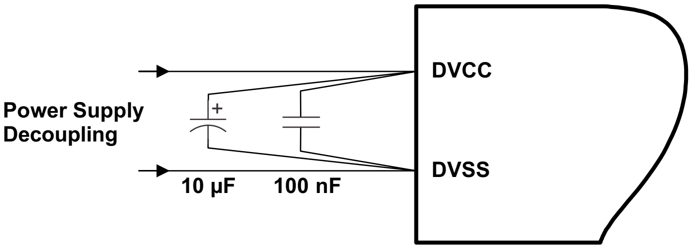


Figure 16-17. ADC Grounding and Noise Considerations

16.2.7.10 ADC Interrupts

The ADC has six interrupt sources:

- ADCIFG0: conversion ready interrupt

  The ADCIFG0 bit is set when the ADCMEM0 memory register is loaded with the conversion result. An interrupt request is generated if the ADCIE0 bit and the GIE bit are set.
- ADCOVIFG: ADCMEM0 overflow

  The ADCOV condition occurs when a conversion result is written to the ADCMEM0 before its previous conversion result was read.
- ADCTOVIFG: ADC conversion-time overflow

  The ADCTOV condition is generated when another sample-and-conversion is requested before the current conversion is completed.
- ADCLOIFG, ADCINIFG, ADCHIIFG: window comparator interrupt flags

  The window comparator interrupt flags are set corresponding to the description in the Window Comparator section (see Section 16.2.7.7).

16.2.7.10.1 ADCIV, Interrupt Vector Generator

All ADC interrupt sources are prioritized and combined to source a single interrupt vector. Read the interrupt vector register ADCIV to determine which ADC interrupt source requested an interrupt.

The highest-priority enabled ADC interrupt generates a number in the ADCIV register (see Section 16.3.13). This number can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine (see Section 16.2.7.10.2). Disabled ADC interrupts do not affect the ADCIV value.

Read access of the ADCIV register automatically resets the highest-pending interrupt condition and flag. Only the ADCIFG0 is not reset by this ADCIV read access. ADCIFG0 is automatically reset by reading the ADCMEM0 register or may be reset with software.

Write access to the ADCIV 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 ADCOV, ADCHIIFG, and ADCIFG0 interrupts are pending when the interrupt service routine accesses the ADCIV register, the highest priority interrupt (ADCOV interrupt condition) is reset automatically. After the RETI instruction of the interrupt service routine is executed, the ADCHIIFG generates another interrupt.

16.2.7.10.2 ADC Interrupt Handling Software Example

The following example shows the recommended use of the ADCIV. The ADCIV value is added to the PC to automatically jump to the appropriate routine.

ADCIFG0, ADCTOV, and ADCOV: 16 cycles

```
; Interrupt handler for ADC.
INT_ADC                            ; Enter Interrupt Service Routine
 ADD   &ADCIV,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: ADC window comparator high
interrupt
 JMP   ADLO                            ; Vector  8: ADC window comparator low interrupt
 JMP   ADIN                            ; Vector 10: ADC window comparator in interrupt
;
; Handler for ADCIFG0 starts here. No JMP required.
;
ADMEM  MOV &ADCMEM0,xxx              ; Move result, flag is reset
       ...                             ; Other instruction needed?
       RETI                            ; Return ;
ADOV   ...                             ; Handle ADCMEM0 overflow
       RETI                            ; Return ;
ADTOV  ...                             ; Handle Conv. time 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
```

16.3 ADC Registers

The ADC registers are listed in Table 16-2. The base address of the ADC can be found in the device-specific data sheet. The address offset of each ADC register is given in Table 16-2.

Table 16-2. ADC Registers

Offset	Acronym	Register Name	Type	Reset	Section
00h	ADCCTL0	ADC Control 0 register	Read/write	0100h	Section 16.3.1
02h	ADCCTL1	ADC Control 1 register	Read/write	0000h	Section 16.3.2
04h	ADCCTL2	ADC Control 2 register	Read/write	0010h	Section 16.3.3
06h	ADCLO	ADC Window Comparator Low Threshold register	Read/write	0000h	Section 16.3.9
08h	ADCHI	ADC Window Comparator High Threshold register	Read/write	03FFh	Section 16.3.7
0Ah	ADCMCTL0	ADC Memory Control register	Read/write	00h	Section 16.3.6
12h	ADCMEM0	ADC Conversion Memory register	Read/write	undefined	Section 16.3.4
1Ah	ADCIE	ADC Interrupt Enable register	Read/write	0000h	Section 16.3.11
1Ch	ADCIFG	ADC Interrupt Flag register	Read/write	0000h	Section 16.3.12
1Eh	ADCIV	ADC Interrupt Vector register	Read/write	0000h	Section 16.3.13

16.3.1 ADCCTL0 Register

ADC Control Register 0

Figure 16-18. ADCCTL0 Register

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

▨ Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active.

Table 16-3. ADCCTL0 Register Description

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11-8	ADCSHTx	RW	1h	ADC sample-and-hold time. These bits define the number of ADCCLK cycles in the sampling period for the ADC. Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0000b = 4 ADCCLK cycles 0001b = 8 ADCCLK cycles 0010b = 16 ADCCLK cycles 0011b = 32 ADCCLK cycles 0100b = 64 ADCCLK cycles 0101b = 96 ADCCLK cycles 0110b = 128 ADCCLK cycles 0111b = 192 ADCCLK cycles 1000b = 256 ADCCLK cycles 1001b = 384 ADCCLK cycles 1010b = 512 ADCCLK cycles 1011b = 768 ADCCLK cycles 1100b = 1024 ADCCLK cycles 1101b = 1024 ADCCLK cycles 1110b = 1024 ADCCLK cycles 1111b = 1024 ADCCLK cycles
7	ADCMSC	RW	0h	ADC multiple sample-and-conversion. Valid only for sequence or repeated modes. Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0b = The sampling timer requires a rising edge of the SHI signal to trigger each sample-and-convert. 1b = 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	0h	Reserved. Always reads as 0.
4	ADCON	RW	0h	ADC on Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0b = ADC off 1b = ADC on
3-2	Reserved	R	0h	Reserved. Always reads as 0.

Table 16-3. ADCCTL0 Register Description (continued)

Bit	Field	Type	Reset	Description
1	ADCENC	RW	0h	ADC enable conversion 0b = ADC disabled 1b = ADC enabled
0	ADCSC	RW	0h	ADC start conversion. Software-controlled sample-and-conversion start. ADCSC and ADCENC may be set together with one instruction. ADCSC is reset automatically. 0b = No sample-and-conversion-start 1b = Start sample-and-conversion

16.3.2 ADCCTL1 Register

ADC Control Register 1

Figure 16-19. ADCCTL1 Register

15	14	13	12	11	10	9	8
Reserved				ADCSHSx		ADCSHP	ADCISSH
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADCDIVx			ADCSSELx		ADCCONSEQx		ADCBUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

▨ Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active.

Table 16-4. ADCCTL1 Register Description

Bit	Field	Type	Reset	Description
15-12	Reserved	R	0h	Reserved. Always reads as 0.
11-10	ADCSHSx	RW	0h	ADC sample-and-hold source select Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 00b = ADCSC bit 01b = Timer trigger 0 (see device-specific data sheet) 10b = Timer trigger 1 (see device-specific data sheet) 11b = Timer trigger 2 (see device-specific data sheet)
9	ADCSHP	RW	0h	ADC 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. Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0b = SAMPCON signal is sourced from the sample input signal. 1b = SAMPCON signal is sourced from the sampling timer.
8	ADCISSH	RW	0h	ADC invert signal sample-and-hold Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0b = The sample input signal is not inverted. 1b = The sample input signal is inverted.
7-5	ADCDIVx	RW	0h	ADC clock divider Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 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

Table 16-4. ADCCTL1 Register Description (continued)

Bit	Field	Type	Reset	Description
4-3	ADCSSELx	RW	0h	ADC clock source select Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 00b = MODCLK 01b = ACLK 10b = SMCLK 11b = SMCLK
2-1	ADCCONSEQx	RW	0h	ADC conversion sequence mode select Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 00b = Single-channel single-conversion 01b = Sequence-of-channels 10b = Repeat-single-channel 11b = Repeat-sequence-of-channels
0	ADCBUSY	R	0h	ADC busy. This bit indicates an active sample or conversion operation. 0b = No operation is active. 1b = A sequence, sample, or conversion is active**.**

16.3.3 ADCCTL2 Register

ADC Control Register 2

Figure 16-20. ADCCTL2 Register

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

▨ Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active.

Table 16-5. ADCCTL2 Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	R	0h	Reserved. Always reads as 0.
9-8	ADCPDIVx	RW	0h	ADC predivider. This bit pre-divides the selected ADC clock source before it gets divided again using ADCDIVx. 00b = Predivide by 1 01b = Predivide by 4 10b = Predivide by 64 11b = Reserved
7-6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	ADCRES	RW	1h	ADC resolution. This bit defines the conversion result resolution.[1] 00b = 8 bit (10 clock cycle conversion time) 01b = 10 bit (12 clock cycle conversion time) 10b = 12 bit (14 clock cycle conversion time) 11b = Reserved
3	ADCDF	RW	0h	ADC data read-back format. Data is always stored in the binary unsigned format. 0b = Binary unsigned. Theoretically the analog input voltage $-V_{REF}$ results in 0000h, the analog input voltage $+V_{REF}$ results in 03FFh. 1b = Signed binary (2s complement), left aligned. Theoretically the analog input voltage $-V_{REF}$ results in 8000h, the analog input voltage $+V_{REF}$ results in 7FC0h.
2	ADCSR	RW	0h	ADC sampling rate. This bit selects drive capability of the ADC reference buffer for the maximum sampling rate. Setting ADCSR reduces the current consumption of this buffer. 0b = ADC buffer supports up to approximately 200 ksps 1b = ADC buffer supports up to approximately 50 ksps
1	Reserved	R	0h	Reserved. Always reads as 0.
0	Reserved	RW	0h	Reserved. Must be written as 0.

[1] 10-bit or 12-bit setting, please refer to the device-specific data sheet for details.

16.3.4 ADCMEM0 Register

ADC Conversion Memory Register

Figure 16-21. ADCMEM0 Register

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

Table 16-6. ADCMEM0 Register Description

Bit	Field	Type	Reset	Description
15-0	Conversion_Results	RW	undefined	This data format is used if ADCDF = 0 (binary unsigned). The 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 12-bit mode, and bits 15-8 are 0 in 8-bit mode. Bit 9 is the MSB. Bits 15-10 are 0 in 10-bit mode, and bits 15-8 are 0 in 8-bit mode. Writing to the conversion memory register corrupts the results.

16.3.5 ADCMEM0 Register, 2s-Complement Format

ADC Conversion Memory Register, 2s-Complement Format

Figure 16-22. ADCMEM0 Register

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

Table 16-7. ADCMEM0 Register Description

Bit	Field	Type	Reset	Description
15-0	Conversion_Results	RW	undefined	This data format is used if ADCDF = 1 (2s complement). The 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. Bit 15 is the MSB. 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. Writing to the conversion memory register corrupts the results.

16.3.6 ADCMCTL0 Register

ADC Conversion Memory Control Register

Figure 16-23. ADCMCTL0 Register

15	14	13	12	11	10	9	8
Reserved							Reserved
r0	r0	r0	r0	r0	r0	r0	rw-(0)

7	6	5	4	3	2	1	0
Reserved	ADCSREFx			ADCINCHx			
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

▨ Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active.

Table 16-8. ADCMCTL0 Register Description

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0h	Reserved. Always reads as 0.
8	Reserved	RW	0h	Reserved.

6-4	ADCSREFx	RW	0h	Select reference. It is not recommended to change this setting while a conversion is ongoing.
				Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active.
				000b = {V_{R+} = AVCC and V_{R-} = AVSS }
				001b = {V_{R+} = VREF and V_{R-} = AVSS}
				010b = {V_{R+} = VEREF+ buffered and V_{R-} = AVSS}
				011b = {V_{R+} = VEREF+ and V_{R-} = AVSS }
				100b = {V_{R+} = AVCC and V_{R-} = VEREF-}
				101b = {V_{R+} = VREF and V_{R-} = VEREF-}
				110b = {V_{R+} = VEREF+ buffered and V_{R-} = VEREF-}
				111b = {V_{R+} = VEREF+ and V_{R-} = VEREF-}

3-0	ADCINCHx	RW	0h	Input channel select. Writing these bits select the channel for a single-conversion or the highest channel for a sequence of conversions. Reading these bits in ADCCONSEQ = 01,11 returns the channel currently converted. Can be modified only when ADCENC = 0. Resetting ADCENC = 0 by software and changing these fields immediately shows an effect when a conversion is active. 0000b = A0 (see device-specific data sheet) 0001b = A1 (see device-specific data sheet) 0010b = A2 (see device-specific data sheet) 0011b = A3 (see device-specific data sheet) 0100b = A4 (see device-specific data sheet) 0101b = A5 (see device-specific data sheet) 0110b = A6 (see device-specific data sheet) 0111b = A7 (see device-specific data sheet) 1000b = A8 (see device-specific data sheet) 1001b = A9 (see device-specific data sheet) 1010b = A10 (see device-specific data sheet) 1011b = A11 (see device-specific data sheet) 1100b = A12 (see device-specific data sheet) 1101b = A13 (see device-specific data sheet) 1110b = A14 (see device-specific data sheet) 1111b = A15 (see device-specific data sheet)

16.3.7 ADCHI Register

ADC Window Comparator High Threshold Register

Figure 16-24. ADCHI Register

15	14	13	12	11	10	9	8
High_Threshold							
r0	r0	r0	r0	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 16-9. ADCHI Register Description

Bit	Field	Type	Reset	Description
15-0	High_Threshold	RW	3FFh	This data format is used when ADCDF = 0 (binary unsigned). The threshold value must be 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. Bit 9 is the MSB. Bits 15–10 are 0 in 10-bit mode, and bits 15–8 are 0 in 8-bit mode.

16.3.8 ADCHI Register, 2s-Complement Format

ADC Window Comparator High Threshold Register, 2s-Complement Format

Figure 16-25. ADCHI Register

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

Table 16-10. ADCHI Register Description

Bit	Field	Type	Reset	Description
15-0	High_Threshold	RW	1FFh	This data format is used when ADCDF = 1 (2s complement). The threshold value must be left justified. 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. Bit 15 is the MSB. Bits 5–0 are 0 in 10-bit mode, and bits 7–0 are 0 in 8-bit mode.

16.3.9 ADCLO Register

ADC Window Comparator Low Threshold Register

Figure 16-26. ADCLO Register

15	14	13	12	11	10	9	8
Low_Threshold							
r0	r0	r0	r0	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 16-11. ADCLO Register Description

Bit	Field	Type	Reset	Description
15-0	Low_Threshold	RW	0h	This data format is used if ADCDF = 0 (binary unsigned). The threshold value must be 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. Bit 9 is the MSB. Bits 15–10 are 0 in 10-bit mode, and bits 15–8 are 0 in 8-bit mode.

16.3.10 ADCLO Register, 2s-Complement Format

ADC Window Comparator Low Threshold Register, 2s-Complement Format

Figure 16-27. ADCLO Register

15	14	13	12	11	10	9	8
Low_Threshold							
rw-(1)	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)	r0	r0	r0	r0

Table 16-12. ADCLO Register Description

Bit	Field	Type	Reset	Description
15-0	Low_Threshold	RW	200h	This data format is used if ADCDF = 1 (2s complement). The threshold value must be left justified if 2s-complement format is chosen. 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. Bit 15 is the MSB. Bits 5-0 are 0 in 10-bit mode, and bits 7-0 are 0 in 8-bit mode.

16.3.11 ADCIE Register

ADC Interrupt Enable Register

Figure 16-28. ADCIE 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		ADCTOVIE	ADCOVIE	ADCHIIE	ADCLOIE	ADCINIE	ADCIE0
r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 16-13. ADCIE Register Description

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5	ADCTOVIE	RW	0h	ADC conversion-time-overflow interrupt enable. 0b = Conversion-time overflow interrupt disabled 1b = Conversion-time overflow interrupt enabled
4	ADCOVIE	RW	0h	ADCMEM0 overflow interrupt enable. 0b = Overflow interrupt disabled 1b = Overflow interrupt enabled
3	ADCHIIE	RW	0h	Interrupt enable for the above upper threshold interrupt of the window comparator. 0b = Above upper threshold interrupt disabled 1b = Above upper threshold interrupt enabled
2	ADCLOIE	RW	0h	Interrupt enable for the below lower threshold interrupt of the window comparator. 0b = Below lower threshold interrupt disabled 1b = Below lower threshold interrupt enabled
1	ADCINIE	RW	0h	Interrupt enable for the inside of window interrupt of the window comparator. 0b = Inside of window interrupt disabled 1b = Inside of window interrupt enabled
0	ADCIE0	RW	0h	Interrupt enable. This bits enable or disable the interrupt request for a completed ADC conversion. 0b = Interrupt disabled 1b = Interrupt enabled

16.3.12 ADCIFG Register

ADC Interrupt Flag Register

Figure 16-29. ADCIFG 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		ADCTOVIFG	ADCOVIFG	ADCHIIFG	ADCLOIFG	ADCINIFG	ADCIFG0
r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 16-14. ADCIFG Register Description

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5	ADCTOVIFG	RW	0h	The ADCTOVIFG is set when an ADC conversion is triggered before the actual conversion has completed. 0b = No interrupt pending 1b = Interrupt pending
4	ADCOVIFG	RW	0h	The ADCOVIFG is set when the ADCMEM0 register is written before the last conversion result has been read. 0b = No interrupt pending 1b = Interrupt pending
3	ADCHIIFG	RW	0h	The ADCHIIFG is set when the result of the current ADC conversion is greater than the upper threshold defined by the window comparator upper threshold register. 0b = No interrupt pending 1b = Interrupt pending
2	ADCLOIFG	RW	0h	The ADCLOIFG is set when the result of the current ADC conversion is below the lower threshold defined by the window comparator lower threshold register. 0b = No interrupt pending 1b = Interrupt pending
1	ADCINIFG	RW	0h	The ADCINIFG is set when the result of the current ADC conversion is within the thresholds defined by the window comparator threshold registers. 0b = No interrupt pending 1b = Interrupt pending
0	ADCIFG0	RW	0h	The ADCIFG0 is set when an ADC conversion is completed. This bit is reset when the ADCMEM0 get read, or it may be reset by software. 0b = No interrupt pending 1b = Interrupt pending

16.3.13 ADCIV Register

ADC Interrupt Vector Register

Figure 16-30. ADCIV Register

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

Table 16-15. ADCIV Register Description

Bit	Field	Type	Reset	Description
15-0	ADCIVx	R	0h	ADC 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: ADCMEM0 overflow; Interrupt Flag: ADCOVIFG; Interrupt Priority: Highest 04h = Interrupt Source: Conversion time overflow; Interrupt Flag: ADCTOVIFG 06h = Interrupt Source: ADCHI Interrupt flag; Interrupt Flag: ADCHIIFG 08h = Interrupt Source: ADCLO Interrupt flag; Interrupt Flag: ADCLOIFG 0Ah = Interrupt Source: ADCIN Interrupt flag; Interrupt Flag: ADCINIFG 0Ch = Interrupt Source: ADC memory Interrupt flag; Interrupt Flag: ADCIFG0; Interrupt Priority: Lowest

16.3.14 MSP430FR413x SYSCFG2 Register (absolute address = 0164h) [reset = 0000h]

System Configuration Register 2. In MSP430FR413x devices, the ADC pins are controlled by System Configuration Register 2. This is a mirror of the register description from SYS Chapter.

Figure 16-31. SYSCFG2 Register

15	14	13	12	11	10	9	8
Reserved			LCDPCTL	Reserved	Reserved	ADCPCTL9	ADCPCTL8
r0	r0	r0	rw-0	r0	r0	rw-0	rw-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADCPCTL7	ADCPCTL6	ADCPCTL5	ADCPCTL4	ADCPCTL3	ADCPCTL2	ADCPCTL1	ADCPCTL0
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 16-16. SYSCFG2 Register Description

Bit	Field	Type	Reset	Description
15-13	Reserved	R	0h	Reserved. Always read as 0.
12	LCDPCTL	RW	0h	LCD power pin (LCDCAP0, LCDCAP1, R13, R23, R33) control. 0b = LCD power pin disabled 1b = LCD power pin enabled
11-10	Reserved	R	0h	Reserved. Always read as 0.
9	ADCPCTL9	RW	0h	ADC input A9 pin select 0b = ADC input A9 disabled 1b = ADC input A9 enabled
8	ADCPCTL8	RW	0h	ADC input A8 pin select 0b = ADC input A8 disabled 1b = ADC input A8 enabled
7	ADCPCTL7	RW	0h	ADC input A7 pin select 0b = ADC input A7 disabled 1b = ADC input A7 enabled
6	ADCPCTL6	RW	0h	ADC input A6 pin select 0b = ADC input A6 disabled 1b = ADC input A6 enabled
5	ADCPCTL5	RW	0h	ADC input A5 pin select 0b = ADC input A5 disabled 1b = ADC input A5 enabled
4	ADCPCTL4	RW	0h	ADC input A4 pin select 0b = ADC input A4 disabled 1b = ADC input A4 enabled
3	ADCPCTL3	RW	0h	ADC input A3 pin select 0b = ADC input A3 disabled 1b = ADC input A3 enabled
2	ADCPCTL2	RW	0h	ADC input A2 pin select 0b = ADC input A2 disabled 1b = ADC input A2 enabled
1	ADCPCTL1	RW	0h	ADC input A1 pin select 0b = ADC input A1 disabled 1b = ADC input A1 enabled
0	ADCPCTL0	RW	0h	ADC input A0 pin select 0b = ADC input A0 disabled 1b = ADC input A0 enabled

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 describes the operation of the asynchronous UART mode.

Topic **Page**

17.1 Enhanced Universal Serial Communication Interface A (eUSCI_A) Overview 482
17.2 eUSCI_A Introduction – UART Mode 482
17.3 eUSCI_A Operation – UART Mode 484
17.4 eUSCI_A UART Registers 499

17.1 Enhanced Universal Serial Communication Interface A (eUSCI_A) Overview

The eUSCI_A module supports two serial communication modes:

- UART mode
- SPI mode

17.2 eUSCI_A Introduction – UART Mode

In asynchronous mode, the eUSCI_Ax modules connect the device to an external system through 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 no 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 automatic 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 17-1 shows the eUSCI_Ax when configured for UART mode.

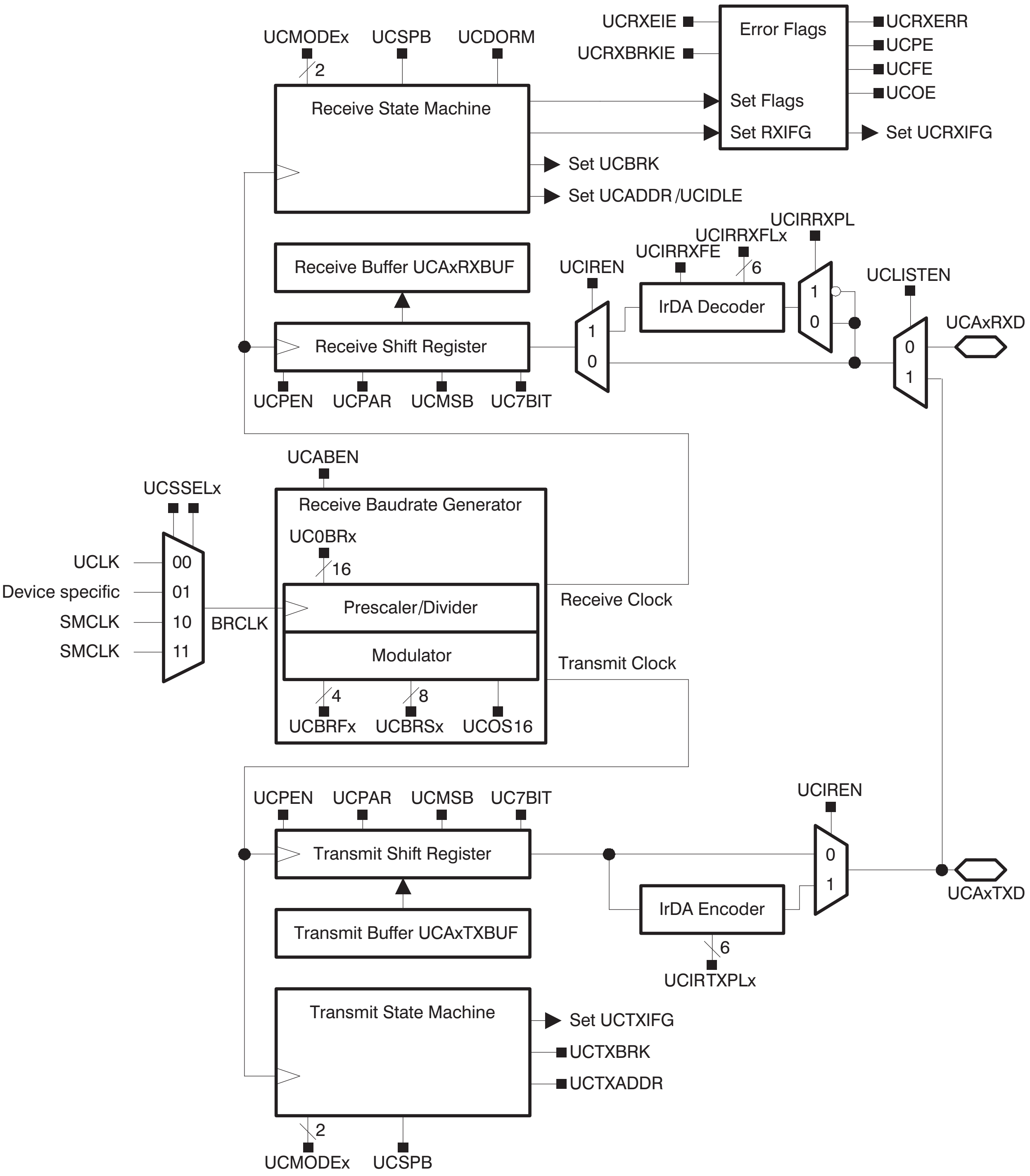


Figure 17-1. eUSCI_Ax Block Diagram – UART Mode (UCSYNC = 0)

17.3 eUSCI_A Operation – UART Mode

In UART mode, the eUSCI_A transmits and receives characters at a bit rate that is 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.

17.3.1 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.

To avoid unpredictable behavior, configure or reconfigure the eUSCI_A module when UCSWRST is set.

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 while UCSWRST = 1 (including UCAxCTL1).
3. Configure ports.
4. Clear UCSWRST by software (`BIC.B #UCSWRST,&UCAxCTL1`).
5. Enable interrupts (optional) using UCRXIE or UCTXIE.

17.3.2 Character Format

The UART character format (see Figure 17-2) 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.

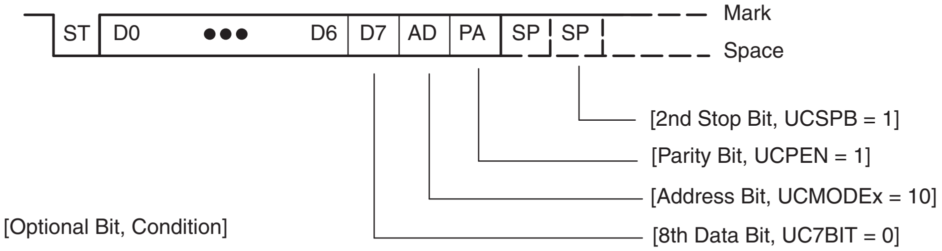

Mark
Space
ST
D0
●●●
D6
D7
AD
PA
SP
SP
[2nd Stop Bit, UCSPB = 1]
[Parity Bit, UCPEN = 1]
[Address Bit, UCMODEx = 10]
[Optional Bit, Condition]
[8th Data Bit, UC7BIT = 0]

Figure 17-2. Character Format

17.3.3 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.

17.3.3.1 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 17-3). 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.

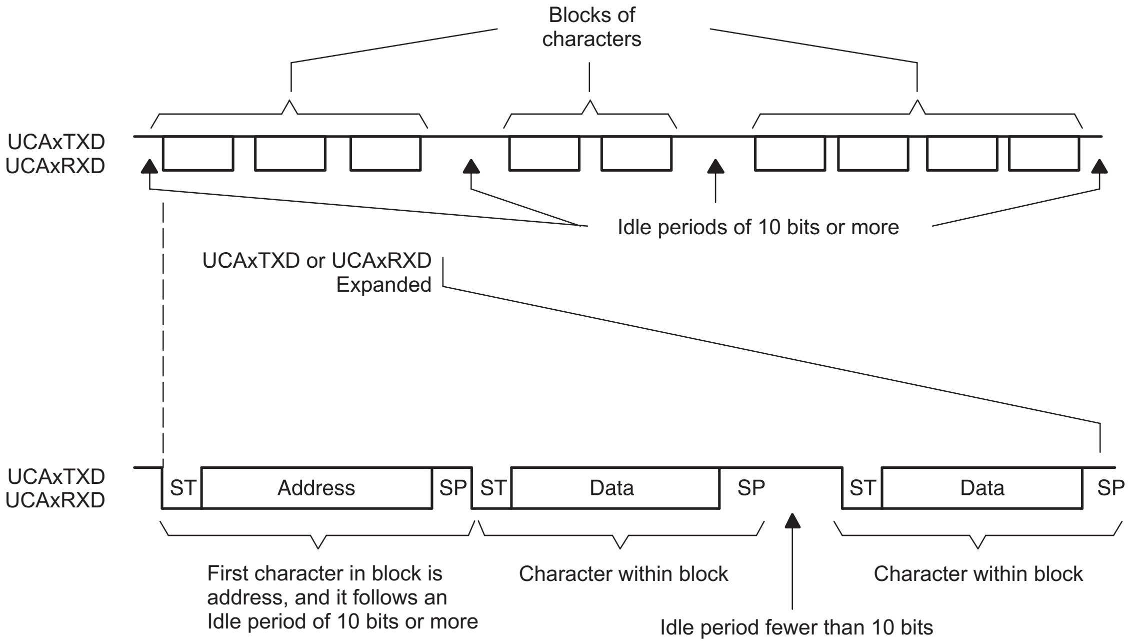

Blocks of characters
UCAxTXD
UCAxRXD
Idle periods of 10 bits or more
UCAxTXD or UCAxRXD Expanded
UCAxTXD
UCAxRXD
ST
Address
SP
ST
Data
SP
ST
Data
SP
First character in block is address, and it follows an Idle period of 10 bits or more
Character within block
Character within block
Idle period fewer than 10 bits

Figure 17-3. Idle-Line Format

The UCDORM bit is used to control data reception in the idle-line multiprocessor format. When UCDORM = 1, all nonaddress 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 completes. 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.

17.3.3.1.1 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.

17.3.3.2 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 17-4). 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.

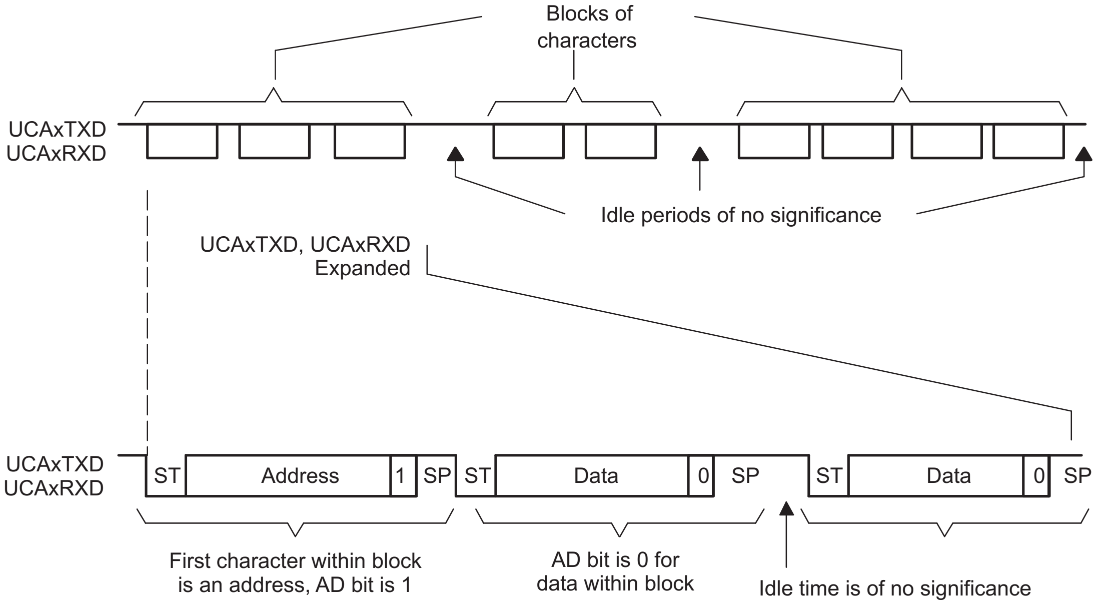


Figure 17-4. Address-Bit Multiprocessor Format

17.3.3.2.1 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.

17.3.4 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 17-5.

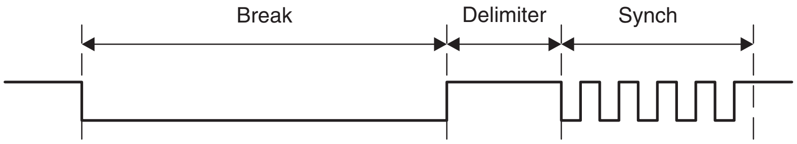


Figure 17-5. Auto Baud-Rate Detection – Break/Synch Sequence

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 17-6). 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.

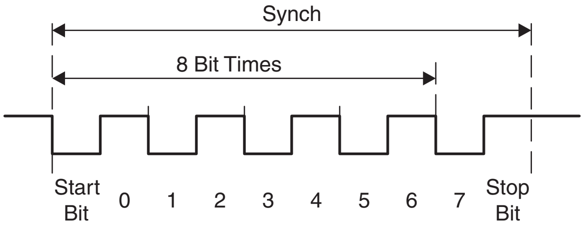


Figure 17-6. Auto Baud-Rate Detection – Synch Field

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 baud rate 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.

17.3.4.1 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.

17.3.5 IrDA Encoding and Decoding

When UCIREN is set, the IrDA encoder and decoder are enabled and provide hardware bit shaping for IrDA communication.

17.3.5.1 IrDA Encoding

The encoder sends a pulse for every zero bit in the transmit bit stream coming from the UART (see Figure 17-7). The pulse duration is defined by UCIRTXPLx bits specifying the number of one-half clock periods of the clock selected by UCIRTXCLK.

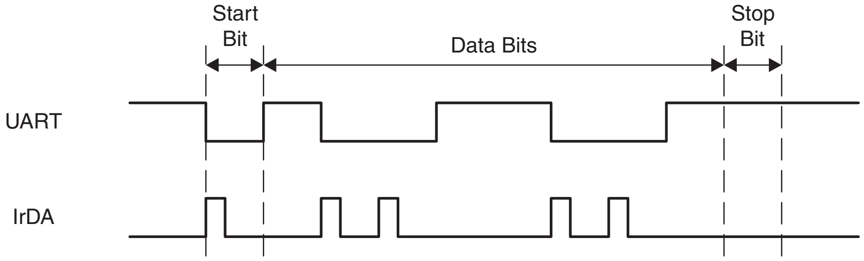


Figure 17-7. UART vs IrDA Data Format

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.

17.3.5.2 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.

17.3.6 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 17-1.

Table 17-1. 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.

17.3.7 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.

17.3.7.1 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 17-8 (see the device-specific data sheet for parameters). The deglitch time t_t can be set to four different values using the UCGLITx bits.

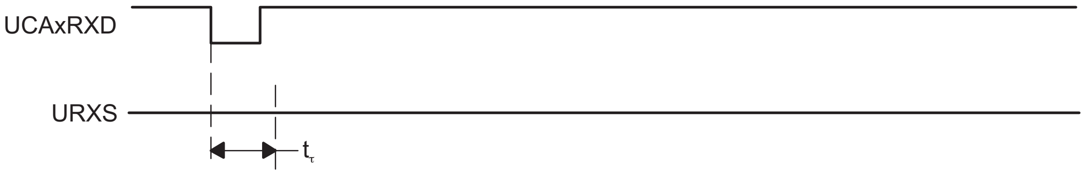


Figure 17-8. Glitch Suppression, eUSCI_A Receive Not Started

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 17-9). If the majority vote fails to detect a start bit, the eUSCI_A halts character reception.

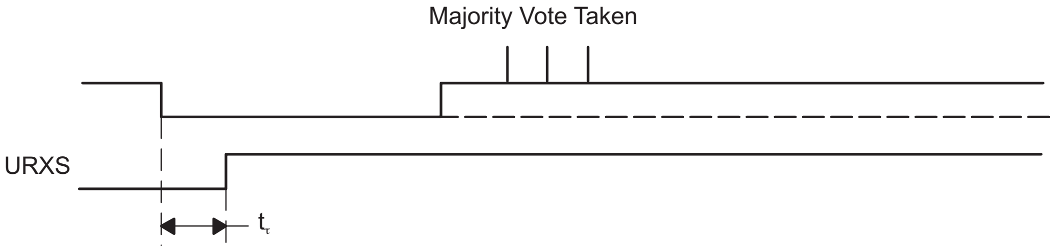


Figure 17-9. Glitch Suppression, eUSCI_A Activated

17.3.8 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.

17.3.9 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 baud rate settings for the eUSCI_A can be found in Section 17.3.10.

17.3.9.1 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 17-10. 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.

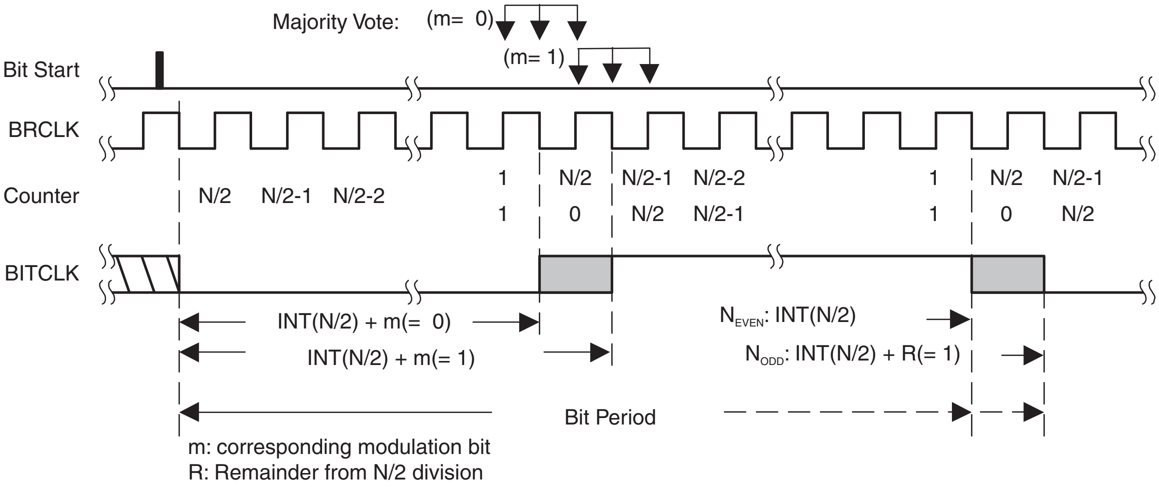


Figure 17-10. BITCLK Baud-Rate Timing With UCOS16 = 0

Modulation is based on the UCBRSx setting as shown in Table 17-2. 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 17-2. 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 17.3.10.

17.3.9.2 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 17-3). 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 17-3. 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

17.3.10 Setting a Baud Rate

For a given BRCLK clock source, the baud rate used determines the required division factor N:

$N = f_{BRCLK}/\text{Baud Rate}$

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: Baud Rate settings quick set up

To calculate the correct the correct settings for the baud rate generation, perform these steps:

1. Calculate $N = f_{BRCLK}/\text{Baud Rate}$ [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 17-4
5. If OS16 = 0 was chosen, a detailed error calculation is recommended to be performed

Table 17-4 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 17-4. UCBRSx Settings for Fractional Portion of $N = f_{BRCLK}/\text{Baud Rate}$

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

17.3.10.1 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 17-5).

17.3.10.2 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 17-4 and the fractional part of N = f_{BRCLK}/Baud Rate.

17.3.11 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.

17.3.11.1 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

17.3.11.2 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:

$$t_{bit,TX}[i] = \frac{1}{f_{BRCLK}}\left((16 \times UCBRx) + \sum_{j=0}^{15} m_{UCBRFx}[j] + m_{UCBRSx}[i]\right)$$

Where:

$\leq \sum_{j=0}^{15} m_{UCBRFx}[j]$ = Sum of ones from the corresponding row in Table 17-3

$m_{UCBRSx}[i]$ = 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:

$$t_{bit,TX}[i] = \overset{i}{\underset{j=0}{S}} t_{bit,TX}[j]$$

To calculate bit error, this time is compared to the ideal bit time $t_{bit,ideal,TX}[i]$:

$t_{bit,ideal,TX}[i] = (1/\text{Baud Rate})(i + 1)$

This results in an error normalized to one ideal bit time (1/baud rate):

$Error_{TX}[i] = (t_{bit,TX}[i] – t_{bit,ideal,TX}[i]) \times \text{Baud Rate} \times 100\%$

17.3.12 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 17-11 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 BRCLKs, independent of the selected baud-rate generation mode.

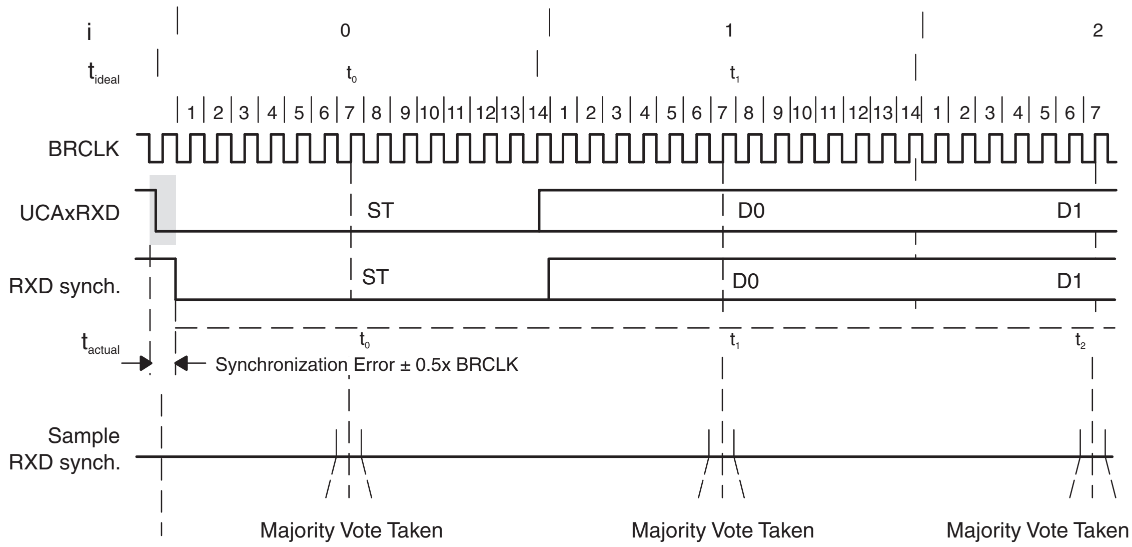


Figure 17-11. Receive Error

The ideal sampling time $t_{bit,ideal,RX}[i]$ is in the middle of a bit period:

$t_{bit,ideal,RX}[i] = (1/\text{Baud Rate})(i + 0.5)$

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:

$$t_{bit,RX}[i] = \frac{1}{f_{BRCLK}}\left((16 \times UCBRx) + \sum_{j=0}^{15} m_{UCBRFx}[j] + m_{UCBRSx}[i]\right)$$

$\sum_{j=0}^{7+m_{UCBRSx}[i]} m_{UCBRFx}[j]$ = Sum of ones from columns 0 to (7 + $m_{UCBRSx}[i]$) from the corresponding row in Table 17-3.

$m_{UCBRSx}[i]$ = Modulation of bit i of UCBRSx

This results in an error normalized to one ideal bit time (1/baud rate) according to the following formula:

$Error_{RX}[i] = (t_{bit,RX}[i] – t_{bit,ideal,RX}[i]) \times \text{Baud Rate} \times 100\%$

17.3.13 Typical Baud Rates and Errors

Standard baud-rate data for UCBRx, UCBRSx, and UCBRFx are listed in Table 17-5 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 17-5. Recommended Settings for Typical Crystals and Baud Rates

BRCLK	Baud Rate	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 17-5. Recommended Settings for Typical Crystals and Baud Rates (continued)

BRCLK	Baud Rate	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

17.3.14 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.

17.3.15 eUSCI_A Interrupts

The eUSCI_A has only one interrupt vector that is shared for transmission and for reception.

17.3.15.1 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.

17.3.15.2 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 UCRXEIE = 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 can set UCRXIFG.
- When UCBRKIE = 1, a break condition sets the UCBRK bit and the UCRXIFG flag.

17.3.15.3 eUSCI_A Receive Interrupt Operation

Table 17-6 describes the I^{2}C state change interrupt flags.

Table 17-6. 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.

17.3.15.4 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 17-1 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 17-1. 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;
    }
}
```

17.4 eUSCI_A UART Registers

The eUSCI_A registers applicable in UART mode and their address offsets are listed in Table 17-7. The base address can be found in the device-specific data sheet.

Table 17-7. eUSCI_A UART Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCAxCTLW0	eUSCI_Ax Control Word 0	Read/write	Word	0001h	Section 17.4.1
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 17.4.2
06h	UCAxBRW	eUSCI_Ax Baud Rate Control Word	Read/write	Word	0000h	Section 17.4.3
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 17.4.4
0Ah	UCAxSTATW	eUSCI_Ax Status	Read/write	Word	00h	Section 17.4.5
0Ch	UCAxRXBUF	eUSCI_Ax Receive Buffer	Read/write	Word	00h	Section 17.4.6
0Eh	UCAxTXBUF	eUSCI_Ax Transmit Buffer	Read/write	Word	00h	Section 17.4.7
10h	UCAxABCTL	eUSCI_Ax Auto Baud Rate Control	Read/write	Word	00h	Section 17.4.8
12h	UCAxIRCTL	eUSCI_Ax IrDA Control	Read/write	Word	0000h	Section 17.4.9
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 17.4.10
1Ch	UCAxIFG	eUSCI_Ax Interrupt Flag	Read/write	Word	02h	Section 17.4.11
1Eh	UCAxIV	eUSCI_Ax Interrupt Vector	Read	Word	0000h	Section 17.4.12

[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".

17.4.1 UCAxCTLW0 Register

eUSCI_Ax Control Word Register 0

Figure 17-12. 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 17-8. 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 = Device specific 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 17-8. 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.

17.4.2 UCAxCTLW1 Register

eUSCI_Ax Control Word Register 1

Figure 17-13. 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 17-9. 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

17.4.3 UCAxBRW Register

eUSCI_Ax Baud Rate Control Word Register

Figure 17-14. 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 17-10. UCAxBRW Register Description

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Clock prescaler setting of the Baud rate generator

17.4.4 UCAxMCTLW Register

eUSCI_Ax Modulation Control Word Register

Figure 17-15. 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 17-11. 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

17.4.5 UCAxSTATW Register

eUSCI_Ax Status Register

Figure 17-16. 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 17-12. 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

17.4.6 UCAxRXBUF Register

eUSCI_Ax Receive Buffer Register

Figure 17-17. 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 17-13. 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.

17.4.7 UCAxTXBUF Register

eUSCI_Ax Transmit Buffer Register

Figure 17-18. 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 17-14. 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.

17.4.8 UCAxABCTL Register

eUSCI_Ax Auto Baud Rate Control Register

Figure 17-19. 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 17-15. 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.

17.4.9 UCAxIRCTL Register

eUSCI_Ax IrDA Control Word Register

Figure 17-20. 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 17-16. 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

17.4.10 UCAxIE Register

eUSCI_Ax Interrupt Enable Register

Figure 17-21. 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 17-17. 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

17.4.11 UCAxIFG Register

eUSCI_Ax Interrupt Flag Register

Figure 17-22. 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 17-18. 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

17.4.12 UCAxIV Register

eUSCI_Ax Interrupt Vector Register

Figure 17-23. 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 17-19. 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 describes the operation of the synchronous peripheral interface (SPI) mode.

Topic **Page**

18.1 Enhanced Universal Serial Communication Interfaces (eUSCI_A, eUSCI_B) Overview 511
18.2 eUSCI Introduction – SPI Mode 511
18.3 eUSCI Operation – SPI Mode 513
18.4 eUSCI_A SPI Registers 519
18.5 eUSCI_B SPI Registers 526

18.1 Enhanced Universal Serial Communication Interfaces (eUSCI_A, eUSCI_B) Overview

Both the eUSCI_A and the eUSCI_B support serial communication in SPI mode.

18.2 eUSCI Introduction – SPI Mode

In synchronous mode, the eUSCI connects the device to an external system through 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 or 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 18-1 shows the eUSCI when configured for SPI mode.

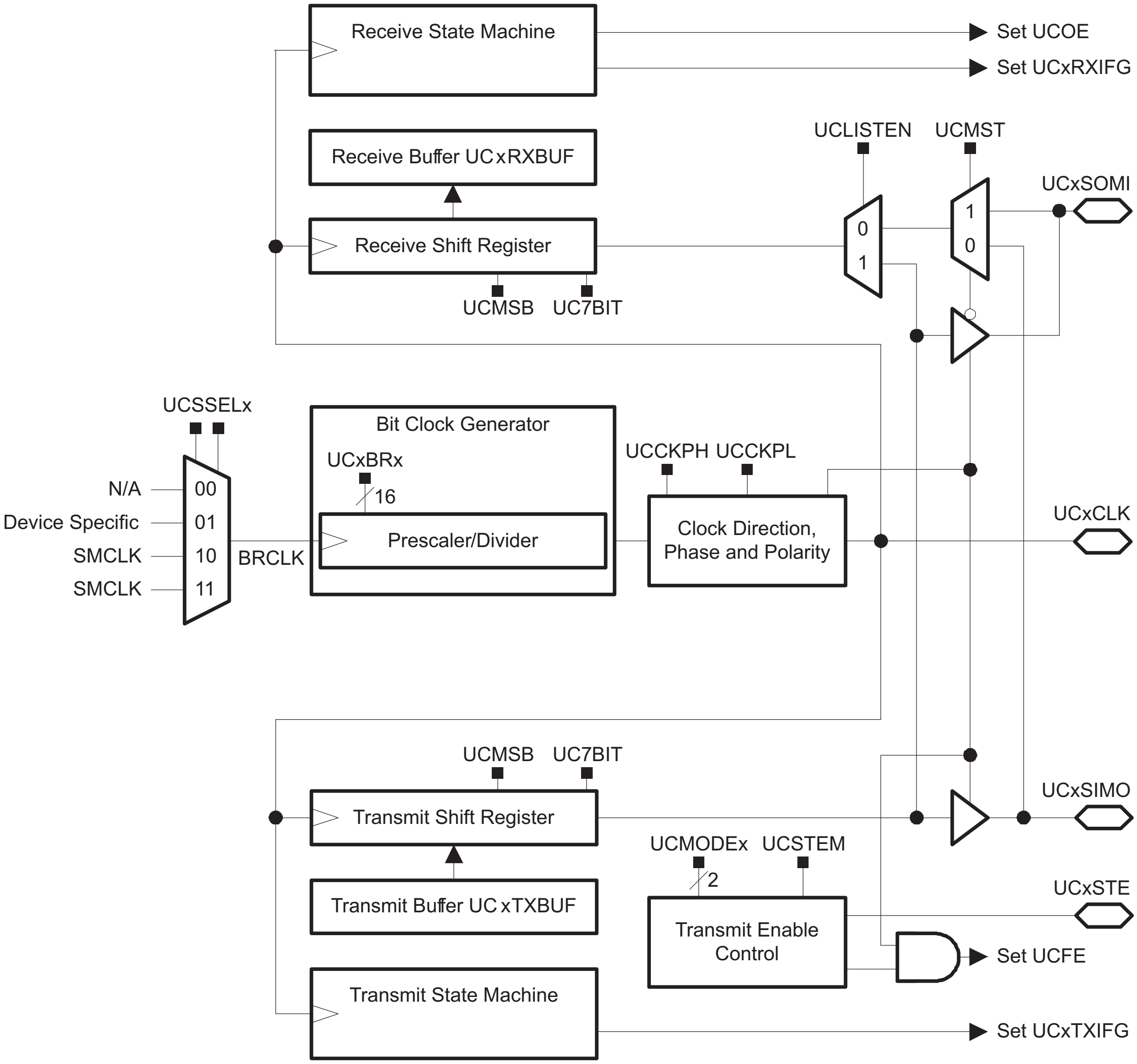


Figure 18-1. eUSCI Block Diagram – SPI Mode

18.3 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 18-1 describes the UCxSTE operation.

Table 18-1. 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

18.3.1 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, which keeps 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.

To avoid unpredictable behavior, configure or reconfigure the eUSCI module when UCSWRST is set.

NOTE: Initializing or reconfiguring the eUSCI module

The recommended eUSCI initialization or reconfiguration process is:

1. Set UCSWRST.

   ```
   BIS.B #UCSWRST,&UCxCTL1
   ```

2. Initialize all eUSCI registers while UCSWRST = 1 (including UCxCTL1).
3. Configure ports.
4. Clear UCSWRST in software.

   ```
   BIC.B #UCSWRST,&UCxCTL1
   ```

5. Enable interrupts (optional) by setting UCRXIE or UCTXIE.

18.3.2 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.

18.3.3 Master Mode

Figure 18-2 shows the eUSCI as a master in both 3-pin and 4-pin configurations.

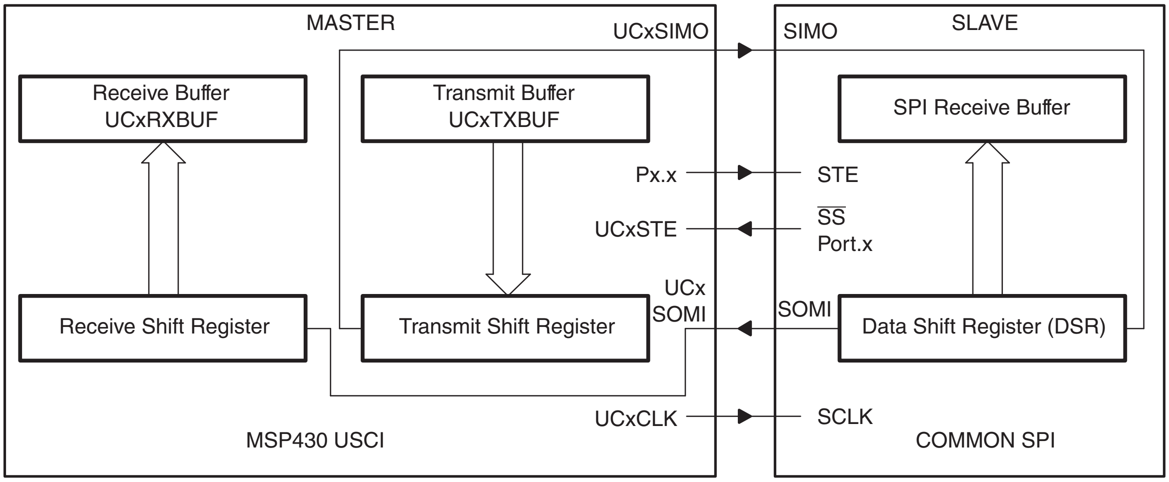


Figure 18-2. eUSCI Master and External Slave (UCSTEM = 0)

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 that the RX or 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 or 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 following 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.

18.3.3.1 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 18-1. 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.

18.3.3.2 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 18-4.

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.

18.3.4 Slave Mode

Figure 18-3 shows the eUSCI as a slave in both 3-pin and 4-pin configurations.

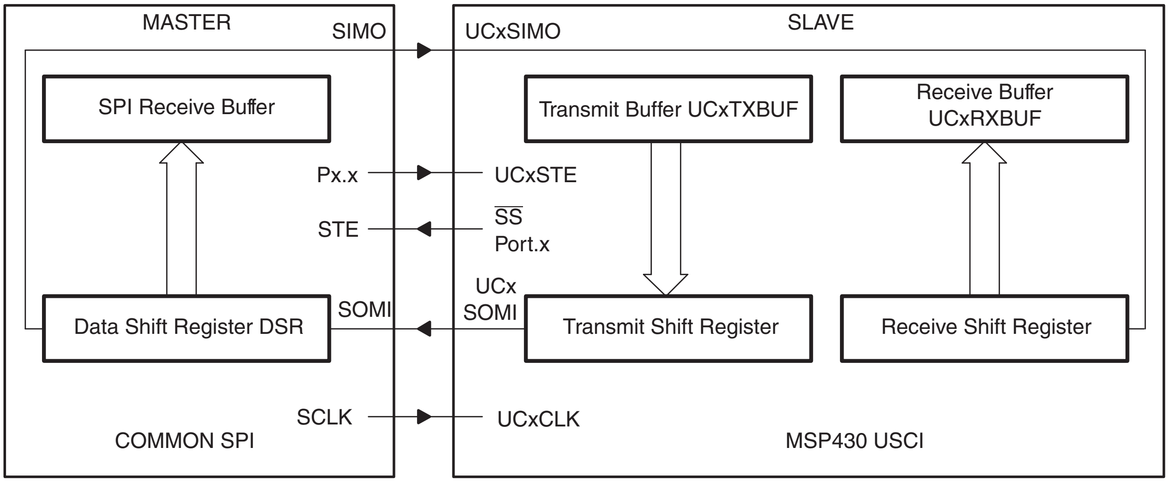


Figure 18-3. eUSCI Slave and External Master

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.

18.3.4.1 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.

18.3.5 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.

18.3.5.1 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.

18.3.5.2 Receive Enable

The SPI receives data when a transmission is active. Receive and transmit operations operate concurrently.

18.3.6 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 or UCBxCLK frequency is given by Equation 12.

$$f_{BitClock} = f_{BRCLK} / UCBRx \qquad (12)$$

When UCBRx = 0, no division is applied to BRCLK, and the bit clock equals BRCLK ($f_{BitClock} = f_{BRCLK}$).

Even UCBRx settings result in even divisions and, thus, generate a bit clock with a 50/50 duty cycle.

Odd 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.

18.3.6.1 Serial Clock Polarity and Phase

The polarity and phase of UCxCLK are independently configured with the UCCKPL and UCCKPH control bits of the eUSCI. Timing for each case is shown in Figure 18-4.

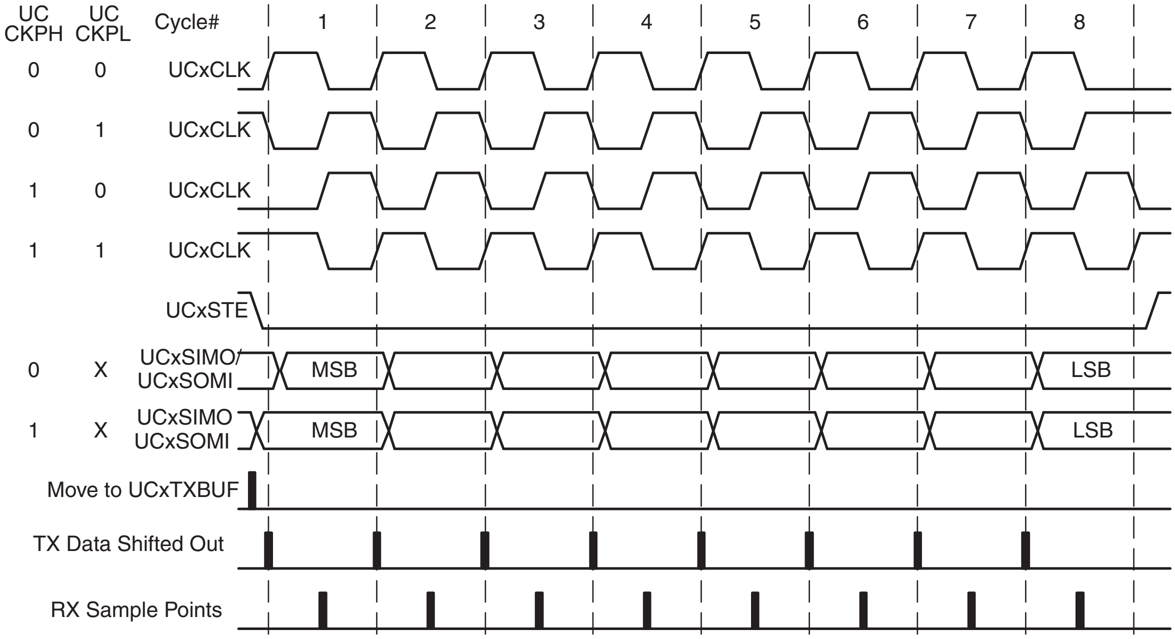


Figure 18-4. eUSCI SPI Timing With UCMSB = 1

18.3.7 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.

When receiving multiple bytes as a slave in LPM4 the wakeup time of the CPU needs to be considered. If the wake-up time of the CPU is, for example, 150 µs (see device-specific data-sheet), it needs to be ensured that the CPU serves the TXIFG of the first received byte before the second byte is completely received by the eUSCI_A or eUSCI_B. Otherwise an overrun error occurs.

18.3.8 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.

18.3.8.1 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.

18.3.8.2 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.

18.3.8.3 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.

18.3.8.3.1 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
```

18.4 eUSCI_A SPI Registers

The eUSCI_A registers applicable in SPI mode and their address offsets are listed in Table 18-2. The base addresses can be found in the device-specific data sheet.

Table 18-2. eUSCI_A SPI Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCAxCTLW0	eUSCI_Ax Control Word 0	Read/write	Word	0001h	Section 18.4.1
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 18.4.2
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 18.4.3
0Ch	UCAxRXBUF	eUSCI_Ax Receive Buffer	Read/write	Word	00h	Section 18.4.4
0Eh	UCAxTXBUF	eUSCI_Ax Transmit Buffer	Read/write	Word	00h	Section 18.4.5
1Ah	UCAxIE	eUSCI_Ax Interrupt Enable	Read/write	Word	00h	Section 18.4.6
1Ch	UCAxIFG	eUSCI_Ax Interrupt Flag	Read/write	Word	02h	Section 18.4.7
1Eh	UCAxIV	eUSCI_Ax Interrupt Vector	Read	Word	0000h	Section 18.4.8

18.4.1 UCAxCTLW0 Register

eUSCI_Ax Control Register 0

Figure 18-5. 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 18-3. UCAxCTLW0 Register Description

Bit	Field	Type	Reset	Description
15	UCCKPH	RW	0h	Clock phase select. Modify only when UCSWRST = 1. 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. Modify only when UCSWRST = 1. 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. Modify only when UCSWRST = 1. 0b = LSB first 1b = MSB first
12	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. Modify only when UCSWRST = 1. 0b = 8-bit data 1b = 7-bit data
11	UCMST	RW	0h	Master mode select. Modify only when UCSWRST = 1. 0b = Slave mode 1b = Master mode
10-9	UCMODEx	RW	0h	eUSCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. Modify only when UCSWRST = 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	0h	Synchronous mode enable. Modify only when UCSWRST = 1. 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. Modify only when UCSWRST = 1. 00b = Reserved 01b = Device specific 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. Modify only when UCSWRST = 1. 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 18-3. 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.

18.4.2 UCAxBRW Register

eUSCI_Ax Bit Rate Control Register 1

Figure 18-6. 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 18-4. UCAxBRW Register Description

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Bit clock prescaler setting. Modify only when UCSWRST = 1. $f_{BitClock} = f_{BRCLK}$ / UCBRx If UCBRx = 0, $f_{BitClock} = f_{BRCLK}$

18.4.3 UCAxSTATW Register

eUSCI_Ax Status Register

Figure 18-7. 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 18-5. 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. Modify only when UCSWRST = 1. 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

18.4.4 UCAxRXBUF Register

eUSCI_Ax Receive Buffer Register

Figure 18-8. 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 18-6. 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.

18.4.5 UCAxTXBUF Register

eUSCI_Ax Transmit Buffer Register

Figure 18-9. 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 18-7. 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.

18.4.6 UCAxIE Register

eUSCI_Ax Interrupt Enable Register

Figure 18-10. 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						UCTXIE	UCRXIE
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Table 18-8. 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

18.4.7 UCAxIFG Register

eUSCI_Ax Interrupt Flag Register

Figure 18-11. 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 18-9. 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 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

18.4.8 UCAxIV Register

eUSCI_Ax Interrupt Vector Register

Figure 18-12. 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 18-10. 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

18.5 eUSCI_B SPI Registers

The eUSCI_B registers applicable in SPI mode and their address offsets are listed in Table 18-11. The base addresses can be found in the device-specific data sheet.

Table 18-11. eUSCI_B SPI Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCBxCTLW0	eUSCI_Bx Control Word 0	Read/write	Word	01C1h	Section 18.5.1
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 18.5.2
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 18.5.3
0Ch	UCBxRXBUF	eUSCI_Bx Receive Buffer	Read/write	Word	00h	Section 18.5.4
0Eh	UCBxTXBUF	eUSCI_Bx Transmit Buffer	Read/write	Word	00h	Section 18.5.5
2Ah	UCBxIE	eUSCI_Bx Interrupt Enable	Read/write	Word	00h	Section 18.5.6
2Ch	UCBxIFG	eUSCI_Bx Interrupt Flag	Read/write	Word	02h	Section 18.5.7
2Eh	UCBxIV	eUSCI_Bx Interrupt Vector	Read	Word	0000h	Section 18.5.8

18.5.1 UCBxCTLW0 Register

eUSCI_Bx Control Register 0

Figure 18-13. 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 18-12. UCBxCTLW0 Register Description

Bit	Field	Type	Reset	Description
15	UCCKPH	RW	0h	Clock phase select. Modify only when UCSWRST = 1. 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. Modify only when UCSWRST = 1. 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. Modify only when UCSWRST = 1. 0b = LSB first 1b = MSB first
12	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. Modify only when UCSWRST = 1. 0b = 8-bit data 1b = 7-bit data
11	UCMST	RW	0h	Master mode select. Modify only when UCSWRST = 1. 0b = Slave mode 1b = Master mode
10-9	UCMODEx	RW	0h	eUSCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. Modify only when UCSWRST = 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. Modify only when UCSWRST = 1. 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. Modify only when UCSWRST = 1. 00b = Reserved 01b = Device Specific 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. Modify only when UCSWRST = 1. 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 18-12. 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.

18.5.2 UCBxBRW Register

eUSCI_Bx Bit Rate Control Register 1

Figure 18-14. 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 18-13. UCBxBRW Register Description

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Bit clock prescaler setting. Modify only when UCSWRST = 1. $f_{BitClock} = f_{BRCLK}$ / UCBRx If UCBRx = 0, $f_{BitClock} = f_{BRCLK}$

18.5.3 UCBxSTATW Register

eUSCI_Bx Status Register

Figure 18-15. 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 18-14. 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. Modify only when UCSWRST = 1. 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 mode 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

18.5.4 UCBxRXBUF Register

eUSCI_Bx Receive Buffer Register

Figure 18-16. UCBxRXBUF Register

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td colspan="8">Reserved</td></tr>
<tr><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">UCRXBUFx</td></tr>
<tr><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td></tr>
</table>

Table 18-15. 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.

18.5.5 UCBxTXBUF Register

eUSCI_Bx Transmit Buffer Register

Figure 18-17. UCBxTXBUF Register

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td colspan="8">Reserved</td></tr>
<tr><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">UCTXBUFx</td></tr>
<tr><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td><td>rw</td></tr>
</table>

Table 18-16. 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.

18.5.6 UCBxIE Register

eUSCI_Bx Interrupt Enable Register

Figure 18-18. 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 18-17. 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

18.5.7 UCBxIFG Register

eUSCI_Bx Interrupt Flag Register

Figure 18-19. 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 18-18. 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 UCBxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	RW	0h	Receive interrupt flag. UCRXIFG is set when UCBxRXBUF has received a complete character. 0b = No interrupt pending 1b = Interrupt pending

18.5.8 UCBxIV Register

eUSCI_Bx Interrupt Vector Register

Figure 18-20. 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 18-19. 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 describes the operation of the I^2C mode.

Topic **Page**

19.1 Enhanced Universal Serial Communication Interface B (eUSCI_B) Overview 534
19.2 eUSCI_B Introduction – I^2C Mode 534
19.3 eUSCI_B Operation – I^2C Mode 535
19.4 eUSCI_B I2C Registers 555

19.1 Enhanced Universal Serial Communication Interface B (eUSCI_B) Overview

The eUSCI_B module supports two serial communication modes:

- I^{2}C 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.

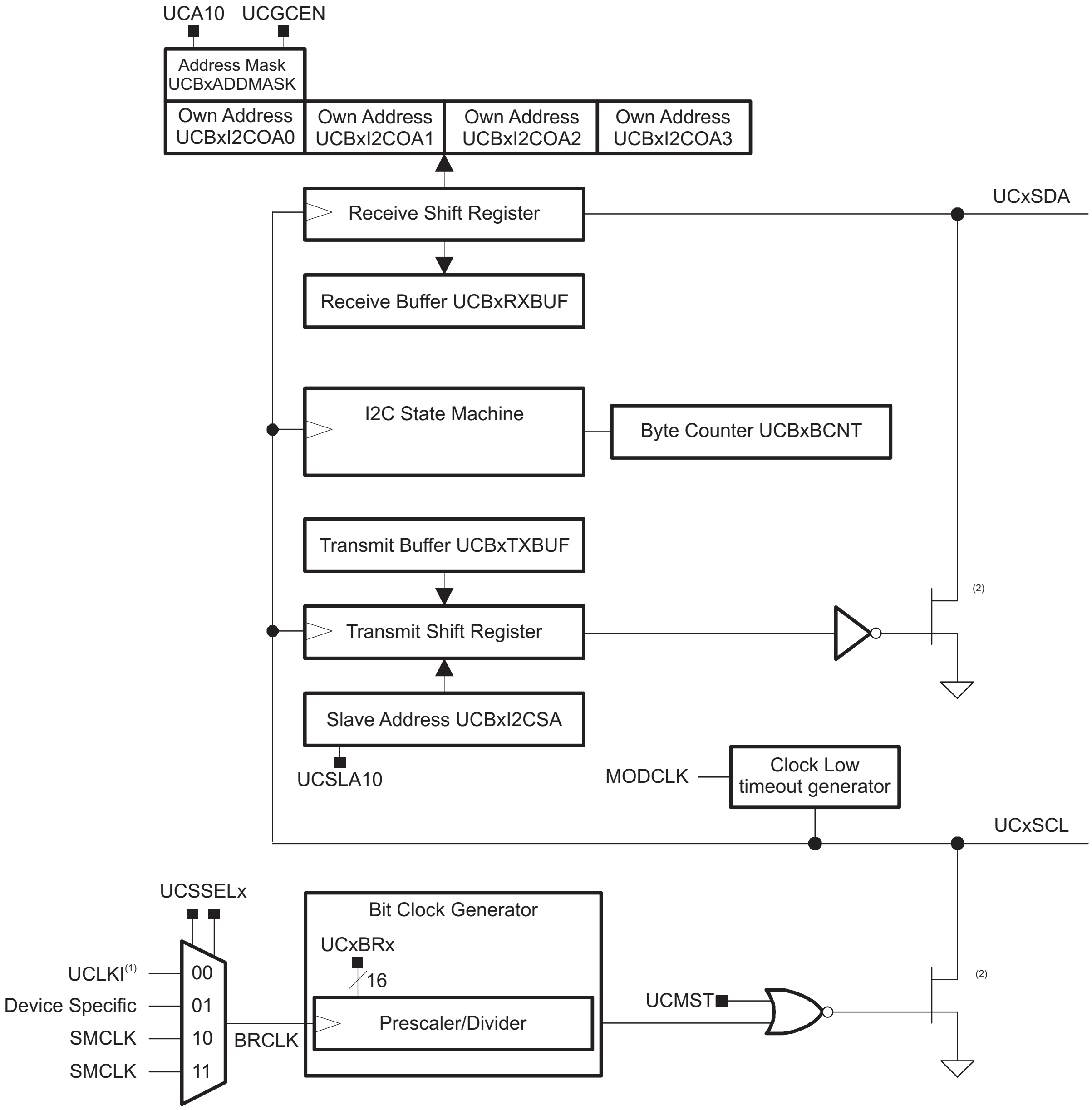
19.2 eUSCI_B Introduction – I^{2}C Mode

In I^{2}C mode, the eUSCI_B module provides an interface between the device and I^{2}C-compatible devices connected by the two-wire I^{2}C serial bus. External components attached to the I^{2}C bus serially transmit or receive serial data to or from the eUSCI_B module through the 2-wire I^{2}C interface.

The eUSCI_B I^{2}C mode features include:

- 7-bit and 10-bit device addressing modes
- General call
- START, RESTART, STOP
- Multiple-master transmitter or receiver mode
- Slave receiver or transmitter mode
- Supports standard mode up to 100 kbps and fast mode up to 400 kbps
- 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
- Mask register for slave address and address received interrupt
- Clock low time-out 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 19-1 shows the eUSCI_B when configured in I^{2}C mode.



[1] Externally provided clock on the eUSCI_B SPI clock input pin

[2] Not the actual implementation (transistor not located in eUSCI_B module)

Figure 19-1. eUSCI_B Block Diagram – I^{2}C Mode

19.3 eUSCI_B Operation – I^{2}C Mode

The I^{2}C mode supports any slave or master I^{2}C-compatible device. Figure 19-2 shows an example of an I^{2}C bus. Each I^{2}C device is recognized by a unique address and can operate as either a transmitter or a receiver. A device connected to the I^{2}C 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.

I^{2}C 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.

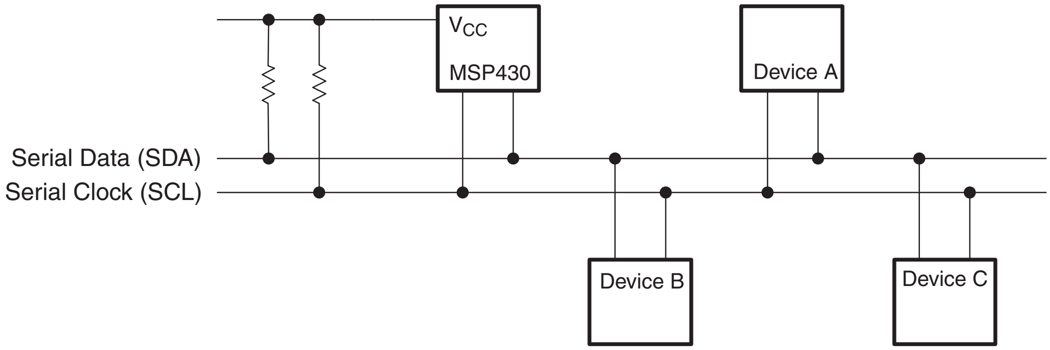


Figure 19-2. I²C Bus Connection Diagram

NOTE: **SDA and SCL levels**

The SDA and SCL pins must not be pulled up above the device V_{CC} level.

19.3.1 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 using software (`BIC.B #UCSWRST,&UCxCTL1`).
5. Enable interrupts (optional).

19.3.2 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 19-3.

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.

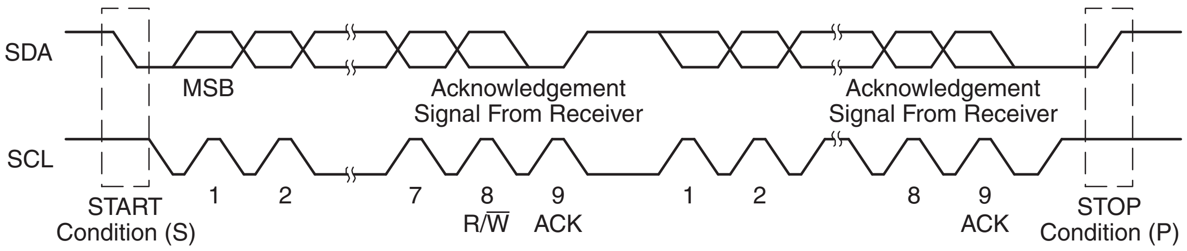


Figure 19-3. I^{2}C Module Data Transfer

START and STOP conditions are generated by the master and are shown in Figure 19-3. 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 19-4). The high and low state of SDA can change only when SCL is low, otherwise START or STOP conditions are generated.

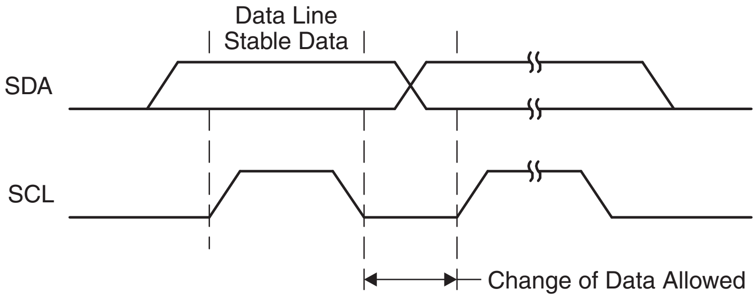


Figure 19-4. Bit Transfer on I^{2}C Bus

19.3.3 I^{2}C Addressing Modes

The I^{2}C mode supports 7-bit and 10-bit addressing modes.

19.3.3.1 7-Bit Addressing

In the 7-bit addressing format (see Figure 19-5), the first byte is the 7-bit slave address and the R/$\overline{\text{W}}$ bit. The ACK bit is sent from the receiver after each byte.

1	7	1	1	8	1	8	1	1
S	Slave Address	R/$\overline{\text{W}}$	ACK	Data	ACK	Data	ACK	P

Figure 19-5. I^{2}C Module 7-Bit Addressing Format

19.3.3.2 10-Bit Addressing

In the 10-bit addressing format (see Figure 19-6), the first byte is made up of 11110b plus the two MSBs of the 10-bit slave address and the R/$\overline{\text{W}}$ 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.

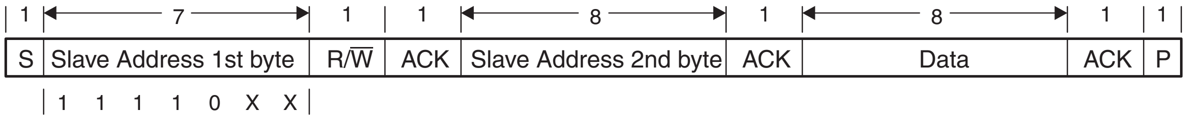


Figure 19-6. I^{2}C Module 10-Bit Addressing Format

19.3.3.3 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 19-7.

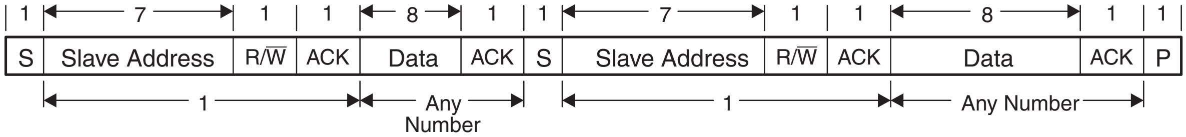


Figure 19-7. I^{2}C Module Addressing Format With Repeated START Condition

19.3.4 I^{2}C 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 19.3.5.

The latest code examples can be found on the MSP430 website 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 19-1).

Example 19-1. Master TX With 7-Bit Address

```
UCBxCTL1 |= UCSWRST;            // put eUSCI_B in reset state
UCBxCTLW0 |= UCMODE_3 + UCMST;  // I2C master mode
UCBxBRW = 0x0008;               // baud rate = SMCLK / 8
UCBxCTLW1 = UCASTP_2;           // automatic 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 I^{2}C operation of the eUSCI_B, UCMODE must be set accordingly. The baud rate 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 UCBxTXBUF inside the interrupt service routine. Example 19-3 shows the recommended structure of the interrupt service routine.

Example 19-2 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 19-2. 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 19-2, all configurations must be done while UCSWRST is set. For the slave, I^{2}C 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 19-3.

19.3.5 I^{2}C Module Operating Modes

In I^{2}C 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 19-8 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.

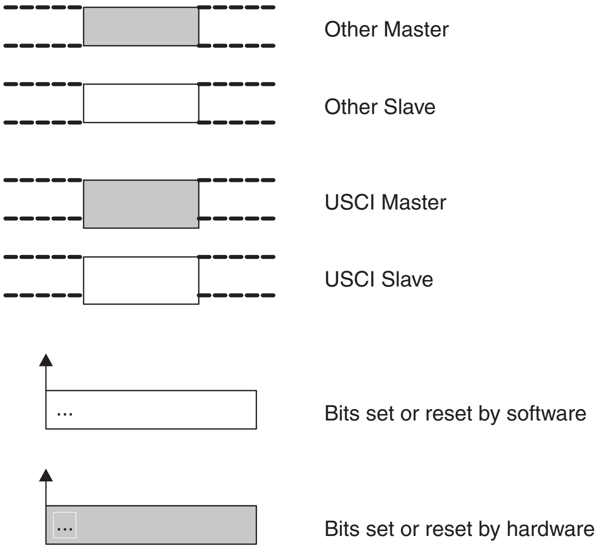


Figure 19-8. I²C Time-Line Legend

19.3.5.1 Slave Mode

The eUSCI_B module is configured as an I²C slave by UCMODEx = 11, UCSYNC = 1, and UCMST = 0.

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 19.3.9. 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.

19.3.5.1.1 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 19-9 shows the slave transmitter operation.

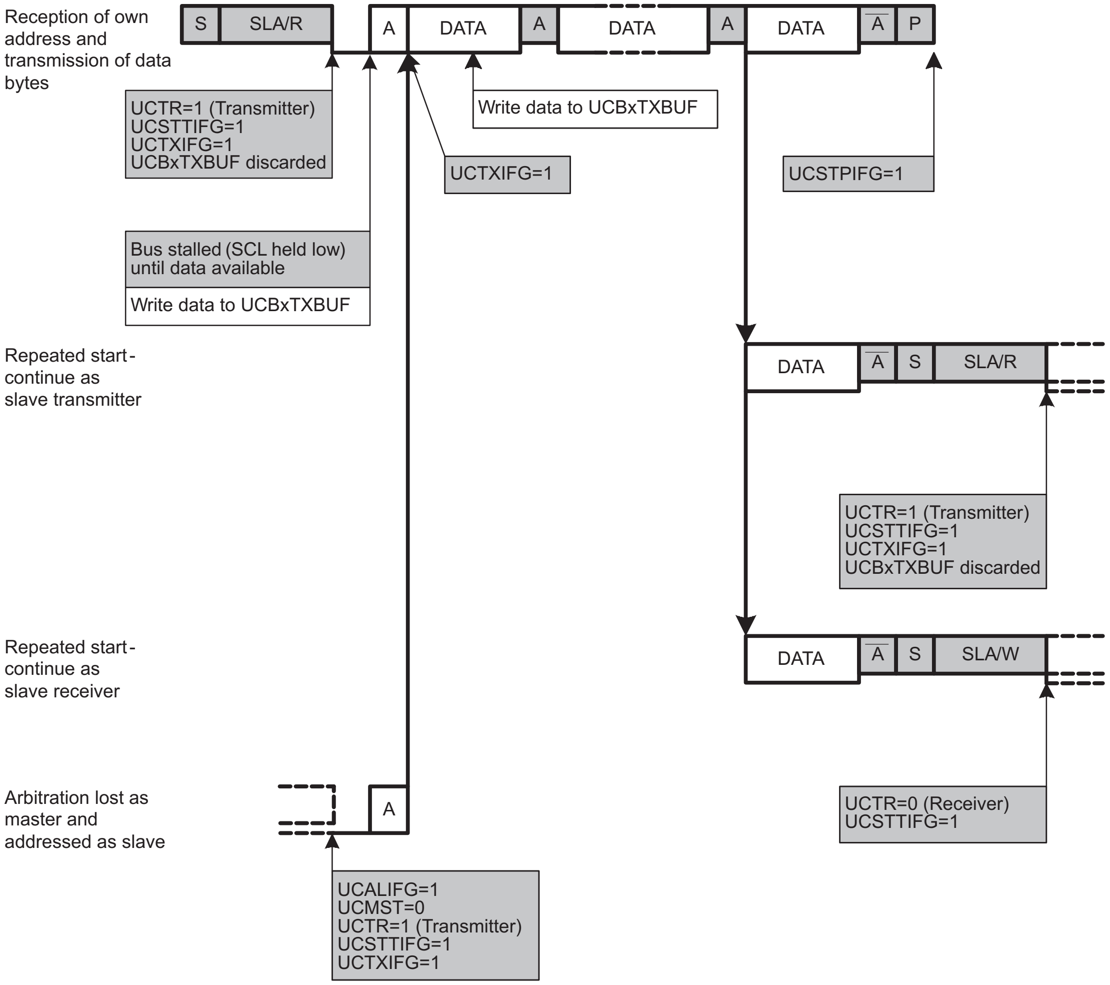


Figure 19-9. I²C Slave Transmitter Mode

19.3.5.1.2 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/W̄ 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 19-10 shows the I²C slave receiver operation.

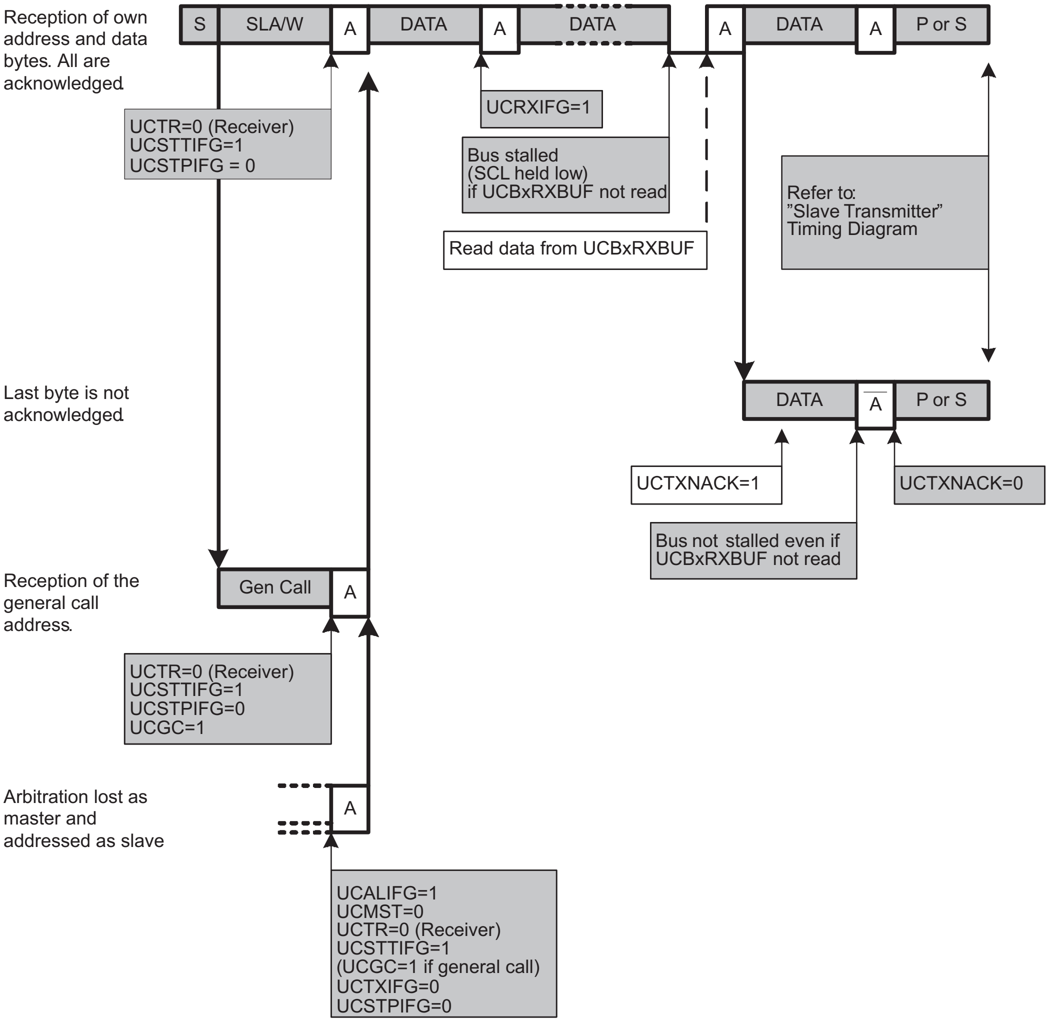

Reception of own address and data bytes. All are acknowledged
S
SLA/W
A
DATA
A
DATA
A
DATA
A
P or S
UCTR=0 (Receiver)
UCSTTIFG=1
UCSTPIFG = 0
UCRXIFG=1
Bus stalled (SCL held low) if UCBxRXBUF not read
Read data from UCBxRXBUF
Refer to: "Slave Transmitter" Timing Diagram
Last byte is not acknowledged
DATA
A
P or S
UCTXNACK=1
UCTXNACK=0
Bus not stalled even if UCBxRXBUF not read
Reception of the general call address.
Gen Call
A
UCTR=0 (Receiver)
UCSTTIFG=1
UCSTPIFG=0
UCGC=1
Arbitration lost as master and addressed as slave
A
UCALIFG=1
UCMST=0
UCTR=0 (Receiver)
UCSTTIFG=1
(UCGC=1 if general call)
UCTXIFG=0
UCSTPIFG=0

Figure 19-10. I²C Slave Receiver Mode

19.3.5.1.3 I²C Slave 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCA10 = 1 and is as shown in Figure 19-11. 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.

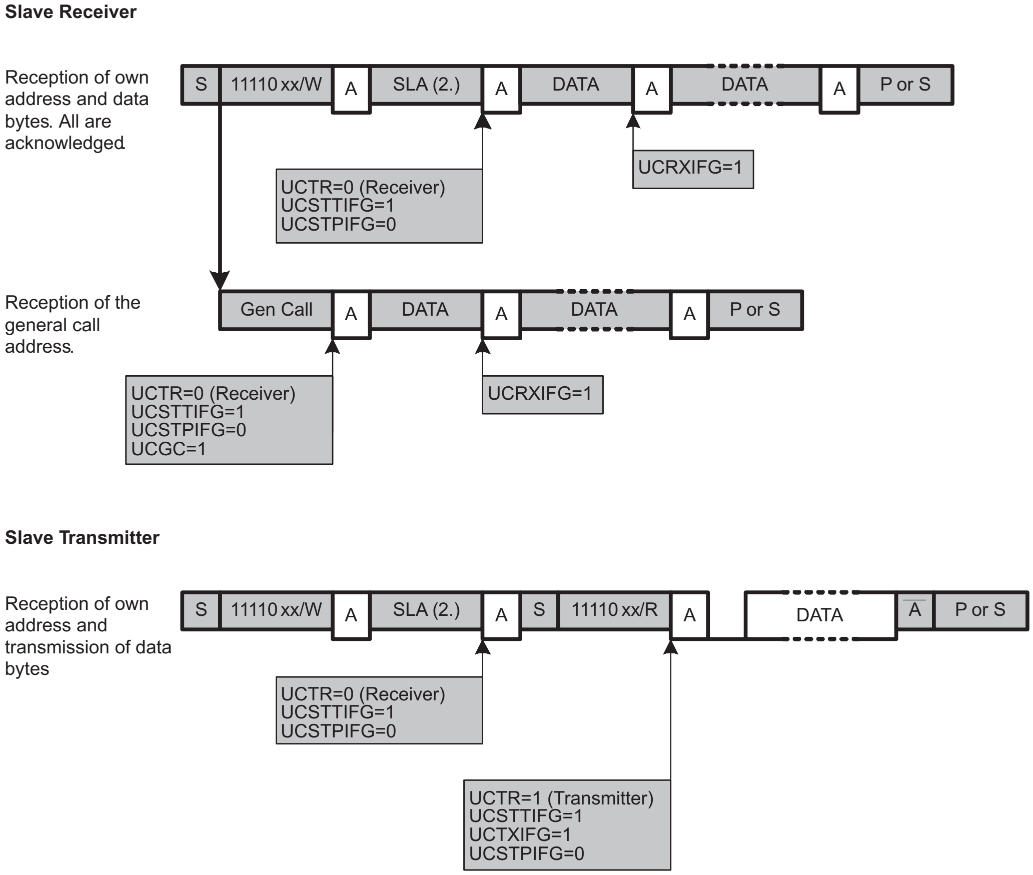


Figure 19-11. I²C Slave 10-Bit Addressing Mode

19.3.5.2 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 multiple-master system, UCMM must be set and its own address must be programmed into the UCBxI2COA0 register. Support for multiple slave addresses is described in Section 19.3.9. 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 multiple-master systems

In master mode with own-address detection enabled (UCOAEN = 1)—especially in multiple-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.

19.3.5.2.1 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 19-12 shows the I^{2}C master transmitter operation.

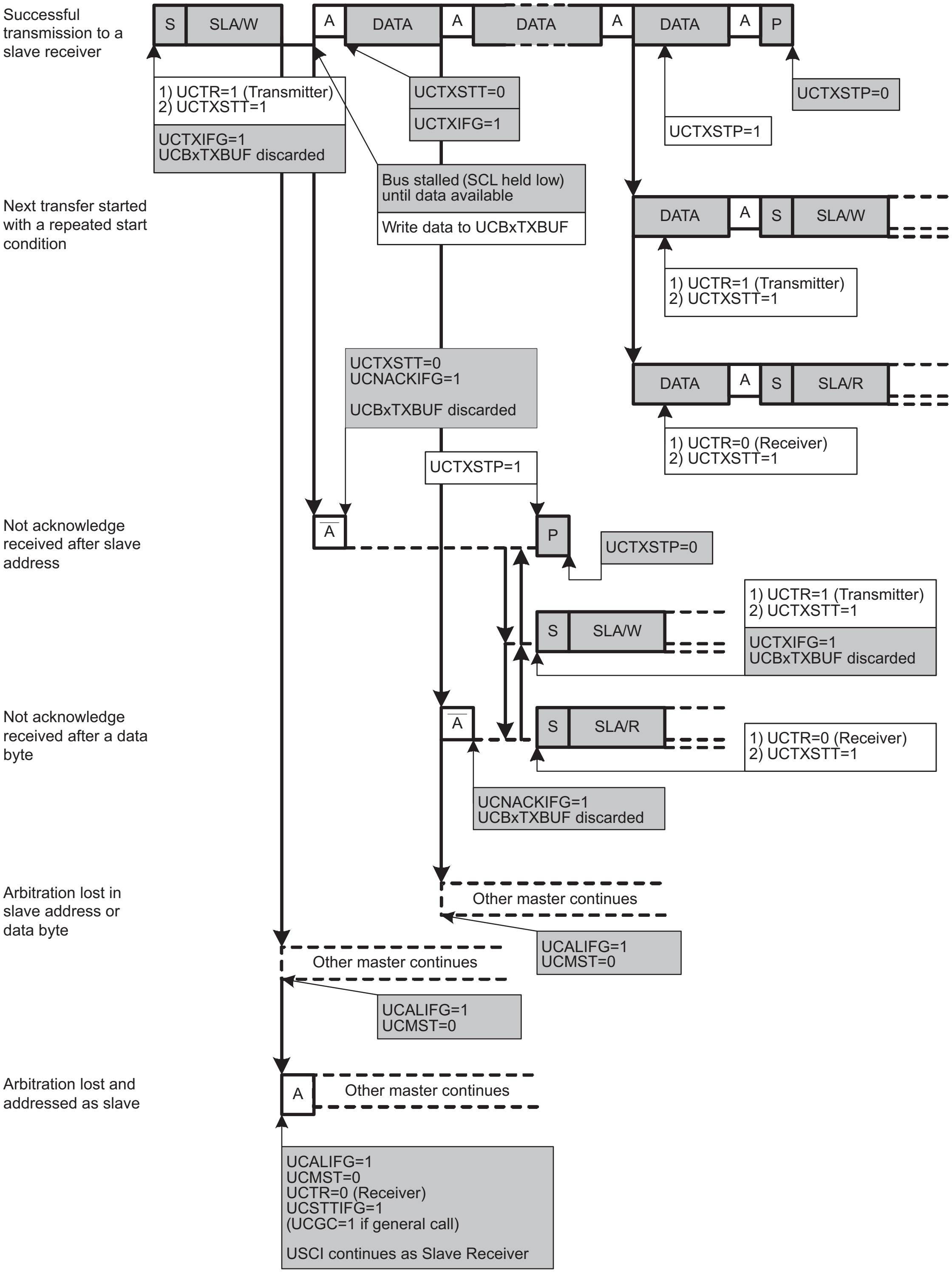


Figure 19-12. I²C Master Transmitter Mode

19.3.5.2.2 I^{2}C 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 19-13 shows the I^{2}C master receiver operation.

NOTE: Consecutive master transactions without repeated START

When performing multiple consecutive I^{2}C 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 I^{2}C transaction is initiated with setting UCTXSTT = 1. Otherwise, the current transaction might be affected.

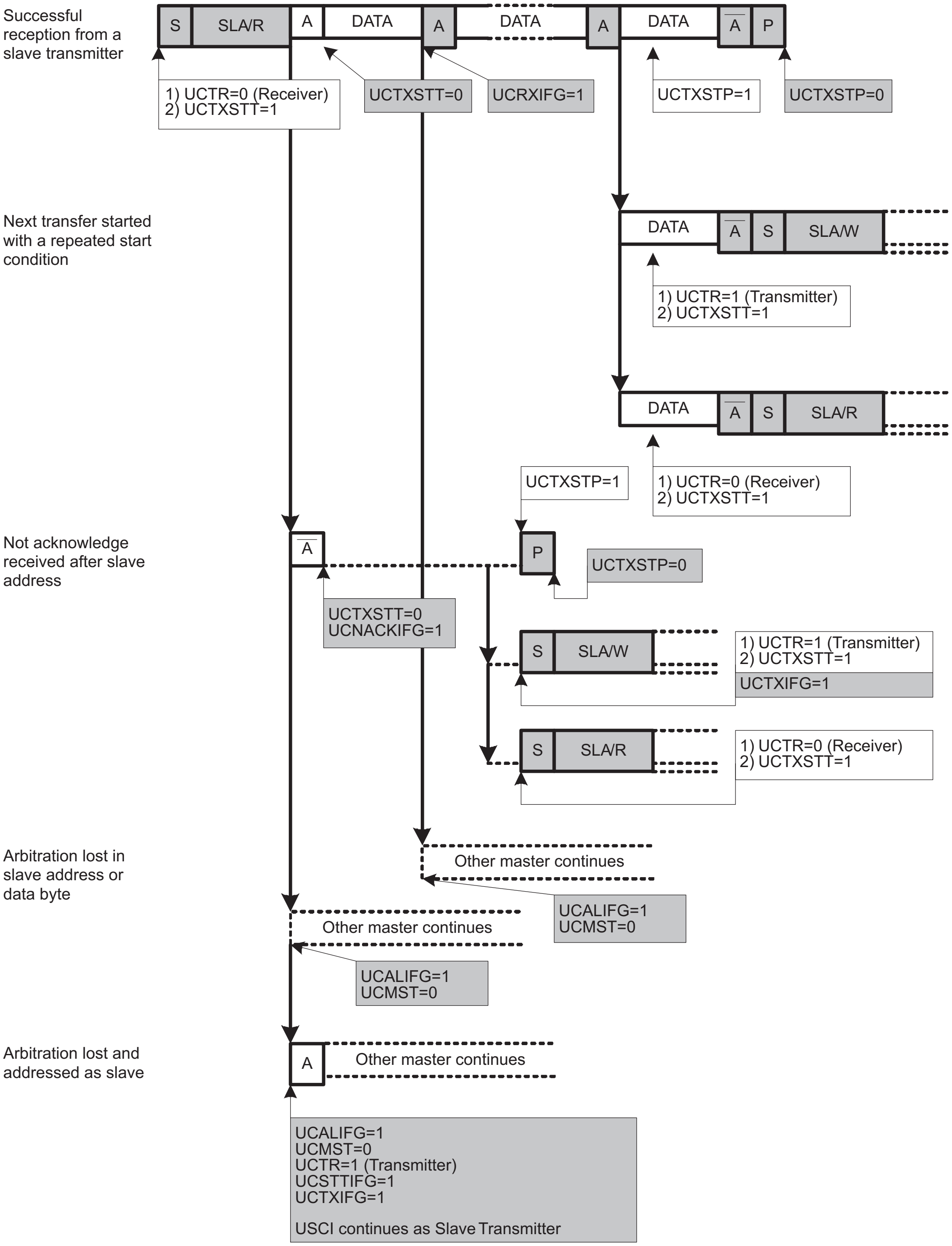


Figure 19-13. I^2C Master Receiver Mode

19.3.5.2.3 I^2C Master 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCSLA10 = 1 and is shown in Figure 19-14.

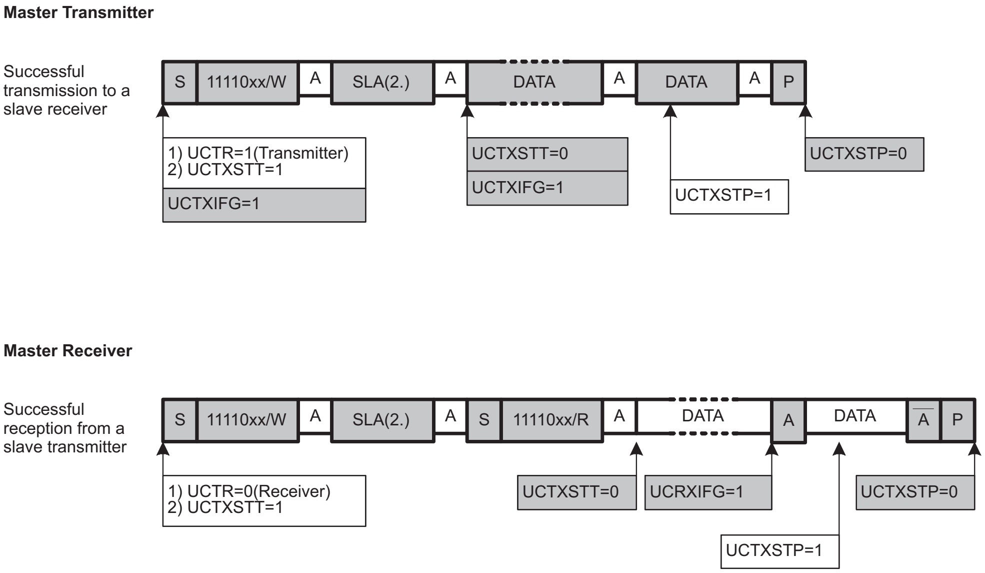


Figure 19-14. I^2C Master 10-Bit Addressing Mode

19.3.5.3 Arbitration

If two or more master transmitters simultaneously start a transmission on the bus, an arbitration procedure is invoked. Figure 19-15 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.

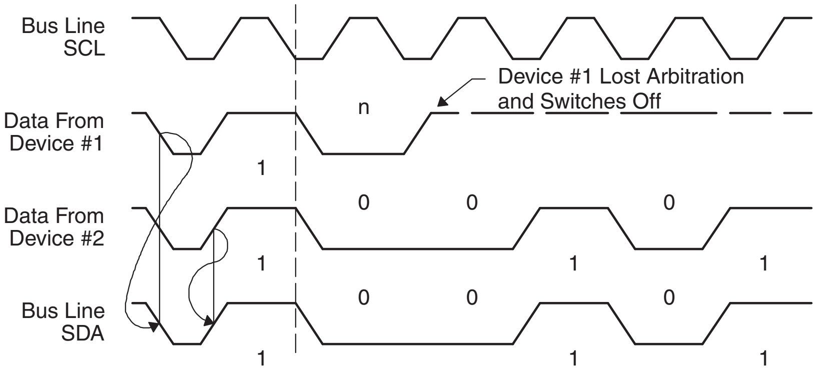


Figure 19-15. Arbitration Procedure Between Two Master Transmitters

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.

19.3.6 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 19-1. 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

19.3.7 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 multiple-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:

$$t_{LOW,MIN} = t_{HIGH,MIN} = (UCBRx/2)/f_{BRCLK} \text{ when UCBRx is even}$$

$$t_{LOW,MIN} = t_{HIGH,MIN} = ((UCBRx - 1)/2)/f_{BRCLK} \text{ when UCBRx is odd}$$

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 19-16 shows the clock synchronization. This allows a slow slave to slow down a fast master.

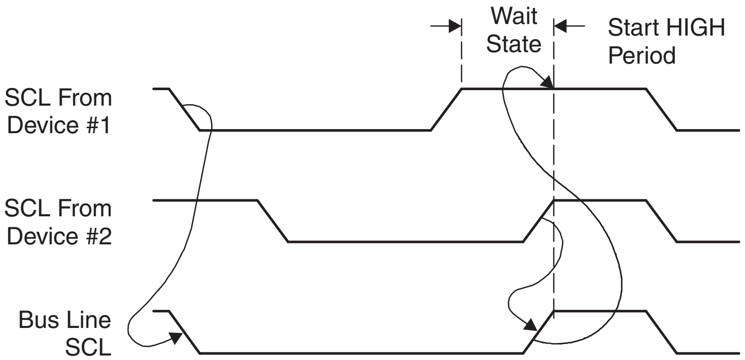


Figure 19-16. Synchronization of Two I²C Clock Generators During Arbitration

19.3.7.1 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.

19.3.7.2 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.

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 19.3.11.2).

19.3.7.3 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 time-out feature is enabled using the UCCLTO bits. It is possible to select one of three predefined times for the clock low time-out. 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.

19.3.8 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.

19.3.8.1 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.

19.3.8.2 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.

19.3.9 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
- Software support for up to 2^{10} different slave addresses all sharing one interrupt

19.3.9.1 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.

19.3.9.2 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.

19.3.10 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.

19.3.11 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.

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.

19.3.11.1 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.

19.3.11.2 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.

19.3.11.3 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.

19.3.11.4 I²C State Change Interrupt Operation

Table 19-2 describes the I²C state change interrupt flags.

Table 19-2. 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 time-out. This interrupt flag is set, if the clock is held low longer than defined by the UCCLTO bits.

Table 19-2. I²C State Change Interrupt Flags (continued)

Interrupt Flag	Interrupt Condition
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.

19.3.11.5 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 19-3 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 19-3. 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 time-out
                   break;
        case 0x1e: ... // Vector 30: 9th bit
                   break;
        default:   break;
    }
}
```

19.4 eUSCI_B I2C Registers

The eUSCI_B registers applicable in I2C mode and their address offsets are listed in Table 19-3. The base address can be found in the device-specific data sheet.

Table 19-3. eUSCI_B Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCBxCTLW0	eUSCI_Bx Control Word 0	Read/write	Word	01C1h	Section 19.4.1
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 19.4.2
06h	UCBxBRW	eUSCI_Bx Bit Rate Control Word	Read/write	Word	0000h	Section 19.4.3
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 19.4.4
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 19.4.5
0Ch	UCBxRXBUF	eUSCI_Bx Receive Buffer	Read/write	Word	00h	Section 19.4.6
0Eh	UCBxTXBUF	eUSCI_Bx Transmit Buffer	Read/write	Word	00h	Section 19.4.7
14h	UCBxI2COA0	eUSCI_Bx I2C Own Address 0	Read/write	Word	0000h	Section 19.4.8
16h	UCBxI2COA1	eUSCI_Bx I2C Own Address 1	Read/write	Word	0000h	Section 19.4.9
18h	UCBxI2COA2	eUSCI_Bx I2C Own Address 2	Read/write	Word	0000h	Section 19.4.10
1Ah	UCBxI2COA3	eUSCI_Bx I2C Own Address 3	Read/write	Word	0000h	Section 19.4.11
1Ch	UCBxADDRX	eUSCI_Bx Received Address Register	Read	Word		Section 19.4.12
1Eh	UCBxADDMASK	eUSCI_Bx Address Mask Register	Read/write	Word	03FFh	Section 19.4.13
20h	UCBxI2CSA	eUSCI_Bx I2C Slave Address	Read/write	Word	0000h	Section 19.4.14
2Ah	UCBxIE	eUSCI_Bx Interrupt Enable	Read/write	Word	0000h	Section 19.4.15
2Ch	UCBxIFG	eUSCI_Bx Interrupt Flag	Read/write	Word	2A02h	Section 19.4.16
2Eh	UCBxIV	eUSCI_Bx Interrupt Vector	Read	Word	0000h	Section 19.4.17

19.4.1 UCBxCTLW0 Register

eUSCI_Bx Control Word Register 0

Figure 19-17. 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 19-4. 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 = Multiple-master environment
12	Reserved	R	0h	Reserved
11	UCMST	RW	0h	Master mode select. When a master loses arbitration in a multiple-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 = I^{2}C 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 = Device specific 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 19-4. 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.

19.4.2 UCBxCTLW1 Register

eUSCI_Bx Control Word Register 1

Figure 19-18. 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 19-5. 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 time-out select. Modify only when UCSWRST = 1. 00b = Disable clock low time-out 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 19-5. 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

19.4.3 UCBxBRW Register

eUSCI_Bx Bit Rate Control Word Register

Figure 19-19. 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 19-6. UCBxBRW Register Description

Bit	Field	Type	Reset	Description
15-0	UCBRx	RW	0h	Bit clock prescaler. Modify only when UCSWRST = 1.

19.4.4 UCBxSTATW

eUSCI_Bx Status Word Register

Figure 19-20. 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 19-7. 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

19.4.5 UCBxTBCNT Register

eUSCI_Bx Byte Counter Threshold Register

Figure 19-21. 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 19-8. 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.

19.4.6 UCBxRXBUF Register

eUSCI_Bx Receive Buffer Register

Figure 19-22. 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 19-9. 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.

19.4.7 UCBxTXBUF

eUSCI_Bx Transmit Buffer Register

Figure 19-23. 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 19-10. 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.

19.4.8 UCBxI2COA0 Register

eUSCI_Bx I2C Own Address 0 Register

Figure 19-24. 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 19-11. 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.

19.4.9 UCBxI2COA1 Register

eUSCI_Bx I2C Own Address 1 Register

Figure 19-25. 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 19-12. 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.

19.4.10 UCBxI2COA2 Register

eUSCI_Bx I2C Own Address 2 Register

Figure 19-26. 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 19-13. 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.

19.4.11 UCBxI2COA3 Register

eUSCI_Bx I2C Own Address 3 Register

Figure 19-27. 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 19-14. 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.

19.4.12 UCBxADDRX Register

eUSCI_Bx I2C Received Address Register

Figure 19-28. 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 19-15. 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.

19.4.13 UCBxADDMASK Register

eUSCI_Bx I2C Address Mask Register

Figure 19-29. 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 19-16. 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.

19.4.14 UCBxI2CSA Register

eUSCI_Bx I2C Slave Address Register

Figure 19-30. 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 19-17. 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.

19.4.15 UCBxIE Register

eUSCI_Bx I2C Interrupt Enable Register

Figure 19-31. 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 19-18. 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 time-out 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 19-18. 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

19.4.16 UCBxIFG Register

eUSCI_Bx I2C Interrupt Flag Register

Figure 19-32. UCBxIFG Register

15	14	13	12	11	10	9	8
Reserved	UCBIT9IFG	UCTXIFG3	UCRXIFG3	UCTXIFG2	UCRXIFG2	UCTXIFG1	UCRXIFG1
r0	rw-0	rw-1	rw-0	rw-1	rw-0	rw-1	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 19-19. 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 time-out 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 19-19. 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

19.4.17 UCBxIV Register

eUSCI_Bx Interrupt Vector Register

Figure 19-33. 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 19-20. 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 time-out; Interrupt Flag: UCCLTOIFG 1Eh = Interrupt Source: Nineth bit position; Interrupt Flag: UCBIT9IFG; Priority: Lowest

Chapter 20

SLAU445D–October 2014–Revised December 2015

Embedded Emulation Module (EEM)

This chapter describes the embedded emulation module (EEM) that is implemented in all devices.

Topic **Page**

20.1 Embedded Emulation Module (EEM) Introduction 573

20.2 EEM Building Blocks 575

20.3 EEM Configurations 576

20.1 Embedded Emulation Module (EEM) Introduction

Every device in this family 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 20.3 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 and 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 20-1 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™ IDE (CCS), see the application report *Advanced Debugging Using the Enhanced Emulation Module* (SLAA393) at www.msp430.com. Most other debuggers that support the MSP430 devices have the same or a similar feature set. For details, see the user's guide of the applicable debugger.

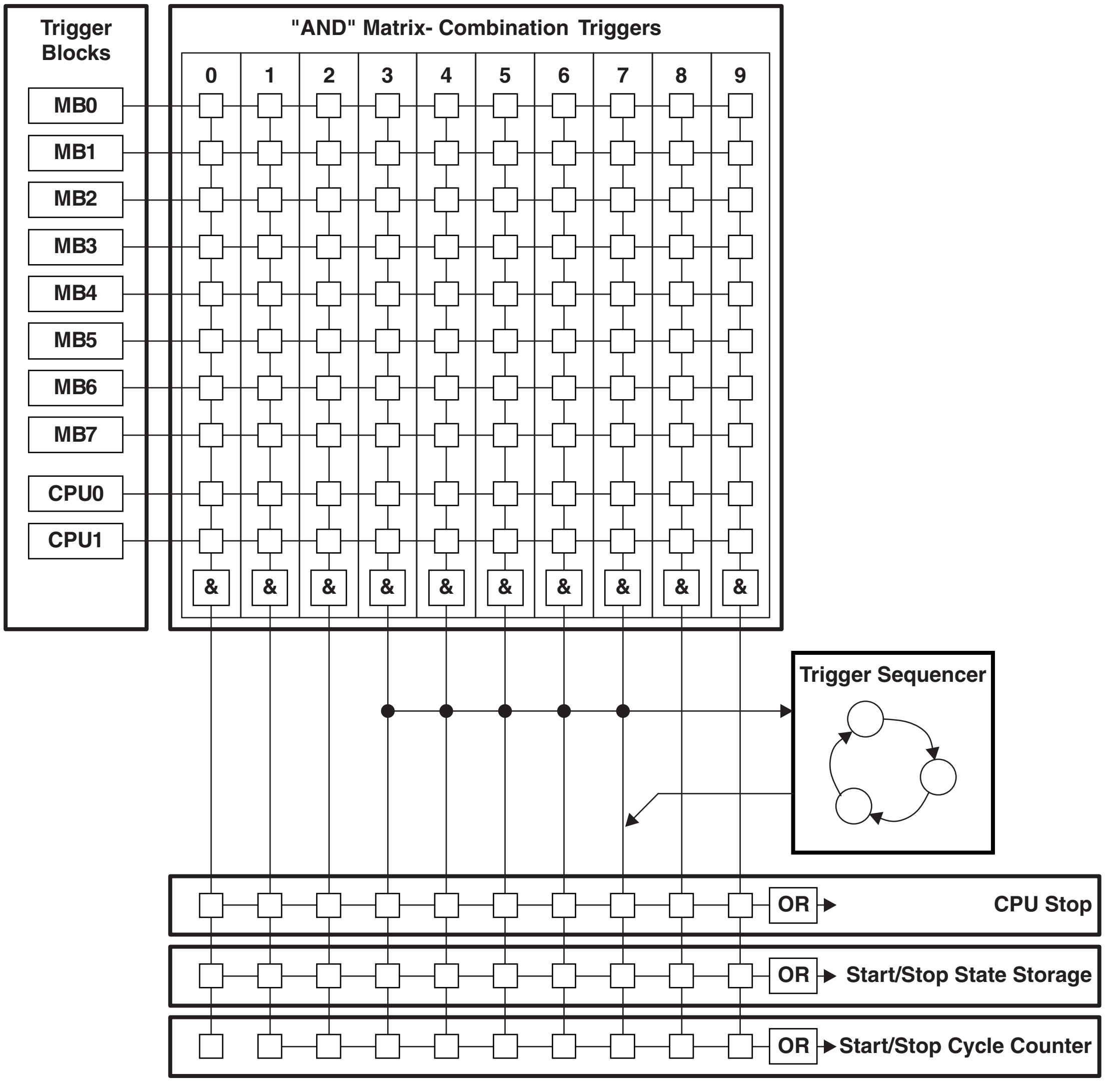


Figure 20-1. Large Implementation of EEM

20.2 EEM Building Blocks

20.2.1 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.

20.2.2 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.

20.2.3 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.

20.2.4 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.

20.2.5 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.

20.3 EEM Configurations

Table 20-1 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 the following documents:

Advanced Debugging Using the Enhanced Emulation Module (EEM) With CCS Version 4 (SLAA393)

IAR Embedded Workbench for MSP430 User's Guide (SLAU138)

Code Composer Studio for MSP430 User’s Guide (SLAU157)

Table 20-1. 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 November 6, 2015 to December 9, 2015	Page
• Added row for LPM4 to Table 3-1, *Clock Request System and Power Modes*	97
• Added Section 9.3, *CapTIvate Registers*	318

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